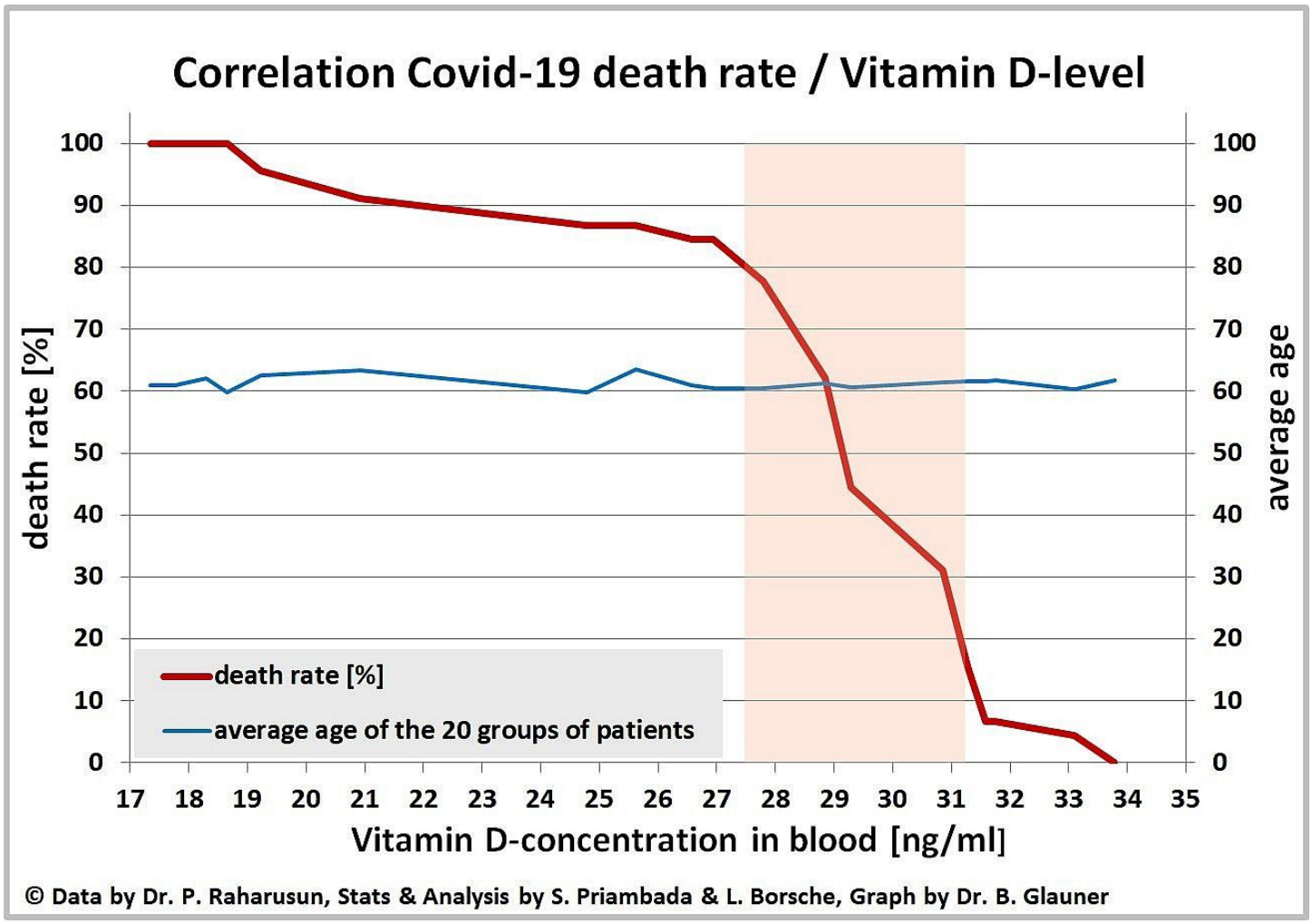
Security Now! #778 - 08-04-20

BootHole

This week on Security Now!

This week we touch on the recent update of Firefox to v79, we check back on the Twitter hack with the news of the identity of the accused perpetrators, we also have more information about the Garmin ransomware hack, we look at the behavior of another disgruntled vulnerability researcher and consider another aspect of the ethics of vulnerability disclosure, we examine Zoom's bug of the week and the consequences of Microsoft's removal of all SHA-1 signed downloads. And QNAP NAS devices are still suffering from real trouble and neglect by their owners. I'm going to check-in with the SpinRite work. Then we take a look at the week's biggest security event... the discovery of a boot security bypass for Linux.

Not to belabor the point, but this should not go unobserved:



https://borsche.de/res/Vitamin_D_Essentials_EN.pdf

Full report with URL links to all of the source material research

Browser News

Firefox is now at v79.

No big news on the Firefox front. The biggest new feature is a credential export added to Firefox's built-in "Lockwise" password manager. This exports the Firefox database into a CSV formatted text file. From there it could be dropped into a spreadsheet or imported into some other password manager. And, it goes without saying, that while in text form it's readily discoverable by anyone / anything scanning your machine... so storing it in a password-protected 7zip archive would be smart. 7-Zip uses well-designed archive encryption. It derives a 256-bit AES key from a password-based key derivation function which uses a high iteration count (as a brute forcing delay) SHA-256 hash. So pick a good password and you should be safe with your passwords exported and kept in an encrypted archive.

