Security Now! #968 - 04-02-24
A Cautionary Tale

This week on Security Now!

Why should all Linux users update their systems if they haven't since February? What do 73 million current and past AT&T customers all have in common? What additional and welcome, though very different, new features await Signal and Telegram users? Which major IT supplier has left Russia early? What did Ghostery's ad blocking profile reveal about Internet users? Whatever happened with that Incognito-mode lawsuit against Google? And how are things going in the open source repository world? And then, after I share something kinda special that happened Sunday involving my Wife, SpinRite and her laptop — and it's probably not what you think — we're going to take a look at another rather horrifying bullet that the Internet dodged again.

Where there's a will…

Security News

A near-Universal (Local) Linux Elevation of Privilege vulnerability

Last Friday (Good Friday) was a rough day for Linux. Not only did the world first learn of a near-miss on the dissemination of an SSH backdoor – which we'll be covering in detail later – but the public learned of a widespread elevation of privilege vulnerability affecting the Linux kernel from at least versions 5.14 through 6.6.14. Running the exploit as a normal user – which was released as a proof of concept on GitHub – on a vulnerable machine will grant root access allowing the exploiter to do whatever they wish. Being a local-only exploit, this could be used by rogue insiders or malware already on a computer to cause further damage and problems.

Affected distributions include Debian, Ubuntu, Red Hat, Fedora, and doubtless many other Linux's and it had a 99.4 percent success rate given 1000 tries. In other words, it worked 994 times and only failed 6 times.

I noted that the public learned of it last Friday because it was responsibly disclosed to Linux insiders in January and since the end of January updates to this CVSS 7.8 flaw have been rolling out. The fix was clear and simple, so it did not take long to patch nor test. And it's because the updated code had been available for two months that on Friday its discoverer posted everything about it on GitHub and also published an extremely detailed description of what he found.

The takeaway for anyone using Linux machines that may not have been updated since before February, is that knowledge of this is now public and widespread. So if anyone untrusted might have or gain access to any unpatched machines, it might be good to get them updated.

https://pwning.tech/nftables/ https://github.com/Notselwyn/CVE-2024-1086

TechCrunch informed AT&T of a 5 year old data breach

TechCrunch has been on top of the news of a significant breach of very sensitive customer data from AT&T. The story begins five years ago, which TechCrunch first reported ten days ago on March 22nd under the headline *"AT&T won't say how its customers' data spilled online"*. TechCrunch wrote:

Three years after a hacker first teased an alleged massive theft of AT&T customer data, a breach seller this week dumped the full dataset online. It contains the personal information of 73 million AT&T customers.

A new analysis of the fully leaked dataset — containing names, home addresses, phone numbers, Social Security numbers, and dates of birth [everything needed for identity theft] — points to the data being authentic. Some AT&T customers have confirmed their leaked customer data is accurate. But AT&T still hasn't said how its customers' data spilled online.

The hacker, who first claimed in August 2021 to have stolen millions of AT&T customers' data, only published a small sample of the leaked records at the time, making it difficult to verify its authenticity.

AT&T, the largest phone carrier in the United States, said back in 2021 that the leaked data "does not appear to have come from our systems," but it chose not to speculate as to where the data had originated or whether it was valid.

*Troy Hunt, a security researcher and owner of data breach notification site **Have I Been Pwned**, recently obtained a copy of the full leaked dataset. Hunt concluded the leaked data was real by asking AT&T customers if their leaked records were accurate.*

In a blog post analyzing the data, Hunt said that of the 73 million leaked records, the data contained 49 million unique email addresses, 44 million Social Security numbers, and customer dates of birth.

When reached for comment, AT&T spokesperson Stephen Stokes told TechCrunch in a statement: "We have no indications of a compromise of our systems. We determined in 2021 that the information offered on this online forum did not appear to have come from our systems. This appears to be the same dataset that has been recycled several times on this forum."

The AT&T spokesperson did not respond to follow-up emails by TechCrunch asking if the alleged customer data was valid or where its customers' data came from.

As Troy Hunt notes, the source of the breach remains inconclusive. And it's not clear if AT&T even knows where the data came from. Hunt said it's plausible that the data originated either from AT&T or "a third-party processor they use or from another entity altogether that's entirely unrelated."

What is clear is that even three years later, we're still no closer to solving this mystery breach, nor can AT&T say how its customers' data ended up online.

Investigating data breaches and leaks takes time. But by now AT&T should be able to provide a better explanation as to why millions of its customers' data is online for all to see.

So that's what TechCrunch reported ten days ago. In follow-up reporting last Saturday, under their headline *"AT&T resets account passcodes after millions of customer records leak online"* TechCrunch wrote:

TechCrunch has exclusively learned that phone giant AT&T has reset millions of customer account passcodes after a huge cache of data containing AT&T customer records was dumped online earlier this month. The U.S. telco giant initiated the passcode mass-reset after TechCrunch informed AT&T on Monday that the leaked data contained encrypted passcodes that could be used to access AT&T customer accounts.

A security researcher who analyzed the leaked data told TechCrunch that the encrypted account passcodes are easy to decipher. TechCrunch alerted AT&T to the security researcher's findings.

In a statement provided Saturday, AT&T said: "AT&T has launched a robust investigation supported by internal and external cybersecurity experts. Based on our preliminary analysis, the data set appears to be from 2019 or earlier, impacting approximately 7.6 million current AT&T account holders and approximately 65.4 million former account holders."

The statement also said: "AT&T does not have evidence of unauthorized access to its systems resulting in exfiltration of the data set."

TechCrunch withheld the publication of this story until AT&T could begin resetting customer account passcodes. AT&T also has a post on what customers can do to keep their accounts secure.

AT&T customer account passcodes are typically four-digit numbers that are used as an additional layer of security when accessing a customer's account, such as calling AT&T customer service, in retail stores, and online.

This is the first time AT&T has acknowledged that the leaked data belongs to its customers, some three years after a hacker claimed the theft of 73 million AT&T customer records. AT&T had denied a breach of its systems, but the source of the leak remains inconclusive.

AT&T said Saturday that "it is not yet known whether the data in those fields originated from AT&T or one of its vendors."

