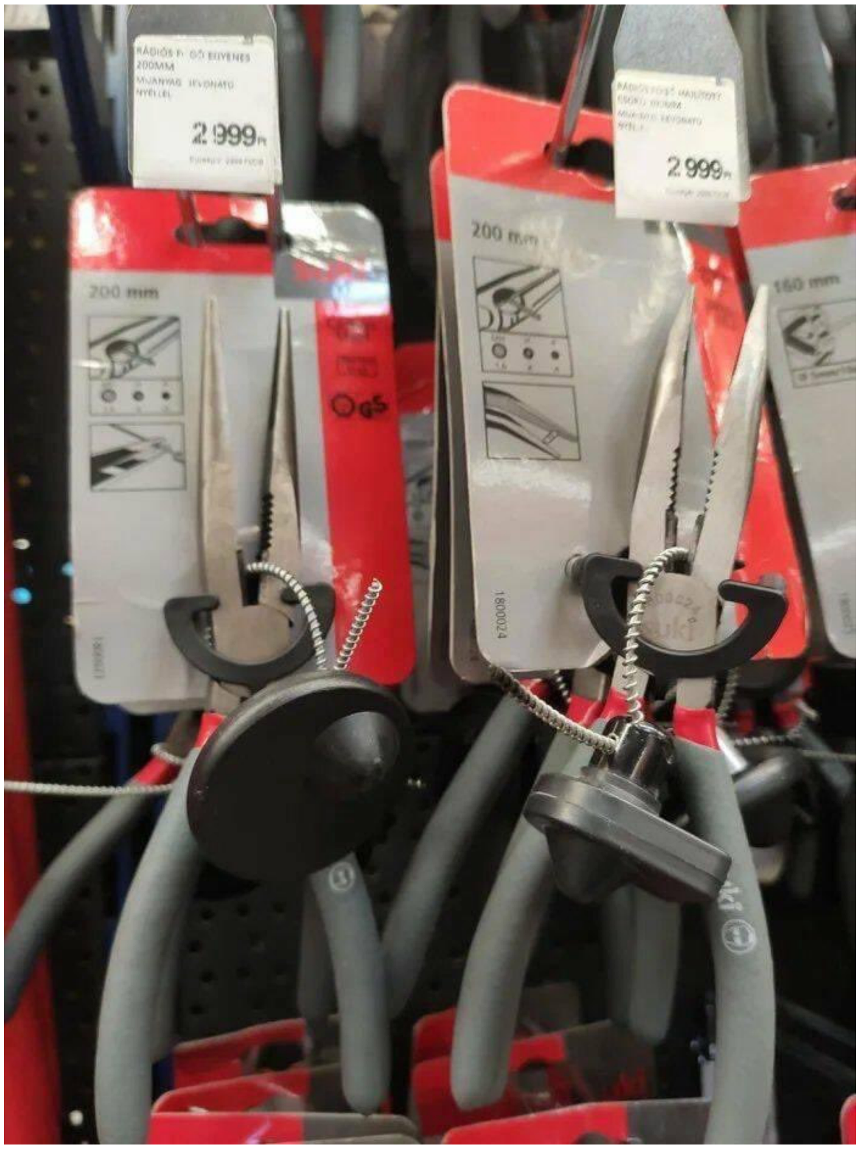
Security Now! #919 - 04-18-23
Forced Entry

This week on Security Now!

So... what happened with last week's Patch Tuesday? was there anything of note? If we took a quick overview of just a tiny bit of last week's news, what would that look like? and what would those stories all have in common? What new developer-centric service is Google making freely available for the good of the open source community? What moves is WhatsApp making to improve the security for the world's most popular secure messaging system? What happens when a European psychotherapy clinic apparently doesn't care enough to provide even minimal security for the patient's records? And finally, in this week's deep dive, we're going to answer the question: What could researchers have found inside a piece of the NSO Group's Pegasys smartphone spyware that actually terrified them? And why?

What's wrong with this picture?? (Security Theatre?)



Security News

Patch Tuesday Review

Last Tuesday was our monthly celebration of all the things we know of -- hopefully all the things we know of -- that we wish were different in the details of our Windows software.

I would like to say that, to that end, we received new and improved software. But since we never appear to be getting ahead, we're basically just treading water hoping to keep our heads above, what that suggests we actually received last week was software that is broken in new and different ways than what we had before. What's different, is that we haven't yet discovered all of the new ways that this replacement software is broken. But we have eliminated a bunch of the ways that we knew the old software was broken.

So, we'll be back next month and the month after and the month after that, ad infinitum. This will never end. Think about that. This shows no sign of ever ending. It's not as if the number of problems is diminishing over time, patch counts are dropping and there's any chance of us not needing one of these monthly rituals. I'm not really up on pop culture. Leo typically helps me with these things. But I think that the official technical term for this monthly dance, that ultimately goes nowhere, is "whack-a-mole."

