Transcript of Episode #780

Microsoft's 0-Day Folly

Description: This week we discuss the "Achilles" Snapdragon DPS flaw affecting more than one billion Android Smartphones, last week's third-largest Patch Tuesday in history, Mozilla's sadly uncertain future, the other shoe dropping after the ransomware attack on Canon, the nature of the so-called "software glitch" preventing California from accurately tallying Coronavirus lab test results, the significance of Microsoft's addition of their Control Flow Guard technology to the Rust and LLVM code bases, Threema's addition of video calling to their super-secure communications platform, a bit of closing-the-loop feedback, news of a SpinRite technology decision, and then we take a sad look at Microsoft's recent seeming unconscionable behavior with regard to the two zero-day vulnerabilities that were finally patched last week.

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-780.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-780-lq.mp3

SHOW TEASE: It's time for Security Now!. Steve Gibson is here. Lots to talk about. The biggest Patch Tuesday ever, again, for the third month in a row. We'll talk about that wild Qualcomm Snapdragon problem that is basically unpatchable on half of all Android devices. That's a billion devices globally. And then Steve's got a little spanking to do. Microsoft knew about a zero-day security flaw in Windows for two years, and just fixed it this week. It's all coming up next - what? - on Security Now!.

Leo Laporte: This is Security Now! with Steve Gibson, Episode 780, recorded Tuesday, August 18th, 2020: Microsoft's Zero-Day Folly.

It's time for Security Now!, ladies and gentlemen. Here's where we protect you online with this guy right here, Mr. Steve Gibson of GRC. Hi, Steve.

Steve Gibson: Yo, Leo. Great to be with you again on this hot California heat wave day.

Leo: It's hot hot hot.

Steve: Where we're trying to stay ahead of the rolling blackouts, which sort of make our technology...

Leo: It's kind of hard to do what we do without electricity.

Steve: Yes, it is. So we're at Episode 780. And, oh, I meant to check. I'm pretty sure that it was 15 years ago that we began this, and it's not a folly. "Microsoft's 0-Day Folly" is the title of this podcast, but I think this is the last podcast of Year 15, and that we'll be starting in on Year 16 next week.

Leo: Wow. Who would have thought? Who would have thought; you know?

Steve: So, yeah. We're going to discuss the issue that you mentioned; and you're right, it's a big one. You brought it up last week, and I had seen it, but I didn't grab it for the show. And that's the Achilles, as it's been named, Snapdragon DPS flaw, which as you mentioned affects more than one billion, with a "b," Android smartphones. We've got last week's third-largest Patch Tuesday in history, from which the show's title was derived. But that part of it we're going to put down at the end because I needed to do it justice. Also a note of Mozilla's sadly uncertain future. The other shoe dropping after the ransomware attack on Canon. The nature of the so-called "software glitch" that prevented California from accurately tallying coronavirus lab test results. I actually saw what that was because, you know, I'd heard about it on the news. And I thought, okay, well, you know, software, that happens. But our listeners will get a kick out of the nature of the problem.

We've also got the significance of Microsoft's announcement yesterday of the addition of their what's called Control Flow Guard (CFG) technology into the Rust and the LLVM open source codebases. I wanted to touch on Threema's addition of video calling to their super-secure communications platform. We've got two pieces of closing-the-loop feedback. I've got a little bit of news about SpinRite, a decision that was made a couple days ago after an extensive round of testing of some technology for SpinRite. And then we're going to take what is a sad look at Microsoft's recent seemingly unconscionable - oh, I guess I did know how to spell it - behavior with regard to the two zero-day vulnerabilities that were finally patched last week. But the nature of them and how long ago they were told is folly. And we're all, like, living with it. Well, you're not, Leo, because you've wisely left the ship.

Leo: I have got rid of Windows, yeah. I just - it's just too raggedy. All right.

Steve: And we have a fun Picture of the Week, which is apropos of many discussions we've had in the past.

Leo: I see. Some great stuff coming up. I can't wait.

Steve: Our Picture of the Week, the headline is "Tabs, Tabs Everywhere." And of course our listeners know what a tab fanatic I am.

Leo: You've a tab lover.

Steve: I am. And this was just so apropos because, you know, guilty as charged. We have a four-frame cartoon. The first one has this woman in a pink dress saying, "Hold on, I want to show you this thing online." And then the person she's showing it to says, "Oh my gosh! You have a thousand tabs open!" And she says, "I know. They're all critical."

And he says, "You're actually going to go back and look at all those tabs?" And she asserts, "Yes." And he says, "But there's a thousand..." And she interrupts him, and she says, "AND THEY'RE ALL CRITICAL." So I do have some affection for tabs.

Leo: And they're all...

Steve: I am hoping, actually, when we talk a little bit about what seems to be going on with Firefox, I'm hoping that the forthcoming vertical tab feature in Microsoft's version of Chromium, you know, their Edge...

Leo: Oh, don't say that. Don't say that. We've got to save Firefox. We've got to save it.

Steve: I know. I know.

Leo: There's no alternative.

Steve: I dislike the idea of having a mono browser culture.

Leo: Yup. Yup.

Steve: Yeah. So as I mentioned at the top, your pointer last week about the Snapdragon problem in Android is significant. And as I dug into it to explain it, to look at it, I got a good sense for what has apparently happened. So Check Point's security researchers decided to take a look at what's known as the cDSP, the computational digital signal processor, which is part of Qualcomm's Snapdragon system on a chip. The idea is, the so-called system-on-a-chip, one of the very expensive things to do is interconnect, to interconnect lots of small pieces of silicon with each other. It's mechanically expensive. It reduces reliability. And as our ability to reliably produce larger and larger slabs of silicon improved, it became increasingly practical to, instead of having lots of little chips, to just put the whole system on a single chip.

The problem originally was that, if a large wafer had too few large chip attempts, a single flaw anywhere within that large piece of real estate would bring the whole large piece of real estate down. Like the whole thing would die. So the original thinking was, well, let's do lots of small things. That way a flaw will only affect a much smaller piece of the component of the larger wafer. So but anyway, over time we got better at making things, and the incidence of problems dropped to the point where we can make much larger chips. And, yes, then you've got to cool them, and there's problems. But so that sort of introduced this system-on-a-chip idea.

So the best way to introduce our listeners to the many, and I mean many, like 400 security vulnerabilities recently discovered by Check Point security researchers, is to share the short teaser that they recently offered to potentially interested participants during the virtual Black Hat security conference a couple weeks ago.

They said: "Qualcomm Snapdragon SoC (system-on-a-chip) integrates multiple subsystems. Each one is customized for a particular application domain. Compute digital-signal processor (cDSP) is a subsystem which allows a mobile device to process simple sets of data with high performance for low power. In the talk we will show that this little-studied proprietary subsystem has many security problems that open the door to malicious Android applications for privilege elevation and denial of service (DoS) attacks on the device.

"For security reasons, this cDSP is licensed for programming by OEMs and a limited number of third-party software vendors." In other words, it's not just widely open. "The code running on the DSP is signed by Qualcomm. However, we will demonstrate how an Android application can bypass Qualcomm's signature to execute privileged code on the DSP, and what further security issues this can lead to.

"Hexagon SDK," they wrote, "is the official way for the vendors to prepare DSP-related code. We discovered serious bugs in the SDK that have led to the hundreds of hidden vulnerabilities in both Qualcomm-owned and vendor code. The truth is that almost all DSP executable libraries embedded in Qualcomm-based smartphones are vulnerable to attacks due to issues in the Hexagon SDK. We're going to highlight the auto-generated security holes in the DSP software and then exploit them."

So anyway, I did some digging, and I learned that Hexagon, which Check Point fingers as the culprit underlying these myriad problems, is the name of Qualcomm's proprietary DSP architecture, and the Hexagon SDK was created by Qualcomm. The documentation for the Hexagon DSP is freely available, and I took a look through it. But the development platform is available only under license from Qualcomm, which allows them to maintain some control over it and to only disclose the proprietary aspects to people who qualify.