Note that I'm still using LastPass under Firefox because I can also use it under Chrome and Edge and Safari and across all platforms.

Security News

You may have heard that Twitter was not hacked by some powerful foreign state-sponsored cybercrime gang. Nope. Just a 17-year old kid. According to a local Florida news channel WFLA, this is "Graham Clark" from Tampa Bay, Florida. Graham is almost certainly the juvenile that U.S. Department of Justice mentioned in its press release. Graham has reportedly been charged with 30 felonies relating to computer communications and organized fraud for scamming hundreds of people using compromised accounts.

According to a press release from Hillsborough State Attorney Andrew Warren's office, Graham now faces a grand total of 30 felony charges, including:

- 1 count of organized fraud of over $50,000
- 17 counts of communications fraud of over $300
- 1 count of fraudulent use of personal information over $100,000 or 30 or more victims.
- 10 counts of fraudulent use of personal information
- 1 count of access to computer or electronic device without authority - scheme to defraud.

Although state Attorney Andrew Warren didn't initially indicate whether Clark had partners in crime, a few hours after the press conference announcement, the world learned that the US Department of Justice had also filed charges against two other suspects believed to have helped Clark in the hack. The first of those was identified as Mason Sheppard, aka "Chaewon," 19 years old, of Bognor Regis, in the UK, and the other was identified as Nima Fazeli, aka "Rolex," a 22 year old residing in Orlando, Florida.

The U.S. Attorney Anderson said: "There is a false belief within the criminal hacker community that attacks like the Twitter hack can be perpetrated anonymously and without consequence. Today's charging announcement demonstrates that the elation of nefarious hacking into a secure environment for fun or profit will be short-lived. Criminal conduct over the Internet may feel

stealthy to the people who perpetrate it, but there is nothing stealthy about it. In particular, I want to say to would-be offenders, break the law, and we will find you."

Twitter, for their part, has disclosed a bit more about the nature of the attacks. They said that the phone-based social engineering attack allowed the attackers to obtain the credentials of a limited set of employees which made it possible to gain access to Twitter's internal network and support tools. Although not all of the employees that were initially targeted had permissions to use account management tools, the attackers were able to use their credentials to access Twitter's internal systems and gain information about Twitter's processes. That expanded knowledge then enabled the attackers to target additional employees who did have access to Twitter's privileged account support tools.

Reuters News Service also reported something that I had not seen elsewhere, which was that as of earlier this year more than a thousand Twitter employees and contractors had access to Twitter's internal tools that could change user account settings and hand control to others. One Thousand! This was according to two former Twitter employees. As we know, such widespread access makes it difficult, if not impossible, to defend against the sort of hacking that occurred.

My take on this is that it's just what you might call "managerial inertia", and it's kind of natural. When Twitter was born, it wasn't initially taken very seriously. Let's remember that in the beginning it was a ridiculously-limited text-only patently insecure messaging service. I recall thinking "wait... you said 140 characters max"? What's that good for? But obviously, over time, Twitter's importance has changed dramatically. Today, heads of industry and state use Twitter to reach their followers, including, famously, our U.S. President who uses it to directly reach each of his 84.5 million followers. And more than likely, Twitter also didn't take itself very seriously at the start. After all, as we've noted, there never really was any clear plan for how this free service was supposed to make any money.

But over time that changed. And my point is, Twitter's importance doubtless crept up on it. Over the course of many years Twitter slowly grew into a truly important global communications facility. It certainly didn't start out as one, and it clearly is today. That didn't happen all at once. It crept up on Twitter. So I think this security breach is mostly a consequence of Twitter doing things the way it always has, and of any change to that status quo having lagged behind. This ultra-high-profile security breach was probably the best thing that could have happened to Twitter: It didn't result in any huge amount of damage, and it was obviously a very much needed wake up call delivered at the right time.

For anyone who wants more, I've read everything I could find and ZDnet provided the most detail about the hack, including Discord chat logs where Graham is soliciting the participation of the other two, claiming to work for Twitter and offering to prove it by modifying their Twitter accounts. He also provided his bitcoin address and sold them access to some high-value Twitter accounts such as @xx, @dark, @vampire and @drug. The link with all the details is in the show notes for anyone who's interested:

https://www.zdnet.com/article/how-the-fbi-tracked-down-the-twitter-hackers/

The Garmin Hack

Lawrence Abrams' Bleeping Computer has always had a strong interest in ransomware. So I'm not surprised that his coverage of the Garmin ransomware attack was the most detailed of any I've seen... nor that he's had some Garmin insiders reaching out to provide some tasty bits:

Among other things, an employee inside Garmin informed him that the initial ransom request was for $10 million. We don't know what ransom was finally paid, but it seems more certain than ever that Garmin did pay up. Lawrence wrote: "After a four day outage, Garmin suddenly announced that they were starting to restore services, and it made us suspect that they paid the ransom to receive a decryptor."

