SECURITY NOW!

Transcript of Episode #850

It's a Log4j Christmas

Description: There was no way that a massively widespread vulnerability in Java with a CVSS score of 10.0 would be wrapped up in a week. So this week we'll look at the further consequences of the Log4j vulnerabilities, including the two additional updates the Apache group have since released. But before that we'll look at what will hopefully be Chrome's final zero-day patch of the year, Firefox's surprise refusal to take its users to Microsoft.com, and Mozilla's decision to protect its users from Windows 10 cloud-based clipboard sharing. We have a new and interesting means of increasing the power of fraudulent cell tower Stingray attacks, and a continuing threat from cross-radio WiFi-to-Bluetooth leakage. We'll touch on a sci-fi reminder and a SpinRite update, then dig into what's happened since last week on the Log4j front.

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-850.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-850-lq.mp3

SHOW TEASE: It's time for Security Now!. Steve Gibson is here. We have yes, of course, more on Log4j, the worst security exploit of all time. At least that's what I'm thinking as Steve continues to describe how bad it is. Yet another, let's hope it's the last, zero-day for Chrome. Yes, they're going to fix it. And clipboard in the cloud? What could possible go wrong? It's all coming up next on Security Now!.

Leo Laporte: This is Security Now! with Steve Gibson, Episode 850, recorded Tuesday, December 21st, 2021: It's a Log4j Christmas.

It's time for Security Now!. Yes, indeed. We're going to soon rename this to The Log4Shell Show. Steve Gibson is here. He's the Security Officer in Chief. Hello, Steve.

Steve Gibson: Hello, Leo. That's an unofficial title, by the way.

Leo: I just gave it to you.

Steve: But I do respond to that, so that's fine.

Leo: Yes. Yes.

Steve: This is Episode 850. And it's like, wow.

Leo: Oh. That's remarkable.

Steve: That makes the math really easy because it means we've got 150 left, and then...

Leo: Why do you keep bringing that up? You're going to make me cry, dude. You're going to make me cry.

Steve: Well, that's going to be - that's three years, Leo. That really should do it. I think Log4j will finally be resolved. Oh, actually there are some opinions about that. We'll be talking about that, about the depths to which this thing has been submerged into our Java libraries. But the other nice number is 122121, which is today, which I guess means Christmas is not far off.

Leo: Not far away.

Steve: I should go get some wrapping paper. Anyway...

Leo: Yes, me, too. First get presents to put in the wrapping paper, then wrapping paper.

Steve: But yes. Today's title is "It's a Log4j Christmas." And actually the CISA has forbidden any IT people to open any presents before they have fully remediated their federal enterprises of Log4j, which is nice to say. I mean, you know, you can say it. But it's not happening. And we'll be talking about that because Google did some interesting analysis. But of course there was no way that a massively widespread vulnerability in Java carrying a CVSS score of 10.0 could possibly have been wrapped up last week. So this week we'll be looking at the further consequences of the Log4j vulnerabilities, including the two additional updates the Apache Group have since released as a consequence of more eyeballs looking at the code and going, oh.

Leo: Hey. You missed this.

Steve: What about this little problem over here?

Leo: God. Oh, my god.

Steve: But before that we're going to look at what will hopefully be Chrome's final zero-day patch of the year; Firefox's surprising refusal last week to take its users over to Microsoft.com - what?; and Mozilla's decision to protect its users from Windows 10 cloud-based clipboard sharing. Cloud-based clipboard sharing, you know, that just - you don't even have to wonder what could possibly go wrong with that.

Leo: Yeah, yeah, yeah, yeah, yeah.

Steve: Oh. We have a new and interesting means of increasing the power of fraudulent cell tower Stingray attacks, and a continuing threat from cross-radio WiFi-to-Bluetooth leakage within our handsets. I want to touch briefly on a sci-fi reminder about what's happening tomorrow. I'll be on the couch with popcorn. And I've got a SpinRite update, and that's actually - Lorrie and I will both be on the couch with sore shoulders because we have a 9:00 a.m. appointment in the morning.

Leo: Oh, good for you. Yay.

Steve: To boost ourselves.

Leo: Yeah.

Steve: For a while we were thinking, well, actually I was wanting to wait to see if Omicron would produce some change. And I was gratified to hear in the background one of the talking heads saying that, although Moderna is looking at an altered booster, one of the consequences of Omicron appears to be the booster fall-off is much more rapid than it was with Delta. It's providing on the order of only a few months of protection against Omicron as opposed to...

Leo: I'm going to start getting a monthly shot, I said. Whatever it takes; you know?

Steve: What a mess, yeah.

Leo: What a mess.

Steve: Anyway, so the sci-fi reminder, an update on some interesting stuff that has happened within my SpinRite world this past week, and then we're going to dig into everything that's happened in the previous week on the Log4j front.

Leo: Oh, so much.

Steve: So I think a final podcast, not forever, just of the year. We've got three more to go at least.

Leo: Actually I want to, while I'm doing this ad, point you to - you probably read it, the Project Zero analysis of how the Pegasus exploit got around Apple's Blaster. Have you read it yet?

Steve: Oh, I did not read that, no.

Leo: Oh, my god, you will get such a kick out of it because essentially the way it worked was, as you know, it's these rendering engines that are always the flaws.

Steve: Oh, it's an interpreter.

Leo: And the PDF rendering engine that they use for Apple's messages is an old open source Xpdf program. And it turns out it's got a buffer overflow, a little integer overflow. These things happen. And but the bad guys, I guess the NSO group, figured out that by pretending that a GIF file was a GIF file and actually making a PDF would trigger the PDF renderer, and then they could trigger the overflow. And then they discovered that you could in effect put four logic gates in a random location in memory: an AND gate, an OR gate - AND gate, OR gate, XOR, and there's one more, another OR.

Steve: NOR.

Leo: NOR, yes, I think it's an XNOR or something like that. But with those four they could make a Turing-complete system. They basically built an operating system and code rendering engine that gave them registers, it gave them comparators, gave them everything.

Steve: Oh, my lord.

Leo: You know, if it weren't so evil, it's really a work of genius. It's amazing.

Steve: Yeah. So you're saying that they located four locations where they could get that work done.

Leo: Exactly.

Steve: And then they knit those locations together.

Leo: Exactly. Exactly.

Steve: In order to create basically pseudocode out of those.

Leo: Yes. Yes. It's brilliant.

Steve: Wow.

Leo: It's a two-parter the Project Zero team put up at Google Project Zero to Blogspot.com. And the second part is not out yet. But I think you will - be good Christmas Eve reading. Maybe you and Lorrie can gather 'round the fire, and you can read it out loud. I think...

Steve: Well, and if there's any more surprises that haven't already been given away, maybe it's our first podcast of the new year. Sounds like it would be...

Leo: It's be a - I would love to hear you talk about this, yes.

Steve: This one nominally has a Christmas theme since it's about wrapping things. We have Uncle Sam in his flag-theme stovepipe hat. And he's apparently bringing to the table to be wrapped a box. On one edge it says "U.S. Gov't." And then proudly in the front it says "Control of End-to-End Encryption." And so that's got to get wrapped. And we have a corporate media schmo who in the talk bubble is asking Uncle Sam, "How would you like this wrapped?" And then against the back wall we see two rolls of wrapping paper, one titled "Anti-Terrorism" and the other "Protect the Kids."

Leo: There should be a third. "Think of the Women." I guess that's out of date now, but yeah.

Steve: Oh, yeah, that's right. How are we going to sell this sucker?

Okay. So early last week Google pushed an emergency Chrome update to resolve the 16th in-the-wild zero-day being actively used in attacks against its users. By this time, and even in this case I, which surprised me, it's like, oh, good, I don't have to ask for it, everyone using Chrome on Windows, Mac, or Linux desktops should be at 96.0.4664.110. That eliminates a high-severity, once again, use-after-free flaw, which really seems to be the, like, they're all over the place in Chrome. This is in Chrome's V8 JavaScript engine which someone had found and was using. And as usual, because it's Google, that's all we know about it, since Google has no motivation, no need or reason to share anything more. Like, why would they? What good's that going to do?

They wrote: "Google is aware of reports that an exploit for CVE-2021-4102 exists in the wild." And they thanked an anonymous security researcher for their report which allowed them to fix this. And since we're wrapping up the year in this week's podcast, we've got one more to hold our breath after this, we'll note that along with Chrome's extreme popularity comes a big bulls-eye painted on it since the bad guys get the most bang for their buck by attacking the market-leading web browser, and thus all of its many users.