This month we reportedly cured 97 flaws in our software from Microsoft, including one actively exploited 0-day flaw. There were 7 flaws which received a critical rating because they allowed for remote code execution, however overall, nearly HALF of this month's vulnerabilities allowed for remote code execution. Wow. So, here was the breakdown:

- 45 Remote Code Execution Vulnerabilities
- 20 Elevation of Privilege Vulnerabilities
- 10 Information Disclosure Vulnerabilities
- 9 Denial of Service Vulnerabilities
- 8 Security Feature Bypass Vulnerabilities
- 6 Spoofing Vulnerabilities

As we know, the two most potent vulnerability classes are remote code execution and elevation of privilege. Together they formed 65 out of the total 97 patched problems. Oh, and that count is separate from the 17 newly discovered vulnerabilities which Microsoft fixed for us the week before in their Edge browser.

The 0-day vulnerability is an Elevation of Privilege (EoP) problem in the Windows Common Log File System Driver. This tends to be a problematic component of Windows which we're hearing about too often. The problem with Windows EoP bugs in core component drivers, such as the common long file driver, is that they are always there. They are installed on every Windows computer. Consequently, they allow attackers with few or no significant access privileges to promote themselves directly to the SYSTEM account — you know: root — which gives them free reign over the machine. This is why elevation of privilege, while it sounds less scary than remote code execution, can be at least as troublesome.

One of the critical-rated remote code execution bugs may be a bit extra-worrisome. Microsoft disclosed a remote flaw in their Microsoft Message Queue system, known as MSMQ. It provides a robust means for inter-system message passing communications when absolute reliability in messaging is critical. Since it's a bit special-purpose, it's not always enabled. But when it is, it can be trouble, because Microsoft gave it a 9.8 CVSS score. And apparently, from Microsoft's description: *"To exploit this vulnerability, an attacker would need to send a specially crafted malicious MSMQ packet to a MSMQ server. This could result in remote code execution on the server side."* Let's hope that MSMQ servers are entirely constrained within enterprise intranets and that none are exposed to the public Internet. We now know with absolute certainty that it's not only security researchers who compare pre-patched software to post-patched software in over to reverse-engineer the flaws that were fixed. So it won't be long before bad guys are working to gain entry to enterprises that (a) haven't yet applied the April patches to their border servers and (b) have the MSMQ service publicly exposed.

Another worrisome critical bug is a remote code execution flaw in Microsoft's DHCP server service. As we know, DHCP is the Dynamic Host Configuration Protocol which is the way most of the computers in the world receive their IP addresses. It's a slick system that allows a machine that knows nothing about the network it's connected to to emit a totally generic Ethernet broadcast packet out onto whatever network it's attached to – wired or wireless.
That packet, which essentially says *"Hey!... Uhhh... I'm here... Help!"* will be received by a listening DHCP server, which is then able to reply with all of the instructions and settings necessary to get that machine's IP stack configured with a unique, non-conflicting local IP address, the IP of the network's gateway and the addresses of some DNS servers.

Although GRC's server network is 100% statically configured, at home I have a combination of static IPs and DHCP. Some machines I want to always have at the same location whereas things like iPhones, iPads, Roku's and IoT thermostats don't matter and there's no point in typing them down. One of the cool things that a DHCP server can be configured for is the range of IPs that it is allowed to allocate from. So, for example, in a /24 network like those which most residential users have, which begins with 192.168.0 or .1, the final byte can range from 0 to 254. So that residential network's DHCP server can be configured to automatically assign IPs from the upper range 100 to 254 which leaves the lower 100 IPs 0-99 guaranteed to always be available for manual static assignment.

In any event, DHCP server flaws have historically and understandably been quite frightening because, if you stop to think about it, the entire nature of the way DHCP servers operate is inherently unauthenticated. By design, they are there for anyone who asks for their help, without question. This means that DHCP servers must be robust enough to accept and reply to packets from unknown, untrusted and potentially hostile devices. Fortunately, although we're seen very serious problems with DHCP services in the past, today's only rates an 8.8 out of 10 since the flaw is a post-authentication problem which, while still bad, is at least less so. Again quoting Microsoft: *"An authenticated attacker could leverage a specially crafted RPC call to the DHCP service to exploit this vulnerability."* And, being DHCP, it's going to be on the LAN, not the WAN, so exploitation of this vulnerability requires that an attacker first have access to the local internal network.

Since no serious post-patch meltdowns have been reported, there's no obvious reason not to update our machines to eliminate the latest batch of known problems while apparently, given all of the evidence, introduce at least as many new unknown problems. We'll see you next month.