Then, on Saturday Lawrence posted: "Today, BleepingComputer gained access to an executable created by the Garmin IT department to decrypt a workstation and then install a variety of security software on the machine. Since WastedLocker is an enterprise-targeting ransomware with no known weaknesses in their encryption algorithm a decryptor cannot be made for free. To obtain a working decryption key, Garmin must have paid the ransom to the attackers. It is not known how much was paid, but as previously stated, an employee had told BleepingComputer that the original ransom demand was for $10 million. When extracted, this restoration package includes various security software installers, a decryption key, a WastedLocker decryptor, and a script to run them all. When executed, the restoration package decrypts the computer and then preps the machine with security software. Garmin's script contains a timestamp of '07/25/2020', which indicates that the ransom was paid either on July 24th or July 25th. Using the sample of WastedLocker from the Garmin attack, BleepingComputer encrypted a virtual machine and tested the decryptor to see if it would decrypt our files. In our test, the decryptor had no problems decrypting our files."

What was interesting was that the package received by BleepingComputer included references to both the cybersecurity firm Emsisoft and the ransomware negotiation service "Coveware". When BleepingComputer subsequently reached out to Coveware they were told that they do not comment on any ransomware incidents reported in the media.

And, similarly, Emsisoft told BleepingComputer that they could not comment on any cases, that they create decryption tools, and are not involved in ransom payments. Brett Callow, a threat analyst at Emsisoft said "I cannot comment on specific cases, but generally speaking, Emsisoft has no involvement whatsoever in negotiating or transacting ransom payments. We simply create decryption tools."

It might seem odd for a reputable security firm such as Emsisoft to have anything to do with ransomware, but they have an interesting angle. As we know, the decryption side of the ransomware mess sometimes receives much less attention from the bad guys than the encryption side. Consequently the decryptors tend to be buggy, to crash, or to for some reason fail to fully undo the damage that they did, despite having a valid key. That's where Emsisoft comes in. They reverse-engineer questionable ransomware decryptors — for which the decryption key is known — to create a more robust and reliable decryptor for a victim's systems. Emsisoft's ransomware recovery services page states: "If the ransom has been paid but the attacker-provided decryptor is slow or faulty, we can extract the decryption code and create a custom-built solution that decrypts up to 50 percent faster with less risk of data damage or loss"

This also explains why the decryption package Garmin finally used also contained legitimate security software. That extra security software, along with an improved decryptor, was provided by Emsisoft.

Now that Evil Corp has been attributed as the creator of WastedLocker and has been placed on the US sanctions list for using Dridex to cause more than $100 million in financial damages, paying this ransom could lead to hefty fines from the government. So, due to these sanctions, sources familiar with Coveware have told BleepingComputer that the negotiation company has placed WastedLocker on their restricted list in early July, and are not handling negotiations for related attacks.

Tor and Dr. Krawetz

Tor has recently been in a bit of a back and forth with a security researcher by the name of Dr. Neal Krawetz. Neal obtained his PhD in Computer Science from Texas A&M University and his bachelors from UC Santa Cruz. He has a long history of finding and reporting problems with the Tor Network and he operates multiple Tor nodes himself. He appears to have long been a bit of a thorn in the side of the Tor engineers. And, frankly, not all of his concerns over Tor's privacy guarantees appear to warrant undue concern.

https://www.hackerfactor.com/blog/index.php?/archives/888-Tor-0day-Stopping-Tor-Connections.html

For example, the good doctor wrote:

> *"Over three years ago, I tried to report a vulnerability in the Tor Browser to the Tor Project. The bug is simple enough: using JavaScript, you can identify the scrollbar width. Each operating system has a different default scrollbar size, so an attacker can identify the underlying operating system. This is a distinct attribute that can be used to help uniquely track Tor users. (Many users think that Tor makes them anonymous. But Tor users can be tracked online; they are not anonymous.)"*

He's obviously not wrong. But in the past three years, despite trying, he has not managed to get this fixed. During that time the Tor Project joined HackerOne, and officially credited him for the discovery of the bug. Its resolution was then pushed off to Mozilla who, after some time of doing nothing with it, dropped it entirely.

We've seen examples of this before. A security researcher finds a problem that he or she believes to be highly critical and needs everyone's attention right away, if not yesterday. But for whatever reason they don't obtain satisfaction from the affected parties. They feel unappreciated for their efforts, stiffed and ignored. So what comes next? In the case of Doctor Neal Krawetz, under the subheading "Dropping 0Days", he explains on his blog:

> **Dropping 0Days**
> A "0day" is any exploit that has no known patch or wide-spread solution. [As we know, I disagree with this definition. It's just an unknown vulnerability unless and until it is found to be exploited. Anyway, he continues...] A 0day doesn't need to be unique or novel; it just needs to

have no solution. I'm currently sitting on dozens of 0days for the Tor Browser and Tor network. [And, again, I would say that he claims to be sitting on dozens of unknown vulnerabilities.] [He says] Since the Tor Project does not respond to security vulnerabilities, I'm just going to start making them public. While I found each of these on my own, I know that I'm not the first person to find many of them.

[So, here we have the unfortunate phenomenon of the security researcher whose original white hat begins to darken as his or her work doesn't receive the attention they believe it deserves.]