Chrome had its first zero-day flaw patched at the start of February, then two in March, another two in April, one in May. June offered us a pair. July had only one. September kept the patchers very busy by closing five zero-days just here in this past September. October had one. November snuck past without any. And assuming that we're able to close out the year without any others, December will have brought us the 16th and final...

Leo: Geez. Good lord.

Steve: ...zero-day, I know, for 2021. And as I've noted before, I don't think this means anything about Chrome other than it's receiving more than its fair share of scrutiny. One of the best models I think this podcast has developed for security is this idea of porosity. Complex software security barriers are just - they're not secure. There's no other way to read the last 17 years of this podcast's evidence. You know, we haven't figured out how to do that yet. Or at least we haven't made it a priority, probably because it doesn't seem to really be such a big problem. You know? Other people are being attacked.

So, okay. But if our software security barriers are not absolutely secure, then what are they? You know? They resist penetration, though imperfectly. And this is why I'm enamored of the idea that our barriers are porous. The more pressure the bad guys place against a barrier, the more problems sneak through. And the closer we look at our software, the more problems we find. There was only one known problem with Log4j before researchers began staring at it closely for the first time ever, which revealed two additional, or three depending upon how you count, new problems. And we'll be talking about those developments at the end of the podcast. None of those was quite as bad as the first, but all needing fixing. One got a 9.0, which is pretty high up there. And the same thing, as we know, happened with Microsoft's ill-fated Exchange Server and Windows Printing throughout all of 2021.

And this is why, of course, there's money to be made from bug bounties. Companies will pay to learn of problems in their own software. Their own people should have found them, or never created those problems in the first place. But like I said, we don't really seem to care enough to prevent them. So there's revenue available for anyone who enjoys looking at code. And speaking of looking at code, there is absolutely, and I know you'll agree with this, Leo, no better way to improve one's own coding skill than by reading the work of other coders.

Leo: Oh, yeah. Absolutely.

Steve: Anyone who hasn't will be stunned by how much you can learn by looking at the way other people have solved problems. It's, I mean, you could spend half your life, if you were like really into coding, reading other people's code and saying, hey, I didn't know you could do that. That's really cool.

Leo: Yeah, yeah, yeah, yup, exactly right, yup.

Steve: Yeah, it just happens. And I think that's, for you and me, that's why we are so passionate about code is that it is so big. It is so open-ended. It is, you know, it can ask as much from you as you're willing to give it. So anyway, as for browser vulnerabilities, there's currently a lot to be annoyed with Microsoft about. But I just want to say again, because we talked about this over the last couple weeks, that that idea they had of backing off of Chromium's extreme optimization in order to reduce the browser's attack surface at the minimal cost of what is probably unneeded performance gain, I really think that's quite compelling. That's one that I hope Microsoft ends up proving and backports into the Chromium build and maybe gives to Chrome users and other users of the Chromium engine. I think that seems like a real win.

Okay, now, get a load of this one. I encountered it, too. Firefox had a bit of an adventure last week when it began refusing to connect to Microsoft.com. And I don't recall now what had me going to Microsoft.com. I don't go often. But I encountered the error myself, and it caused quite a stir online.

Now, back in the early days of this podcast we discussed the many problems surrounding web server certificate revocation. In fact, it got to be a hobbyhorse of mine. I created revoked.grc.com just to demonstrate how poorly browsers were handling revoked certificates by like putting one there and saying, look, your browser doesn't know it's been revoked. It's been revoked a long time. No one cares.

So what happens is certificates expired only - and in the old days, you could get five years, or three for some. Of course that keeps getting whittled down as a means of still trying to deal with this problem of stale certificates that might have gotten out of someone's control. So one of the solutions to the problem of erroneously trusting a revoked certificate was to have the web browser perform, that is, the user's web browser perform an on-the-fly query of the certificate issuer's OCSP server, where OCSP stands for Online Certificate Status Protocol. So in this way a real-time verification of the current validity of an identity certificate which had just been received from a remote web server could be verified by the browser client.

Now, the problem with this is that it takes time to do that. And back when OCSP was still young, OCSP servers were often overloaded or sometimes unresponsive. And the last thing a browser wanted to do was to slow down its users. There's no faster way to lose your user base than to be a slow browser. So OCSP was often set, kind of uselessly, to "fail open," meaning that only if an affirmative negative response came back would the site be blocked. Either an affirmative positive response or no response at all would let the user keep using that site. And the problem was that the bad guys who may have stolen someone else's valid certificate might arrange to block a negative OCSP response from getting back to its user, thus causing their browser to erroneously trust a fraudulent website.

Okay. So the concept of "stapling" was introduced, which I just think is brilliant. With stapling, rather than asking the user's web browser to go get an OCSP attestation, as they're referred to, the web server itself, the web server itself could periodically obtain a recently signed and timestamped assertion from its own certificate authority to accompany the certificate it was providing to browsers. And it would, and this was the term, "staple" that new fresh assurance to its own multiyear lifelong identity certificate. The stapled OCSP attestations would only be valid for a short time, perhaps just a few hours, so the web server would periodically reach out to renew this attestation that it was stapling to all of its outgoing certificates well before they would expire.

And the final bit of perfection comes from the web server's certificate having a flag known as "OCSP Must Staple" set. This completes and locks down the system by affirmatively telling any browser that receives this certificate that it is only to be considered valid, that is, the certificate itself is only to be considered valid if it also is accompanied by a valid, recently signed, stapled OCSP certificate. So that measure protects the certificate holder from the theft of their certificate. If it were found to be stolen, it could be immediately revoked by its original issuer, who would then refuse to issue any further valid OCSP certificates for stapling. And since by design all OCSP certificates expire quickly, the stolen certificate would quickly become useless because the certificate itself said this certificate must be accompanied by a valid stapled OCSP attestation.

So what happened? Last week, with this wonderful system having long been in place at Microsoft, and everything going along swimmingly, the now outdated SHA-1 signature was silently dropped from the Certificate ID fields present in Microsoft's OCSP certificates. And apparently no one else has done this. Microsoft led the charge. Since the newer SHA-256 signature hash was still present, Safari and Chrome had no complaint and didn't even notice the change, nor did their users. But somehow Firefox had never enabled their browser's support for SHA-256 hashes in OCSP stapled certs. The code was in there and ready to go, but someone forgot to turn it on.

So around the middle of last week, Firefox began refusing to allow any of its users to connect to most of Microsoft's various domains, including Microsoft.com itself. The server's certs were flagged, as I said, as "OCSP Must Staple," but from Firefox's vantage point the stapled certs appeared to be unsigned and thus invalid. So Firefox properly, kind of, refused to allow its users to go any further.

Fortunately, the problem was quickly remedied. As I said, it was a matter of turning it on and pushing out an update. Last Thursday's December 16th release of Firefox 95.0.1 fixed this immediately. Mozilla's bug number 1745600 is titled: "Fixed frequent MOZILLA_PKIX_ERROR_OCSP_RESPONSE_FOR_CERT_MISSING error messages when trying to connect to various Microsoft.com domains." So anyway, I just thought that was interesting. It popped up on my Firefox; and I thought, oh, what do you know? There's a problem with OCSP. I didn't dig any further. I didn't, like, look into turning it off or anything because, you know, I've got Chrome right next to it, so I just used Chrome to do whatever I had to do, and that worked, and I figured they'd sort things out, as they indeed did.

But anyway, just sort of interesting little misfire from, I mean, and this is - there is sort of a lesson here, right, which is it is certainly possible to tighten the screws down so much so that absolutely nothing will get past. But boy, if you do that, you really need to make sure that all of the links in the chain are working correctly. You know, I'm put in mind of the HSTS, the HTTP Strict Transport Security, where that's the header that you're able to add to your server that tells the browser, you've got a secure connection now. And here's an expiration on how long we want you to remember that you should only get secure connections from this website. And best practice is that you set that to infinity, essentially. But if you do that, you'd better absolutely know that you're going to be able to have TLS certificates for the end of time. Otherwise users are just not going to be able to get to your site. So, yeah, again, neat that we have these technologies that allow us to tighten things down so much. But you need to make sure they don't go wrong.

Okay. Since we've talked about Firefox and Microsoft, here as I mentioned at the top was another one of those "what could possibly go wrong" features in Windows, which to their credit Mozilla has dealt with. It's known that as, well, the feature is known as Windows Cloud Clipboard. It was added to Windows 10 way back in September of 2018, back with Win10's 1809 release. And it is what it sounds like, a feature that allows users to sync their local PC's clipboard histories to their Microsoft accounts.