Risky Business News

As I was perusing the news of the past week, the *"Breaches and hacks"* section of the *"Risky Business News"* newsletter had me shaking my head. I remind myself that our listeners are only aware, at least through this podcast, of those things that rise to the top of the week's news enough to seem worthy of mention, further exploration, and some deeper dives. But that can have the effect of inadvertently obscuring a broader view of just how much nonsense is continually going on out there in the wider world. Risky Business's tag line is *"It's a Jungle Out There"*. So I wanted to share what I encountered in just this one short section covering the past week's news. I'm just going to read the headlines and their short blurb... but you'll get the idea:

NCR gets ransomwared: *NCR, the world's largest banking and payments software maker, has confirmed that a recent data center outage was caused by a ransomware attack. According to NCR, the incident has impacted the availability of its Aloha PoS payments platform. The intrusion was claimed by the AlphV ransomware gang, which temporarily listed the company on its dark web leak site.*

Cyber-attack on irrigation systems: *A cyber-attack is suspected of having disrupted the operations of irrigation systems in Israel's Upper Galilee region. Water controllers for irrigating fields in the Jordan Valley and control systems for the Galil Sewage Corporation were down last week during yearly cyberattacks aimed at Israeli targets known as OpsIsrael. While several hacktivist groups announced their participation in OpsIsrael this year, the identity of the attacker remains unknown, the Jerusalem Post reported.*

Hyundai data breach: *Hyundai's Italian and French branches have leaked customers' details. Exposed data includes email addresses, telephone numbers, physical addresses, and vehicle chassis numbers.*

MSI ransomware attack: *Taiwanese hardware vendor MSI has confirmed a security breach after the MoneyMessage ransomware gang claimed to have breached and encrypted some of the company's systems.*

Hundred Finance crypto-heist: *The Hundred Finance platform was hacked for $7.4 million following a flash-loan attack.*

GDAC crypto-heist: *South Korean cryptocurrency platform GDAC was hacked earlier this month for $13 million worth of assets.*

Bitrue crypto-heist: *The Bitrue cryptocurrency exchange says it was hacked and lost $23 million worth of assets following an attack on one of its hot wallets on April 14. Bitrue says it's currently investigating the incident and doesn't know how the hack took place.*

Yearn Finance crypto-heist: *A threat actor exploited a bug in the Yearn Finance platform to steal $11.6 million worth of cryptocurrency assets.*

Terraport crypto-heist: *The Terraport DeFi platform was hacked for $4 million worth of crypto-assets.*

What a week!! Nineteen years ago, when Leo first proposed this podcast to me, none of this sort of nonsense was going on. We had cute little eMail macro viruses that liked to send out eMail to all of the contacts in someone's address book. You'd receive a bizarre eMail from your mom and call her up and say *"Uhh... mom... Unless we have a Nigerian cousin that you've never told me about, I think that maybe your computer has a virus."* And that was pretty much the extent of it, right? Ah... simpler times. But thinking about that brief news blurb summary that I just read, you know the one thing that they all have in common?; the thread that winds and weaves through them all? Cryptocurrency. That's today's common factor. Partly it's because it enables ransoms – payments of which are now facilitated by the inherent anonymity offered by cryptocurrency – but even moreso, it's the cryptocurrency itself. The software being used to implement cryptocurrency systems is apparently, given the evidence, no better than software anywhere else. But rather than getting the font wrong when an item is selected in a word processor, when a hole is found in cryptocurrency software, many millions of dollars of actual tradeable fiat currency drain out through that hole. And where there's money, there's bad guys.

Google Assured Open Source Software

Okay. Right about now we need some good news. And we have some courtesy of Google. Last Wednesday the 12th, Google announced that their so-called "Google Assured Open Source Software" service would now be generally available. Here's what Google posted:

> *Threats to the software supply chain and open source software (OSS) security continue to be major areas of concern for organizations creating apps and their developers. According to Mandiant's M-Trends 2022 report, 17% of all security breaches start with a supply chain attack, the initial infection vector second only to exploits.*
>
> *Building on Google's efforts to improve open source software (OSS) security, we are announcing the general availability of the Assured Open Source Software (Assured OSS) service for Java and Python ecosystems. Available today at no cost, Assured OSS gives any organization that uses open source software the opportunity to leverage the security and experience Google applies to open source dependencies by incorporating the same OSS packages that Google secures and uses in their own developer workflows.*

In other words, just to be clear, this new service will provide free use of the same vetted and carefully reviewed Java and Python packages that Google themselves uses internally for their own applications. Google explains:

> *Using Assured OSS, organizations can:*
> - *Obtain their OSS packages from a trusted and known supplier.*
> - *Know more about their ingredients with Assured SBOMs provided in industry standard formats like SPDX and VEX.*
> - *Reduce risk, as Google is actively scanning, finding, and fixing new vulnerabilities in curated packages.*
> - *Increase confidence in the integrity of the ingredients they're using through signed, tamper-evident provenance.*
> - *Choose from more than 1,000 of the most popular Java and Python packages, including common machine learning and artificial intelligence projects like TensorFlow and Pandas.*

Since our public preview announcement in May 2022 and integrating Assured OSS as a key component in Software Delivery Shield the following October, we have received an overwhelmingly positive response and interest from our customers.