So we have some sense for how there could possibly be some 400 problems - 400 - in the resulting code that's produced by this SDK. It must be that the SDK contains buggy libraries that work, but that are not very resistant to abuse. Of course we see that all the time in stuff we're talking about on this podcast. And it might also be that the compiler produces DSP code that has problems. That's certainly possible. They're probably translating C or something into RISC-based DSP-like code, and it might not be doing a good job. It could work, but consistently make that translation problematical.

And if that compiler was - if the compiler which they created to translate high-level code into DSP RISC code was also used to produce their libraries, which provide big chunks of subsystem stuff, then the libraries would be buggy, as now seems likely. And then that might help to explain how all this happened. That is, how you could actually have 400 problems is that the compiler was the core source. And so the libraries they offer in the SDK, which were compiled with this buggy compiler, they're all going to have problems. And any vendor code that used the SDK and compiled to this RISC architecture is going to be in trouble, too.

So that explains how something this bad could have happened. Check Point introduced their research by writing: "In this research dubbed 'Achilles'" - so this is Check Point, not the Black Hat conference stuff. So this is Check Point saying, like, disclosing what they could. They have been responsible, thank goodness, because this otherwise would be really bad. So they said: "In this research dubbed 'Achilles,' we performed an extensive security review of a DSP chip from one of the leading manufacturers, Qualcomm Technologies. Qualcomm provides a wide variety of chips that are embedded into devices that make up over 40% of the mobile phone market, including high-end phones from Google, Samsung, LG, Xiaomi, OnePlus, and more.

"More than 400 vulnerable pieces of code were found within the DSP chip we tested, and these vulnerabilities could have the following impact on users of phones with the affected chip. Attackers could turn the phone into a perfect spying tool, without any user interaction required. The information can be exfiltrated from the phone including photos, videos, call recordings, real-time microphone data, GPS and location data, et cetera." Basically, it's wide open.

"Attackers may be able to render the mobile phone constantly unresponsive, making all the information stored on this phone permanently unavailable, including photos, videos, contact details, et cetera. In other words, a targeted denial-of-service attack," meaning it's obvious with all these bugs to crash the thing. And malware and other malicious code can completely hide their activities and become unremovable. So sort of blanket rootkitness, bootkitness as part of this. So as you pointed out, Leo, last week, this is really bad.

So today somewhere in the neighborhood of a billion Android devices are vulnerable to these many hacks because this highly vulnerable Snapdragon DSP is embedded in approximately 40% of the three-plus billion Android devices in use today. This allows them to be turned into various forms of spying tools and malware carriers. The vulnerabilities can be exploited when a target plays an MP3, downloads a video, displays a web page, or renders any other content that is handled even briefly by the device's DSP. And the idea is that, as we know, in a mobile device, minimizing power consumption is crucial. So using a standard processor can be more expensive because you're using less specific generalized instructions to process media.

What a DSP is, is specialized instructions designed for processing media, which means they're able to do far more for less power than a general purpose processor that is able to do much more general things better than a DSP could. So the point is, essentially by creating silicon in each case best fit to what it's going to be doing, you're able to reduce the overall power consumption. Consequently, this DSP chip is brought to bear all over the place for all kinds of things, where you're able to sort of say, oh, rendering a JPEG, give it to the DSP. We've got some code for that in the DSP. Unfortunately, the code is buggy. And so if you know how the code is buggy, and you create a specially crafted JPEG, it could be malicious.

So that's where we are. And because the problem is, in this case a billion Android devices, is so pervasive and potentially devastating, Check Point research has withheld any publication of the technical details of these vulnerabilities. So the Black Hat presentation really was something more of a tease than a how to go do it yourself at home that afternoon. But this could not be unleashed upon an unsuspecting world. They plan, Check Point plans, to keep these secrets to themselves until it's safe for them to be released.

And frankly, that begs the question, when will that be? Because, as we know, many hundreds of millions of older Android devices are already past end of patch life and are riddled with many other known bugs, and we just added 400 more to the pile. So in Samsung's article titled - this is Samsung, the smartphone maker - titled "Understanding the Product Lifecycle of a Smartphone," they explain. "The product lifecycle of a smartphone is not just about physical attributes. It's heavily impacted by software in order to ensure security, reliability, and continuity. So when deploying smartphones, enterprise mobility managers should think about not only protecting the device itself" - meaning give it a good case - "but also whether it will continue to be supported by the manufacturer with ongoing firmware and security updates." Yeah, no kidding, Samsung.

So they say: "Making the right decision about which smartphones to deploy within your business is critical, and a major part of that is ensuring the model you choose will continue to be available and supported for business continuity. With the Galaxy Note 8 and Galaxy S9 Enterprise Edition smartphones, Samsung addresses this pain point, giving businesses an extra level of reassurance, thanks to its commitment to two years of market availability - from date of general availability - and three years of monthly security updates.

"The product lifecycle of any smartphone can be cut short when software updates don't happen regularly or are not fully completed. An enterprise's corporate mobile device policy should ensure that employees keep their smartphones updated at all times." And, finally: "One of the most important reasons to keep your smartphone's software up to date is to ensure that you have the latest security measures in place to protect against a wide range of cyberattacks. To help companies protect their investment, Samsung provides support for monthly security updates for three years from general availability."

Leo: Yeah, that's an increase. That's to match Google's three years, which is good, yeah.

Steve: Right, right. "With this setup, IT departments won't have to worry about running outdated security software or that they'll have to change their update protocol to continue supported versions." So what this means, though, is that there really is a drop-dead date for the use of phones. A smartphone, as we know, is a sophisticated, connected, pocket computer. And it's going to wind up containing and storing a significant amount of personal information, some financial, and probably a lot of logins to other sensitive services. So no such device should ever be purchased from some random low-end fly-by-night manufacturer who's not going to stand by it. It's just not worth it. And given the imperative of Google, Samsung, and the other high-end vendors, they regard a smartphone as having an expected life.

Now it's looking, as you said, Leo, Samsung is now matching 36 months. So I really think that everybody who's being responsible at this point needs to think in terms of a phone you buy new having a three-year use horizon. And really, when you think about the evolution that we're seeing in this technology, a three-year-old phone is lacking some of the new useful features that, I mean, that sort of seems reasonable. But the idea of using a phone past its end-of-security cycle?

Leo: Bad idea.

Steve: No. Yeah. And...

Leo: After three years these phones, yes, admittedly there's new technology, but there's nothing wrong with them. They'll operate for years to come. So we're creating waste, basically.

Steve: Right, yes.

Leo: And you don't want to hand it down to somebody because it's insecure.

Steve: Right. And think of all of the hackers who are salivating at the knowledge that Snapdragon has 400 problems. The community, the underworld community is going to figure out what they are, and there is a billion targets. We have to subtract out the updates that Google and Samsung and others who are being responsible will offer to their users. But that still leaves a huge base of maybe not high-value targets; but at some level, if you can get a lot of them, that's aggregated value. It's not good.

Speaking of not good, last week's Patch Tuesday, when ZDNet sums up last week's Patch Tuesday saying: "Microsoft says attackers have used a Windows zero-day to spoof file signatures, and another remote code execution in the Internet Explore scripting engine to execute code on user's devices," we need to take a closer look. And actually, those two things are the subject of the podcast that we will get to because it's just hard to believe what a closer look reveals.

But we have 120 new flaws in Microsoft's software fixed last week, making it the third-largest patch bundle of all time, topped only by each of the previous two months, with June and July weighing in with 129 and 123 fixes, respectively. This month's bundle carried a bit more urgency than usual since one of those 17 flaws which were classified critical was a zero-day under active attack at the time of the updates. And one of the remaining more than 100 flaws rated as merely "important" was also a zero-day being exploited in the wild and publicly disclosed, so not even secret.