Security researcher Sam Croley told TechCrunch that each record in the leaked data also contains the AT&T customer's account passcode in an encrypted format. Croley double-checked his findings by looking up records in the leaked data against AT&T account passcodes known only to him. Croley said it was not necessary to crack the encryption cipher to unscramble the passcode data.

He took all of the encrypted passcodes from the 73 million dataset and removed every duplicate. The result amounted to about 10,000 unique encrypted values, which correlates to each four-digit passcode permutation ranging from 0000 to 9999, with a few outliers for the small number of AT&T customers with account passcodes longer than four digits.

In other words, AT&T used a fixed cipher with no salt to encrypt for the entire dataset and that encryption was never changed. Every encryption maps one-to-one to a four-digit passcode. This means that there's no mystery here. TechCrunch wrote that:

"According to Croley, the insufficient randomness of the encrypted data means it's possible to guess the customer's four-digit account passcode based on surrounding information in the leaked dataset." <unquote>

And since all 73 million users were restricted to using 4-digit passcodes, and all passcodes are identically encrypted, once any single user's passcode can be guessed, and we see what that encrypts to, we also know everyone else who chose the same 4 digits because they will have an identical encrypted code. It's kind of a mess. TechCrunch wrote:

It's not uncommon for people to set passcodes — particularly if limited to four digits — that mean something to them. That might be the last four digits of a Social Security number or the person's phone number, the year of someone's birth, or even the four digits of a house number. All of this surrounding data is found in almost every record in the leaked dataset.

By correlating encrypted account passcodes to surrounding account data — such as customer dates of birth, house numbers, and partial Social Security numbers and phone numbers —

Croley was able to reverse-engineer which encrypted values matched which plaintext passcode.

Yep. Like I said, a mess.

*AT&T said it will contact all of the 7.6 million **existing** customers whose passcodes it reset, as well as current and former customers whose personal information was compromised.*

Let's hope that they have a new and improved means of encrypting passcodes, hopefully using a unique per-user random salt so that it's no longer possible to create a simple one-to-one mapping of passcodes to encryptions. Doing this without sale is really so last century.

Signal to get very useful cloud backups
I haven't (and don't have) much need for cross platform secure messaging. iMessage does everything I need. But as I mentioned years ago, the talented PHP programmer, Rasmus Vind, suggested that we use Signal to converse while were were working to add SQRL login to GRC's XenForo forums. So that exposed me to Signal, and to its many restrictions.

So I was interested and pleased to read that Signal is beginning to loosen up a bit. The site "SignalUpdateInfo.com" which apparently exists to watch and report on Signal happenings wrote:

Cloud Backup / Status: In Development

There has not been any official announcement, but it appears that Signal is currently working on cloud backup for iOS and Android. This feature would allow you to create cloud backups of your messages and media.

While we have known about cloud backups since a commit to Signal-iOS on October 20, 2023, a recent commit to Signal-Android has revealed many more details. It looks like there could be both free and paid plans. The free plan could provide full message text backups, but only media backups from the last 30 days. The paid plan could provide full message text and media backups with a storage limit of 1TB.

From the commit, we also know that Signal backups will be end-to-end encrypted and completely optional, with the ability to delete them at any time. Cloud backups could significantly improve usability on all platforms by preventing complete data-loss in the event that you lose your phone or encounter some other issue.

Apparently this suggests that there may also be a means for migrating content between devices, which would be a nice boost in usability.

Telegram to allow restricted incoming
While we're on the topic of secure messaging apps, Telegram users, initially located in Russia, Belarus, and Ukraine, have received a welcome new feature which allows users to restrict who

can message them. Last Wednesday, Telegram's founder, Pavel Durov, sent the following message through Telegram. I had it in Russian, so I had Google translate it, which did a pretty good job. Pavel wrote:

> *Four days ago, Russian-speaking Telegram users began to complain about messages from strangers calling for terrorist attacks. Within an hour of receiving such complaints, we applied a number of technical and organizational measures in order to prevent this activity.*
>
> *As a result, tens of thousands of attempts to send such messages were stopped, and thousands of users participating in this flash mob faced permanent blocking of their Telegram accounts.*
>
> *From the beginning of next week, all users from Russia, Ukraine and Belarus will be able to limit the circle of those who can send them private messages. We are also implementing AI solutions to handle complaints even more efficiently.*
>
> *Telegram is not a place for spam mailings and calls for violence*✋

And speaking of Google Translation, when I fired up Chrome to view the page, had it perform the translation then copied and posted the result over to Firefox, I left Chrome running. It had that one page open, doing nothing... yet it caused my machine's fans to spin up. When I noticed that I opened Task Manager and saw that Chrome was at the top of the process utilization list, consuming half of my machine while doing nothing. I had forgotten that it used to do that when I was using it for a while and before switching back to Firefox. Thanks anyway.