Jon Meadows, managing director and Citi Tech Fellow, Cyber Security at Citi said, "Citi has been an advocate and active leader in the industry's efforts to secure enterprise software supply chains. Both Citi and Google see untrusted and unverified open source dependencies as a key risk vector. This is why we've been excited to be an early adopter of Google Cloud's new Assured OSS product. Assured OSS can help reduce risk and protect open source software components commonly used by enterprises like us."

Assured OSS guards OSS packages against attacks and risk by:

- *Continuously mirroring key external ecosystems to manage end-to-end security without creating forks.*
- *Managing the security and integrity of the mirrored repos and end-to-end build tool chain with tamper-evident provenance and attestations.*
- *Continuously scanning for, fuzz testing, and fixing critical vulnerabilities, which are then quickly contributed back upstream to limit the exposure time and blast radius.*
- *Operating a critical patching team to support covered packages.*

Melinda Marks, senior analyst with ESG, said: "As organizations increasingly utilize OSS for faster development cycles, they need trusted sources of secure open source packages. Without proper vetting and verification or metadata to help track OSS access and usage, organizations risk exposure to potential security vulnerabilities and other risks in their software supply chain. By partnering with a trusted supplier, organizations can mitigate these risks and ensure the integrity of their software supply chain to better protect their business applications."

There are significant security benefits to Assured OSS adopters and the larger community from the curation process. Since our Assured OSS team curated the first 278 packages, we have been the first to find 48% of the new vulnerabilities (CVE) — each of these CVEs has been fixed and upstreamed.

To me, this seems like all good news. I do have the sense that this might cause free software open source purists to blow a gasket if they sense an effort to in any way privatize or taint the openness and freeness of open source software. But we all know too well now that public open source software registries are under constant and growing attack. So while the ideal of having an open community of like-minded contributors is terrific, the reality is that not everyone is like-minded.

So I, for one, think that it makes all kinds of sense to have access to a vetted and curated source of open source software. So, bravo, Google!

WhatsApp Improvements

I'm not a big social media user. As Leo often notes, it took him quite a while to get me onto Twitter. And since I'm not moving state secrets securely — and since I'm skeptical that it's actually possible to do so with any mainstream technology — the only encrypted messaging service I bother with is iMessage. And it wouldn't matter much to me if it wasn't encrypted. I have some friends with green bubbles, so those SMS messages are flying in the clear.
But, when the world's leading secure messaging platform – WhatsApp – announces security

improvements, it's something that we need to at least take note of. So, last week, WhatsApp received three new security features: Account Protect, Device Verification and Automatic Security Codes. Here's how WhatsApp described these new features. The first is “Account Protect”:

Account Protect: *If you need to switch your WhatsApp account to a new device – we want to double check that it’s really you. From now on, we may ask you on your old device to verify that you want to take this step as an extra security check. This feature can help alert you to an unauthorized attempt to move your account to another device.*

Okay. So, that one seems kind of obvious. We often see that when we’re changing an eMail account, a notification is sent to both the old and the new accounts to help prevent a malicious eMail change. This is the same sort of thing. And, sort of similarly, when a new trust relationship is being established, both ends are asked to affirm that they both wish to trust the other. So this new so-called “Account Protect” feature is obviously useful, but you have to wonder why it’s only happening now.

Okay, next up we have what WhatsApp is calling “Device Verification.” They write:

Device Verification: *Mobile device malware is one of the biggest threats to people’s privacy and security today because it can take advantage of your phone without your permission and use your WhatsApp to send unwanted messages. To help prevent this, we have added checks to help authenticate your account - with no action needed from you - and better protect you if your device is compromised. This lets you continue using WhatsApp uninterrupted.*

So, “Device Verification” is some back-end technology designed to prevent malware which crawls into a user's phone from obtaining the user's authentication token and then impersonating them in subsequent secure messages. But exactly what is going on here isn’t yet clear. So here's how WhatApp describes the situation:

WhatsApp uses several cryptographic keys to ensure that communications across the app are end-to-end encrypted. One of these is the authentication key, which allows a WhatsApp client to connect to the WhatsApp server to re-establish a trusted connection. This authentication key allows people to use WhatsApp without having to enter a password, PIN, SMS code, or other credential every time they turn on the app.

This mechanism is secure because the authentication key cannot be intercepted by any third party including WhatsApp. If a device is infected with malware, however, the authentication key can be stolen.

We are primarily concerned about the popularity of unofficial WhatsApp clients that contain malware designed for this purpose. These unofficial apps put users’ security at risk – and it is why we encourage everyone using WhatsApp to use the official WhatsApp app.

