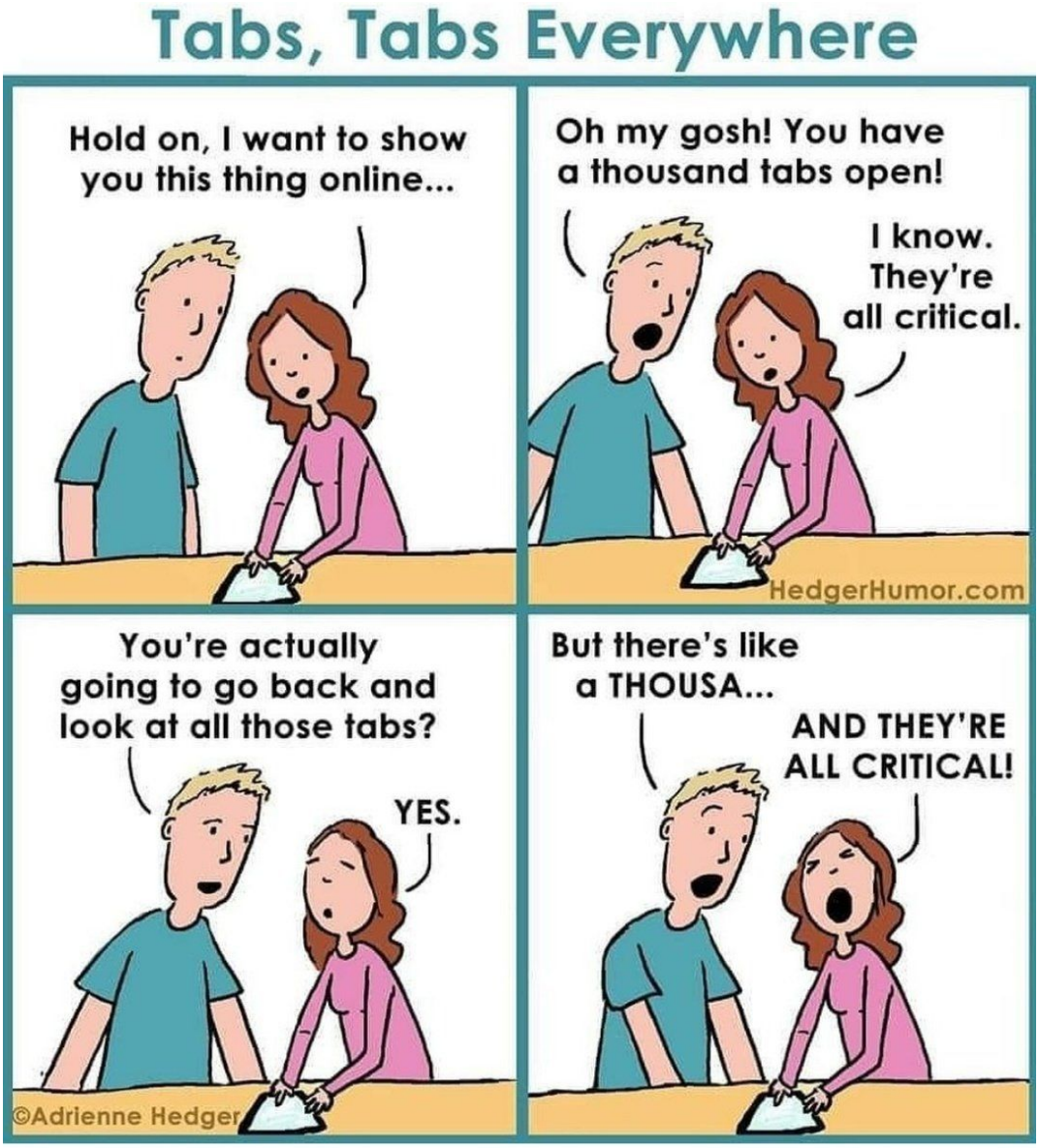
Security Now! #780 - 08-18-20

Microsoft’s 0-day Folly

This week on Security Now!