[He continues…] The scrollbar profiling vulnerability is an example of a 0day in the Tor Browser. But there are also 0days for the Tor network. One 0day for the Tor network was reported by me to the Tor Project on 27-Dec-2017 (about 2.5 years ago). The Tor Project closed it out as a known issue, won't fix, and "informative".

Let's start with a basic premise: let's say you're like some of my clients -- you're a big corporation with an explicit "no Tor on the corporate network" rule. This is usually done to mitigate the risks from malware. For example, most corporations have a scanning proxy for internet traffic that tries to flag and stop malware before it gets downloaded to a computer in the company. Since Tor prevents the proxy from decoding network traffic and detecting malware, Tor isn't permitted. Similarly, Tor is often used for illegal activities (child porn, drugs, etc.); blocking Tor reduces the risk from employees using Tor for illegal purposes. Although denying Tor can also mitigate the risk from corporate espionage, that's usually a lesser risk than malware infections and legal concerns. (Keep in mind, these same block and filtering requirements apply to nation-states, like China and Syria, that want to control and censor all network traffic. But I'm going to focus on the corporate environment.)

It's one thing to have a written policy that says "Don't use Tor." However, it's much better to have a technical solution that enforces the policy. So how do you stop users from connecting to the Tor network? The easy way is to download the list of Tor relays. A network administrator can add a firewall rule blocking access to each Tor node.

0Day #1: Blocking Tor Connections the Smart Way

There are two problems with the "block them all" approach. First, there are thousands of Tor nodes. Checking every network connection against every possible Tor node takes time. This is fine if you have a slow network or low traffic volume, but it doesn't scale well for high-volume networks. Second, the list of nodes changes often. This creates a race condition, where there may be a new Tor node that is seen by Tor users but isn't in your block list yet.

However, what if there was a distinct packet signature provided by every Tor node that can be used to detect a Tor network connection? Then you could set the filter to look for the signature and stop all Tor connections. As it turns out, this packet signature is not theoretical.

In his blog posting he then goes on to describe in great detail Tor's TLS handshake and the unique properties of the TLS certificates which Tor node servers generate on the fly.

Validating the Vulnerability
Back in 2017, I used a scanner and Shodan to search for TLS certificates. In theory, it is possible for there to be some server with a server-side TLS certificate that matches this signature but that isn't a Tor node. In practice, every match was a Tor node. I even found servers running the Tor daemon and with open onion routing (OR) ports that were not in the list of known Tor nodes. (Some were non-public bridges. Others were private Tor nodes.)

Similarly, I scanned every known Tor node. Each matched this Tor-specific certificate profile. That makes the detection 100% accurate; no false positives and no false negatives. (Although now that I've made this public, someone could intentionally generate false-positive or false-negative certificates. The false-positives are relatively easy to construct. The false-negatives will require editing the Tor daemon's source code.)

While a scanner could be used to identify and document every Tor server, corporations don't need to do that. Corporations already use stateful packet inspection on their network perimeters to scan for potential malware. With a single rule, they can also check every new connection for this Tor signature. Without using large lists of network addresses, you can spot every connection to a Tor node and shut it down before the session layer (TLS) finishes initializing, and before any data is transferred out of the network.

He then explains that he reported his discovery of a simple way of detecting and thus blocking all Tor traffic he wrote: "I reported this simple way to detect Tor traffic to the Tor Project on 27-Dec-2017 (HackerOne bug #300826). The response that I got back was disappointing."

Hello and thanks for reporting this issue!

This is a known issue affecting public bridges (the ones distributed via bridgedb); see ticket #7349 for more details. This issue does not affect private bridges (the ones that are distributed in a P2P ad hoc way). As indicated in the ticket, to fix this problem, we are aiming to make it possible to shutdown the ORPort of Tor relays. In our opinion, we should not try to imitate normal SSL certs because that's a fight we can't win (they will always look differently or have distinguishers, as has been the case in the pluggable transports arms race).

Unfortunately, ticket #7349 is not straightforward to implement and has various engineering complexities; please see the ticket for more information

Due to the issue being known and planned to be fixed, I'm marking this issue as Informative.

The doctor was displeased and his blog posting itemized his disagreements with this decision. Then he concludes with some commentary on bug bounties and a promise for more "0days" -- which I think are simply presently unknown vulnerabilities. He wrote:

More Soon
If you have ever worked with bug bounties, then you are certain to recognize the name Katie Moussouris. She created the first bug bounty programs at Microsoft and the Department of Defense. She was the Chief Policy Officer at HackerOne (the bug bounty service), and she spear-headed NTIA's Awareness and Adoption Group's effort to standardize vulnerability

disclosure and reporting. (Full disclosure: I was part of the same NTIA working group for a year. I found Katie to be a positive and upbeat person. She is very sharp, fair-minded, and realistic.)

Earlier this month, Katie was interviewed by the Vergecast podcast. I had expected her to praise the benefits of vulnerability disclosure and bug bounty programs. However, she surprised me. She has become disenchanted by how corporations are using bug bounties. She noted that corporate bug bounties have mostly been failures. Companies often prefer to outsource liability rather than solve problems. And they view the bug bounties as a way to pay for the bug and keep it quiet rather than fix the issue.