Once malware is present on user devices, attackers can use the malware to capture the authentication key and use it to impersonate the victim to send spam, scams, phishing attempts, etc. to other potential victims. Device Verification will help WhatsApp identify these *scenarios and protect the user's account without interruption.*

Now, I'll just note that WhatsApp is being optimistic if not disingenuous, since serious spy malware, such as we'll be looking at closely as our final topic today, is purpose-designed to do this with the official WhatsApp app. In any event, the problem is certainly real. So here's how they're solving this problem:

WhatsApp has built Device Verification to benefit from how people typically read and react to messages sent to their device. When someone receives a message, their WhatsApp client wakes up and retrieves the offline message from WhatsApp server. This process cannot be impersonated by malware that steals the authentication key and attempts to send messages from outside the user's device.

Device Verification introduces three new parameters:

- *A security-token that's stored on the user's device.*
- *A nonce that is used to identify if a client is connecting to retrieve a message from WhatsApp server.*
- *An authentication-challenge that is used to asynchronously ping the user's device.*

These three parameters help prevent malware from stealing the authentication key and connecting to WhatsApp server from outside the user's device. Every time someone retrieves an offline message, the security-token is updated to allow seamless reconnection attempts in the future. This process is called bootstrapping the security-token.

Every time a WhatsApp client connects to the WhatsApp server, we require the client to send us the security-token that's on their device. This allows us to detect suspicious connections from malware that is trying to connect to the WhatsApp server from outside the user's device.

An authentication-challenge is an invisible ping from the WhatsApp server to a user's device. We only send these challenges on suspicious connections. There are three possible responses to the challenge:

- *Success: The client responds to the challenge from the connecting device.*
- *Failure: The client responds to the challenge from a different device. This means the connection being challenged is very likely from an attacker and the connection will be blocked.*
- *No Response: The client doesn't respond to the challenge. This situation is rare and indicates that the connection being challenged is suspicious. We retry sending the challenge a few more times. If the client still doesn't respond, the connection will be blocked.*

Then the finished by adding:

Device Verification has been rolled out to 100% of WhatsApp users on Android and is in the process of being rolled out to iOS users. It enables us to increase our users' security without interrupting their service or adding an additional step they need to take. Device Verification will serve as an important and additional tool at WhatsApp's disposal to address rare key-theft security challenges. We will continue to evaluate new security features to protect the privacy *of our users.*

Okay. So what they've described is a simple protocol for detecting when two physically separate clients are both checking-in for messages. The idea is that every WhatsApp client will now be maintaining some state in the form of this new token which they have received from the WhatsApp server. It's probably just a simple random nonce that was most recently received from a WhatsApp server. This nonce will be changed by the server and sent every time a WhatApp client connects to transact messages. And at the start of each new connection, the client will return the last nonce it received to the server. Clearly, if only one client is doing this, the server will always receive the nonce that it had previously sent to the client. But if a user's WhatsApp identity authentication has been stolen and is being used by a physically separate client, then two clients will both be returning nonces and only one of them will have the nonce that was most recently sent to that user's account by the server. When the client that didn't most recently connect to the WhatsApp server does so, it will provide an obsolete nonce to the server and the server will know that something's not right. So it's a clean and simple solution to this cloned-client problem.

The final new feature they're calling "Automatic Security Codes," which they describe:

Automatic Security Codes: *Our most security conscious users have always been able to take advantage of our security code verification feature, which helps ensure you are chatting with the intended recipient. You can check this manually by going to the encryption tab under a contact's info. To make this process easier and more accessible to everyone, we're rolling out a security feature based on a process called "Key Transparency" that allows you to automatically verify that you have a secure connection. What it means for you is that when you click on the encryption tab, you'll be able to verify right away that your personal conversation is secured.*

I read through WhatsApp's description of what they're calling "Automatic Security Codes." What it amounts to is a public, auditable, append-only (meaning nothing can ever be deleted) log of the use of WhatsApp user account public keys.

The WhatsApp app already allows users to display, share and verify their public keys with either a QR code or by verifying a 60-digit number. But doing that requires that both parties arrange to interact in real time, preferably face to face with their phones to exchange QR codes. As we know, this was a feature that the Threema secure messaging app has used since it's inception.

So what WhatsApp is now creating under the name "Key Transparency" is a directory of user accounts and their public keys which will allow one-sided verification of the public keys of user's contacts on WhatsApp. The user's experience is that they're able to press a button to verify the public key of any of their contacts, and that verification happens immediately. So, another nice feature implemented in a way that makes sense.

And this is good since, as I noted, WhatsApp remains the global leader in secure messaging.

Bad Security? Go to jail!

Remember back near the beginning of February, it was during our episode #909, Leo and I talked about a particularly astonishing and horrifying data breach at a Psychotherapy clinic.