Fortunately, what seems like an abuse-prone feature is disabled by default. Yeah, and I mean that seriously. You'll see. But when enabled, it allows Windows users to access the Cloud Clipboard by pressing the Windows Key+V shortcut. So instead of like, you know, CTRL+V for local paste, you do Windows+V for galactic cloud paste. And this grants users access to clipboard data from all their other devices. Again, as I said, what could possibly go wrong? And more than that, the service also maintains a clipboard history, allowing users to go through past items they copied or cut and then repaste the same data into new contexts, whatever that might be.

So what recently came to light, and what Firefox fixed in last month's Firefox 94, remember just last week was 95.0.1 to fix this Microsoft.com mess. So last month was Firefox 94. What happened was they realized that usernames and passwords copied from the browser's password section were being recorded on demand, as with everything else, into this shared Cloud Clipboard repository with history. So is that a bug, or is that a feature? Either way, Mozilla decided it was not good. And there's a workaround; right? If you really want that, then copy it to Notepad, and then copy it from Notepad to the cloud. But not directly. Mozilla says no. We don't want people to like, when you're just copying it for some reason from Mozilla, that should not be recorded in your permanent clipboard history shared by all your PCs.

So in a blog post last Wednesday, Mozilla said that with release 94 they've modified Firefox so that usernames and passwords copied from the browser's password section - you get to that with about:logins in the URL - will no longer be stored in the Windows Cloud Clipboard, but instead will be stored only locally, in a separate clipboard section.

Mozilla said it considered this behavior, that is, the normal behavior, dangerous - uh-huh - since any threat actor with access to a synced device could simply press the Windows+V keyboard combination and access any clipboard data from a user's past activity on other devices. Now, I suppose that as long as a Windows power user understands the inherent danger, this clipboard cloud sharing could come in handy. But Mozilla felt that this was especially dangerous in the case of the exportation of the browser's stored usernames and passwords since no trails or local logs of any kind are retained. So you wouldn't even see that this had happened. Mozilla also said that they had added even more protection to their Private Browsing windows so that nothing copied from a Firefox private window would be synced to the Windows Cloud Clipboard, not just an exclusion for user credentials but everything.

And for what it's worth, be warned, no Chromium-based browsers include this protection, and usernames and passwords from Chrome were synced to Microsoft cloud servers when this feature was tested. So users should be aware of who might access their passwords if they use the Cloud Clipboard, you know, if it's enabled, and are using any non-Firefox browser on Windows. But as I said, even that is still some weak protection. It's not that difficult to get around it, if you wanted to. So, wow. You know, the idea of synchronizing your clipboard to the cloud really has the feeling, Leo, like a feature in desperate search of need.

Leo: You know, Apple has a sort of a similar thing which is if you copy something onto the clipboard on a Mac, it will then go to your phone. But it doesn't go to the cloud. It just goes to your phone. Why put your clipboard in the cloud? That seems like a - it does. It's a selling point, it's not a...

Steve: Yeah, and not just the last thing you pasted, but like...

Leo: The whole thing.

Steve: ...your history.

Leo: Yeah. That's, you know, that's passwords. That's everything.

Steve: Yeah.

Leo: That's not a good idea.

Steve: Yeah.

Leo: Obviously. Just the name of it.

Steve: So turns out there is, not a huge weakness, but sort of an interesting problem which has existed in all cellular networks since 2G. A paper was just published under the title "Don't Hand It Over: Vulnerabilities in the Handover Procedure of Cellular Communications."

Now, "handover" is the official name for what we normally call cellular inter-tower handoff. It's the process by which, as we know, a phone call or a data session gets transferred from one cell site base station to another without losing any connectivity during the transmission. And of course it was the key innovation of the whole cellular radio system, and a brilliant invention. And of course it's crucial to establishing and maintaining cellular communications while the user's on the move so that you're driving along, and everything seems fine, yet your connection is being handed off from one cell tower along the side of the freeway to the next as you drive.

Okay. So here's how this works in a little more detail which pertains to what has been the exploit that's been worked out. The user's equipment, typically a handset, sends its received signal strength, that is, the signal strength it's receiving from multiple nearby towers to the network so that the network can determine whether a handoff is necessary and, if so, that network facilitates a switch between base stations when a more suitable target station is available or is soon to be available. Although these signal readings are cryptographically protected, it turns out that the content of these reports is not verified. And this lack of verification allows an attacker to spoof these reports and to thereby force a device to move to a cell site of their choosing, typically operated by an attacker.

So this attack arises because, as I said, these unverifiable signal strength reports are undetectable. They're part of the system that was just, you know, in this long-established protocol this aspect was never properly designed to be spoof proof and has indeed been proven now not to be. It doesn't create a huge new problem, but it does mean that the effectiveness of fake cell stations, often called Stingrays, can now be increased. It used to be that a Stingray needed to be located in close proximity to its target so that it would have the stronger signal, and that the normal signal strength juggling process would cause the targeted user to switch their traffic over to the Stingray.

But this new research dramatically reduces this requirement. So now any user within range can have their traffic rerouted through a man in the middle, which is what these Stingrays are typically used for. It can be used as a denial of service if there's some benefit for, like, disconnecting someone from the cell system. You can essentially commandeer their phone by switching it to a Stingray that doesn't proxy for the regular network as is normally done.

In the researchers' experimental setup, they found that all devices, they tested a OnePlus 6, an Apple iPhone 5 - that's an oldie - a Samsung S10 5G, a Huawei Pro P40 5G, and others. Every one of them was susceptible to both a denial of service, meaning just cutting you off from the network, and man-in-the-middle attacks. They presented their findings at the Annual Computer Security Applications Conference (ACSAC) earlier this month.

So again, it's not like any new big problem crack in cell coverage. But literally everything from 2G all the way up through 5, there is no - and of course this is never going to change now; right? You can't change this protocol. It's too late. All of the infrastructure is deployed. All of the devices exist. All you can do is know about it. And maybe, certainly to the degree that now we're, in the case of data, we're encrypting our traffic before it goes over a cell connection, thanks to HTTPS, there's not much that a bad guy could do.

We already have, we already know that HTTPS itself is providing strong protection against any man-in-the-middle attacks. You know, a denial of service would be a vulnerability that would be effective, although potentially less devastating. But it might, you know, be used cleverly to get someone to move from where they are, trying to, you know, like can you hear me now? Can you hear me now? Who knows? But anyway, the technology exists. It's public. And so given the way things are these days, we could expect it to be incorporated into next-gen Stingray systems before long.

Another thing that's not going to change, unfortunately, is protection against this next problem, which is cross-WiFi-to-Bluetooth leakage. A team of German and Italian researchers have significantly updated their previous research. A couple years ago they first, like, raised the alarm, actually it was at a Black Hat conference several summers ago, and no one really took that much notice of it. The problem being, or the reason being probably is that it is not easy to fix. So they updated their research regarding a class of what they call "coexistence attacks" against WiFi and Bluetooth. Their research paper is titled "Attacks on Wireless Coexistence: Exploiting Cross-Technology Performance Features for Inter-Chip Privilege Escalation."

And the abstract of their paper explains what they've done. They wrote: "Modern mobile devices feature multiple wireless technologies such as Bluetooth, WiFi, and LTE." You know, cellular. They said: "Each of them is implemented with a separate wireless chip, sometimes packaged as combo chips. However, these chips share components and resources such as the same antenna or wireless spectrum. Wireless coexistence interfaces enable them to schedule packets without collisions despite sharing these resources, which is essential to maximizing networking performance.

"Today's hardwired coexistent interfaces hinder clear security boundaries and separation between chips and chip components." They said: "This paper shows practical coexistence attacks on Broadcom, Cypress, and Silicon Labs chips deployed in billions of devices. For example, we demonstrate," they said, "that a Bluetooth chip can directly extract network passwords" - meaning WiFi - "and manipulate traffic on a WiFi chip. Coexistence attacks enable a novel type of lateral privilege escalation across chip boundaries. We responsibly disclosed the vulnerabilities to the vendors, yet only partial fixes were released for existing hardware since wireless chips would need to be redesigned from the ground up to prevent the presented attacks on coexistence." In other words, don't hold your breath.

Okay. So to put some additional context to this, I want to share the top of their paper's introduction, which I think everyone will find interesting and, unfortunately, somewhat worrisome. They explain: "Wireless communication is enabled by Systems on a Chip (SoC), implementing technologies such as WiFi, Bluetooth, LTE, and 5G. While SoCs are constantly optimized towards energy efficiency, high throughput, and low-latency communication, their security has not always been prioritized. New exploits are published continuously. Firmware patching to mitigate flaws requires strong collaboration between vendors and manufacturers, leading to asynchronous, incomplete, and slow patch cycles. In addition, firmware patch diffing can provide attackers with Systems on a Chip vulnerabilities months before their public disclosure." Not to speak of their actual installation into their users' phones, if that ever happens.