The first of the two is titled - it's CVE-2020-1380, Scripting Engine Memory Corruption Vulnerability. Being a scripting engine problem, we should not be surprised to learn that the source of the trouble is IE11. It was reported by a researcher at Kaspersky Lab. And since it can be invoked by a malicious Office document, the belief is that it was probably spotted being used in a phishing campaign.

Microsoft had this to say about it. They said: "In a web-based attack scenario, an attacker could host a specially crafted website that is designed to exploit the vulnerability through IE and then convince a user to view the website. An attacker could also embed an ActiveX control marked 'safe for initialization' in an application or Microsoft Office document that hosts the IE rendering engine. The attacker could also take advantage of compromised websites and websites that accept or host user-provided content or advertisements. These websites could contain specially crafted content that could exploit the vulnerability."

In other words, anything that puts content on a website that is able to invoke IE, which we know they can, can do this. So keep this in mind when we get to the other end of this podcast because it's unbelievable what the history of this is. So that remains a threat to anybody who hasn't yet applied last Tuesday's updates to their installation of Windows 10, so obviously it would be good to do that.

The second zero-day, despite being actively exploited in the wild and publicly known, is only rated as "important," which seems odd since it is CVE-2020-1464 and labeled, somewhat innocuously, as a "Windows Spoofing Vulnerability." Okay. I suppose the scale of the problem should relate to what's being spoofed. The bug's description will catch your attention because it allows attackers to spoof the identities of other companies when digitally signing an executable. Now, that's the way the press covered it. We will get to the details a bit later.

In Microsoft's words, they said: "These spoofed signatures could allow an attacker to bypass security features intended to prevent improperly signed files from being loaded." Now, all of this is a bit of a misdirection because the signatures are actually not being spoofed. I'll explain that later. So this, too, is not good; but we'll cover the details at the end.

Beyond those two zero-days, five of the other critical bug fixes are for Microsoft's Windows Media Foundation, the multimedia framework and infrastructure which has been used to render digital content ever since Windows 7 and since Windows Server 2008. In these cases, successful exploitation would allow an attacker to install malicious software, manipulate data, or create new accounts.

And among the rest - because, again, we had 120 to choose from - there's also 2020-1046, another nasty one in the .NET framework affecting versions 2.0 through 4.8. It's a Remote Code Execution flaw in the way .NET handles imports. An attacker could exploit this vulnerability to gain admin-level control of the vulnerable system. This vulnerability would be exploited by uploading a specially crafted file to a web app, which is of course not a heavy lift these days. There's all kinds of web apps that are saying, you know, that involve uploading user-submitted stuff. This allows that to be exploited. So as always, don't wait too long before you make time to allow Windows 10 to eliminate another 120 previously obscure but now known flaws from its codebase. And as I said, I will have much more to say about these two zero-day flaws, since they have become the topic of today's podcast.

Speaking of Mozilla, in pre-COVID January of this year, Mozilla cut 70, seven zero, people from its workforce of approximately a thousand. And now, citing the need to respond to COVID as the impetus, another 250 have been laid off. Mozilla said that the primary casualties of last week's layoffs were the developers working on the company's experimental Servo browser engine and Mozilla's threat management security team. The threat management security team is the team that investigates security reports and performs incident response. The team that fixes bugs in Mozilla's products is still in place, according to sources and a Mozilla spokesperson.

So going forward, Mozilla said that they will be rethinking their core business model, whatever that means, and putting more focus on financially viable products. Mozilla Corporation's CEO and the Foundation's chairwoman said: "Recognizing that the old model where everything was free has consequences means we must explore a range of different business opportunities and alternative value exchanges." Again, okay, whatever that means. She said: "We must learn and expand different ways to support ourselves and build a business that isn't what we see today."

So some assume that this might include a stronger focus on their new VPN offering. They have high hopes for that, as we covered it when it was formally launched last month. It turns out that VPN apps and services are one of the biggest moneymakers in tech today; and, despite having arrived late to the game, Mozilla believes that they may be in a good position to leverage their reputation, their strong "privacy first" position as a civil and privacy rights advocate. So clearly they're looking around for something they have of value to keep them afloat. And unfortunately - sorry?

Leo: I think they don't do their own VPN. I think they're relabeling a third-party VPN, which doesn't really mean much at all. It just, you know...

Steve: Yeah. Doesn't say exactly; right. And of course adding further uncertainty to the mix is the fact that Mozilla's contract to include Google as the default search provider inside Firefox is expected to be expiring. Well, the existing contract is expiring later this year, and so far it has not been renewed with Google.

Leo: Yeah, that's a big deal. That's most of their revenue, yeah.

Steve: Exactly, 90%. The Google deal has historically accounted for 90% of Mozilla's revenue. And without that, Mozilla's future is uncertain, to say the least. And we talked about this at the top of the show. I strongly dislike the idea of having just two dominant browsers, pretty much two browsers at all in the world, Safari and Chromium, since everybody else except Firefox is Chromium-based.

Leo: Best solution, everybody use Firefox. You want to support it, use it because they make money when you use it. You don't have to buy something, just use it.

Steve: Yeah. And so I'm helping them that way. I know you are.

Leo: That's my favorite, yeah.

Steve: Yeah. As we've noted, the browser has become most people's effective operating system. I know, Leo, when someone calls you on the weekend during your Tech Guy radio show, and asks whether they should get a Windows machine or a new Mac, you know, you first ask them what do they need a computer for at all?

Leo: Really, get a Chromebook. You're using the browser.

Steve: And then you say, "Get a Chromebook." Because you're right, that's all they need. And a Chromebook is basically a bootable Internet browser. So my point is that...

Leo: By the way, that might be one of the things that's hurting Firefox is iPad and Chromebooks, neither of which use Firefox.

Steve: Yeah, yeah. So browsers have grown insanely complex, with so many features and bells and whistles, and they are so prone to attack because they are the piece of ourselves that we stick out there on the Internet.

Leo: By the way, just a couple days ago, they did do that deal with Google. So the default search engine within Firefox until 2023 will be Google. So at least they've got that.

Steve: Yay.

Leo: That's worth between, I think, 400 and 500 million a year.

Steve: Wow. Good. So maybe they just need to scale their ambitions down. I don't understand how COVID affected them, but that was what they said. So, good. Let's hope they stay around. And I did have a note in our Miscellany section which simply observed, because I thought this was sort of interesting, that Firefox Send is still offline. I went there to check yesterday, and yup, it just says we're temporarily offline while we retool. And clearly they have other things on their minds at Mozilla. So I wouldn't want to hold my breath for that free service returning. It was my favorite, but I may need to return to Filemail, which is what I'd been using before that.

Leo: Steve has been refreshed with a long...

Steve: And I imagine Canon wishes they had Barracuda.

Leo: Oh, no.

Steve: Checking their email.

Leo: Is Canon the latest?

Steve: Yeah, well, remember we talked about it last week. They got hit, Canon got hit by ransomware and had all their properties offline. Remember there was the loss of their customers' images at image.canon.

Leo: Oh, that's right, that's right. Oh, forgot about that.

Steve: What you were saying last week as being even worse than encryption is what has happened. Their proprietary files are leaking.

Leo: They exfiltrate and encrypt.

Steve: Exactly.

Leo: Double punch.

Steve: Exactly. When ransomware began, one of the things that was so clever about it was that it did not need any bandwidth because it would do encryption in place, and many places there just wasn't connectivity. There wasn't bandwidth to allow exfiltration. Well, that problem no longer exists, especially at major large targets. So the Maze ransomware gang have apparently started publishing data which they allege was stolen from Canon. And once again, BleepingComputer is breaking the news that Canon may have elected not to pay the ransomware.

Lawrence Abrams at BleepingComputer wrote: "As Canon was able to restore many systems in a short time, BleepingComputer believed that they had paid the ransom. It looks like we were wrong, as Maze has started to publish Canon's stolen data, which is only done after a ransom is not paid." In this case the published file is a 2.2GB ZIP archive titled StrategicPlanningPart62.zip. BleepingComputer was told it contains marketing materials and videos, as well as files related to Canon's website.