Every problem that Katie brought up about the vulnerability disclosure process echoed my experience with the Tor Project. The Tor Project made it hard to report vulnerabilities. They fail to fix vulnerabilities. They marked issues as 'resolved' when they were never fixed. They outsource simple issues, like passing a simple scrollbar issue upstream to Firefox where it is never fixed. And they make excuses for not addressing serious security issues.

During the interview, she mentioned that researchers and people reporting vulnerabilities only have a few options: try to report it, sell it, or go public. I've tried reporting and repeatedly failed. I've sold working exploits, but I also know that they can be used against me and my systems if the core issues are not fixed. (And even the people who buy exploits from me would rather have these vulnerabilities fixed.) That leaves public disclosure.

In future blog posts, I will be disclosing more Tor 0day vulnerabilities. Most (but probably not all) are already known to the Tor Project. I have a list of vulnerabilities ready to drop. (And for the Tor fanboys who think "use bridges" will get around this certificate profiling exploit: don't worry, I'll burn bridges next.)

I thought this story was interesting and worth covering and sharing with our listeners because it illuminates another facet of the weird security industry that we spend time looking at every week.

It’s certainly the case that a vulnerability hunter lacks the ability to force their discoveries to be fixed. But I think that “forcing discoveries to be fixed” is probably the wrong goal. If, after having been informed of it, an organization should choose not to repair a defect in their system, for whatever reason, isn’t that entirely their business? I understand the ego involvement and the temptation to force the issue. The security researcher is in possession of knowledge the public is not. But attempting to publicly shame an organization into bending to the hacker’s will feels wrong... especially when that public shaming must, to be effective, inherently put other users of the organization’s systems at some form of increased risk.

It’s clear how a formal bug bounty program such as HackerOne could be abused by an organization to purchase and then sit on their bugs. But, again, isn’t that exactly the right they have purchased? That’s part of the bargain. The bug hunter agrees not to disclose, and in return receives payment both for the documentation of the discovered problem and for their silence. What happens after that, is no longer the hackers business. The information has been sold.

I think that the core lesson of this next story is, in this day and age, in 2020 and beyond, it's truly necessary to do everything right... which brings us to the latest Zoom mistake. This one is not a bug. It's a design mistake:

Enabling Zoom Meeting Hacking

Back in April, in response to the flurry of interest in Zoom, both by the overworld, who wanted to use it, and the underworld who wanted to abuse it, "Zoom Bombing" became a thing which caused us to title an episode "Zoom go Boom." The trouble was that Zoom meeting were not required to have any password protection. Just the meeting code was sufficient to allow otherwise uninvited visitors to break into and disturb Zoom conferences of all kinds.

Zoom responded by adding a 6-digit PIN to protect entry into all recurring Zoom meetings. As we know, a 6-digit pin can provide useful security, but it must be deployed with care because it does not, by itself, contain much entropy. And, sure enough, Tom Anthony, the VP in charge of Product at SearchPilot in the UK, discovered that Zoom's implementation of 6-digit PINs had made a classic rookie mistake. And, frankly, it's kind of unforgivable in this day and age. But I'm sure they were in a hurry to close down the Zoom Bombing problem. What Tom discovered, somewhat to his amazement, was that Zoom had failed to implement any sort of rate-limiting to prevent high-speed brute-force guessing of these comparatively short 6-numeric-digit PINs. Like I said... a rookie mistake.

https://www.tomanthony.co.uk/blog/zoom-security-exploit-crack-private-meeting-passwords/
As Tom wrote in his write up, this enabled: "an attacker to attempt all 1 million possible PINs in a matter of minutes and gain access to other people's private Zoom meetings."

In the absence of any checks for repeated incorrect password attempts, no lockout, nor any rate-limiting for mistakes (and, really, correctly entering a 6-digit PIN? How difficult is that to get correct?), an attacker can leverage Zoom's web client (https://zoom.us/j/MEETING_ID) to continuously send HTTP requests try all the one million combinations in relatively short order. Tom noted: "With improved threading, and distributing across 4-5 cloud servers the entire 6-digit space could be checked within a few minutes."

He responsibly reported this glaring oversight to Zoom on April 1st. along with a Python-based proof-of-concept script. The next day Zoom took their web client offline since that was the largest exploit vector, and a week later they fixed the flaw.

So it's obviously good that this was caught early and fixed quickly. But the lesson here is that this should really never have happened in the first place. The problem we have today is that truly important security pieces are still be implemented in an ad hoc fashion. Everyone is still needing to reinvent the same wheel over and over again. This approach invites mistakes. Every developer rolls their own web UI to suit their particular needs, so every one is different. Everyone handles login differently. There's no uniformity about passwords. Can I use special characters? How long can it be? What if I forget it? Everyone handles account recovery differently. Today, what we have is a mess. And not only does user convenience suffer dramatically, so does security.

It's going to be interesting to see how this gets resolved.