They said: "Mobile device vendors account for potentially insecure wireless SoCs by isolating them from the mobile Operating System and hardening the OS against escalation strategies. For example, on Android, the Bluetooth daemon residing on top of OS drivers runs with limited privileges, is sandboxed, and is currently being reimplemented in a memory-safe language. As a result, recent wireless exploit chains targeting the mobile OS instead of the System on a Chip are rather complex and need to find a bypass for each mitigation."

They said: "We provide empirical evidence that coexistence, i.e., the coordination of cross-technology wireless transmissions, is an unexplored attack surface. Instead of escalating directly into the mobile OS, wireless chips can escalate their privileges into other wireless chips by exploiting the same mechanism they use to arbitrate their access to the resources they share, for example, the transmitting antenna and the wireless medium. This new model of wireless system exploitation is comparable to well-known threats that occur when multiple threads or users are able to share resources like processors or memory.

"This paper demonstrates lateral privilege escalations from a Bluetooth chip to code execution on a WiFi chip. The WiFi chip encrypts network traffic and holds the current WiFi credentials, thereby providing the attacker with further information. Moreover, an attacker can execute code on a WiFi chip even if it is not connected to a wireless network. In the opposite direction, we observe Bluetooth packet types from a WiFi chip. This allows determining keystroke timings on Bluetooth keyboards, which can allow reconstructing texts entered on the keyboard." And we've talked about keyboard timing attacks years ago, and they're real. And surprisingly effective when you put enough AI behind the timing.

And they finish: "Since wireless chips communicate directly through hardwired coexistence interfaces, the OS drivers cannot filter any events to prevent this novel attack. Despite reporting the first security issues on these interfaces more than two years ago, the inter-chip interfaces remain vulnerable to most of our attacks." They said: "For instance, Bluetooth/WiFi code execution is still possible on iOS 14.7 and Android 11. Wireless coexistence is indispensable for high-performance wireless transmissions on any modern device. We experimentally confirm that these interfaces exist and are vulnerable today." So maybe in 2022 we'll be talking a little bit about the mischief that has been leveraged, taking advantage of inter-radio leakage.

Leo: Unfortunately, I think the mischief will be kind of mysterious because it's going to be things like those DC Stingrays that are going to get better, and nobody's going to know anything about it; right?

Steve: Yup.

Leo: Right? It's going to be kind of transparent.

Steve: Yup, yup. Okay, so I said a brief sci-fi reminder. And that's for "The Matrix Resurrections." Yeah.

Leo: I hope it's good. Because 1 was great. 2 and 3 weren't so great. You know what I'm saying?

Steve: I'm in complete agreement. I'm in complete agreement. It starts airing at like 3:00 a.m. or something in the morning tomorrow. It's co-releasing, because it's Warner Bros. it's co-releasing in the theaters and on HBO Max. And, you know, despite the fact that IMDB currently pegs it at a 6.8, which falls beneath my normal IMDB threshold of 7.0, I find that that's sort of about the level of pain, I'm quite certain that Lorrie and I, having received as I mentioned our vaccine boosters at 9:00 a.m. tomorrow morning, will be snuggled up on our couch.

Leo: You won't be able to move. So you'll be forced to watch "The Matrix."

Steve: We won't be able to move. Yeah, and maybe we ought to be shot in different shoulders.

Leo: Kind of like taking the red pill; you know?

Steve: That way we could put the popcorn in between us.

Leo: Yeah, yeah, one hand can move, the other one will just lie limp.

Steve: And Leo, I'm not expecting much, and I don't expect to be wowed or surprised. Movies don't seem to be that much anymore.

Leo: No.

Steve: You know? There are a lot of special effects.

Leo: I liked "Dune." You liked "Dune"; right? That was good.

Steve: That's true. That is, but - yeah. But it wasn't like <gasping>. When have we had that?

Leo: No, it's been a while. It's mostly long-form TV shows really these days are the things we get excited about.

Steve: Yeah.

Leo: I don't know. I'll be interested.

Steve: Oh, I can't wait. I just lost two and a half hours. Goodbye.

Leo: Yeah. Yeah. "Midnight Alley" - is that what it's called? - looks good, too. Anyway, there are some good movies coming out this week, thank goodness.

Steve: And are they available without going to a theater?

Leo: Yes, yes, yes. Because I ain't going to no theaters.

Steve: Unh-unh. No, unh-unh. No. Okay. So as planned, last week I brought up GRC's own instance of GitLab. It's a big, lumbering Nginx and Ruby on Rails conglomeration that also drags in all manner of other accoutrements. And watching go in, it's really so clear why security has become such a problem for our industry. I mean, you push a few buttons, and then you stand back while screen after screen scrolls by as module after module, I mean, hundreds and hundreds of the little buggers download, install themselves, and set up shop. You know? So just stand back and hold your breath.

What are they all doing? No one knows. This one needs that one, and that one needs another one, and before you know it 150 million bytes of your mass storage has been commandeered by the forces of darkness. Processes are now running that are presumably all supposed to be running. You know? And what choice do you have? It's free. Do you want it or not? And what are you going to do? Write it yourself? And actually there was a comment over on GRC's spinrite.dev newsgroup urging me to please not spin off and write a bug tracking system in assembly language. Not to worry, that won't be happening ever.

So I put this thing, because I don't trust it, you know, trust has to be earned. I put this thing on its own physical FreeBSD server and sequestered its network behind a physical firewall with its own dedicated port. The only thing it has any access to is the outside world. It can see out, but it can't look around. And all that said, GitLab really is a very nice piece of work. I don't feel like I know it fully yet, or really trust it. But now that we have it, I think it makes sense to use it.

As I mentioned before our threaded conversation textual-only newsgroups are really perfect for what they are. They're fast and simple for casual use. And they allow us to move with great speed when we're all moving forward as a group. And they've served us well until now. But they're not ideal when we get stuck on a problem. We already have the web forums, but those are really targeted at our future SpinRite end users. That's where I want, you know, that'll be the official GRC support side. And GitLab is clearly not aimed at the uninitiated end user. You know, there's a lot of "Where's the button I'm supposed to push?" certainly at the start.

But I, GRC, and SpinRite have been blessed with a bunch of really amazing and terrific testers who have large arrays of hardware at their disposal and who have shown an enduring interest and willingness in actively participating in SpinRite's development testing. So I think that this GitLab instance will be there for them to use if they choose to use it. Many testers are understandably less involved in this entire process, and that's fine. We need them, too. They tend to swing by every week or two to give the latest release a try. And they generally just post that it worked again, as usual, as they all do. And I don't want to mess with that, or giving them any kind of a burden. So we'll still have the newsgroups for quick, just drive-by posting of go/no-go notes of successive SpinRite test releases. But I think that GRC's GitLab instance will be a very valuable resource for those who are interested in working with me when sticky problems arise.

Oh, and I did want to mention one thing, just because I hadn't, and it changed my life a few months ago. Last week I posted a shout-out to the free service whose performance has stunned me. What I tweeted was "A shout-out to @StopForumSpam." I said: "GRC's forums were drowning in forum spam because forum spammers are people, typically in the Eastern bloc, who create temporary throwaway Gmail accounts and manually bypass all CAPTCHA challenges. But after adding StopForumSpam, not a single fake registration." So it really works.

I looked high and low for some solution, and turns out it was built into XenForo. It already had support for it, but for some reason I hadn't turned it on in the beginning. So the moment I did, this problem disappeared.

Leo: Do you get a lot of false positives, though, like people who aren't spammers?

Steve: One so far.

Leo: Okay.

Steve: One in six months.

Leo: That's pretty good, yeah.

Steve: I'm very impressed by that. And when it happens, they use the contact form to say, "Hey, what's up?"

Leo: Hey, what happened?

Steve: And I say, "Sorry about that," and I approve them. So what I'm hoping is that I can find this for GitLab because I just don't want - it just bugs me, you know, sort of just from a "get off my lawn" standpoint. I just don't want a bunch of crap in my forums or burdening my GitLab instance. That's just - it's annoying. And eventually they're going to be posting crap, and they ultimately get around to doing that, which is annoying, too.