HP exits Russia ahead of schedule

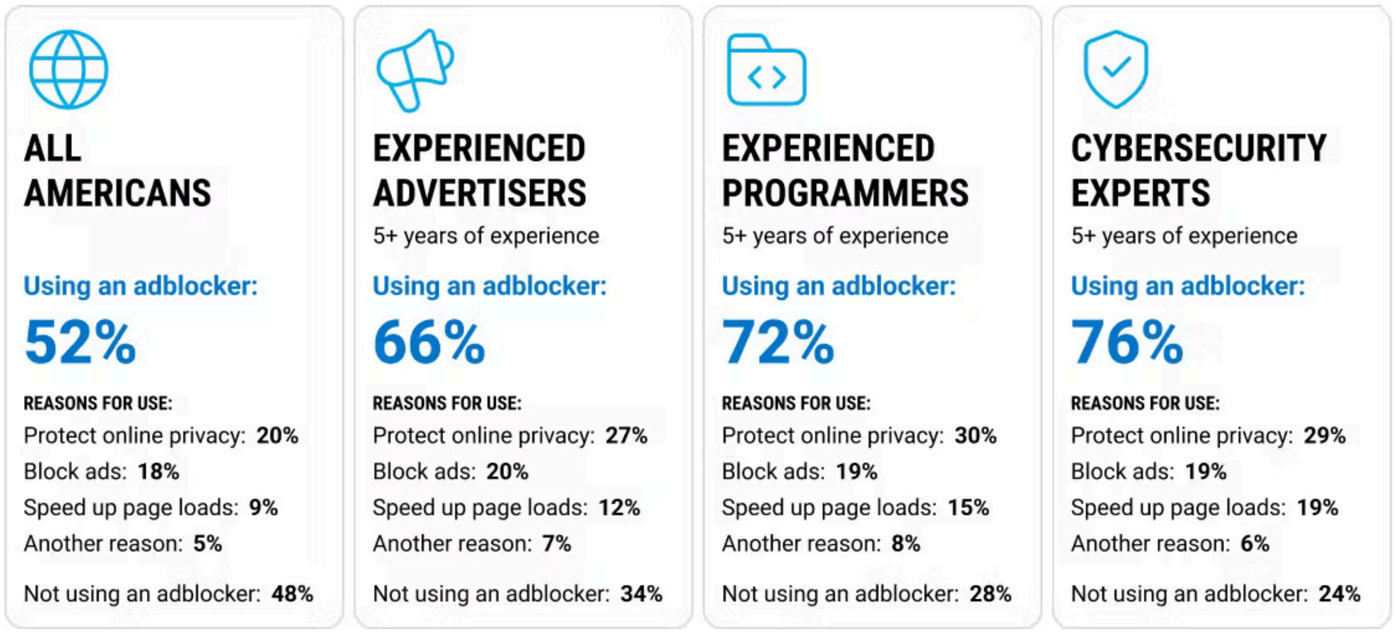
Speaking of Russia, HP has ceased all operations and has exited the Russian market ahead of schedule. HP had terminated all shipments into Russia two years ago, in February of 2022 after Russia's invasion of Ukraine. Three months later, in May of 2022, HP began the slow process of winding down operations and had planned to make their final exit next month in May 2024. But HP pulled out of Russia last week at the end of March, two months ahead of its planned departure. This move has surprised Russian companies, which are no longer able to update drivers or contact support.

Advertisers are heavier users of Ad Blockers that average Americans!

Ghostery published their Tracker and Ad Blocker report resulting from research conducted by third-party research firm Censuswide. It found that:

- Individuals who have experience in advertising, programming and cybersecurity are significantly more likely to use ad blockers than the average American.
- These industry insiders are more skeptical of their online safety, underscoring concerns about the current severity of user tracking.
- Americans are underestimating the dangers of health and political tracking.
- Lesser-known Big Tech players sow distrust among these experts.

The infographic in their report shows that whereas 52% of all Americans use an Ad Blocker (which is actually a far higher number than I would have guessed), 66% of experienced **advertisers** block ads. I suppose they don't wish to be tracked, either!



ALL AMERICANS	EXPERIENCED ADVERTISERS	EXPERIENCED PROGRAMMERS	CYBERSECURITY EXPERTS
	5+ years of experience	5+ years of experience	5+ years of experience
Using an adblocker: **52%**	Using an adblocker: **66%**	Using an adblocker: **72%**	Using an adblocker: **76%**
REASONS FOR USE:	REASONS FOR USE:	REASONS FOR USE:	REASONS FOR USE:
Protect online privacy: **20%**	Protect online privacy: **27%**	Protect online privacy: **30%**	Protect online privacy: **29%**
Block ads: **18%**	Block ads: **20%**	Block ads: **19%**	Block ads: **19%**
Speed up page loads: **9%**	Speed up page loads: **12%**	Speed up page loads: **15%**	Speed up page loads: **19%**
Another reason: **5%**	Another reason: **7%**	Another reason: **8%**	Another reason: **6%**
Not using an adblocker: **48%**	Not using an adblocker: **34%**	Not using an adblocker: **28%**	Not using an adblocker: **24%**

The infographic divides all those surveyed into four groups: Everyone, experienced advertisers, experienced programmers and cybersecurity experts. And that's also the order of increasing use of ad blockers: Everyone, advertisers, coders, and security experts with the percentages of use, respectively, being 52%, 66, 72 and 76%. So just over three quarters (76%) of cybersecurity experts surveyed are using ad blockers.

The other interesting question each group was asked was why those who are using ad blockers are doing so. They were asked to choose among one of four reasons: for the protection of their online privacy, to block ads, to speed up page loading or anything else. Interestingly, for every group of users, the ranking among those four reasons: Privacy, Freedom from Ads, Web Speed or Other was identical and in that order. The only difference was in the distribution among those reasons. The generic "All Americans" group was the most evenly split between privacy and not wishing to be confronted with ads at 20% and 18% respectively, and only 9% were using ad blockers for speed. But among programmers and cybersecurity experts the split was much greater at 30% wishing for privacy and 19% wanting not to see ads.

With the changes Google has not only been promising but is now well on the way to delivering, with the implementation and enforcement of their Privacy Sandbox technologies in Chrome, the connection between advertising and privacy encroachment is truly being broken for the first time ever. But it's such a significant change that we can expect the recognition of this to take quite some time to spread – probably much more time than we expect. I've heard from some of our listeners who simply don't believe it. They don't really care about the technology behind it. For them it's just blah, blah blah, blah. Even if they understand it. They're so jaded that they cannot conceive of a world where we are not implicitly paying for the web by relinquishing our personal information.

I believe in technology that I understand. And thanks to this podcast, which forced me to invest in acquiring an understanding of it so that I could share that understanding, I understand what Google has wrought. It is good, it is right and it makes sense. If it was me doing this, you know, like me doing SQRL, it wouldn't matter how good it was. But it's not me. It's Google. And it's Chrome. And it is going to matter. It's just going to take a long time for the rest of the world to catch up. And that's not a bad thing either. Inertia creates stability. And all of those techies who have been coding tracking technologies will need some time to find news jobs.

As I noted recently, it's the presence of advertising that's financing the web. And that shows no sign of changing. Advertisers appear to be willing to pay double when they know that their ads are being well-targeted – and "double" can easily make the difference between a website being financially viable and not. But as the statistics which Ghostery collected showed, everyone knows that the way advertising initially evolved has not been privacy friendly. So we've needed a long term solution for giving advertisers the ad targeting that they're willing to pay for while not trading away our personal information and privacy.

Google's final solution, which amounts to moving all advertising auctioning and ad choosing into the user's web browser is brilliant. It flips everything on its head. But it's a massive change that will not happen overnight.