Another SHA-1 Deprecation
In a posting titled: "SHA-1 Windows content to be retired August 3, 2020" (in other words, yesterday), last week Microsoft made the following announcement:

https://techcommunity.microsoft.com/t5/windows-it-pro-blog/sha-1-windows-content-to-be-retired-august-3-2020/ba-p/1544373

> To support evolving industry security standards, and continue to keep you protected and productive, Microsoft will retire content that is Windows-signed for Secure Hash Algorithm 1 (SHA-1) from the Microsoft Download Center on August 3, 2020. This is the next step in our continued efforts to adopt Secure Hash Algorithm 2 (SHA-2), which better meets modern security requirements and offers added protections from common attack vectors.
>
> SHA-1 is a legacy cryptographic hash that many in the security community believe is no longer secure. Using the SHA-1 hashing algorithm in digital certificates could allow an attacker to spoof content, perform phishing attacks, or perform man-in-the-middle attacks.
>
> Microsoft no longer uses SHA-1 to authenticate Windows operating system updates due to security concerns associated with the algorithm, and has provided the appropriate updates to move customers to SHA-2 as previously announced. Accordingly, beginning in August 2019, devices without SHA-2 support have not received Windows updates. If you are still reliant upon SHA-1, we recommend that you move to a currently supported version of Windows and to stronger alternatives, such as SHA-2.

The only consequences I can see this having would be for those among us who have reason to set up and update older versions of the operating system. For example, I was testing SpinRite's forthcoming USB prep technology, which I packaged as "InitDisk" under Windows XP because it needs to run there. And Windows 7, which continues to valiantly hold onto to a bit more than one quarter of the desktop market share, at 26.72%, originally shipped without support for SHA-256. So, I wonder whether there will be a bit of a Catch-22... because there are Windows updates that are required to add SHA-256 support to WinXP and Win7... so they must still be signed with SHA-1 in order to allow SHA-256 support to be bootstrapped into those earlier versions. Fortunately, I long again created kits of all of the required files. So I'm okay. And I imagine that's also true for most others who have a similar interest in archaeology.

QNAP and QSnatch
QNAP NAS devices have been giving their owners and the security industry some serious problems for nearly a year, though the troubles appear to date as far back as 2014. We've touched on this before, but it's worthy of a brief refresher because there's been a recent escalation, and last week the cybersecurity agencies in both the US and UK issued a joint advisory about a massive ongoing malware threat which is infecting the network attached storage appliances of this Taiwanese company's products.

The malware goes by the name QSnatch (or, for some reason, Derek). It is credential- and data-stealing malware which has compromised more than 62,000 devices since reports began last October, and there's a high degree of infection in North America and Western Europe.

It's probably for the best that the exact infection vector is not publicly known, but the US Cybersecurity and Infrastructure Security Agency (CISA) and the UK's National Cyber Security Centre (NCSC) wrote in their joint alert that "All QNAP NAS devices are potentially vulnerable to QSnatch malware if not updated with the latest security fixes." And also that "Once a device has been infected, attackers can prevent administrators from successfully running firmware updates." Talk about a Catch-22!

QSnatch has an assortment of features which are implemented as modules. Therefore, these include, but are not necessarily limited to:

- A password logger which installs a fake device admin login page for obtaining credentials
- A more generic credential scrapper
- An SSH backdoor enabling the attackers to run arbitrary code on infected devices
- A web shell to provide malware operators with remote access to compromised NAS devices
- A data theft module which steals a predefined list of files (including logs and system configuration) and sends them in encrypted form to attacker-controlled servers

So, suffice to say, if you own QNAP device, or are aware of anyone who does, you should make time to follow QNAP's Security Advisory guidelines. I have a link in the show notes.

One thing in particular is worth noting. They say: "QSnatch collects confidential information from infected devices, such as login credentials and system configuration. Due to these data breach concerns, QNAP devices that had been infected may still be vulnerable to reinfection after removing the malware." In other words, don't flush the malware but retain the use of your favorite passwords in the device for the sake of connectivity with other applications of users or whatever. You should assume that a complete compromise has taken place, including all accounts on the device. It's a pain, but they must all be abandoned.

https://www.qnap.com/en-us/security-advisory/nas-201911-01

SpinRite

My discussion, last week, of the forthcoming mass storage benchmark, which will be another development spinoff of the ongoing work toward SpinRite v6.1, generated a lot of interest and feedback. Our listeners wanted to know where it was, and how they could run it. So I need to quickly note that it's still in development and is not yet ready for general use. Believe me when I say that at this point it would cause far more confusion, frustration and questions than it would answer. But as soon as it's ready for general purpose use, it will be easy to find and I'll formally invite all of our listeners to experiment with it.

I mentioned last week that I was going to add further granularity to the benchmark, to look at the timing of the 32 individual 32-megabyte transfers within the larger 1 gigabyte benchmark. We did that, and the results were interesting. We definitely found spots where drives, both spinning and solid state, but primarily solid state, were a great deal slower to respond. And in general we're seeing much more evidence that highly-used regions of SSDs, typically at the front of the drive, under the operating system, are consistently performing much more slowly, sometimes at as little as half the speed, when compared to unused areas.

We know that SSDs broadly employ two management schemes to make up for the technology's inherent lack of write endurance: They perform wear levelling which relocates the data in more highly used regions to lesser used regions. And just as with hard drives, SSDs are generously over-provisioned to allow regions that finally wear out to be replaced with fresh storage that had been set aside for that purpose.