So it turns out that there's an old project on GitHub from seven years ago. StopForumSpam has a very simple web API. So it's simple for some code to reach out and query the IP, the email address, and the username to see if it's on the blacklist. And boy, does it work. So maybe there'll be some way, an easy way to get this added to GitLab. That would be my dream. But for what it's worth, forum spam, I did some looking around when I was trying to find a good solution before it occurred to me, oh, just turn it on. It's a real, it's a big problem for a lot of people.

Leo: Yeah. I just manually approve everybody.

Steve: Yes.

Leo: And you can usually tell a spammer. It's funny, they don't try to - I shouldn't say this out loud - get into our forums very often.

Steve: Good.

Leo: But they try to get in our Mastodon instance. Usually about nine to one requests to join are from spammers.

Steve: Yes. I know. In fact, when I whittled them down, we lost three quarters of what looked like people who, like, members on GRC's forums. It was crazy.

Leo: It's bad, yeah.

Steve: It was just crap.

Leo: Our forums mustn't yet be on the list of attackable forums. I won't tell anybody.

Steve: Yeah.

Leo: Don't mention it again. What forum? We don't have a forum, no.

Steve: So as for SpinRite, I think a pattern is beginning to emerge. I mentioned last week that I had purchased some stuff on eBay. All of that new - actually three different motherboards have come in. What I'm seeing is that all of my new SpinRite code generally works perfectly on all newer hardware which is behaving correctly. Pretty much anything I do just works right off the bat there. Which is why the vast majority of people who are in the spinrite.dev newsgroup just say, yeah, you know, I've never had a problem. I don't know what's up with you other guys, but everything Steve does just works.

But it's on older hardware, typically very old hardware, where what I think early and probably slightly incompatible operation was originally being hidden behind those older systems' BIOSes. But now that SpinRite is bypassing the BIOS, we're encountering a few gotchas which is what I'm now in the process of addressing.

I got one, it was an ASUS M4A78 motherboard or something, like from - the last BIOS was 2010, so 11 years old. And when I got it and ran SpinRite, the benchmark was like stuttering. And I immediately noticed that the progress bar wasn't moving smoothly for 7.25 seconds as it's supposed to. Well, it's being moved by the clock tick, which occurs at 18.2 Hz. And that's the highest priority interrupt in any PC is that timer because you don't want to lose track of time. And it's very brief. It just increments a counter and then returns control.

My point is that progress bar can't hesitate. I mean, it can't. So something is very wrong somewhere if it's only advancing as the drive's transfer completes, which is on an old, I had a 300GB Maxtor drive, IDE, and it took a substantial amount of time to do one of these big block transfers, a 16MB transfer. And what I was noticing was the progress bar was jumping ahead by a big chunk every time a transfer finished, as if somehow like the clock interrupt was not happening. It's like, what? So anyway, that's my project for tonight is to roll - I'm going to rub my hands together.

Oh, I forgot to mention that when I - so it was doing that but not crashing, where it was crashing on the two other systems that the two people who have these were trying, were testing SpinRite on. But it also had an old BIOS. When I updated to the latest BIOS, now it crashes. So, like, what? Anyway, so why should - I'm not using the BIOS. So why should changing the BIOS have any effect on the hardware? I don't know. Anyway, these are the problems I live for, and I've got three motherboards that are causing these problems at the moment. They'll probably be fixed by the time everyone hears from me again in two weeks.

Leo: Oh, my god.

Steve: Because we have vacation next week. Anyway.

Leo: We're ready to hear about more troubles with Log4j.

Steve: Well, so we all knew that nothing as ubiquitous and pernicious as Log4j was going to be wrapped up in a week. And indeed there's a lot more to talk about this week. And I guess if last week's podcast served to introduce the problem, this week we're going to gain some perspective on its consequences.

As we know, Log4j earned the, I don't know if we've ever seen I before, CVSS score of 10.0.

Leo: You don't get much worse than this, I think it's fair to say.

Steve: And Leo, I'm put in mind of that old joke about the volume controls of massive stereo amplifiers which, because they have so much power, need to have their volume controls range from 0 to 11. If there was ever a case for a CVSS score of 11 out of 10...

Leo: Goes to 11.

Steve: Yeah, Log4j is as good a case as any. Okay. So since last week, much has happened. On the remediation front, while the audio was still reverberating from last week's podcast recording, the Apache Software Foundation published an updated fix for the Log4j vulnerability after it was discovered that the previous patch, the one from the previous Thursday was incomplete in certain non-default configurations. So it was technically the second but related vulnerability, so it was given its own CVE, 2021-45046.

And because it didn't induce nearly as much hyperventilation among security experts, it was initially rated at a CVSS of 3.7, which, you know, you don't even get out bed for that, because it was initially believed to only have DoS, you know, as in you could crash something, denial of service consequences. But apparently the bad guys couldn't do any worse than that. But that unexciting score was soon amended to a brightly glowing 9.0 once it became clear that an attacker could send specially crafted strings which could lead to information leakage and remote code execution in some environments, and local code execution in all environments. Whoopsie. So that was important.

So the second discovery affected all versions of Log4j, all the way from 2.0-beta9, which is where the whole Log4j 2 began, up through 2.12.1 and 2.13 through 2.15.0. And remember that 2.15 was the big fix that the Log4j project maintainers had just shipped the week before. That was obsolete now, so we had 2.16. And this JNDI Lookup was essentially disabled by default as a consequence. After all the headache with 2.15 and saying, well, you know, we're going to filter this, and we're going to try to eliminate that, they decided no. JNDI, this is just too big a mess. Nobody really needs it in loggings. And if you do, okay, then as we mentioned last week, you could turn it on.

Anyway, it is much more firmly disabled now in 2.16. And that's for Java 8. If for some reason you're on Java 7, then look for 2.12.2 as soon as it becomes available. That's the one that you'll need for Java versions 7, you know, Java major version 7.

The Apache Software Foundation's Ralph Goers said: "Dealing with CVE-2021-44228" - that was the original one -

"has shown that JNDI has significant" - it's hard to even say that with a straight face - "significant security issues." Yeah, really? "While we have mitigated what we are aware of, it would be safer for users to completely disable it by default, especially since the large majority are unlikely to be using it." But until now, of course, even though it's always been true that the large majority are unlikely to be using it, we had it turned on. Okay? Lesson learned.

And since last week the line-up of well-known affected organizations has, of course, only grown. Here we are on Tuesday. Last week the list in alphabetical order of the biggies was "Akamai, Amazon, Apache, Apereo, Atlassian, Broadcom, Cisco, Cloudera, ConnectWise, Debian, Docker, Fortinet, Google, IBM, Intel, Juniper Networks, Microsoft, Okta, Oracle, Red Hat, SolarWinds, SonicWall, Splunk, Ubuntu, VMware, Zscaler, and Zoho."

Leo: Holy moly.

Steve: So A through Z, baby. And we'll be looking at why in a minute here. Also as of last week, 10 different attacking groups have been identified, and roughly 44% of corporate networks globally have been under attack. The U.S. CISA, the awkwardly named Cybersecurity and Infrastructure Security Agency (CISA) has given all federal agencies the deadline of December 24th, that's Friday, this Friday, to fully incorporate all patches for the vulnerability. And I'm going to explain why that can't possibly happen. No one gets to have Christmas until every agency's Log4j vulnerabilities have been fully remediated. The only way to do that is to pull the plug. Because the fixes are not available, and Google is estimating we've got years before that's going to happen, and I'll explain why in a minute.

Sophos's senior threat researcher Sean Gallagher said that: "Adversaries are likely grabbing as much access to whatever they can get right now with the view to monetize and/or capitalize on it later on. There's a lull before the storm," he's saying, "in terms of more nefarious activity from the Log4Shell or Log4j vulnerability. The most immediate priority for defenders is to reduce exposure by patching and mitigating all corners of their infrastructure and investigate exposed and potentially compromised systems. This vulnerability can be everywhere," he said.

Okay. So that took us from 2.15 to 2.16. But wait, there's more. Later in the week researchers discovered an entirely new Log4j attack vector. And we knew that was going to happen. We talked about it; right? I mean, there are other ways to get there, which enables adversaries to exploit servers locally by using a Java WebSocket connection.

Matthew Warner, the Chief Technology Officer for Blumira said, and get a load of this one: "This newly discovered attack vector means that anyone with a vulnerable Log4j version on their machine or local private network can browse a website and have this vulnerability triggered." Wow.

Leo: Now, of course you have to be logging, using Log4j to log that traffic. I mean, it won't just - it can't just be sitting there not running and be affected. It has to be running; right?

Steve: Uh, well, if Log4j.jar is in your process tree somewhere...