Here's how I described it then:

The news was that French authorities have detained a 25-year-old Finnish national who is accused of hacking the Vastaamo Psychotherapy Center. For reasons we'll see, this hack of Vastaamo is considered to be one of the worst in the country's history. Okay. Now, it occurred back in 2018 and 2019, so I guess this kid was, what, 20 years old then, when he allegedly stole the personal medical records of the clinic's patients and attempted to extort the clinic. To put pressure on the company, the hacker leaked extremely sensitive client files on the dark web. When that failed, he sent emails with ransom demands to more than 30,000 of the clinic's patients, asking them each for 200 euros and threatening to publish their medical records if they did not pay up.

To which Leo replied: “Oh, boy.” And I continued...

Uh-huh. Finnish authorities formally identified the hacker in October last year when they issued a European arrest warrant for his arrest, and they detained him last week. Okay, so this is brazen and bad; right? The hacker obtained extremely sensitive personal medical information and chose to use it to extort both the clinic and its past patients, all 30,000 of them. And it was that number of files and patient histories that raised my eyebrows, 30,000. Okay. No matter how large and busy this clinic might be, they cannot be currently treating 30,000 patients. And in fact there are 260 working days a year, five times 52. So if the clinic averaged 10 new patients per day, which seems like a high-side number, 30,000 patient records would be 11.5 years' worth of patient files at the rate of 10 per day.

So that was then. Today, we have an update.

For one thing, the Finnish psychotherapy clinic is now bankrupt. What a surprise! That's what'll happen when apparently the entire past hyper-confidential history of your clients is publicly exposed to the Internet. Who's going to make an appointment there?

What's a bit more interesting is that the ex-CEO of the clinic also faced criminal charges. And it's difficult to feel sorry for him since, as we know, mistakes can happen by mistake, right? ... but policies do not happen by mistake. They happen by policy. So someone needed to be held accountable for this clinic's online preservation of the past records of its 30,000 patients. Now, that might not have been sufficient, but the truth turned out to be:

In the process of digging around in the facts of this data breach, it came to light that in addition to failing to take the sort of data security precautions that any medical patient would reasonably assume were in place, and that the law would both expect and require, the clinic's CEO knew about his company’s sloppy cybersecurity for up to two years before the blackmail took place in 2020.

Even worse, he presumably knew about the problems because, while under his watch, the clinic had suffered several previous breaches in 2018 and 2019, and failed to report them, presumably hoping that no traceable cybercrimes would arise as a result, and thus that the company would never be held to account. Of course, the final breach was, what in the security industry technically call "a doozy!" so there was no sweeping that one under the rug.

Moreover, current breach disclosure and data protection regulations, such as our favorite GDPR in Europe, make it very clear that data breaches can no longer simply be ignored with the hope that no one will find out. No. They must be promptly disclosed for the greater good of all.

So, the news from Finland is that our criminally negligent CEO was, in fact, found by the courts to be criminally negligent and has been convicted and given a prison sentence. (I wonder if he'll have the cell next to the young hacker who stole his data?) The hope is that this will serve to remind other business leaders that merely promising to look after other people's personal data is not good enough. You actually have to at least be able to demonstrate that you have tried to do so.

Forced Entry

My original title for today's podcast was "KingsPawn," the name that's been given to a powerful piece of spyware being offered by one of those Israeli spyware purveyors **other** than the NSO Group who, as we know, offer their "too well known for comfort" Pegasys spyware to various governments. But while doing the background legwork for **that** story I ran across the fascinating technical details of another earlier piece of Spyware that was used by both this second group and by Pegasys. I knew that our listeners would find the technical details of this interesting. So I decided to push our discussion of "KingsPawn" to next week, unless something even more juicy comes up. But we'll get to it. Today, I want to take us through what Google's Project Zero team discovered about the so-called "ForcedEntry" exploit that has been successfully deployed by several malware vendors to gain entry into Apple iPhones.

Citizen Lab, which is at the Munk School of Global Affairs and Public Policy at the University of Toronto in Canada, provided Google's Project Zero team with a sample of the ForcedEntry exploit and Apple's Security Engineering and Architecture group collaborated with Google in working through exactly how it was possible for a GIF thumbnail image to takeover any iPhone despite all of Apple's many layers of protection to make that impossible.

So what we had here with ForcedEntry was the holy grail of exploits providing attackers with remote code execution via a zero-click iMessage.

Citizen Lab somehow arranged to capture an NSO Group generated iMessage-based zero-click exploit while it was being used to target a Saudi activist. At the time that Project Zero wrote about this, they said: *"Based on our research and findings, we assess this to be one of the most technically sophisticated exploits we've ever seen, further demonstrating that the capabilities NSO provides rival those previously thought to be accessible to only a handful of nation states."*

And, of course, what makes this so disturbing is that this is for sale so that it's not only a handful of nation states that now have access to this sort of technology. Any qualified government can now purchase this advanced targeting malware. Here's how Project Zero described the NSO group. They wrote:

NSO Group is one of the highest-profile providers of "access-as-a-service", selling packaged hacking solutions which enable nation state actors without a home-grown offensive cyber capability to "pay-to-play", vastly expanding the number of nations with such cyber capabilities.