And Lawrence indicated that, from the small number of samples they had reviewed, they believed that the files do appear to belong to Canon USA. A source who reviewed the archive stated that they do not appear to contain any financial information, employee information, or other sensitive data. So BleepingComputer reached out to Canon for comment; but, not surprisingly, did not hear back. But of course there is the carrot of pay us, and we'll decrypt your machines. Then there's the stick of, and if you don't, we're going to hurt you further.

Leo: Wow. Wow.

Steve: So that's sort of the, okay, well, you think you're fancy because you've got backups. Well, so do we. Yikes.

Okay. I just got a kick out of this one. This is a quickie. What was the software glitch in California's widely reported COVID case reporting problems? As a Californian, I noted with passing interest the several times I heard in the news that an unspecified "software glitch" was preventing the accurate reporting of somewhere between a quarter million and 300,000 COVID lab test results for the State of California. From time to time we've noted when - okay, are you sitting down? - an expired web server certificate catches its owners by surprise, that can be embarrassing. And believe it or not, this was the underlying cause of California's recent trouble.

Leo: Oh, my god.

Steve: I know. The certificate for the server that the third-party labs such as Quest and LabCorp use to upload their lab test results expired when no one was paying attention. Whoops. And I should note, too, for the record, that I have an expired certificate. The revoked.grc.com cert expired a few months ago, and I haven't replaced it with another revoked cert because now it's not only revoked, but it's expired, so no one really cares that it's revoked because it's expired. And I haven't been, or hadn't been monitoring that facility that I created years ago. And I didn't realize that the site's page was still today getting an average of 561 visitors who were wanting to play with certificate revocation. But recently I've been receiving notes from people who miss having it working the way it's supposed to.

So I'll obtain a new cert from my favorite certificate provider, DigiCert, and then have them immediately revoke it. And frankly, now, here we are in mid-August. It's the perfect time to do that since it's at the end of this month, starting September 1st, that no certificate issued after August can have a lifetime of greater than a year and a month. So I'm happy to be doing that now because it gives me an extra year before I have to think about this again, which for a site that I'd sort of, you know, it's sort of a backwater site, I'm happy not to worry about it for an extra year.

So for anybody else who's listening who might have certs that are going to be expiring sometime, if you care, you can renew them. Most certificate providers, I know that DigiCert does, will credit you with the available remaining life on a cert if you renew early. And if you do have the not-valid-before date this side of September 1st, 2020, then that certificate can have as long a life as it wants, as long as the provider will issue to you. And you don't have to worry about it a year from now. Everybody else falls into the one-year plan starting the end of this month.

Yesterday, Monday the 17th, Microsoft announced that they had completed the work of incorporating their so-called Control Flow Guard technology (CFG) into both the Rust language and the LLVM open source projects. For the significance of this to be understood, let's review a bit. First of all, the Rust programming language is, as we know, rapidly growing in popularity. Its syntax is very similar to C++, and it provides strong memory safety without the need for the garbage collection employed by dynamic languages. And it's now Microsoft's language of choice for implementing secure and safety critical software components.

And, you know, it would be nice, maybe, if Microsoft sent some money Mozilla's way because Rust was originally designed by the engineers of Mozilla, who refined its definition while they were writing Mozilla's Servo layout browser engine and their Rust compiler. Rust is free and open source, and it's dual-licensed under both the MIT and the

Apache license 2.0. So the fact that that's getting CFG is very cool, and we'll get to that in a second.

The other major project that Microsoft just finished enhancing with this Control Flow Guard technology is LLVM. It's best thought of as a compiler infrastructure which provides a set of tools on the front end for implementing new languages, that is, for understanding a new type of language; and on the back end for essentially understanding a new type of computer for compiling to a new computer instruction target, instruction set target. And then in between there's also the so-called "intermediate representation," which provides a language-neutral and instruction set-neutral, so that is to say it's neutral facing the front to the language and facing the back to the code emission. It's an instruction set-neutral representation which serves as a portable high-level assembly language that can be optimized through a variety of transformations over multiple passes.

Leo: It's really cool.

Steve: It is.

Leo: LLVM is very future forward.

Steve: It's the way to do this, yes. It's the way you solve this problem. So although LLVM was originally designed to implement C and C++, its language-agnostic design has allowed it to be the underlying technology for, in alphabetical order, ActionScript, Ada, C# and of course C and C++, Common Lisp, Crystal, CUDA, D, Delphi, Dylan, Fortran, the Graphical G programming language, Halide, Haskell, Java bytecode, Julia, Kotlin, Lua, Objective-C, OpenCL, several different SQLs, Ruby, Rust, Scala, Swift, and Xojo.

Leo: Everything.

Steve: Yeah, exactly.

Leo: It's Chris Lattner. He's just a genius.

Steve: This is the way you do that now.

Leo: It's truly amazing, yeah.

Steve: Yeah. So in other words, bringing security improvements to the LLVM project automatically imbues all of those LLVM-driven language compilers with the enhanced security provided by Microsoft's Control Flow Guard. And so after this point, when CFG is enabled in the compiler, anything emitted by any of the compilers of any of those LLVM-based languages will contain CFG's protection. Which brings us to what is the enhanced security provided by Microsoft's Control Flow Guard? This will probably sound familiar to at least some of our listeners because we have discussed this at some length back when it was proposed and introduced by Microsoft.

As we know, one of the ways attackers attack is by arranging to cause a victim program's path of execution to vary from the way it was designed to, to the way the attacker wants it to. We've learned from the effectiveness of so-called Return Oriented Programming that a surprising amount of damage to the system's security can be caused simply by jumping near the end of a subroutine to execute existing authentic and authorized code, but doing that out of order. Essentially, the attacker knits together the code they want to execute out of existing code which is authorized to run.

So this bypasses the previous strong protections offered by preventing data on the stack from being executed as if it were code, or data in other non-stack data regions such as communication buffers from being executed as code. We have the so-called "no execute bit" which is added in the memory managers to mark these regions of memory as data so that the processor will refuse to execute any data in there. So we solved that problem. And so we said, okay, yeah, great, problem solved. And of course then the attackers said, okay, you think you're so smart, we're going to execute your own code to accomplish our nefarious goals. Thus Return Oriented Programming was born.

So how do we prevent that now? A functional subroutine has a so-called entry point. This means that, although it might be used by a great many different parts of a system, everyone who uses it jumps to its so-called entry point. In modern languages, the beginning of a function does a bunch of critical setup work. If the function defines some local memory working variables, as most do, then some space must be set aside on the stack to hold them. And since some functions may require not to modify any of the caller's registers, some of those that might be needed by the function for its own working storage will have to be put on the stack and kept safe while the function uses them.

So because of all this, it's not possible for a program to simply jump into the middle of a function and bypass or miss that setup. Everyone must enter through the front door so that the function's local environment can be properly initialized. But deliberately not entering through the front door is precisely what Return Oriented Programming does to abuse the system. So in other words, no valid code will ever jump anywhere but to a function's entry point. Only malicious code will ever do so.

Therefore, if we had some means of detecting any attempt by a program to jump into the middle of its own code, even when it thinks it wants to, since that never happens by design, we would be able to stop Return Oriented Programming abuse in its tracks. And that's exactly what Microsoft's CFG (Control Flow Guard Technology) does. CFG operates by creating a per-process bitmap where a set bit indicates that the address is a valid destination for a jump. And just before jumping to any function's entry point, the application checks with this bitmap to see whether the destination address is represented by a one bit set in this allowed jump destination's bitmap.

If the destination address has a zero bit in the bitmap, the program immediately terminates. This makes it much more difficult for an attacker to exploit many of the so-called gadgets that are used, such as use-after-free flaws that we're often talking about. It's not perfect protection, and it does come at the cost of some overhead. But for security-sensitive code it probably makes sense to have it turned on, and now it can be, since it further raises the bar to complicate the task of malicious subversion.