The Google Incognito Mode Lawsuit

And speaking of Google and Privacy, remember that confusion over just exactly how incognito those using Chrome's "incognito mode" really were? Google's defense was that its users who were upset to learn that they were still being tracked and profiled while using "incognito mode" had simply failed to read the fine print about what, exactly, incognito mode did – and mostly did not – protect them from. Well, Google, it's time to massively delete everything you learned about your Chrome users while they believed that they were incognito.

The Hacker News just carried this bit of news this morning, and what's been learned during lawsuit discovery and under oath depositions is definitely worth sharing. They wrote:

Google has agreed to purge billions of data records reflecting users' browsing activities to settle a class action lawsuit that claimed the search giant tracked them without their knowledge or consent in its Chrome browser.

The class action, filed in 2020, alleged the company misled users by tracking their internet browsing activity who thought that it remained private when using the "incognito" or "private" mode on web browsers like Chrome. In late December 2023, it emerged that the company had consented to settle the lawsuit. The deal is currently pending approval by the U.S. District Judge Yvonne Gonzalez Rogers.

A court filing yesterday (on April Fool's day) said: "The settlement provides broad relief regardless of any challenges presented by Google's limited record keeping. Much of the private browsing data in these logs will be deleted in their entirety, including billions of event level data records that reflect class members' private browsing activities."

As part of the data remediation process, Google is also required to delete information that makes private browsing data identifiable by redacting data points like IP addresses, generalizing User-Agent strings, and remove detailed URLs within a specific website (i.e., retain only domain-level portion of the URL).

In addition, it has been asked to delete the so-called X-Client-Data header field, which Google described as a Chrome-Variations header that captures the "state of the installation of Chrome itself, including active variations, as well as server-side experiments that may affect the installation." What's significant here is that this header is generated from a seed value, making it potentially unique enough to identify specific Chrome users. Whoops!

Other settlement terms require Google to block third-party cookies within Chrome's Incognito Mode for five years, a setting the company has already implemented for all users. The tech company has separately announced plans to eliminate tracking cookies by default by the end of the year.

Google has since also updated the wording of Incognito Mode as of January 2024 to clarify that the setting will not change "how data is collected by websites you visit and the services they use, including Google."

The lawsuit extracted admissions from Google employees that characterized the browser's Incognito browsing mode as a "confusing mess," "effectively a lie," and a "problem of professional ethics and basic honesty." Ouch! It further laid bare internal exchanges in which executives argued Incognito Mode shouldn't be called "private" because it risked "exacerbating known misconceptions."

Canonical fights malicious Ubuntu store apps
We'll be talking, shortly, about the problems of malice in the open source world. It's a problem everyone is having to deal with. The NPM repository discovered eight new malicious packages last week and PyPI was again forced to disable new user account creation and package uploads after being hit by a malware submission wave. And Ubuntu's caretaker, Canonical, has just switched to manual reviews for all apps submitted to the Ubuntu OS app store. They needed to do this after multiple publishers attempted to upload malicious crypto-wallet applications to the store over the past few weeks. The problem focused enough upon crypto-wallet apps that they plan to create a separate app submission policy for cryptocurrency wallets going forward.

So it's kind of a mess out there.

SpinRite

The day before yesterday, my wonderful wife Lorrie had her first experience of what SpinRite can mean for long-term solid state storage management and maintenance. It meant a lot to me, so I wanted to share it with our listeners.

Sunday morning, she set up a Dell Laptop to take a "Q" (or QEEG), which is short for Quantitative Electroencephalogram (EEG). This is the first phase for delivering EEG biofeedback therapy to clients. A client sits motionless for 15 minutes with their eyes open and another 15 minutes with their eyes closed. They're wearing an elastic cap covered with EEG electrodes recording the surface electrical potentials across the exterior of their skull. Lorrie does these on Sunday mornings since there's no noise from nearby gardeners.

While getting everything ready, she turned the Dell laptop on to boot it up. And she began waiting. And waiting. And waiting. I don't recall whether I happened to wander by or she called me over. But the spinning dots were circling and the little hard drive light/icon was on solid.

She was impatient and beginning to grow worried. But I pointed to the hard drive light and told her that that machine was still working on getting booted up. Finally, after a truly interminable wait, Windows showed signs of life. When the desktop finally appeared, she started moving the cursor around. But the desktop was incomplete and I pointed to the little hard drive light again, showing her that even though the desktop was visible, Windows was still working to finish getting itself ready.

After another full minute or so, the rest of the tray icons finally populated and the drive light began flickering as Windows finally began to wrap up its work to get settled.

The program she uses to record EEGs is a true monstrosity. It was written by the engineers who designed and built the multi-channel EEG amplifier – which is a true work of art – but the software that goes with it? Not so much. So she wondered how long THAT would take to start up. She launched it and again waited and waited and waited. And it, too, finally showed up. She was relieved and knew that when the family she would be working with arrived she'd be ready.

Having observed all this, I said "It looks like it would be worth running SpinRite on that laptop." So a few hours later after a successful session of EEG recording, she brought the laptop to me. I plugged it in, booted into SpinRite, and set it going on Level 3 to perform a full rewrite/refresh of the drive. And I had to push past 6.1's new warning screen reminding me that I was going to be rewriting mass storage media that preferred being read to being written.

I did that and started SpinRite running on Lorrie's EEG recording laptop. Looking at the Real Time Monitoring display, where the reading and writing cycles are obvious, huge amounts of time were being spent reading and once a 16 megabyte block was successfully read, rewriting it was just a flicker on the screen. There were little bursts of rapidly alternating read and write flicker... but mostly very long pauses where the SSD was attempting to read its own media.

It was a 256 gigabyte SSD of which about 89 gigs were in use. And I noticed that by the time SpinRite was about half way through, the read and write phases were flickering back and forth almost without interruption with occasional pauses during writing while the SSD buffered, flushed and rearranged its data. So by that point I knew we were through with the region that mattered.

I interrupted SpinRite, removed the little USB boot drive and rebooted. And the difference was truly astonishing. Windows booted right up and was ready in maybe 15 seconds. So I shut it down and moved it back to where the rest of her EEG equipment still was that she had been using earlier.

Before we left the house for Easter dinner, I told her that SpinRite had finished with her machine and invited her to sit down and boot it up. Needless to say, she was astonished by the difference. She then launched the EEG recording software and it, too, started almost immediately.