For years, groups like Citizen Lab and Amnesty International have been tracking the use of NSO's mobile spyware package "Pegasus". Despite NSO's claims that they "[evaluate] the potential for adverse human rights impacts arising from the misuse of NSO products" Pegasus has been linked to the hacking of the New York Times journalist Ben Hubbard by the Saudi regime, hacking of human rights defenders in Morocco and Bahrain, the targeting of Amnesty International staff and dozens of other cases.

The United States has added NSO to the "Entity List", to severely restrict the ability of US *companies to do business with NSO and stating in a press release that "[NSO's tools] enabled foreign governments to conduct transnational repression, which is the practice of authoritarian governments targeting dissidents, journalists and activists outside of their sovereign borders to silence dissent."*

Citizen Lab was able to recover these Pegasus exploits from an iPhone and therefore this analysis covers NSO's capabilities against iPhone. We are aware that NSO sells similar zero-click capabilities which target Android devices; Project Zero does not have samples of these exploits but if you do, please reach out.

In previous cases analyzed, targets were sent links in SMS messages. And while, as we know, these sorts of phishing-style attacks are often successful, high-risk individuals may also be more highly trained and thus less apt to click on something that doesn't look absolutely 100% legitimate. So although a one-click exploit might work, the holy grail is the exploit that requires zero action on the part of the target; the exploit just works silently in the background. Short of not using a device, there is no way to prevent exploitation by a zero-click exploit; it's a weapon against which there is no defense.

Since the initial entry point for Pegasus on iPhone using this attack is iMessage. This means that a victim can be targeted just using their phone number or AppleID username.

iMessage offers native support for GIF images which can be directly sent and received in iMessage chats. They show up in the chat window's chat log timeline. As Leo and his Discord denizens are all too aware, GIF images can also be animated by causing the GIF to contain a series of frames which are displayed in succession. Apple wanted to make those GIFs loop endlessly, rather than only play through once, so very early in the incoming iMessage parsing and processing pipeline, after a message has been received but well before the message is displayed, iMessage calls a method in the IMTranscoderAgent process to which is passes any image file it receives with the extension .gif. The method called is named "IMGIFUtils" with "CopyFromPath" and a "ToDestinationPath" parameters.

As suggested by the selector's name, the intention was presumably to just copy the GIF file before editing the loop count field to make the GIF repeat endlessly. The method uses the CoreGraphics APIs to render the source image to a new GIF file at the destination path. And although the source filename does need to end in .gif, that doesn't mean it's actually a GIF file.

So, apparently we're going to learn this lesson all over again. Or at least a variation of it: Apple's ImageIO library is used to determine the actual format of the source file and then parse it, **completely and deliberately ignoring the file extension**. This was actually done to prevent file type confusion attacks, but in this instance the protection backfired. Thanks to the use of this "fake gif" trick, over 20 different image processing codecs are suddenly part of the iMessage zero-click attack surface. Unfortunately, this includes some very obscure and complex formats. As a consequence of this unintentional chain of dependencies, hundreds of thousands of lines of code would now be remotely exposed. Last week's podcast "A Dangerous Interpretation" was just one example of how difficult it can be to get complex codecs exactly right.

So... what codec do you imagine the NSO Group chose to invoke with their "fake GIF in an iMessage" attack? Would you believe a PDF. <sigh> Yes. Believe it or not. The unintended chain of dependencies in iMessage's processing of incoming GIF images allowed external remote malicious parties to send anyone a PDF which the receiving iPhone would then attempt to parse and display. For how many years were we, as an industry, dealing with malicious PDF's?

As we know, PDFs were one of the more popular exploitation targets due to the file format's ubiquity and the complexity of its – wait for it – it's interpreter. And on top of the difficulty of rendering a document image from a typesetting specification, PDFs later acquired the ability to interpret JavaScript; which is where we all intone in unison: "WHAT could possibly go wrong?" Fortunately, Apple's CoreGraphics PDF parser doesn't appear to interpret JavaScript. But those evil geniuses at the NSO Group found something sufficiently powerful lying inside the CoreGraphics PDF parser...

Turning back the clock to the late 1990's, we can all remember a time when bandwidth and storage were much more scarce than they are today – and when it made much more sense to code in assembly language. Back then, a compression standard known as JBIG2 was created. JBIG2 had a very narrow and very specific purpose. It was an image codec designed to highly compress monochrome images where pixels were only be black or white. It was developed to achieve extremely high compression ratios for scans of text documents and was implemented and used in high-end office scanner/printer devices. And if you were to use the direct "Scan to PDF" feature of such a machine, a PDF file would be produced which was mostly just a thin PDF wrapper around a JBIG2 stream.

