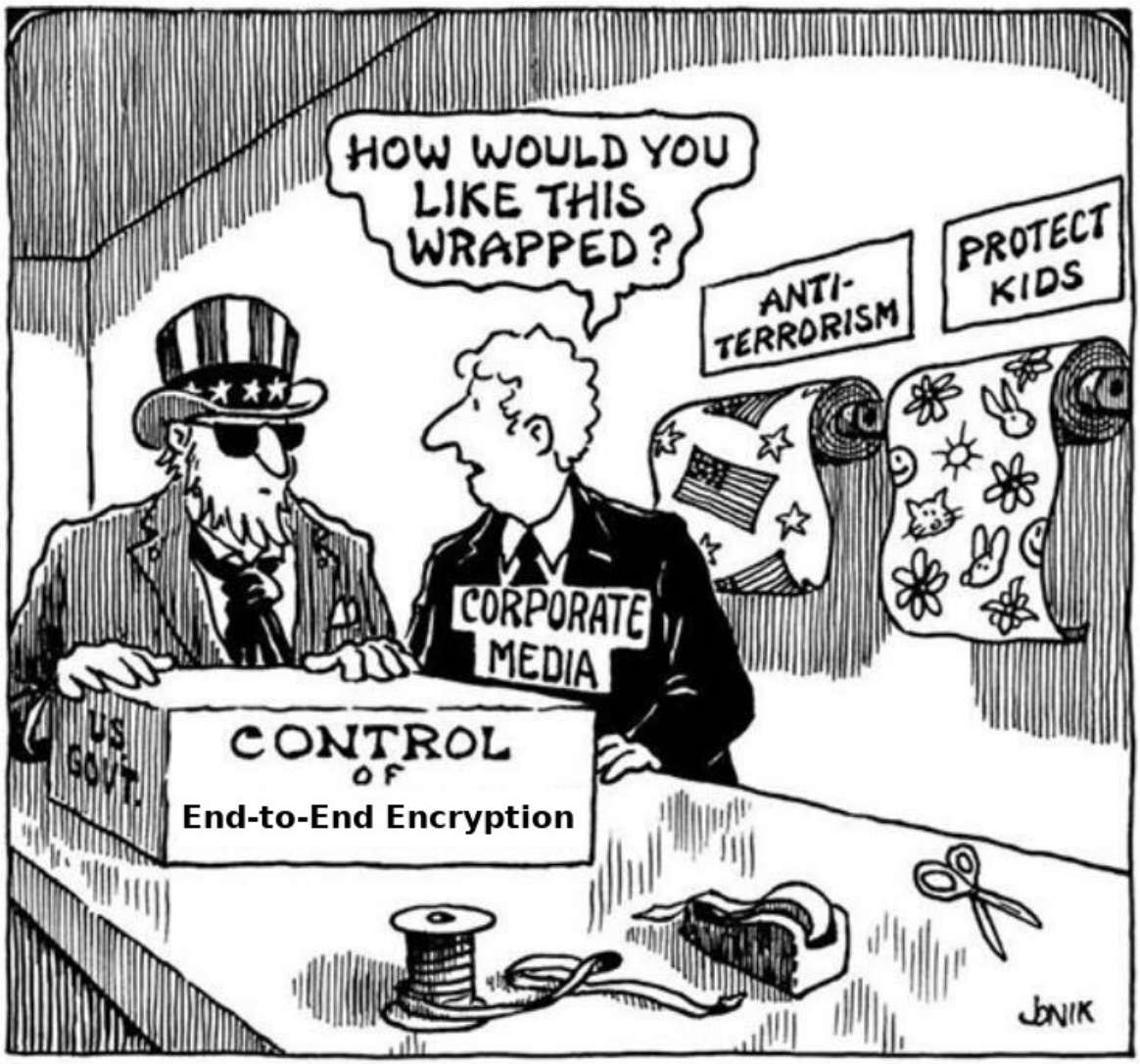
Security Now! #850 - 12-21-21

It's a Log4j Christmas

This week on Security Now!

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 0-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.



0-Day Watch

Google's 16th exploited Chrome 0-day of the year

Early last week, Google pushed an emergency Chrome update to resolve the 16th in-the-wild 0-day being used in attacks against its users.

Everyone using Chrome on Windows, Mac, or Linux desktops should now be at Chrome 96.0.4664.110. It eliminates a high-severity user-after-free flaw in Chrome's V8 JavaScript engine which someone had found and was using. And, as usual, that's about all we know, since Google has no need or reason to share anything more. 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.