She's put up with me for the past three and a half years, working every day and weekends and nearly every evening without a break or vacation. She's taken a few trips to visit friends while I've stayed home to work. And she's let me, because she knows it's what I have wanted to do more than anything else. But for this work to have made that much difference in the performance and operation of a machine she depends upon... well, it was a gratifying moment. “Yes, honey, your husband’s not insane after all.”

We discovered this phenomenon early in SpinRite’s development after I wrote the ReadSpeed utility as a platform for testing SpinRite's new drivers. We know that spinning hard drives transfer data more slowly as we move toward the end of the drive. But we naturally expected solid state storage to have uniform performance across its media -- and brand new SSDs do. What we discovered, was that SSDs that had been in service for a while had grown astonishingly slow at the front where files live that are often read but not often rewritten. What's fascinating to me as an engineer, is that the SSD in Lorrie's laptop could have become **that slow** while still eventually managing to return error free results.

We know that inside SSDs are analog bit storage cells where each cell contains a voltage that represents some number of binary bits. Originally, that was just one bit that was either on or off; high or low; fully charged or fully discharged. There’s always pressure to achieve higher density. But the cells were already tiny. So designers decided to store four voltages per cell instead of only two. Since four voltages could be used to represent two binary bits instead of just one, that would allow them to instantly double the storage density of the entire system. But why stop there? Eight voltages would store three bits and sixteen voltages (god help you) could be used to store four bits.

There's a well known problem with SSDs known as "read disturb" where reading from a block of cells subtly disturbs the storage of neighboring cells. We're witnessing that as time passes, SSDs gradually require more and more time to perform error correction, or error correction is being used across a greater and greater portion to fulfill a request. And it may also be that an SSD’s error correction is not fast since it's assumed not to be needed often.

And it’s not a stretch to imagine that if things are allowed to get bad enough, a solid state drive will eventually report that it's unable to read and recover the contents of a block of data. This is why preventative maintenance makes just as much sense for solid state media as it does for spinning platter drives. This was not something that anyone really understood and appreciated until we began working on SpinRite v6.1.

An SSD is not only being restored to "factory fresh" performance, but the reason it's performance is restored is that the voltages in the storage cells that had gradually drifted away over time from their original ideal levels will be reset. That means no more extensive and slow error correction, much faster performance, and presumably restored reliability.

So as I said, it meant a lot to me that Lorrie was able to finally see what I’d been up to for the past three and a half years. And I’m looking forward to having the rest of the world see the same thing for themselves. All of this was done with the second release of 6.1. The built-in RAM memory tester is finished and finalized and I've made a few other small improvements. After today's podcast, after some final testing, I'll publish the 3rd release of v6.1 and get back to work on bringing GRC's eMail system up so that I’m able to tell all of SpinRite’s current v6.0 owners that a free upgrade is waiting for them. And I know how much our listeners are looking forward to being able to send feedback via eMail. That’s not far off, either.

A Cautionary Tale

The runner-up title for today's "A Cautionary Tale" podcast was "A Close Call for Linux" because it was only due to some very small mistakes made by a very clever malicious developer that the scheme was discovered. What was discovered was that by employing a diabolically circuitous route, the system SSH daemon, which is to say the SSH incoming connection accepting server for Linux would have had a secret and invisible backdoor installed in it that would have then allowed someone, anyone, anywhere, using a specific RSA public key, to authenticate and login to any Linux system on the planet. To say that this would have been huge hardly does it justice.

And to be clear, unlike last week's coverage of the GoFetch vulnerability where Apple was informed 107 days before its public disclosure by a team of responsible researchers... someone actually created this extremely complex and very well hidden backdoor and, had it not been discovered, by chance, due to some small errors made by this malefactor, **it would have happened.** This would have happened. The Linux distros would have been updated and refreshed and they would have gradually moved out into the world to eventually become widespread. And this code was so well and cleverly hidden that it might have remained unseen for months or years – fortunately, we'll never know. I imagine it would have been eventually discovered... but who knows what damage might have been done before then. And imagine the frenzied panic that would have ensued when it ***was*** discovered that all builds of Linux with publicly exposed SSH services contained a remotely accessible keyed backdoor.

I went with the title "A Cautionary Tale" because the bigger takeaway lesson here should not be this specific instance, which I'll describe in further detail in a minute. It should be that there's nothing about this that's necessarily a one-off. This is the threat and the dark side of a massive community of developers working collectively and largely anonymously. There's no question that by far nearly every developer is well meaning. I am certain that's the case. But the asymmetry of the struggle for security, where we must get every single thing right to be secure and only one mistake needs to be made to introduce an insecurity, means that, unfortunately, the good guys will always be fighting an uphill battle. What has just been discovered present in Linux demonstrates that the same asymmetric principle applies to large scale software development, where just one bad seed, just one sufficiently clever malicious developer, can have an outsized effect upon the security of everything else.

I'll give everyone the TL;DR first because this is so cool and so diabolically clever:

How do you go about hiding malicious code in a highly scrutinized open source environment which worships code in its source form, so that no one can see what you've done? You focus your efforts upon a compression library project. Compression library projects contain test files which are used to verify the still-proper behavior of recently modified and recompiled source code. These compression test files are, and are expected to be, opaque binary blobs. So you very cleverly arrange to place your malicious binary code into one of the supposedly compressed compression test files for that library... where no one would ever think to look.

In their reporting of this, Ars Technica interviewed Will Dormann, a senior vulnerability analyst at the security firm Analygence. Will noted that because the backdoor was discovered before the malicious versions of xz Utils were added to production versions of Linux, <quote> *“it's not really affecting anyone in the real world. BUT that's only because it was discovered early due to bad actor sloppiness. Had it not been discovered, it would have been catastrophic to the world.”*

All of this erupted last Friday (Good Friday) with a posting to the oss-security list by the Microsoftie who stumbled upon this, Andreas Freund. As a consequence of this being a huge deal, security firms and developers plowed into and reverse engineered all of the available evidence. I’m sure that Ars Technica felt last Friday, being a news outlet, that they needed to say something. So Dan Goodin wrote what he could given the limited amount of information available to him at the time. But then late Sunday morning Dan published a second piece that did a much better job of pulling this entire escapade together and characterizing what’s known as well as some of the intriguing back story. I’ve edited it for the podcast, but here’s what we know with thanks to Ars Technica and Dan Goodin:

On Friday, a lone Microsoft developer rocked the world when he revealed a backdoor had been intentionally planted in xz Utils, an open source data compression utility available on almost all installations of Linux and other Unix-like operating systems. The person or people behind this project likely spent years on it. They were likely very close to seeing the backdoor update merged into Debian and Red Hat, the two biggest distributions of Linux, when an eagle-eyed software developer spotted something fishy.