Leo: Yeah. Running.

Steve: And your browser can be the way to get to it.

Leo: Trigger it, yeah, yeah.

Steve: Yes, right. So it won't launch it.

Leo: Most home users aren't running Log4j.

Steve: No, no, no. I mean, it can be any of the, let's see, Google counted it as something like 34,000 different Java things. So any of those being used brought Log4j along with it.

Leo: Right, right, right.

Steve: Yes, yes.

Leo: But I think mostly those are going to be associated with some, I mean, you don't log, well, maybe you do. I don't know. I mean, there's constant logs.

Steve: Oh, Leo, have you ever - have you looked like at some of your directories?

Leo: Yeah, yeah, I know I'm logging. But I'm not using - I don't think Apple or Microsoft uses Log4j to log the console log. Maybe, I mean...

Steve: Oh, no, no. It's the...

Leo: It's mostly the servers.

Steve: I don't know. I mean...

Leo: I hope.

Steve: We'll have plenty to talk about next year.

Leo: I mean, that's the real question, yeah, I mean, is how much should people who are not knowingly running servers worry about this. Might they have Log4j? I mean, I run Minecraft Server, so I know I probably have Log4j running.

Steve: You think?

Leo: But if you're running - but I know because I'm a server. Most people are not running servers out of their house.

Steve: Right.

Leo: I'm hoping that most people would not be exposed to this. But I might be wrong. I'm just curious. I might be wrong.

Steve: So it's not only for servers. Servers were the initial vulnerability because they listen for bad guys on the outside. I've got some, I don't know if it's Java-based. I think it is. It's a RAID. Intel's RAID monitoring package is Java-based, so it could be cross-platform. And so they brought it along as part of the easy way of implementing things these days.

Leo: And it's normal for software to run logs because...

Steve: Yes. In fact...

Leo: To keep track of errors and things like that. Every operating system is.

Steve: Yes. And I was going to say, if you've ever like dug deep into some, like, directories under your own user tree in Windows, there's, like, logs. It's like, what is all this crap? And it's like, you know, but it's doing stuff.

Leo: Oh, yeah. Occasionally you use them, you know.

Steve: Yeah, and it's on - and I'm on a workstation, yet it's still happening.

Leo: Oh, yeah, me, too. Now, there's a request in the chat room, and I agree with it. You need to write NeverLog4j. You need to do that right now. Forget it. Just go out, write it, NeverLog4j. Put it up on the site. You'll have a million downloads because there is really, right now, to my knowledge, there's no way to say, I mean, you could attempt to trigger it, I guess. But there's no way to say, oh, I'm not running it; you know?

Steve: Yeah, that's a very good point. It would be worth someone - I'm going to be debugging SpinRite tonight.

Leo: No, I know, you're busy. I know. You've got to watch a movie about Neo. I understand. You've got priorities. You don't work for us 24/7. But, I mean, in theory, I mean, you could - your IoT devices may be running it. I mean, many of these IoT devices could be running it.

Steve: Yes.

Leo: I wouldn't be surprised.

Steve: Well, where are they going to put their log? But you're right, they might be rolling their log daily.

Leo: Right.

Steve: And having it available in case something crashes.

Leo: In case you have a problem. Then you can...

Steve: Yup.

Leo: How often do you see that in the preferences? Well, you know, here, you can upload the log to the support.

Steve: Yes. Yes, yes. You know? Who knows what's in our routers?

Leo: Exactly.

Steve: Yes. Anyway, so Matthew Warner said: "At this early stage there is no proof of active exploitation." Meaning that your browser can be the entry point. He said: "But this vector significantly expands the attack surface and can impact services even running as localhost which were not exposed to any network." And so that's what makes this latest local attack vector such a concern. So although this latest discovery can be resolved by updating all local development and Internet-facing environments, last Friday Apache rolled out yet another version of Log4j, 2.17.0. So we now have the original CVE 44228 with its CVSS score of 10.0, fixed by version 2.15. Then CVE-2021-45046 with its score upgraded from 3.7 to 9.0, and that was fixed by 2.16.0. And finally, CVE-2021-45105 with its CVSS of 7.5, because it's local only, but still can get you, which is fixed by 2.17.0.

Leo: By the way, my router is running Log4j. I forgot. Because I'm using the Ubiquiti router.

Steve: Oh, that's right. Ubiquiti's on the list.

Leo: UniFi uses it, yup.

Steve: Yup.

Leo: That's nice.

Steve: Yeah, there you go. What could possibly... Anyway, Jake Williams, the CTO and co-founder of the incident response firm BreachQuest, said: "We shouldn't" - as in should not - "be surprised that additional vulnerabilities were discovered in Log4j given the additional specific focus on the library. Similar to Log4j, this summer the original" - and this is Jake saying - "PrintNightmare vulnerability disclosure led to the discovery of multiple additional distinct vulnerabilities. The discovery of additional vulnerabilities in Log4j should not cause concern about the security of Log4j itself. If anything, Log4j is more secure because of the additional attention paid by researchers."

And I would just say turn off that variable interpretation and expansion. That's where all of the problem came from. Not from logging, but from the fact that it was like ridiculously open-ended and over-powered so that you could cause variable expansion to go do a DNS or an LDAP query or download Java and run it. I mean, that's just nuts.

Okay. So not too surprisingly, additional problems were found with the original vulnerability; right? And the initial fixes needed fixing. So that's happened. So now anybody who thought, okay, good, I installed 2.15, well, no. Now you need 2.17. So that would probably be a good idea.

So, what have we learned about the extent of Log4j's usage which need to be found and fixed? And this is really interesting. Okay. So one source of appraisal is Google, who had their open-source team scan Maven Central. Maven Central is today's largest Java package repository. What they found was that 35,863 individual Java packages were either directly or indirectly using vulnerable versions of Apache's Log4j library. Their scan looked for packages that used Log4j versions which were vulnerable to both the original Log4Shell exploit we first talked about last week, and also the second remote code execution bug discovered in the Log4Shell patch.

Members of the Google Open Source Insights Team said when a major Java security flaw is found - major; right? I mean, past major Java security flaws generally affect around 2% of the Maven Central index. However, thanks to the prevalence of the use of Log4j, the 35,863 Java packages vulnerable to Log4Shell account for roughly 8% of Maven Central's total of around 440,000 Java packages. This was an impact that the team members characterized as "enormous."

And despite the effort and focus, fixing everything is a monumental task. Google published their report at the end of last week on the 17th where, at that point, 4,620 out of the total 35,863 packages had been updated. But we've seen that these things tend to taper off exponentially, right, with the most work immediately being done and with the pace inevitably slowing with each passing day and week and month. Because of this, the Google guys said they didn't expect the Log4Shell issue to be patched in full, at least for years.

One contributing reason is because Log4j is, more often than not, an indirect dependency. Java libraries are built by writing some code which uses functions available in other Java libraries, which they import. And those are built by writing some code which uses functions from other Java libraries and so on. That's really how it happens. So the problem is more often not that a Java library directly uses Log4j; 17% do. It's that the other 83% instead use another Java library which might use Log4j, or might use another library that does, and so on. And as a result of this multiple depth inheritance, maintainers of vulnerable Java packages are forced to wait until other developers and maintainers of the packages they depend upon have updated their own libraries.

To put some numbers to this, Google explained - and Leo, I have a really cool histogram on the next page, page 11 on the show notes. Google explained that Log4j is only a direct dependency in approximately 7,000 packages of the more than 35,000 vulnerable libraries. So many Java developers either need to wait for the packages they use to be updated or search around for and switch to alternative solutions that are not vulnerable.

When Google was asked why fixing the JVM ecosystem was so hard, their open-source team explained that by far most JAVA modules depend upon Log4j indirectly. And the deeper the vulnerability is down the dependency chain, the more steps are required for it to be fixed since the bottommost module, where the direct dependency exists, must be fixed first. Then the module that depends upon that module needs to be rebuilt using its dependency which is now fixed. And then the module above that one needs to be rebuilt, and so on.

So Google's report used a histogram to elegantly summarize the dependency chain depths they encountered when they were exploring each of the approximately 440,000 dependency chains in search of any that use vulnerable Log4j instances. And I've got, as I said, I've got that chart in the show notes. You'd expect to see more or less decreasing incidence statistically as the depth increases. And overall that's what we do see. But there's a weird jump upward at a dependency depth of 5. The histogram shows that about 14.75 percent of all dependencies occur at a depth of 4.