As of Windows 10 Creators Update, which was at v1703, the entire Windows kernel has been compiled with CFG on, in place and running. And the kernel uses the Hyper-V technology to prevent malicious kernel code from overwriting the CFG bitmap. So even malicious code in the kernel that has kernel privileges is unable to mess with that CFG bitmap, since obviously that would be a way to give aberrant code permission to jump where it should not.

So yesterday Microsoft announced that Rust and the LLVM Project now has this technology, and it certainly represents a very nice step forward. So although I'm going to be very tough on Microsoft at the end of this podcast, I certainly want to give them props for this because I would argue that authors should compile their LLVM-derived code both with CFG on and with it off, and see if they sense a performance hit. You can selectively turn it on only in, for example, in security-sensitive modules. Like if you had a larger subsystem, you might turn it on in the communications portion, which is going to be handling potentially unsolicited communications, or maybe in some media rendering portion, if you can afford whatever performance hit it might create.

Anyway, I just wanted to say, you know, give Microsoft props and make sure that developers know that it is now part of Rust and all of the LLVM-derived compilers. You might experiment with turning it on and getting some security against the malicious abuse of your own code that way. Nice piece of work.

Threema gets end-to-end encrypted video calls. I wanted to mention that Threema added WebRTC-based end-to-end encrypted video calling to their existing text and voice messaging platform. And, you know, I'm still tempted to say that only when users manage their own keys can they truly feel that the solution they're using is as secure as it could be. If someone else, as I've said before, is managing those keys for you to make the process more automatic and convenient, which it inarguably does, and if the keys are the key, as they are, then true security is necessarily reduced.

Signal offers optional key verification, as we know, which is nice. But it's not enforced. Since Threema makes key management entirely the user's responsibility, it arguably offers a modicum of improved security at the cost of some convenience. Now, the problem with my argument is the classic "weakest link" security dilemma. If you or your counter party at the other end of the connection are using Threema on a five-year-old Android phone that hasn't been updated in ages, then it really doesn't matter how good the encryption and the keying of the link might be. You're already hosed. An attacker won't care what crypto tech you're using. They'll just siphon off the entire conversation outside of the encrypted tunnel, thank you very much.

So, okay, sure. But I like Threema. And if I needed to truly hold a private video or messaging call, if my work mode had me doing that, if I needed to have really secure video conferencing and messaging, local or transcontinental, and I wanted what I felt was the best security possible, I really do think Threema would be my choice. It's so clear that they're doing the best job possible.

Here's how they explained their news. They said: "After a thorough beta test, video calls are now ready for everyday use. Simply switch on the camera during a Threema call, and your contact will see you in stunning image quality, while you can rest assured that no one else is able to access the video. Since video calls contain personally identifiable information of the purest form, they are particularly worthy of protection. As to be expected," they wrote, "Threema's video calls are designed from the ground up with security and privacy in mind. Not only the transmitted data but also signaling is fully end-to-end encrypted; and before a call is accepted, no data is transmitted.

"When it comes to metadata, video calls also meet Threema's rigorous standard. In order to ensure full end-to-end encryption of all metadata, including real-time metadata such as camera orientation, our team had to make corrections to the widely used base technology, WebRTC. This security improvement will be incorporated into the WebRTC project, meaning that countless other communication services benefit from our patch in the future. Technical details concerning the security aspects of Threema's video calls are documented in the Cryptography Whitepaper." And they have a link in their announcement.

So, yeah. Those are the people whose technology I would trust. And note that the improvements they were forced to bring to WebRTC so that it would be able to meet their standards for real-time communication privacy meant that all of the previous so-called "secure applications" which also use WebRTC, claiming total privacy and encryption, were not truly providing that. This is why Threema would be my choice. So now with video.

Two little pieces of feedback. Timo Gruen, who signed out as Tim Green, he said: "Hi, Steve. I desperately want to participate in the SpinRite beta, but I can't afford $20 a month to subscribe to a news server just to get into your newsgroup. Is there any other way to get the betas and send you feedback? I haven't been able to find any information at all about the beta on your website. All the best, Tim Green."

So a couple of quick clarifications. First of all, the GRC newsgroups are wholly self-contained. It is true that NNTP - Network News, what is it, Transmission Protocol? - NNTP, TCP over port 119, that does form a global network of connected NNTP servers, and a client is able to obtain access to a server like the kind of $20 a month news server that Tim's talking about, and then gain access to that global newsgroup system. You won't find any of GRC there at all. We proactively block the leakage of GRC's stuff out onto the global NNTP network, the reason being, if our postings were going out, then people would respond, and we would never know. We would never see them.

So what GRC runs is an island. It's its own NNTP news server, which you don't need to pay anything to use. It's at news.grc.com, which you can access with Thunderbird that has a built-in NNTP client, you know, Mozilla's Thunderbird; or Gravity, my favorite client, or any other. There's a whole bunch of clients. If you go to GRC.com/discussions, there's information there, a list of all the clients we know of, links to how to get started and going. So it doesn't cost anything.

However, there's no beta. We're not there yet. I would say the technology is in beta, and I'll have something to say about that in a second. But there isn't a SpinRite beta that is yet available. There will be. And existing SpinRite owners, anyone who has SpinRite 6, will be able to get access to the beta at no charge while it's getting packaged up and ready for release. I'm not going to waste any time doing that, but it'll take some time.

And SpinRite owners in the newsgroup will at some point switch from using the freeware that we're using right now, this benchmark I've been talking about, to actually testing a pre-release version of SpinRite. And I don't know how long it'll be between then and its official release, but there's no reason not to let people play with it. So everyone who owns SpinRite will be able to use their existing serial number and transaction code in order to download it.

And this second one was just - I just got a kick out of this, Leo. Max, he said. "@SGgrc Steve, I have an air filter" - and I'm presuming he means like for his home HVAC, you know, his heater and air conditioning. "I have an air filter with a BUILT-IN," in all caps, "Bluetooth IoT device. An air filter." He says: "I can't believe I just paired my phone with an air filter." So I thought that was kind of cool. I did a little looking around. It turns out there is one. It's this little pod sitting on the middle of your classic heater/air circulating system in your home.

I didn't look to see anything more about it, but I would imagine, I mean, if it were my Bluetooth IoT device that I was designing for an air filter, I would be looking at the differential pressure from one side to the other. And because the point is that as that air filter becomes clogged over time with just particulate dust, which is why you have it in the first place, the differential air pressure from one side to the other will increase because the filter will be blocking more from the suction of your HVAC, and at some point it could send an alert to your phone, which is presumably what it does, to say, "Time to change me, I'm getting dirty." So anyway, I thought that was a little cool bit of IoT technology.

And a little bit of an update on SpinRite. I was talking last week about the two different ways of reading a disk, the standard way of actually doing a read, which is what we've been benchmarking. But one of the things that I started to benchmark was the concept of a verify, that the verify command is used instead of the read. Essentially you tell the device that I want you to verify that you can read the following block of sectors, but don't bother sending it to me.

Well, so for a hard disk on a SATA link, that doesn't buy you much because the maximum transfer rate of today's state-of-the-art hard drives is slower than a SATA III link which is 6GB per second, or about 800MB per second. No hard drives are able to actually physically transfer at 800MB per second. They can't get there. Is it 800 or 600? Maybe it's 600MB per second. I think it's 600MB per second. I'm just pulling this off the top of my head. But yes, it's six. Still, they're not there.

Not so necessarily with SSDs. In theory, if the SSD is parallel enough in its core, that is to say, if when you ask for a bunch of sectors, those sectors are all coming from non-volatile memory spread across the physical real estate, and if that memory is fast enough, it is theoretically possible for today's SSDs to exceed the speed of SATA III. And in fact that's why there's the next generation way of talking to solid-state memory, so-called NVME, Non-Volatile Memory. That's a different interface that can keep up with today's SSDs. SATA III in some cases can't.