This week we discuss the “Achilles” Snapdragon DPS flaw affecting more than one billion Android Smartphones, last week's 3rd-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” that prevent 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 0-day vulnerabilities that were finally patched last week.



Security News

Snap Your Dragon / "Achilles: Small Chip, Big Peril"

The troubles recently discovered by Check Point's security researchers surround one of the modules, the cDSP — computational digital signal processor — of the Qualcomm SnapDragon system-on-a-chip. The best way to introduce our listeners to the many — and by "many" I mean 400 — security vulnerabilities recently discovered is to share the short teaser they recently offered to potentially interested participants during the recent Black Hat security conference:

> Qualcomm Snapdragon SoC 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 on 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 PE (privilege elevation) and DoS attacks of the device.
>
> For security reasons, the cDSP is licensed for programming by OEMs and by a limited number of third-party software vendors. The code running on the DSP is signed by Qualcomm. However, we will demonstrate how an Android application can bypass Qualcomm's signature and execute privileged code on DSP, and what further security issues this can lead to.
>
> Hexagon SDK 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 are going to highlight the auto generated security holes in the DSP software and then exploit them.

"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.

So we have some sense for how there could possibly be some 400 problems 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. We've seen that often enough. And it might also be that the compiler produces DSP code that has problems. And if that compiler was used to produce the buggy libraries -- as seems likely -- then that might help to explain how this happened. The next thing to understand, then, is what it means.

Check Point introduced their research by writing:

> 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 can turn the phone into a perfect spying tool, without any user interaction required – The information that can be exfiltrated from the phone include photos, videos, call-recording, real-time microphone data, GPS and location data, etc.

- 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, etc – in other words, a targeted denial-of-service attack.

- Malware and other malicious code can completely hide their activities and become un-removable.

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 3+ 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's handled by the device's DSP chip. And targets can also be attacked by installing malicious apps that require no permissions at all.

Because the problem 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 of a tease rather than a how-to-do-it. But this could not be unleashed upon an unsuspecting world. Check Point plans to keep these secrets until it's safe for them to be released.

Which begs the question... when will that be? As we know, many hundreds of millions of older Android devices are already past-end-of-patch-life and are riddled with many known bugs. So what we have now amounts to a pile on.

In Samsung's article titled "Understanding the Product Life Cycle of a Smartphone" they explain: https://insights.samsung.com/2018/05/10/understanding-the-product-life-cycle-of-a-smartphone-2/

The product life cycle of a smartphone is not just about physical attributes; it is 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, but also whether it will continue to be supported by the manufacturer with ongoing firmware and security updates.

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 Note8 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 life cycle 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.

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. 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.

As we know, a SmartPhone 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. No such device should never be purchased from a random low-end fly-by-night manufacturer. And given the imperative of Google, Samsung and the other high-end vendors, they regard a SmartPhone as having an expected life, from the time of its initial release, of 36 months. I have traditionally preferred to purchase my devices outright. But three years is a long time in the current evolution of a high-end smartphone. So it might be that taking advantage of Smartphone trade-in and upgrade plans makes the most sense.

For their part, Check Point has acted as responsibly as they could. They wrote that they decided to publish a blog posting about the SnapDragon problem to raise awareness of the potential problems. And they have also informed, they said, "relevant government officials, and relevant mobile vendors we have collaborated with on this research to assist them in making their handsets safer. The full research details were revealed to these stakeholders."

Last Week's Patch Tuesday

When ZDNet sums up last week's Patch Tuesday: "Microsoft says attackers have used a Windows zero-day to spoof file signatures and another RCE in the Internet Explorer scripting engine to execute code on users' devices" ... we need to take a closer look.

With 120 new flaws in Microsoft's software fixed, this was the 3rd 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. And this month's bundle carried a bit more urgency than usual, since one of the 17 flaws that were classified "critical" was a 0-day under active attack at the time of the updates, and one of the remaining more than 100 flaws rated "important" was also a 0-day being exploited in the wild and publicly disclosed.

The first of the two is titled "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:

> "In a web-based attack scenario, an attacker could host a specially crafted website that is designed to exploit the vulnerability through Internet Explorer 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."