Software and crypto engineer Filippo Valsorda said of the effort, which came frightfully close to succeeding. “This might be the best executed supply chain attack we've seen described in the open, and it's a nightmare scenario: malicious, competent, authorized upstream in a widely used library.”

xz Utils is nearly ubiquitous in Linux. It provides lossless data compression on virtually all Unix- like operating systems, including Linux. xz Utils provides critical functions for compressing and decompressing data during all kinds of operations. xz Utils also supports the legacy .lzma format, making this component even more crucial.

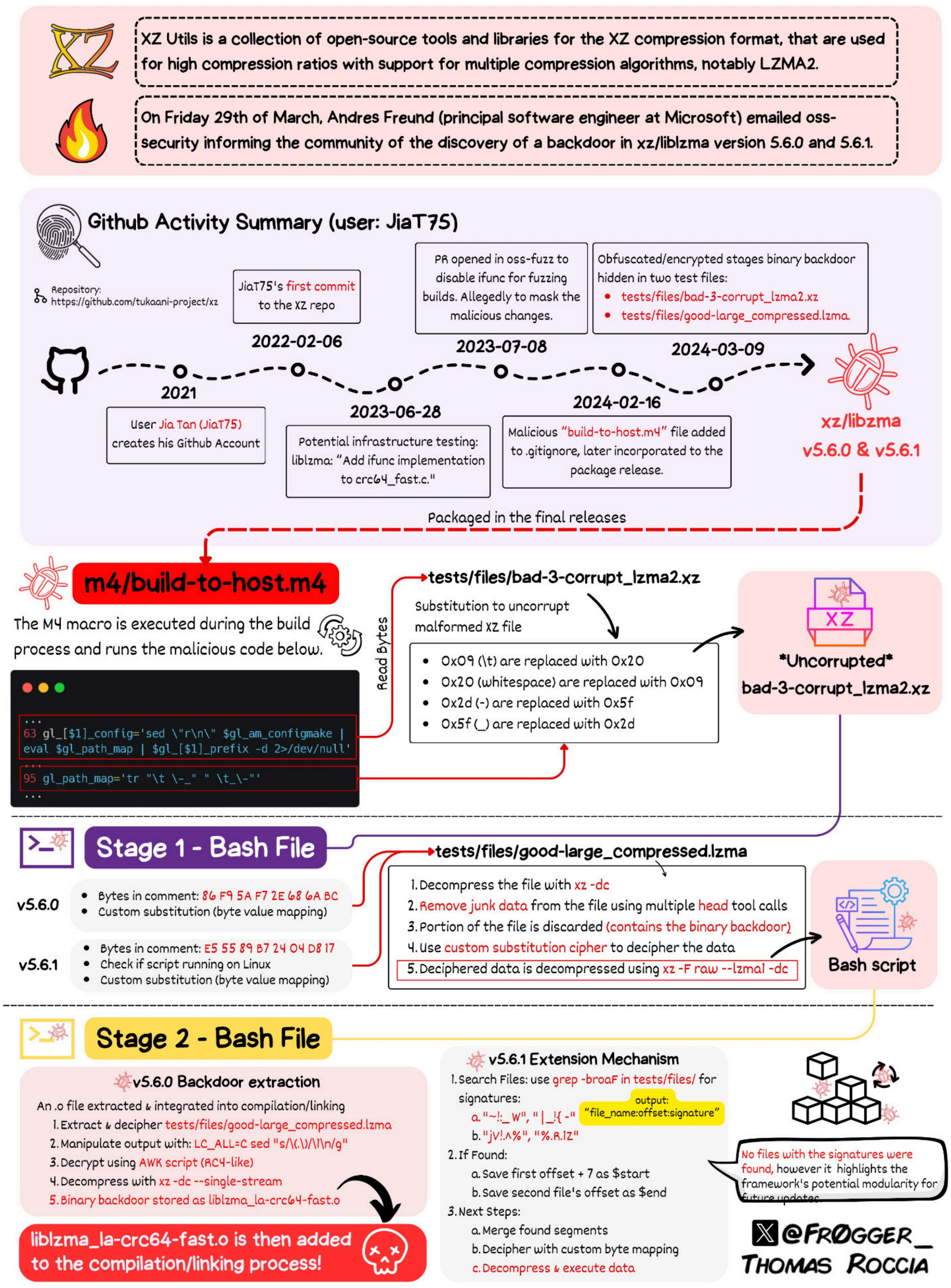
So what happened?

Andres Freund, a developer and engineer working on Microsoft’s PostgreS-Q-L offerings, was recently troubleshooting performance problems a Debian system was experiencing with SSH, the most widely used protocol for remotely logging in to devices over the Internet. Specifically, SSH logins were consuming too many CPU cycles and were generating errors with valgrind, a utility for monitoring computer memory.

Through sheer luck and Freund’s careful eye, he eventually discovered the problems were the result of updates that had been made to xz Utils. On Friday, Freund took to the Open Source Security List to disclose that the updates were the result of someone intentionally planting a backdoor in the compression software.

It's hard to overstate the complexity of the social engineering and the inner workings of the backdoor. Thomas Roccia (roe-cha), a researcher at Microsoft, published a graphic on Mastodon that helps visualize the sprawling extent of the nearly successful endeavor to spread a backdoor with a reach that would have dwarfed the SolarWinds event from 2020.

XZ Outbreak (CVE-2024-3094)



XZ Utils is a collection of open-source tools and libraries for the XZ compression format, that are used for high compression ratios with support for multiple compression algorithms, notably LZMA2.

On Friday 29th of March, Andres Freund (principal software engineer at Microsoft) emailed oss-security informing the community of the discovery of a backdoor in xz/liblzma version 5.6.0 and 5.6.1.