So the allure here was that this next forthcoming SpinRite might be able to detect if an SSD was actually doing a verify, and then just say, okay, verify yourself, and we're going to sit back and wait while you do that because, although you can't tell - you can only tell it to verify 32MB at a time, which is 65536 sectors. And actually now we learned last week you really only can do half. But I'm able to be much faster anyway, so we didn't lose any performance in cutting it down to 16K sectors, or 16MB transfers. No loss of performance there.

So what the benchmarks seemed to be showing was really crazy high performance from SSDs, which was compelling because it would mean that SpinRite could have the SSD do the work. And as soon as it coughed, then SpinRite would zero in on the problem and get to work there, and then tell the SSD, okay, keep going. It turns out that the only, well, we always knew that the only way this would be possible, if we could absolutely guarantee that the SSD internally was truly doing the same work. Then we could rely on that.

So late last week I developed the technology to test this. I first asked everyone's drive to read one past its end and verify that it got an error; then to recover from the error; then read the last sector and verify that it didn't get an error; then save that data in case we need to restore it. Then I deliberately caused that sector to be unreadable. There's a command you can use to do that. In the old days, there were so-called "long reads" and "long writes" where you could tell the hard drive, just give me all of the data. Don't worry if you can't correct it. I want it anyway.

Those days are over. But it is still possible for maintenance purposes to tell a drive you want to mark a sector as unreadable. So this test code marked the last sector as unreadable. Then it tried to read it, and it verified that it could not. Then it tried to verify it, and it verified that it could not. Some drives said, yeah, it's fine, meaning that verify was a no-op. They weren't actually doing the verify.

So we found some drives did not honor the mark unreadable, so we would be unable to check those. Some drives that apparently did, apparently they said, yeah, no problem, I marked it unreadable. Then both the read and the read verify succeeded, which meant it didn't actually mark it unreadable. In some cases, the verify succeeded and the read didn't.

Anyway, what we learned was it was a mess, and SpinRite cannot use it. So the good news is we're still going to be screamingly fast. The bad news is we're not going to be able to go faster than a SATA III link. On the other hand, the good news is a SATA III link can go at 600GB per second. So SpinRite is still going to be extremely fast. So that's where we are. Great round of experiments.

And I'm at this point - I talked about, I remember you chuckled, Leo, when I said that for a couple hours last Sunday there were no known problems. Well, we're back there now. It is running flawlessly on every drive that everybody has on every system that they've got. I think that the AHCI driver is nailed. I've got bulk transfer working at the capacity of the SATA link in every case. I've got now, as a result of this recent set of testing, I've got sector-level error induction and recovery and retry and all that.

So at this point I will now integrate all this with the work I did on the bus mastering IDE, and the AHCI when it's set to its legacy compatibility mode, and then pull this into a final benchmark. And when it's ready, I will let all of our listeners know where they can go get it in order to play with it. And of course my actual motivation is to get a much larger audience to give it a try so I can find problems now. And of course their motivation for trying it is, if they find a problem, they want me to fix it because that will mean that SpinRite will be guaranteed to work for all the stuff they've got, as soon as it's released.

So a win-win for everyone. And, well, except for people who hope that Microsoft will keep doing a good job. We are, after our last break, going to talk about Microsoft's zero-day folly.

Leo: Not a win for them.

Steve: Boy.

Leo: So we were talking about Threema, and I know that you've always liked Threema a lot because of the way they handle keys. It isn't open source. But I was reading up on it, and they have a kind of an interesting way of verifying that they are using the NaCL, open source NaCL library properly. So it's a complicated system, and you have to compile and install a C++ program, open source program that you can then verify. But you're satisfied...

Steve: To duplicate what they're doing, yes.

Leo: Duplicate the encryption. So are you satisfied, though, that even though they're not fully open source - I kind of wish they were, like Signal is - that they are sufficiently open about how they're doing it to be safe? Obviously you must.

Steve: Yeah. And they're not free; right? You have to pay them something.

Leo: That's how they do it, yeah, you pay for it.

Steve: And so that's what I like.

Leo: Right.

Steve: Yes. I think rather than the problem Mozilla is facing, they said, look, we're going to give you really, really, really good security and ask for a little bit of one-time payment.

Leo: It's not expensive, yeah.

Steve: No, it's not.

Leo: And it's one-time only, yeah.

Steve: Yeah.

Leo: And I know you like the three dots, where you shift keys.

Steve: Well, yeah. What I like is that the Threema comes from the three levels of verification, which is, okay, we have encryption, but we haven't verified. We have encryption and we have verified. And then the third one is we actually have the two devices looking at each other in the same physical location and did like a real-time meeting verification.

Leo: Right.

Steve: So, yeah.

Leo: You have to have Threema signing parties. The reason I ask is I'm becoming less and less happy with Signal. You know, the fact that they require to tie it to a real phone number I think is not good. They're starting to do things that I feel like - and I guess it's for monetization reasons. But for whatever reason I feel like they're a little bit less privacy focused. So maybe I'll, I mean, Signal's a gold standard. I don't think a lot of people know about Threema. I don't hear a lot of talk about it.

Steve: Right. And that's of course the problem. It's easier if you say, hey, let's use WhatsApp. It's like, oh, okay, fine.

Leo: Yeah, everybody knows it, yeah.

Steve: And then next down is - or just, you know, what iOS has built in, iMessage. We know that it's as secure as Apple can make it, except that, you know, they're managing the keys.

Leo: If I showed my Threema, I guess it's a QR code, on camera here, would that be the equivalent of an in-person, I mean, people could presume I'm me.

Steve: I think it would be. I think that, as I remember, the second level is you read a signature, like a short token that's created from your password, and then the third - it's been so long since I've looked. But, yeah, I think if you held up your, you know, what you're showing them is your NaCL public key.

Leo: Presumably they know it's me.

Steve: Yeah.

Leo: And then you have a Threema ID which is uniquely generated, but it's not tied to any phone number, which I really like. You don't have to link a phone number to it at all.

Steve: Right.

Leo: I think that's really cool.

Steve: And of course NaCL is the library.

Leo: Yeah.

Steve: It's the one, you know, I built SQRL around, and it's the one that is being used by everybody.

Leo: I trust NaCL. My issue would be, because there is around it a binary blob, you know the encryption's now good, but you don't know what else is going on. You could in theory have a binary blob that decrypted it and sent it to some third party, even if they're using NaCL and doing it properly; right?

Steve: Sure.

Leo: Yeah, that's the problem with binary blobs. Well, I'll put my - here's my QR code if anybody wants to make me a three dot. Just take a picture of that with your phone. It has to be me because here I am. It's like you're in person. It's like you're in person. I've had Threema forever.

Steve: It's better than being in person.

Leo: I'm practically right here.

Steve: Because it's COVID safe.

Leo: You don't have to wear a mask. And now let me just get my paddle out.

Steve: Oh, yes.

Leo: It's time for the flogging.

Steve: It's difficult to know what to think of the fact pattern I'm going to lay out. As we've noted before, it's one thing to make an honest mistake. Anyone can. It happens all the time, to all of us. But when a company who we depend upon, like Microsoft, appears to be deliberately acting irresponsibly, that's a whole 'nother class of problem.

So the story begins last December, 2019, when Microsoft received a report for a serious vulnerability in Windows. And it didn't come through some unknown channel. It was relayed from an anonymous source through Trend Micro's official Zero Day Initiative (ZDI), which we've often talked about. These guys are the real deal. And Trend Micro was quite patient, and gave Microsoft a full six months to fix the problem. Microsoft did nothing. So after half a year of inaction from Microsoft, ZDI, as their policy permits, published an advisory on May 19th of this year.

And because this was the real deal, a serious vulnerability that Microsoft had for whatever reason chosen not to act upon for six months, the now public vulnerability was exploited the next day in an effort now called "Operation PowerFall." An advanced threat actor exploited one of these two zero-day vulnerabilities that Microsoft patched last week in a targeted attack a few months ago. That adversary chained two flaws in Windows, both which were unknown at the time of the attack, to achieve remote code execution and increase their privileges on a compromised machine.