As it happens, the PDFs files produced by those scanners were exceptionally small, often being on the order of only a few kilobytes. How was this possible? There were two novel techniques which JBIG2 used to achieve its extreme compression ratios which are relevant to the exploitation we're talking about here.

The first technique was known as "Segmentation and substitution." If you think about it, a text document contains a bunch of text. And in a language such as English having a relatively small alphabet, the same character, like 's', 't' and 'e' will appear all over the place. Believe it or not, JBIG2 actually segments documents to be compressed into their individual glyphs and uses a simple pattern recognition matching to match and collect all of the glyphs that look the same.

Then rather than storing all of those near duplicate glyphs, JBIG2 pretends that they are all the same and replaces all of them with just one. This replacement of all occurrences of similar-looking glyphs with a copy of just one often yields a document which remains entirely legible and enables very high compression ratios since the compressor just needs to store the coordinates with each of the same-looking glyphs rather than the glyphs themselves.

The output remains perfectly readable but the amount of information to be stored is significantly reduced. Again, rather than needing to store all the original pixel information for the whole page, we only need one compressed version of the "reference glyph" for each character shape and the relative coordinates of all the places where its copies should be made.

This means that the decompression stage treats the output page like a canvas onto which it "paints" that one copy of the glyph at each of the stored locations.

There is one big problem with such a scheme: It’s possible for a poor encoder to accidentally get confused and exchange similar looking characters. This explains why the format fell out of favor and became obscure. It doesn’t explain, though, why Apple never thought to remove its decompression codec from their ImageIO library.

Most of Apple’s CoreGraphics PDF decoder appears to be Apple’s own proprietary code, but the JBIG2 implementation is straight from Xpdf, whose source code is freely available. This means that the NSO Group effectively had the full source code for an extremely complex and not well vetted interpreter to which they were able to provide any amount of data and cause to be invoked on any unsuspecting target’s iPhone.

The vulnerability that was found and exploited was a classic integer overflow which occurs when the code processes a deliberately malformed document containing a JBIG2 compressed image description. This flaw allows the contents of the image to be executed as code, allowing the attackers total freedom to inject and run any code of their choosing. The only problem is that the messages sent remain on the target’s device. And it was the persistence of the attacking code that triggered the unraveling of this entire scheme.

The extent to which the NSO Group went in their implementation of this vulnerability is somewhat astonishing:

Since replacing all of an image’s text with identical duplicate versions of just one master reference character will result in an inherently lossy compression – because the decompressed image will not be identical to the original – an additional mode for lossless compression was added to JBIG2. An error mask could be created to represent the difference between the original image and the lossy super-compressed image. And it turned out that this error mask was not large either. However, this capability gave the decompressor the ability to perform XOR operations, since the mask was an XOR mask which was able to flip bits, as needed, to make the final output identical to the original input. This meant that the decompressor’s interpreter had access to AND, OR, XOR and XNOR logical operators. Given these, any computable function can be computed. We wind up with a machine, a system, which is “Turing Complete.”

JBIG2 does not, itself, have scripting capabilities, but when combined with a vulnerability which was found, it does have the ability to emulate circuits of arbitrary logic gates operating on arbitrary memory. So, these genius attackers must have thought: “why not just use that to build an entire computer architecture from scratch and script it!?” And believe it or not, this is exactly what the NSO Group’s “Forced Entry” exploit does!!

By using over 70,000 JBIG2 interpreted segment commands defining logical bit operations, they defined an entire small computer architecture with features such as registers and a full 64-bit adder and comparator which they use to search memory and perform arithmetic operations. It's not as fast as Javascript, but who cares? It's fundamentally computationally equivalent.

Here’s what the Project Zero guys wrote to describe their amazement and their reactions to what they discovered once they had reverse-engineered this exploit. They wrote:

The bootstrapping operations for the sandbox escape exploit are written to run on this logic circuit and the whole thing runs in this weird, emulated environment created out of a single decompression pass through a JBIG2 stream. It's pretty incredible, and at the same time, pretty terrifying.

From what I've just described it's clear why the Project Zero guys, who are immersed in and live this sort of stuff day and night, nevertheless described this as the single most sophisticated attack they had ever encountered. On the one hand it amazed them, as it would anyone. And it also terrified them to think that there are people out there somewhere who are not only willing to go to these lengths but who have the capability to do so. This is a tour de force in fundamental computer science. These people built a working emulation of a 64-bit computer, replete with registers and math capability out of the **pixels** being manipulated by a JBIG2 image decompressor.

Impressive as this is as an example of astonishing computational mastery, it's also a tragedy that the entire goal of this effort was to subvert iPhones belonging to journalists and human rights defenders for the purpose of illegally spying on their actions and communications.

It seems a shame for such talent and capability to be spent toward such an end.