So, that wasn't good and it will remain a threat for anyone who hasn't yet applied last Tuesday's updates. But the second 0-day, despite being actively exploited in the wild and publicly known, is only rated “important”, which seems odd since it is “CVE-2020-1464” and labeled somewhat innocuously as a “Windows Spoofing Vulnerability.” Okay. But 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. In Microsoft's words: “These spoofed signatures could allow an attacker to 'bypass security features intended to prevent improperly signed files from being loaded.'” That's certainly not good. ... and just WAIT until you get the details at the end of this podcast.

Beyond those two 0-days, five of the other critical bug fixes are for Microsoft’s Windows Media Foundation (WMF), the multimedia framework and infrastructure used to render digital media ever since Windows 7 and 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, CVE-2020-1046 is another nasty one in the .NET framework affecting version 2.0 through 4.8. And 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 not a heavy lift these days.

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 0-day flaws since they have become the topic of today’s podcast.

Mozilla's Uncertain Future

In pre-COVID January, Mozilla cut 70 people from its workforce of approximately 1,000. 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 products is still in place, according to sources and a Mozilla spokesperson.

Going forward, Mozilla said that they will be re-thinking their core business model and put more focus on financially viable products.

Mozilla Corporation's CEO and Mozilla Foundation Chairwoman said "Recognizing that the old model where everything was free has consequences, means we must explore a range of different business opportunities and alternate value exchanges." 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."

Some assume that this might include a stronger focus on their new VPN offering, which was formally launched last month. VPN apps and services are one of the biggest money-makers in tech today, and despite having arrived late to the game, Mozilla believes that they may be set to compete due to their strong privacy-first reputation as a civil and privacy rights advocate.

Adding further uncertainty to the mix is the fact that Mozilla's contract to include Google as the default search provider inside Firefox will likely be expiring later this year, and so far that contract has not been renewed with Google. The Google deal has historically accounted for around 90% of Mozilla's revenue, and without that, Mozilla's future is uncertain, to say the least. I strongly dislike the idea of having just two dominant browsers in the world — Safari and Chromium — but the writing might already be on the walls.

As we've noted, the browser has become most people's operating system. When someone calls Leo on the weekend during his nationally syndicated The Tech Guy radio show and asks whether they should get a new Windows machine or a new Mac, Leo's reply, after asking them what they need a computer for, is often to recommend a Chromebook because that's all they need. It's basically a bootable Internet browser.

My point is, browsers have grown so insanely complex, with so many features and bells and whistles, and they are so prone to attack because they are the piece of yourself that you're sticking out there on the Internet, that they **are** like an operating system. And the task of maintaining a piece of software as incredibly complex as a contemporary web browser is expensive. If Edge implements its forthcoming vertical tabs well, and if Mozilla finally throws in the towel, that might be where I go.

I had a note in our "Miscellany" section simply observing that the "Firefox Send" service is **still** offline. They clearly have other things on their minds at Mozilla, so I would not hold my breath for that free service returning. It was my favorite, but I may need to return to FileMail.

The other shoe drops at Canon

The Maze Ransomware gang have apparently started publishing data which, they allege, was stolen from Canon.

Once again, BleepingComputer is breaking the news that Canon may have elected NOT to pay the ransom. Lawrence Abrams 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."

The published file is a 2.2 GB ZIP archive titled “STRATEGICPLANNINGpart62.zip”. BleepingComputer was told it contains marketing materials and videos, as well as files related to Canon's website. 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. BlleepingComputer reached out to Canon for comment but did not hear back.

What was that software glitch in California's COVID case reporting?
As a Californian, I noted with just passing interest the several times I heard that an unspecified “software glitch” was preventing the reporting of somewhere between 250,000 and 300,000 COVID lab test results for the State of California. From time to time we've noted when an expired web server certificate catches its owners by surprise... And believe it or not, this was the underlying cause of California's recent trouble. The certificate for the server that third-party labs such as Quest and LabCorp use to upload their lab test results had expired when no one was paying attention. Whoops.