What we think we're seeing could be the extra time required for error correction, which would tend to be required to fix low-bit-count errors as memory becomes fatigued, and we might also be seeing evidence of the overhead associated with the management of what eventually becomes physically fragmented solid state storage. In any event, this doesn't appear to be something that there's much awareness of today, but this benchmark reveals it conclusively, and I imagine that our listeners will find this fascinating.

So, stay tuned! :)

BootHole

Early in the history of this podcast, well before modern "Secure Booting" was invented, we talked about how truly insidious rootkits could be. By hooking and subverting the operating system's own file system, a user could be looking right at a directory containing malicious malware and not see it. You would do a directory listing, and the rootkit's API hooks would simply filter out any and all appearance of any files it didn't want you to see. But they would still be right there in front of you, unseen. Rootkits are, therefore, a big problem when the goal is to have a truly trustworthy system.

And we've previously covered the concept of secure booting quite thoroughly in several contexts, both securely booting a PC and also iOS devices. The fundamental idea is the establishment of a chain of trust which is anchored by some root component that can be absolutely trusted and which is then able to examine and verify the trustworthiness of each and every additional stage of the booting process. And with Secure Booting enabled the integrity of the resulting system is both assured and assumed.

So, when two researchers, Mickey Shkatov and Jesse Michael, of Eclypsium, announced their discovery of a vulnerability, which they called "BootHole" in the GRUB2 bootloader utilized by

most Linux systems, which can be used to gain arbitrary code execution during the boot process — even with Secure Boot enabled — this understandably generated quite a stir within the security industry. This meant that attackers could exploit this vulnerability to break boot security and install persistent and stealthy bootkits — another name for rootkits — to provide near-total control over the victim device. "GRUB" by the way, stands for "GRand Unified Boot" loader.

https://eclypsium.com/wp-content/uploads/2020/08/Theres-a-Hole-in-the-Boot.pdf

In their disclosure document the researchers wrote:

> The vulnerability affects systems using Secure Boot, even if they are not using GRUB2. Almost all signed versions of GRUB2 are vulnerable, meaning virtually every Linux distribution is affected. In addition, GRUB2 supports other operating systems, kernels and hypervisors such as Xen. The problem also extends to any Windows device that uses Secure Boot with the standard Microsoft Third Party UEFI Certificate Authority. Thus the majority of laptops, desktops, servers and workstations are affected, as well as network appliances and other special purpose equipment used in industrial, healthcare, financial and other industries. This vulnerability makes these devices susceptible to attackers such as the threat actors recently discovered using malicious UEFI bootloaders.
>
> *[In other words, the idea of corrupting a system's boot is not just theoretical. It's actively happening in the wild, now.]*
>
> [They continue…] Eclypsium has coordinated the responsible disclosure of this vulnerability with a variety of industry entities, including OS vendors, computer manufacturers, and cybersecurity emergency response teams. Mitigation will require new bootloaders to be signed and deployed, and vulnerable bootloaders should be revoked to prevent adversaries from using older, vulnerable versions in an attack. This will likely be a long process and take considerable time for organizations to complete patching.

In other words… Whoopsie!

So, I'm going to cut to the chase because part 2 of this is everything that has happened since: In their disclosure document they explain…

> In the course of Eclypsium's analysis, we have identified a buffer overflow vulnerability in the way that GRUB2 parses content from the GRUB2 config file (grub.cfg). Of note: The GRUB2 config file is a text file and typically is not signed like other files and executables. This vulnerability enables arbitrary code execution within GRUB2 and thus control over the booting of the operating system. As a result, an attacker could modify the contents of the GRUB2 configuration file to ensure that attack code is run before the operating system is loaded. In this way, attackers gain persistence on the device.
>
> Such an attack would require an attacker to have elevated privileges. However, it would provide the attacker with a powerful additional escalation of privilege and persistence on the device, even with Secure Boot enabled and properly performing signature verification on all loaded executables. One of the explicit design goals of Secure Boot is to prevent unauthorized code, even running with administrator privileges, from gaining additional privileges and pre-OS persistence by disabling Secure Boot or otherwise modifying the boot chain.

With the sole exception of one bootable tool vendor who added custom code to perform a signature verification of the grub.cfg config file in addition to the signature verification performed on the GRUB2 executable, all versions of GRUB2 that load commands from an external grub.cfg configuration file are vulnerable. As such, this will require the release of new installers and bootloaders for all versions of Linux. Vendors will need to release new versions of their bootloader shims to be signed by the Microsoft 3rd Party UEFI CA. It is important to note that until all affected versions are added to the dbx revocation list, an attacker would be able to use a vulnerable version of shim and GRUB2 to attack the system. This means that every device that trusts the Microsoft 3rd Party UEFI CA will be vulnerable for that period of time.

First, as regards the buffer overflow, UEFI does **not** employ address space layout randomization, data execution prevention, or any of the other common exploit mitigation protections that have fortunately become standard in operating systems. This means that weaponizing this buffer overflow will be trivial for attackers who already have a foothold on the targeted computer to exploit the flaw. From there, the protections many people expect to prevent boot kits from taking hold can be readily bypassed.