Now, that's - I should also mention that it looks like 17%, as I mentioned before, have a depth of 1, meaning that the package directly uses Log4j. But then looks like around 21% are using a package where that package uses Log4j. So a depth of 2 has 21%. And so of course you have to sum those to get the total packages with 1 or 2. Then a depth of three drops to around, looks like 12%. A depth of 4 for some reason, as I said, goes up to 14.75. But then for some reason that almost doubles at a depth of 5 to 26.5% of all dependencies occurring at the 5, which is more than one in four of all the vulnerable packages.

Leo: I think there must be one package a lot of people use that depends upon, depends upon, depends upon, depends upon Log4j.

Steve: Right, right.

Leo: And so it's got to be - there's got to be one outlier package that has Log4j deep, buried deep within its dependencies that everybody uses. You know, and it might be Apache. It might be, you know, something just widely used. That's got to be it; right?

Steve: Yeah, it's got to...

Leo: It's so weird.

Steve: I think that because, I mean, it just is like a complete statistical outlier because then, as if to make up for that big almost doubling from between 4 and 5...

Leo: It just goes away.

Steve: ...then it crashes; right. Then it drops to 6% of them at a depth of 6 and looks maybe like a half a percent at a depth of 7 or 8. So but what this means is that, if the root Log4j dependency is 5 dependency layers down, then all five - then that package must be fixed. And the packages that depend upon it must be fixed. Then the packages that depend on it must be fixed, and fixed again, and fixed again. And it has to be done in sequence. Right? Because you have to be able to import the packages you're dependent upon, rebuild your library using the updated dependencies. Only then can the person maintaining the library that's dependent upon yours update theirs. And if at any point in that chain somebody's on vacation or got tired of that project and, like, eh, you know, then everything comes to a grinding halt. Nothing beyond that can get updated.

So, I mean, it's really bad. I mean, it's kind of a messed up system from that standpoint. And you have to say, wow, it's amazing it does as well as it does. And of course it reminds of that crazy stack of blocks where there was one little twig holding the whole rickety thing upright, and everyone's hoping that that tower of blocks is not in California where we tend to have earthquakes.

Leo: Yeah. Naive question. I think I know the answer. Why don't I just search for all occurrences of Log4j.jar and delete it?

Steve: Oh. Because it could be a binary inclusion. It could be linked into...

Leo: Ah. So it wouldn't necessarily be a Log4j.jar file.

Steve: Correct. It could be...

Leo: It could be linked in. You might not see it.

Steve: Yes, yes, the Log4j functionality built into a package.

Leo: Not to mention you'd probably break a bunch of stuff if you just delete it.

Steve: Well, exactly. Exactly.

Leo: And some software will go, oh, somebody deleted my Log4j. Let me just download that again. Wow. And I imagine in the manifest, if you had a good package manager - oh, I don't know. It's just a mess.

Steve: There are some that allow you to specify ranges.

Leo: You should say nothing less than whatever it is, 2.15.

Steve: But it may well have been that someone said 2.12 to 2.15 because those are the ones that were known to be compatible. And we don't know about 16 or 17.

Leo: Right, right.

Steve: So it's like, yeah, no, no, I don't want that one.

Leo: We've kind of - we know about this world because this is what happens with Python and a lot of programming languages. And you do, you know, in the manifest you'll specify, no, it has to be this version or this version.

Steve: Right.

Leo: And it just won't compile and won't run.

Steve: Right.

Leo: What a mess.

Steve: So it's difficult to estimate at this early stage. But yes, Leo, it is an incredible mess. We don't know, you know, so much depends upon the maintenance of packages which may not be receiving routine updating. In attempting to get some answer to this question, Google looked at all publicly disclosed past critical advisories, okay, all publicly disclosed past critical advisories affecting Maven packages to get some sense of how quickly other vulnerabilities in the past have been fully addressed. Okay. You ready? Less than half (48%) of Java packages affected by a previous critical vulnerability have ever been fixed.

Leo: That's nice.

Steve: So Google says we might be waiting for a long time, likely on a scale of years.

Leo: Holy moly.

Steve: Yeah. On the other hand, those vulnerabilities may not have made the front page, and there might never have been the pressure to fix them that there has been on this one. So we can hope. In less than a week, 4,620, that's 13% of those known to be vulnerable have been fixed. But we can also assume that those had shallow dependency depths and didn't require some guy, I mean, there's one that's eight layers down. How are you ever going to get all of the packages in an eight-layer dependency tree fixed? I don't know.

Leo: I don't know.

Steve: So what's happening on the attack front? No good news there. Bitdefender, the Romanian AV maker, said it has spotted the first ransomware group using the Log4Shell vulnerability to infect and encrypt unpatched systems. The first attacks were spotted on Sunday, December 11th. Okay. Remember this was announced on the 9th. So it took two days, two days to incorporate Log4j into a ransomware attack. They have been carried out using a new strain of ransomware named Khonsari. Khonsari means "bloodshed" in Persian. The Khonsari ransomware - yeah, great - is coded in .NET and can only target Windows systems, and it has not impressed researchers. The MalwareHunterTeam and

Michael Gillespie, he's that guy that does all of the free decryptors when they can be done. They've both described it as low-effort "skidware" assembled from publicly available code.

Leo: I'm not going to go too far down the road of this, but I think I know what "skidware" implies. I don't want to go too far down that road.

Steve: That's good.

Leo: We'll leave it as an exercise for the reader.

Steve: That's their term. But despite its rather low quality, the ransomware is functional and can successfully encrypt systems after a successful Log4Shell attack. Khonsari uses properly designed encryption, and no flaw so far has been found in it.

A group that doesn't produce "low-effort skidware" is the professional Conti ransomware gang, responsible for hundreds of millions in received ransomware payments. They were the second ransomware group to immediately jump on the vulnerabilities in Log4j with scans and attacks beginning on Monday, December 13th, so four days following the December 9th public disclosure of the vulnerabilities. The Conti gang are specifically targeting, at least now, VMware vCenter servers, which are known to be vulnerable to Log4Shell attacks. When exploited, they gain access to the server and then immediately move laterally across enterprise networks.

So it has to be the case, Leo, that we got a lot of insinuation into networks, which as the researchers have said, no one's jumping on, or not everyone is necessarily immediately jumping on leveraging their newfound footholds. They're just wanting to get in immediately before the openings are closed. And once in...

Leo: We'll figure out what to do later. We're in there. Just get in now. You'll think of something later.

Steve: We'll look around and see who we landed. Yeah. So besides the threat from Khonsari, Bitdefender also reported on other malware groups that have been using vulnerabilities in Log4j to install DDoS and cryptomining botnets. The Chinese security firm Qihoo 360 said last Monday that they've been tracking at least 10 different groups abusing the vulnerability, as well. And Check Point has reported that it has seen at least 60, six zero, variations on the Log4Shell exploit so far, as a result of bad guys working around the quick mitigations which were put in place after the exploit first became public. Remember we talked about that, different ways of obscuring the variable expansion in order to avoid the filters that the people just quickly did.

And of course all the serious players jump on this immediately. The APT (Advanced Persistent Threat) groups based in China, Iran, North Korea, and Turkey were immediately active. The strategy appears to operationalize the vulnerability, as we've said, by discovering and establishing a foothold. And then, yeah, we'll see what we find. 1.8 million attempts to exploit the Log4j vulnerability have been recorded as of early last week. This thing just took off like a rocket. And initial access brokers, remember those are the miscreants who first penetrate systems to then sell their illicit access to others to exploit, Microsoft's Threat Intelligence Center, the MSTIC, said it had observed IABs, these Initial Access Brokers, immediately leveraging the flaws in Log4j to obtain initial access to target networks that were then sold to other ransomware affiliates.

Leo: Oh, that's a good idea. Break into them and just say, I've got access, anybody wants to buy it. That's brilliant.

Steve: Yup. What's it worth to you?

Leo: What's it worth to you? Oh.

Steve: Yup.

Leo: God, this is so bad.

Steve: I know.

Leo: It really is bad.

Steve: Merry Christmas. In addition, Microsoft said that dozens of malware families running the gamut from cryptocurrency coin miners and remote access trojans to botnets and web shells have been identified taking advantage of Log4j. The industrial cybersecurity firm Dragos noted that: "This cross-cutting vulnerability, which is vendor-agnostic and affects both proprietary and open-source software, will leave a wide swath of industries exposed to remote exploitation, including electric power, water, food and beverage, manufacturing, transportation, and more."