I should note that I, too, have an expired certificate. The “revoked.grc.com” cert expired recently and I haven't replaced it with another revoked cert. Frankly, I hadn't been monitoring that facility and I didn't realize that the site’s page was getting 561 visitors per day. But I've received a bunch of notes from people who miss having it working. So I'll obtain a new cert from my favorite certificate provider, DigiCert and then have them immediately revoke it. And now, in mid-August, is the perfect time to do that, since until the end of the month I can obtain a certificate with a 2-year life and thus not need to bother with that one for another two years. But, as we know, after the end of THIS month, the maximum lifetime of any browser certificate will be limited to just one year plus one month.

On Guard!
Yesterday, Microsoft announced that they had completed the work of incorporating their Control Flow Guard technology (CFG) into both the RUST and LLVM open source projects.

For the significance of this to be understood, let’s review a bit. The Rust programming language is rapidly growing in popularity. It has a syntax similar to C++, it provides strong memory safety without the need for the garbage collection employed by dynamic languages and it is now Microsoft's language of choice for implementing secure and safety-critical software components. It was originally designed by engineers at Mozilla who refined its definition while they were writing Mozilla's “Servo” layout browser engine and the Rust compiler. It's free and open-source and dual licensed under both MIT and the Apache License 2.0.

LLVM is a compiler infrastructure project which provides a set of tools on the front end for implementing new languages and on the back end for compiling to new computer instruction sets. And in between there's a so-called “intermediate representation” which provides a language-neutral and instruction set neutral which serves as a portable high-level assembly language that can be optimized through a variety of transformations in multiple passes. 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#,

Common Lisp, Crystal, CUDA, D, Delphi, Dylan, Fortran, Graphical G Programming Language, Halide, Haskell, Java bytecode, Julia, Kotlin, Lua, Objective-C, OpenCL, several SQL's, Ruby, Rust, Scala, Swift, and Xojo.

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. 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 obtain CFG 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 the way an attacker wishes. We've learned from the effectiveness of so-called Return Oriented Programming (ROP) that a surprising amount of damage to the system's security can be caused simply by jumping near the end of subroutine functions of existing authentic and authorized code, but doing so out of order. This bypasses the strong protection offered by preventing data on the stack from being executed as code, or data in other non-stack data regions, such as communications buffers from being executed as code. Once those easier-to-block data-execution attacks were being thwarted, attackers said: “Okay, you think you're so smart? We're going to execute your own code to accomplish our nefarious goals.” And so Return Oriented Programming was born.

So how do we prevent THAT, now? A functional subroutine has a so-called entrypoint. 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 entrypoint. In modern languages, the beginning of a function does a bunch of critical setup work. If the function defines some local temporary working variables, as most do, then space must be set aside on the stack to hold them. And since some functions may be required not to modify the caller's registers, some of those might need to be saved on the stack if the function needs them for working storage. Because of all this, it's not possible for a program to simply jump into the middle of a function and 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 ROP does to abuse the system. So, in other words, NO valid code will EVER jump anywhere but to any function's entrypoint; only malicious code will 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 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 entrypoint, the application checks with this bitmap to see whether the destination address is represented by a '1' bit in the allowed-jump-destinations bitmap. If the destination address has a '0' 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.

It's not perfect protection, and it does come at the cost of some overhead. But for security-sensitive code it probably makes sense, since it further raises the bar to complicate the task of malicious subversion. As of Windows 10 Creators Update (version 1703), the entire Windows kernel is compiled with CFG in place. And the kernel uses Hyper-V to prevent malicious kernel code from overwriting the CFG bitmap... since that would be the obvious way to give aberrant code permission to jump where it should not.

Yesterday Microsoft announced that Rust and the LLVM project now had this technology, and this represents a very nice step forward.

Threema gets E2EE 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.

I'm still tempted to say that only when users manage their own keys can they truly feel that the solution they are using is as secure as it could be. If someone else is managing those keys -- to make the process more automatic and convenient -- and the keys are the key, as they are, then true security is necessarily reduced. Signal offers optional key verification, 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.

The problem with my argument, is the classic weakest-link security dilemma: If you or your counter-party at the other end 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.