As I noted above, this occurred in May and targeted at least one South Korean company. We have no way of knowing who else might have been victimized by Microsoft's six months of inaction on this. But Kaspersky picked up on at least this one attack, and they believe that it might be part of an operation known as DarkHotel. This is a hacker group that likely is operating in one form or another for more than a decade.

Fortunately, Microsoft did then patch this now being actively exploited flaw the next month, three months ago, in June's Patch Tuesday. The so-called Operation PowerFall attack leveraged a remote code execution vulnerability in IE11 that I just talked about earlier, which has just now been patched, and a flaw in Windows GDI Print and Print Spooler API which is what enabled a privilege escalation. That's the one Microsoft sat on for six months and did nothing.

So this should strongly reinforce the idea that privilege escalation flaws are not nothing. We've looked at this before. We've talked about this before. The sequence of events suggests that the attackers already knew of the IE11 remote code execution flaw which was patched last week. The elevation of privilege, that was three months ago. So the evidence suggests that they knew of the IE11 remote code execution flaw, and they were sitting on it because by itself it was impotent and useless without an accompanying escalation of privilege vulnerability. Getting the privilege of the process under which IE was running wouldn't help them. They needed to get root, and so they needed both a remote code execution flaw and to be able to elevate their privilege once they were running code.

So the instant they obtained the news from ZDI's disclosure, having waited six months, of an unpatched privilege escalation, they had everything they needed to launch the attack. If someone inside Microsoft thought, oh, well, it's only a privilege escalation, we don't need to worry about that much, that person should be relieved of their responsibility for making such determinations. And believe it or not, that's not the worst.

I started out by saying it's difficult to know what to think about the fact pattern I'm about to lay out. Well, here's another one. Part 2 is even more egregious. Remember that other zero-day that was actively being exploited in the wild, which Microsoft referred to as a "spoofing vulnerability." Okay? It allows Microsoft installer, you know, .msi files, to be converted into malicious Java executables while retaining a legitimate company's digital signature. That's what they're calling a "spoof." And it's no laughing matter. Believe it or not, this bug was reported to Microsoft precisely two years ago, on August 18th, 2018. Today is August 18th of 2020. Microsoft has known about this for two years, and they stated that they would not be fixing it. What? Yes.

They finally fixed it last week because it was being actively exploited in the wild. So get this. Two years ago, actually a little more than two years ago, a viral sample of an exploit that researchers at the time dubbed GlueBall was first uploaded to VirusTotal. The fact that it was uploaded to VirusTotal means that it existed and was weaponized two years ago. This wasn't some theoretical vulnerability. This was real. This was in the wild. After it was analyzed, it was given the name GlueBall because a malicious Java .jar file had been glued onto the back of a valid install file signed by Google. And even though this MSI file had been tampered with and renamed to .gar, and would now execute as a Java JAR, the Windows OS continued to consider the file to be signed with a valid Google certificate. I have a picture of that certificate in the show notes. There it is. Digital signature is okay, signed by Google.

Now, some security researchers like to troll through VirusTotal looking for interesting submissions. You never know what you're going to find. It's naturally a very fertile territory. And one such researcher was a guy named Bernardo Quintero, the founder of VirusTotal. When Bernardo realized what it was that someone had submitted, he immediately reached out and contacted Microsoft, two years ago to the day, to report this clear flaw in the operation of Microsoft's Authenticode signing verification. As we've discussed here, one of the ways security software, including Microsoft's, manages to deal with the flood of unknown malware is to use the reputation of any software's signature.

I've noted how, after obtaining my own new Authenticode signing certificate a year ago, my properly signed software started being flagged for a few weeks until enough people had downloaded it and run it and nothing bad had happened. That initially new and unknown certificate had acquired a reputation. And thus anything it signed now carried an assumption of trust. That sort of heuristic is the only way we're able to stay afloat in this creaky leaking boat of an industry we have built for ourselves. So it's easy to understand what it would mean if malware was able to piggyback onto something that Google had signed and thus obtained the inherent trust that Google had earned. When such a malicious hybrid was downloaded and run, no warning bells would ring.

So as I said, though it bears repeating, after discovering this flaw, Bernardo Quintero, the founder of VirusTotal, responsibly disclosed it to Microsoft two years ago to the day, on August 18th of 2018. And he was told, nah, we're not going to fix it. What?

Bernardo subsequently explained, this attack vector has been verified in the latest and updated versions of Windows 10 and Java, available at the time of writing, which was Windows 10 v1809 and Java SE Runtime Environment 8, Update 191. Microsoft has decided, he wrote, that it will not be fixing this issue in the current versions of Windows and agreed we are able to blog about this case and our findings publicly.

So on January 15th, 2019, Bernardo blogged about this in a VirusTotal blog entry titled "Distribution of malicious JAR appended to MSI files signed by third parties." He said, this is him speaking in the VirusTotal blog: "Microsoft Windows keeps the Authenticode signature valid after appending any content to the end of a Windows Installer .msi file signed by any software developer. This behavior can be exploited by attackers to bypass some security solutions that rely on Microsoft Windows code signing to decide if files are trusted." Gee, what's the point of signing something with Authenticode?

Anyway, sorry. He continues: "The scenario is especially dangerous when the appended code is a malicious JAR because the resulting file has a valid signature according to Microsoft Windows, and the malware can be directly executed by Java." Now he explains that. "Code signing is the method of using a certificate-based digital signature to sign executables and scripts in order to verify the author's identity and ensure that the code has not been changed or corrupted since it was signed by the author." Right. Change one byte breaks the signature; right? Digital signature, that's the way it works. Different hash - anyway, we know.

"This way, for example, if you modify the content or append any data to a signed Windows PE" - you know, a dot executable, it's called Portable Executable, PE file - the signature of the resulting file will not be valid for Microsoft Windows, as expected. This behavior changes when you append any data to the end of a signed Windows Installer .msi file. The resulting file will pass the verification process of Microsoft Windows and will show just the original signature as valid without any warning.

"This behavior could be used to hide and distribute malicious code in MSI-signed files. In fact, several security solutions rely on the output of Microsoft Windows code-signing validation to avoid an in-depth scan when the file has a valid signature by a well-known and trusted software developer." So he's restating exactly what I was saying before.

"Such an attack vector is not very interesting if the resulting file is not designed to execute the attached payload because the attacker would need an additional component already running in the target to extract and execute the appended malicious code. However, JAR files" - and maybe two years ago Microsoft just missed this fact? I mean, it's the only thing I can understand is that someone said, oh, yeah, we know, you can stick stuff on the end of an MSI file, and it doesn't break the signature. Yeah, we did that on purpose. Maybe they didn't stop to consider Java.

Bernardo writes: "However, JAR files have a characteristic that allows them to run directly in this scenario." And I should point out that Bernardo did tell Microsoft two years ago. Anyway, "...allows them to run directly in this scenario, making them the perfect candidate to take advantage of this situation. A JAR file allows Java runtimes to efficiently deploy an entire application, including its classes and their associated resources, in a single request.

"The interesting part for exploiting the commented scenario is the JAR file format is based on ZIP to store the different components and resources, and this kind of ZIP is correctly identified by the presence of an end of central directory record which is located at the end of the archive to allow the easy appending of new files. When Java opens a JAR file, it looks at the end instead of the beginning of the file, so a JAR file is executed independently of the data at the beginning of the file." In other words, Java ignores whatever that was Google signed for the .msi. It doesn't care. It looks at the end. And what do you know? There's the ZIP directory telling it about the contents and where to find it in the file.

So he says: "In addition, on Microsoft Windows systems, the Java Runtime Environment's installation program will register a default association for JAR files so that double-clicking a JAR file on the desktop will automatically run it with 'javaw -jar' as the arguments.