They said: "Log4j is used heavily in external Internet-facing and internal applications which manage and control industrial processes, leaving many industrial operations like electric power, water, food and beverage, manufacturing, and others exposed to potential remote exploitation and access. It's important to prioritize external and Internet-facing applications over internal applications due to their Internet exposure, although both are vulnerable. As network defenders close off more simplistic exploit paths and advanced adversaries incorporate the vulnerability into their attacks, more sophisticated variations of Log4j exploits will emerge with a higher likelihood of directly impacting Operational Technology networks."

So as we said last week, while I don't plan to dwell on the issue to excess, I have the feeling that the impact of Log4j will be laced throughout the early months of 2022, Leo. And, boy, it really does, when you look at what Google's research showed, the depth of dependencies in the Java package trees, it's going to be some time before this is able to get itself fixed.

Leo: And I don't know how well-documented dependencies are. So if it's compiled into something that you use...

Steve: Right. How do you know?

Leo: I mean, if you're lucky, maybe in the documentation or somewhere it says, I mean, sometimes you hit, you know, you do a man command, and it'll show you dependencies at the bottom. But sometimes not.

Steve: But yeah. Eight layers down?

Leo: Yeah. It wouldn't know.

Steve: Some guy incorporated it, and maybe not even using it, but it's there because, you know, hey, it's free, so let's just say yeah.

Leo: Wow. And really all it takes to trigger it is a malformed string that it logs. So if it's logging stuff, including web traffic, if it logs that malformed string, you're done.

Steve: And it could have set up a RAM buffer to, like, log it in a ring so that it just has, like, you know, if something happens it's able to say here's what was going on. So it doesn't even have to be writing it. It could be just doing it in order to, like, hold a most recent event in memory in case it had a problem. And even that would be triggering this problem.

Leo: There are, the chat room's telling me, Dr. Flay, that there are some vulnerability checkers on GitHub. There's one called Grype. I don't know how well it works. I don't know if it's just comparing stuff you've installed to a list of known problems, or if it's actually testing it. But, I mean, and then there's another one called Log4Shell-tester from Huntress Labs.

Steve: I was glad that when I installed GitLab it was Ruby and not Java.

Leo: Yeah. But that's not a guarantee because you don't know. Down the road there might be a dependency that is in Java. So you're running GitLab now? I'm glad to hear that.

Steve: Yeah.

Leo: Yeah. That's just for your, that's right, I remember you talked about that, for your source code for SpinRite.

Steve: Actually not source. I won't be putting the source up there. It's so that - because all we really have now for vulnerability management is the newsgroups.

Leo: Right.

Steve: And it's fine if everything's sort of moving forward. But like when we get stuck on a motherboard, then like a week ago, and I'm trying to think because I'm like juggling all of this in my head...

Leo: So people can submit.

Steve: ...and say, okay, wait a minute, was it - yes.

Leo: Yeah, yeah, yeah.

Steve: And so the idea is we'll have GitLab available for managing and tracking persistent problems that don't just immediately like, oh, of course, and I, like, you know, produce another update and then it fixes that.

Leo: Git's really great for that. I mean, Git is an amazing tool. I really like it a lot.

Steve: Yeah. And I'm very - I may very well. Right now I'm using Sync.com to sync my source.

Leo: I know you are. And I was going to ask when you had first found that, why don't you just use Git? I might have even asked that.

Steve: You maybe did.

Leo: Yeah.

Steve: And it's just because I never have.

Leo: Right.

Steve: And I would be, I mean, it's one thing to put open source on Git. It's a different thing to put proprietary...

Leo: I wouldn't put anything proprietary in GitHub for sure because you don't know. They might screw up. You have private repositories, but they might screw up. But I keep, for instance, I keep my Emacs dotfiles up there. I keep, as I'm doing the Advent of Code, I'm putting my solutions up there. Perfect for that.

Steve: Yeah, again, stuff where the loss is not critical.

Leo: Yeah, perfect for that. But yeah, proprietary code. But if you're running your own server, that's - as long as Log4j's not in there, you're all right.

Steve: Well, but it is publicly exposed.

Leo: Yeah. It's going to have to be; that's right, yeah.

Steve: Yeah.

Leo: I really think that this is going to hasten the Zero Trust Model because at this point you might as well just assume that you have an attacker inside the network. Right?

Steve: Yeah. I would say, if you're an enterprise, and you have reason, like if your guys are screwing around last week and this week, trying to get this resolved before Christmas, and given the fact that it was so easy to scan for this remotely, and the bad guys jumped on this with a vengeance, I mean, like over the weekend, you know, IT was home on the weekend, and they were scanning over the weekend. So it's, you know, immediately too late.

Leo: Yeah. Yeah. And there's really, I mean, somebody uses it, there's no log of them using it.

Steve: Right.

Leo: So you just don't know unless you catch them, if you're using a Canary or something. But if they just lay low and say, look, we got access, it's going to be tough.

Steve: Now, what you really want is LogLog4j.

Leo: LogLog4j. What could possibly go wrong? Steve Gibson's at GRC.com. When you go there, there's a few things you might want to check out. Of course this show. He's got unusual versions of the show. We both have the 64Kb audio, but Steve has the 16Kb audio. Sounds a little bit like Alexander Graham Bell on his first recording. But, hey, it's small. Also small, the fantastic transcripts Elaine Farris does for every episode, which allow you - they're searchable and allow you to find parts of the show, or just to have it to read along while you listen. And he has full-quality audio. All of that at GRC.com.

While you're there, pick up a copy of SpinRite. Version 6.0 is out right now, as you know, the world's best mass storage recovery and maintenance utility. You will get 6.1 automatically if you buy 6.0 now. And you can also participate in the development of 6.1, imminent. Let's see. There's all sorts of stuff in there. Just browse around. It's kind of a rabbit warren of goodness: GRC.com. I'm sorry, I shouldn't have said that while you were drinking.

Steve: A rabbit warren.

Leo: Of goodness.

Steve: That's right.

Leo: Of goodness. Not a bad rabbit warren. A good rabbit warren. He takes messages. His DMs are open on Twitter. He's @SGgrc. We also have the show at our website, TWiT.tv/sn. You can subscribe, of course, in your favorite podcast application. There's a YouTube channel.

We do the show on Tuesdays. Now, next Tuesday we will not be here because it's the Best Of Security Now!. We picked...

Steve: Yayyyyy.

Leo: ...a bunch of horrible exploits from the year, and we strung them all together. It'll be fun. So next Tuesday there will be a show, but not live. You'll just download it, and you can listen, and it's fun. And then we will all be back in-studio after the New Year, January 4th for the next Security Now!. If you want to tune in and watch it, we do it at about 1:30 p.m. Pacific, 4:30 Eastern, 21:30 UTC on Tuesdays. You can stream it live, audio or video, at live.twit.tv. Chat with us at irc.twit.tv if you're doing that, or in the Discord if you're a Club TWiT member.

Otherwise, I hope, Steve, you and Lorrie have a wonderful Christmas. And are you going to do a New Year's Eve party? Just the two of you, a little champagne?

Steve: You know, she's got asthma, and so...

Leo: You don't want to go out. I'm not suggesting that these days.

Steve: Yeah. And but so, you know, Zoom and FaceTime, we'll say hi to our family.

Leo: Yeah, there you go.

Steve: And sort of hang out with them virtually.

Leo: Yeah.

Steve: And, you know, light the fire and, you know, just enjoy ourselves.

Leo: We have, down the road where my daughter lives, there was a holiday party December 11th. Twenty-eight people got COVID from this little party, and they think it was Omicron. It just spreads so easily. So it's just not worth, even if you're boostered, it's really not worth taking a chance, I think.

Steve: One of the MD talking heads on Sunday was likening it to the mumps in terms of communicability.

Leo: It's just going to spread like crazy, yeah.

Steve: So, yeah.

Leo: And, you know, most people, especially if you're boostered, it'll be sniffles, and you'll be fine.

Steve: And remember Vitamin D status has turned out to be...

Leo: And I'm taking that Vitamin D. I'm drinking my Vitamin C. I'm making sure I - and taking zinc and all that stuff. But it's kind of like Russian roulette, you know, you don't know. You might be the one that really gets sick or gets long COVID. And why take a chance?

Steve: Yeah.

Leo: So we're going to just hunker down, I think is the thing to do.

Steve: Yeah, we don't want to lose any listeners. Our listeners are valuable.

Leo: No, we can't afford to. Every one of you matters. I hope you all have a great holiday, take some time off, spend time with the fam. Enjoy your New Year's Eve, and we'll be back January 4th to continue as we count down the last three years of Security Now!. You son of a gun. Have a great New Year's, Steve. We'll see you next time.

Steve: Bye, everybody. Happy New Year.