But I like Threema, and if I needed to hold truly private video or messaging calls -- local or transcontinental -- and I wanted THE BEST security possible, Threema would be my choice today... because it's so clear that they're doing the best job possible. Here's how they explained their news:

> 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 is to be expected, 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.

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, means 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.

Closing The Loop

Timo Grün / @Khoji
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 news group. 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

Max / @maxstr
@SGgrc Steve I have an air filter with... a BUILT-IN Bluetooth IoT device. On an air filter. I can't believe I just paired my phone with an air filter. 😳

SpinRite

The big "Verify" test... and what we learned.

Microsoft’s 0-day Folly

When Microsoft doesn't act responsibly - Part 1
It's difficult to know what to think of the fact pattern I'm about 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 another entire class of problem.

The story begins last December of 2019 when Microsoft received a report for a serious vulnerability in Windows. And it didn't come through some back channel. It was relayed from an anonymous source through Trend Micro’s official Zero Day Initiative (ZDI).

Trend Data 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 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, the now-public vulnerability was exploited the very next day in an effort now called “Operation PowerFall.”

An advanced threat actor exploited one of the two zero-day vulnerabilities that Microsoft patched on Tuesday in a targeted attack earlier this year. 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 a 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 likely 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.

This so-called “Operation PowerFall,” attack leveraged a remote code execution vulnerability in IE11, which has just now been patched, and that flaw in the Windows GDI Print/Print Spooler API which enabled a privilege escalation. That's the one that Microsoft sat on for six months and did nothing.

And this should strongly reinforce the idea that privilege escalation flaws are not nothing. The sequence of events suggests that the attackers already 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. So, the instant they obtained the news from ZDI's disclosure 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” they should be relieved of their responsibility for making such determinations.

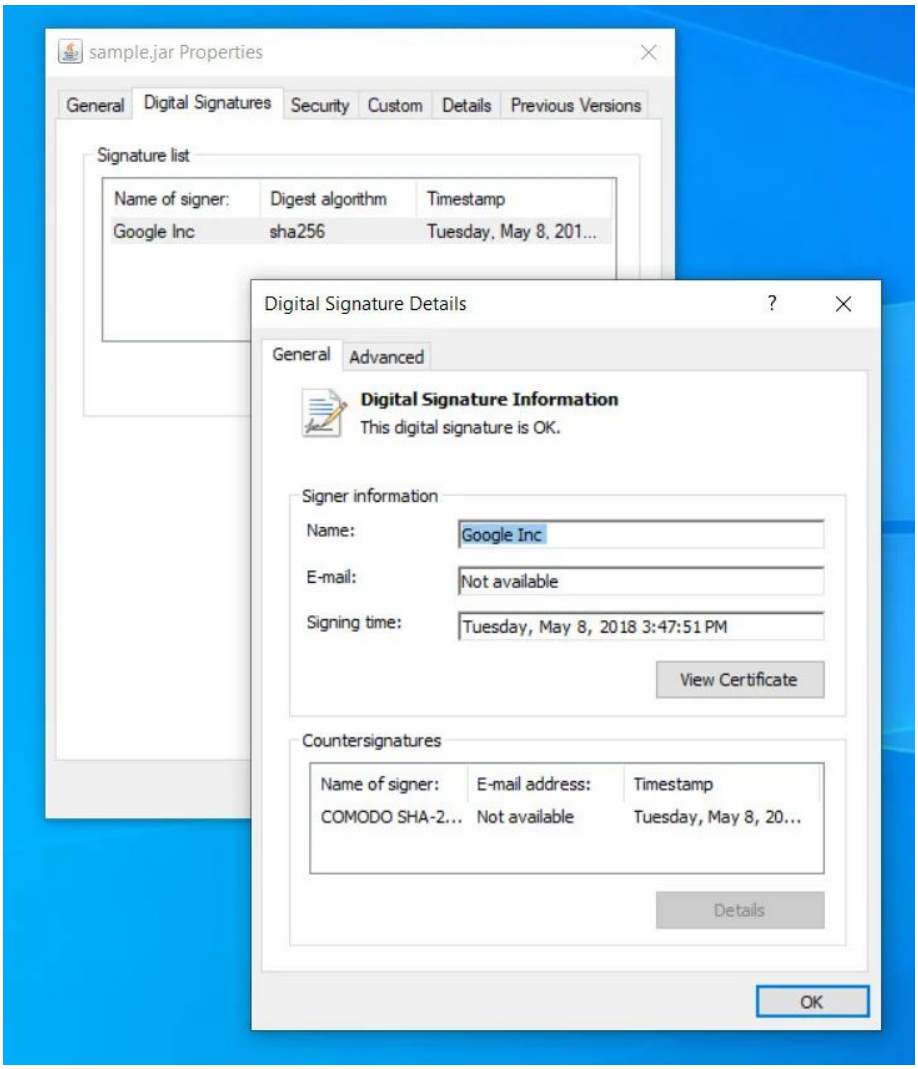
...and, believe it or not, that’s not the worst...