Dependent extensions bundled with the application will also be loaded automatically. This feature makes the end-user runtime environment easier to use on Microsoft Windows systems."

So he finishes: "In short, an attacker can append a malicious JAR to an MSI file signed by a trusted software developer like Microsoft, Google, or any other well-known developer, and the resulting file can be renamed with a .jar extension and will have a valid signature according to Microsoft Windows."

Leo: That's so bad.

Steve: For example, via the command just "copy /b signed.msi + malicious.jar," copy it to "signed_malicious.jar." That is, just simply physically concatenate them. That's all it takes.

Leo: Geez.

Steve: The victim can be infected with just a double-click on such a file, Leo. And for two years Microsoft, eh. Until it was found being exploited.

Leo: Wow.

Steve: So since we now know that someone had been found to be actually doing this, thanks to Microsoft's admission that this """zero-day""" - and I've got that in triple double quotes in the show notes - has been actively used in the wild, we must ask ourselves what could Microsoft have possibly been thinking - or is the term "thinking" way too generous here - when they decided not to fix this two years ago? Most of the world has been using Windows for these two years, while it's been public knowledge that a Java JAR file can be tacked onto the end of any signed and trusted MSI installer file, renamed with a JAR extension, and it will then slide right past any AV system that says to itself, oh, I don't know what that file is. Haven't seen it before. But look, Google signed it. So okay. An objective observer would have to conclude that today's Microsoft is not yesterday's Microsoft.

So finally, as of just last week, the patch for what is being called CVE-2020-1464, Windows will no longer, finally, consider MSI files to be properly signed if they have been tampered with by having a JAR file appended to them. So literally, they finally, after two years, added a special case just for this. And is it even right to call this a zero-day? This appears to have been officially sanctioned behavior by Microsoft for the past two years, until now. A zero-day is supposed to be a surprise. But this wasn't.

Leo: 730 days, I guess.

Steve: Exactly.

Leo: So the thing that baffles me is a certificate should in some way encompass what it's attached to.

Steve: Uh-huh, yes.

Leo: Like with a hash, saying this certificate belongs to this file. Otherwise it sounds like there's no safeguard against the certificate itself being detached and put somewhere else.

Steve: It's nuts.

Leo: You can absolutely say here's the hash of the thing we're attached to. If these don't match, the certificate's no good.

Steve: Right.

Leo: It doesn't even make sense not to do that. What's the point of a certificate?

Steve: Right.

Leo: It's like, oh, I'm going to glue a piece of paper on here that says "Genuine." It doesn't have any meaning.

Steve: Wait a minute. Come to think of it, I've seen stickers like that on my laptops.

Leo: Yeah, genuine. It doesn't - it's meaningless.

Steve: Whoa, yeah, look, it's got a hologram.

Leo: Right? I mean, isn't that the whole idea of what a certificate is, is it's certifying that the contents of this file are unchanged and attached to this ownership.

Steve: Yes.

Leo: It sounds like - and I hope it's more than just a malicious JAR file's been glommed onto here. I hope that it's if the file's been modified in any way. That's what makes me think there must be some functionality. This is always the case with Microsoft. There's some functionality they were protecting.

Steve: Well, exactly. So first of all, the exception to what you're saying, like the hash of everything but the certificate, .msi is the exception. So if this was an EXE that was signed, you can't touch a byte. You can't touch it at all.

Leo: So why is there an MSI exception? There must be some, like, we want you to be able to modify this without recertifying it.

Steve: Well, put stuff on the end, hang other crap on afterwards. I mean, that's what this does. And I do think that Microsoft had to be protecting that function for some reason.

Leo: Yeah.

Steve: But it must have been that poor Bernardo was saying, but, but, but, but, but...

Leo: But it's not even sophisticated. It's a copy command that it concatenates a file with a Java file.

Steve: It's a copy command that glues them together.

Leo: And then renames it.

Steve: That's why they called it the GlueBall attack.

Leo: It's so stupid. I don't even know. I guess a JAR, if you rename it JAR, then Java will just search through the whole binary until it finds main and then say, good, I'll execute here? Because it's got a...

Steve: Actually, it's even better than that. If you rename it JAR, then Java opens - it looks at the end, where the dictionary is.

Leo: Oh.

Steve: That tells it where to find everything else. It ignores the whole front.

Leo: So that's why it works well with Java. It would work with any language that says, well, let me look at the end.

Steve: Yeah.

Leo: Oh, my god. So bizarre. So MSI files are the installer files for Microsoft.

Steve: Right.

Leo: And they're widely used.

Steve: Yes.

Leo: There must have been some functionality where you could - oh, I know what it is. I just thought of what it is. Localization.

Steve: Yes. Different language packs.

Leo: So you'd attach a language pack to the end. Now, you don't want to have to get the whole thing recertified just because I've added a language pack.

Steve: Yup.

Leo: So you go, well, the certificate applies. You can add any localizations you want. I bet you it's something like that.

Steve: Right, right.

Leo: Wow.

Steve: And they made an exception, and the exception bit them. And they ignored a clear problem for two years.

Leo: Wow.

Steve: Just because they said, no, we know you can put crap on the end of MSI. We don't care.

Leo: That's very Microsoft because one of the keys to Microsoft is not to break existing implementations. You never want to break legacy. Even if there's a security reason to do it. You never want to - because some big, you know, Oracle or somebody, some big company has created a whole bunch of files, and this would break everything. They'd have to get new certs.

Steve: Probably is Oracle, actually, because they're Java parents.

Leo: Yeah. Wow. Wow. What a story.

Steve: But, gee, somebody's actually installing malware with it? Oh.

Leo: Sorry. We'd like to.

Steve: I guess we should patch it.

Leo: We'd like to fix it.

Steve: We'd like to patch it right now. We'll just get in a big hurry to patch it.

Leo: Yeah, I mean, this is...

Steve: That Chromebook's looking sweeter every minute.

Leo: I know. Linux. The nice thing about Linux, it's made by a bunch of cranky people who say, but no, this is the right way to do it. I don't care if it breaks your application. We're doing it this way, and suck it. And Microsoft, they're just too nice. They're just too nice.

Steve: Oh, that's right. "Nice" is the word I would use. That's right.

Leo: Somebody in the chatroom, MikeyLikesIt, says "I create MSI files. The language packs can be built as a separate file time MSN. And during runtime the contents of the MSN get streamed into the MSI." So it probably is for localization. Oh, my, my, my. So, Steve, you've done it again. You've used up your time wisely, very wisely, very usefully.

Steve: And the lights stayed on.

Leo: Well, yeah. But we're going to run to All About Android and try to get as much of that show in before they go out because it is getting - I'm watching the power usage in the state of California creep towards that magic line where they say there's no more. Run out, you've run out. And it's getting real close as we get hotter and hotter.

Steve Gibson's at GRC.com. Power's still on there, anyway. Go down and get yourself a copy of SpinRite, the world's best hard drive recovery and maintenance utility, getting better all the time. Soon with AHCI support built right in. That'll be version 6.1, which you will get for free if you buy SpinRite today.

We also would encourage you as you're there to get a copy of the show. He's got 16Kb versions for the bandwidth impaired. He also has 64Kb audio. And he has transcripts, the only place you can get the transcripts of the show. They're very well done by Elaine Farris. All at GRC.com, along with all the other freebies Steve gives away all the time.

We have a copy of audio and video of the show at our website, TWiT.tv/sn. You can also subscribe in your favorite podcast application, audio or video. That way you'll get it automatically. You should start collecting all 780. You know, you could say, oh, I didn't start till Year 16. That would be okay. But start now. There's another many, many shows to come, and you're going to want all of them.

We do the show every Tuesday, 1:30 Pacific, 4:30 Eastern, 20:30 UTC, if you want to watch us do it live. TWiT.tv/live has the streams. Thanks, Steve. Have a great week. Stay cool. Keep the lights on. We'll see you next time on Security Now!.

Steve: Thanks, buddy.