Github Activity Summary (user: JiaT75)

Repository: https://github.com/tukaani-project/xz

2021 User Jia Tan (JiaT75) creates his Github Account

2022-02-06 JiaT75's first commit to the XZ repo

2023-06-28 Potential infrastructure testing: liblzma: "Add ifunc implementation to crc64_fast.c."

2023-07-08 PR opened in oss-fuzz to disable ifunc for fuzzing builds. Allegedly to mask the malicious changes.

2024-02-16 Malicious "build-to-host.m4" file added to .gitignore, later incorporated to the package release.

2024-03-09 Obfuscated/encrypted stages binary backdoor hidden in two test files:

- tests/files/bad-3-corrupt_lzma2.xz
- tests/files/good-large_compressed.lzma

xz/liblzma v5.6.0 & v5.6.1

Packaged in the final releases

m4/build-to-host.m4

The M4 macro is executed during the build process and runs the malicious code below.

```
...
gl_[$1]_config='sed \"r\n\" $gl_am_configmake |
eval $gl_path_map | $gl_[$1]_prefix -d 2>/dev/null'
...
gl_path_map='tr "\t \-_" " \t_\-"'
...
```

Read Bytes

tests/files/bad-3-corrupt_lzma2.xz

Substitution to uncorrupt malformed XZ file

- 0x09 (\t) are replaced with 0x20
- 0x20 (whitespace) are replaced with 0x09
- 0x2d (-) are replaced with 0x5f
- 0x5f (_) are replaced with 0x2d

Uncorrupted bad-3-corrupt_lzma2.xz

Stage 1 - Bash File

v5.6.0

- Bytes in comment: 86 F9 5A F7 2E 68 6A BC
- Custom substitution (byte value mapping)

v5.6.1

- Bytes in comment: E5 55 89 B7 24 04 D8 17
- Check if script running on Linux
- Custom substitution (byte value mapping)

tests/files/good-large_compressed.lzma

1. Decompress the file with xz -dc
2. Remove junk data from the file using multiple head tool calls
3. Portion of the file is discarded (contains the binary backdoor)
4. Use custom substitution cipher to decipher the data
5. Deciphered data is decompressed using xz -F raw --lzmal -dc

Bash script

Stage 2 - Bash File

v5.6.0 Backdoor extraction

An .o file extracted & integrated into compilation/linking

1. Extract & decipher tests/files/good-large_compressed.lzma
2. Manipulate output with: LC_ALL=C sed "s/\(.\)/\1\n/g"
3. Decrypt using AWK script (RC4-like)
4. Decompress with xz -dc --single-stream
5. Binary backdoor stored as liblzma_la-crc64-fast.o

liblzma_la-crc64-fast.o is then added to the compilation/linking process!

v5.6.1 Extension Mechanism

1. Search Files: use grep -broaF in tests/files/ for signatures:
   a. "~!:_W", "|_!{ -" output: "file_name:offset:signature"
   b. "jV!.^%", "%.R.1Z"
2. If Found:
   a. Save first offset + 7 as $start
   b. Save second file's offset as $end
3. Next Steps:
   a. Merge found segments
   b. Decipher with custom byte mapping
   c. Decompress & execute data

No files with the signatures were found, however it highlights the framework's potential modularity for future updates.

@FR0GGER_
THOMAS ROCCIA

So what does the backdoor do?

Malicious code added to xz Utils versions 5.6.0 and 5.6.1 modified the way the software functions. The backdoor manipulated SSHd, the executable file used to make remote SSH connections. Anyone in possession of a predetermined encryption key could stash any code of their choice in an SSH login certificate, upload it, and execute it on the backdoored device. No one has actually seen code uploaded, so it's not known what code the attacker planned to run. In theory, the code could allow for just about anything, including stealing encryption keys or installing malware.

Answering the question about how a compression utility can manipulate any process as security sensitive as SSH, Dan explains:

Any library can tamper with the inner workings of any executable it is linked against. Often, the developer of the executable will establish a link to a library that's needed for it to work properly. OpenSSH, the most popular sshd implementation, doesn’t link the liblzma library, but Debian and many other Linux distributions add a patch to link sshd to systemd, a program that loads a variety of services during the system bootup. Systemd, in turn, links to liblzma, and this allows xz Utils to exert control over sshd.

And, of course, everyone wants to know how this all happened. Dan writes, and this should give everyone some chills...

It would appear that this backdoor was years in the making. In 2021, someone with the username JiaT75 made their first known commit to an open source project. In retrospect, the change to the libarchive project is suspicious, because it replaced the safe_fprint funcion with a variant that has long been recognized as less secure. No one noticed at the time.

The following year, JiaT75 submitted a patch over the xz Utils mailing list, and, almost immediately, a never-before-seen participant named Jigar Kumar joined the discussion and argued that Lasse Collin, the longtime maintainer of xz Utils, had not been updating the software often or fast enough. Kumar, with the support of Dennis Ens and several other people who had never had a presence on the list (and are never seen again), pressured Lasse Collin to bring on an additional developer to maintain the project.

In January 2023, JiaT75 made their first commit to xz Utils. In the months following, JiaT75, who used the name Jia Tan, became increasingly involved in xz Utils affairs. For instance, Tan replaced Collins' contact information with their own on oss-fuzz, a project that scans open source software for vulnerabilities that can be exploited. Tan also requested that oss-fuzz disable the ifunc function during testing, a change that prevented it from detecting the malicious changes Tan would soon make to xz Utils.

And I’ll just note that that sort of request is not unusual. It’s a bit like a website using the /robots.txt file to keep spiders out of places where they might cause damage. It’s possible that aggressive random fuzzing – which is hoping to detect unknown problems – might inadvertently trigger known problems. So asking for fizzing exceptions is not inherently suspicious. In this case the perpetrator did have ulterior motives.

Dan continues:

In February of this year, Tan issued commits for versions 5.6.0 and 5.6.1 of xz Utils. The updates implemented the backdoor. In the following weeks, Tan or others appealed to developers of Ubuntu, Red Hat, and Debian to merge the updates into their OSes. Eventually, according to security firm Tenable, one of the two updates made its way into Fedora Rawhide, Fedora 41, Debian testing, unstable and experimental distros versions 5.5.1alpha-0.1 to 5.6.1-1, openSUSE Tumbleweed and openSUSE MicroOS, and Kali Linux.