When Microsoft doesn't act responsibly - Part 2

I started part one by writing: "It's difficult to know what to think of the fact pattern I'm about to lay out." This part two is no different, if not even more egregious. Remember that other 0-day that was being actively used in the wild, which Microsoft referred to as a "spoofing" vulnerability? Well... it allows Microsoft Installer "MSI" files to be converted into malicious Java executables while retaining a legitimate company's digital signature. That's the 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. And Microsoft stated 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, 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 virus total means that it existed and was weaponized -- two years ago. This wasn't a 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 a .JAR file, and would now execute as a JAVA JAR, Windows still considered the file to be signed with a valid Google certificate!

I have a picture of that certificate in the show notes:



Some security researchers like to troll through VirusTotal looking for interesting submissions. You never know what you're going to find. By its very nature it's fertile territory. And one such researcher was a guy named Bernardo Quintero, the founder of VirusTotal. When Bernardo realized what 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 signing. I've noted how, after obtaining a new Authenticode signing certificate a year ago, my own properly signed software was being flagged for a few weeks until enough people had downloaded 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've 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 obtain the 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 on August 18th, 2018... and he was told... that it would not be fixed.

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 (Windows 10 Version 1809 and Java SE Runtime Environment 8 Update 191). Microsoft has decided 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 of 2019, Bernardo blogged about this in a VirusTotal blog entry titled: "Distribution of malicious JAR appended to MSI files signed by third parties"

https://blog.virustotal.com/2019/01/distribution-of-malicious-jar-appended.html

Microsoft Windows keeps the Authenticode signature valid after appending any content to the end of Windows Installer (.MSI) files signed by any software developer. This behaviour can be exploited by attackers to bypass some security solutions that rely on Microsoft Windows code signing to decide if files are trusted. 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.

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. This way, for example, if you modify the content or append any data to a signed Windows PE (.EXE) file the signature of the resulting file will not be valid for Microsoft Windows, as expected. This behaviour changes when you append any data to the end of a signed Windows Installer (.MSI), the resulting file will pass the verification process of Microsoft Windows and will show just the original signature as valid without any other warning.

This behaviour 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. 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 have a characteristic that 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 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". 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.

In short, an attacker can append a malicious JAR to a MSI file signed by a trusted software developer (like Microsoft Corporation, Google Inc. or any other well-known developer), and the resulting file can be renamed with the .jar extension and will have a valid signature according Microsoft Windows. For example, via the command "copy /b signed.msi + malicious.jar signed_malicious.jar". The victim can be infected with just a double-click in such a file.

So, since we now know that someone had been found to be actually doing this thanks to Microsoft's admission that this """0-day""" was being 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 two years while it's been PUBLIC KNOWLEDGE that a JAVA JAR file can be tacked onto the back of any signed and trusted MSI installer file, renamed with a .JAR extension and that it then will slide right past any A/V system that says to itself "Oh! I don't know that file, 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.

Finally... as of just last week, with the patch for what is being called CVE-2020-1464, Windows will no longer consider MSI files to be properly signed if they have been tampered with by having a JAR file appended to them. What a concept. And do we even call this a 0-day if this appears to have been officially sanctioned behavior by Microsoft, for the last two years, until now? A 0-day is supposed to be a surprise. But this wasn't.