We talked about this before, but it's worth noting that, thankfully, the Secure Boot system was understood to require some form of truly effective revocation mechanism. So every UEFI system which supports Secure Boot contains a pair of protected databases; the "Allow DB" (db) lists approved components, and the "Disallow DB" (dbx) contains a list of known vulnerable or malicious components, including firmware, drivers, and bootloaders. So what this means is that all previously vulnerable boot loading components, GRUB2 and anything else that might be found, will need to be explicitly added to every single trusted motherboard UEFI system.

But wait, there's more!...

In response to Eclypsium's initial vulnerability report, the GRUB2 code came under additional — and, as it turns out, very much needed scrutiny — and a distressing number of additional vulnerabilities were then discovered by the Canonical security team:

- CVE-2020-14308 GRUB2: grub_malloc does not validate allocation size allowing for arithmetic overflow and subsequent heap-based buffer overflow
- CVE-2020-14309 GRUB2: Integer overflow in grub_squash_read_symlink may lead to heap based overflow
- CVE-2020-14310 GRUB2: Integer overflow read_section_from_string may lead to heap based overflow
- CVE-2020-14311 GRUB2: Integer overflow in grub_ext2_read_link leads to heap based buffer overflow
- CVE-2020-15705 GRUB2: avoid loading unsigned kernels when grub is booted directly under secureboot without shim
- CVE-2020-15706 GRUB2 script: Avoid a use-after-free when redefining a function during execution
- CVE-2020-15707 GRUB2: Integer overflow in initrd size handling.

And, given the difficulty of this kind of ecosystem-wide update and revocation, there is a strong desire to avoid having to do this again in six months. So a large effort spanning multiple security

teams at Oracle, Red Hat, Canonical, VMware and Debian, using static analysis tools and manual code review, have identified and fixed dozens of further vulnerabilities and dangerous operations throughout the GRUB2 codebase that do not yet have individual CVEs assigned.

So, what needs to be done to now fully respond to the revelation of this flaw? Broadly, five things:

1. Updates to GRUB2 to address the vulnerability.

2. Linux distributions and other vendors using GRUB2 will need to update their installers, bootloaders, and shims.

3. New shims will need to be signed by the Microsoft 3rd Party UEFI CA.

4. Administrators of affected devices will need to update installed versions of operating systems in the field as well as installer images, including disaster recovery media.

5. Eventually the UEFI revocation list (dbx) needs to be updated in the firmware of each affected system to prevent running this vulnerable code during boot.

We haven't talked about the need for "shims." Open-source projects and other third parties create a small app called a "shim." It contains the vendor's certificate and code that verifies and runs the bootloader (GRUB2). The vendor's shim is verified using the Microsoft 3rd Party UEFI CA and then the shim loads and verifies the GRUB2 bootloader using the vendor certificate embedded inside the shim.

While it's certainty true that Secure Boot loading should be as secure as we can make it, some knowledgeable security industry insiders feel that way too much ado is being made of this whole thing. HD Moore, widely acknowledged to be an expert in vulnerability exploitation who developed the Metasploit Framework, told Ars Technica's Dan Goodin: "I'd argue that Secure Boot is not the foundation of PC security today, because it is rarely effective, and by Eclypsium's own claim, it has been easy to bypass for over a year now, with no long-term fix in sight. I'm not sure what the buffer overflow in GRUB2 is useful for, since there are other problems if the grub.cfg is unsigned. It may be useful as a malware vector, but even then, there is no reason to exploit a buffer overflow when a custom grub.cfg file can be used instead to chain load the real OS."

And, moreover, there's an aspect of this (reminiscent of Spectre and Meltdown) where the cure is arguably worse than the problem, because Red Hat's patch to GRUB2 and the kernel, once applied, is rendering those systems unbootable. The issue has been confirmed to affect Red Hat Enterprise Linux v7.8 and v8.2 and it may also affect v8.1 and v7.9. The derivative distribution CentOS is also affected.

Consequently, Red Hat is currently advising users **not** to apply the GRUB2 security patches until these issues have been resolved. They say, if someone has installed the fix do not reboot your system. Downgrade the affected packages. And if the patches were applied and a system reboot was attempted and failed, users should boot from an RHEL or CentOS DVD in Troubleshooting mode, set up the network, then back out to restore the system's boot.

Additionally, although that problem was first reported in Red Hat Enterprise Linux, apparently related bug reports are rolling in from other distributions from different families as well. Ubuntu and Debian users are reporting systems which cannot boot after installing GRUB2 updates, and Canonical has issued an advisory including instructions for recovery on affected systems.

So, it's certainly good that the GRUB2 code got a clearly much-needed close examination with many fixes. But this particular problem requires the grub.cfg file to first be somehow maliciously altered. And doing that would require physical access to the system or elevated privileges. And, at the moment, updating to fix this might render one's system completely unusable.

The obvious advice, since the sky is not actually falling, would be to wait a while until all of the dust from this has settled and the various kinks have been worked out of the process. Then, have a leisurely update and know that a bunch of potentially exploitable flaws have been fixed.