Additional reporting and research has found that Arch Linux, Alpine Linux, Gentoo, Slackware, PCLinuxOS, and others were also infected. However, this was all recent enough that most of the affected distros were still in their developmental / Unstable releases, so the malicious code didn't make it into many production systems. The macOS Homebrew package manager and the OpenWRT router firmware also included backdoored XZ Utils versions.

Some other reporting into the deeper backstory and various motivations wrote:

All of this started two years ago, in April 2022. The attacker contributed code to XZ Utils, gradually built his reputation and trust overtime, and used various sockpuppet accounts to pressure the XZ Utils project's author (Lasse Collin) to add them as one of the project's maintainers. At the time, Collin was burnt out and dealing with mental health issues and welcomed the help since they were a single developer on a highly popular project.

I want to share one more piece of writing about this, a wonderful narrative by Michael Zalewski. He's a Polish computer security expert, "white hat" hacker from Poland, former Google employee and currently the VP of Security Engineering at Snap Inc. Here's Michael's take on this most recent near-catastrophic misadventure. In his posting Saturday, titled "Techies vs spies: the xz backdoor debate - Diving into some of the dynamics and the interpretations of the brazen ploy to subvert the liblzma compression library", Michael wrote:

Well — we just witnessed one of the most daring infosec capers of my career.

Here's what we know so far: some time ago, an unknown party evidently noticed that liblzma (aka xz) — a relatively obscure open-source compression library — was a dependency of OpenSSH, a security-critical remote administration tool used to manage millions of servers around the world. This dependency existed not because of a deliberate design decision by the developers of OpenSSH, but because of a kludge added by some Linux distributions to integrate the tool with the operating system's newfangled orchestration service, systemd.

*Equipped with this knowledge about xz, the aforementioned party probably invented the persona of "Jia Tan" — **a developer with no prior online footprint** who materialized out of the blue in October 2021 and started making helpful contributions to the library. Up to that point, xz had a single maintainer — Lasse Collin — who was dealing with health issues and was falling behind. Shortly after the arrival of "Jia", several apparent sock puppet accounts showed up and started pressuring Lasse to pass the baton; it seems that he relented at some point in 2023.*

Since then, "Jia" diligently continued the maintenance work — culminating in February 2024 with the seamless introduction of a sophisticated, well-concealed backdoor tucked inside one of the build scripts. Full analysis of the payload is still pending, but it appears to have targeted the pre-authentication crypto functions of OpenSSH; it's probably safe to assume that it added "master key" functionality to let the attackers access all affected servers at will.

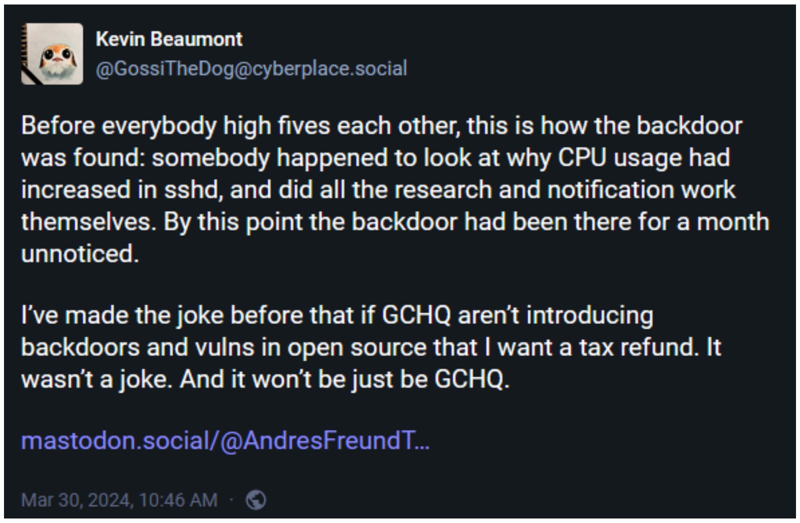
And I'll note that's exactly what was, indeed, being discovered over the weekend while Michael was writing this. The exploit provides for remote code execution with root privileges. Michael continues...

Some time after getting the backdoor in, "Jia" — along with a new cast of sock puppet accounts — started pinging Linux distro maintainers to have the backdoored library packaged and distributed to end users. The scheme worked until Andres Freund — a PostgreS-Q-L developer in the employ of Microsoft — reportedly decided to investigate some unexpected SSH latency caused by a minor bug in the backdoor code.

If this entire exploit timeline is correct, it's not the modus operandi of a hobbyist. In today's world, if you have the technical chops and the patience to pull this off, you can easily land a job that would set you for life without risking any prison time. It's true that we also have some brilliant folks with sociopathic tendencies and poor impulse control — but almost by definition, such "black hat" groups seek instant gratification and don't plan major heists years in advance. In other words, all signs point to this being a professional, for-pay operation — and it wouldn't be surprising if it was paid for by a state actor.

So, As we might imagine, this has really shaken the Linux open source community, because everyone understands just how close these malicious modifications came to being incorporated into all mainstream Linux distributions and filtering out into the world. Were it not for Andres Freund happening to wonder why SSH latencies were slightly higher than expected, how much farther might this have gone.

Kevin Beaumont posted from his cyberplace.social Mastodon account:


Kevin Beaumont
@GossiTheDog@cyberplace.social

Before everybody high fives each other, this is how the backdoor was found: somebody happened to look at why CPU usage had increased in sshd, and did all the research and notification work themselves. By this point the backdoor had been there for a month unnoticed.

I've made the joke before that if GCHQ aren't introducing backdoors and vulns in open source that I want a tax refund. It wasn't a joke. And it won't be just be GCHQ.

mastodon.social/@AndresFreundT...

Mar 30, 2024, 10:46 AM ·

So all of this begs the question, what will be next? And will someone catch that one too before it gets loose? Lately, we appear to be dodging more bullets which are coming more often. In fact, our recent episode #962 from February 20th was titled “The Internet Dodged a Bullet” when researchers stumbled over a way of bringing DNS to its knees. If I hadn’t used that title just six weeks ago I probably would have today. Because we did, again, just dodge another bullet.

Given the inherently precarious nature of security, and that we appear to be dodging more bullets recently, I won’t be surprised if one comes along that we don’t dodge in time. I wonder what lessons we’ll learn from that? Stay tuned!