And since we're wrapping up the year this week, we'll note that along with Chrome's extreme popularity comes a big bullseye target, 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 0-day flaw patched at the start of February. Then two in March, another two in April, May by, but June offered us a pair, July had only one, September kept the patchers busy by closing five 0-days, October had one, November also snuck past without any, and assuming that we're able to close out the year without any other, December will have brought us the 16th and final 0-day of the 2021.

As I noted before, I don't think this means anything about Chrome other than that it is receiving more than its fair share of scrutiny. One of the best models this podcast has developed for security is the idea of "porosity." Complex software security barriers are not absolutely secure. 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 really seem to be such a big problem. Other people are being attacked. But if our software security barriers are not absolutely sure, then what are they? They resist penetration, though imperfectly. This is why I'm enamored of the idea that our barriers are porous. The more pressure the bad guys place against the barrier, the more sneaks through.

Sadly, 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 three additional problems. (We'll be talking about those new developments at the end of the podcast.) None was quite as bad as the first, but all needing fixing. And the same thing happened with Microsoft's ill-fated Exchange Server and Windows Printing throughout 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 them in the first place. But like I said, we don't really seem to care enough to prevent these problems. So there's some revenue available for anyone who enjoys looking at code. And speaking of looking at code... there is absolutely no better way to improve one's own coding skill than by reading the work of other coders. You'll be surprised how much you can learn that way.

And as for browser vulnerabilities, there's currently a lot to be annoyed with Microsoft about. But their idea of backing off of Chromium's extreme optimization to reduce the browser's attack surface at the minimal cost of unneeded performance gains, seems quite compelling.

Browser News

Firefox refuses to do Microsoft.com!

Firefox had a bit of an adventure last week when it started refusing to connect to Microsoft.com. I don't recall why I wanted to go there, but I encountered the error myself and it caused quite a stir online.

Back in the early days of this podcast we discussed the many problems surrounding web server certificate revocation. Certificates expired only every three to five years back then. So one of the solutions to the problem of erroneously trusting a revoked certificate was to have the web browser perform an on-the-fly query to the certificate issuer's OCSP server, where OCSP stands for “Online Certificate Status Protocol”. In this way, a real time verification of the validity of an identity certificate being presented to the client could be checked.

The problem with this is that it takes time, and back when OCSP was still young, OCSP servers were too often overloaded or unresponsive. And the last thing a browser wanted to do was to slow down its users. So OCSP was often set to "fail open" such that only an affirmative negative response would block, but a non-response would not. And the problem with that was that bad guys who had stolen someone’s valid certificate might block a negative OCSP response from getting back to the user, thus causing their browser to erroneously trust a fraudulent website.

So, the concept of "stapling" was introduced. With stapling, rather than asking the user's web browser to go get an OSCP attestation, the web server itself could periodically obtain a recently signed and timestamped assertion from its own certificate authority, and "staple" that assurance to its own multi-year life identity certificate. The stapled OCSP attestations would only be valid for a short time — perhaps a few hours — so the web server would periodically reach out to renew the attestation it was stapling to all of its outgoing certificates well before its expiration.

The final bit of perfection comes from the webserver's certificate having its "OCSP Must Staple" flag 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 if it also has a valid recently stapled OCSP certificate. This measure protects the certificate holder from the theft of their certificate. If it is found to be stolen it can be immediately revoked by its issuer, who will then refuse to issue any further valid OCSP certificates for stapling. And since, by design, all OCSP certificates expire quickly, the stolen certificate will quickly become useless.

So, 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 dropped from the Certificate ID fields present in Microsoft’s OCSP certificates. Since the newer SHA-256 signature hash was still present, Safari and Chrome had no complaint and didn't even notice the change. But, somehow, Firefox had never enabled their browser’s support for SHA-256 hashes in OCSP stapled certs. The code was there and was ready to go, but it hadn’t ever been enabled.

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 “OCSP Must Staple”, but from Firefox’s vantage point the stapled certs appeared to be unsigned and thus invalid. So Firefox properly refused to allow its users to go any further.

Fortunately, the problem was quickly remedied. Last Thursday's December 16th release of Firefox v95.0.1 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"*

Firefox disabled Microsoft's Cloud Clipboard

Since we're talking about Firefox and Microsoft, get a load of this "what could possibly go wrong" feature in Windows:

It's known as the Windows Cloud Clipboard, a feature that was added to Windows 10 way back in September of 2018 with Win10's 1809 release. And it is what it sounds like: a feature that allows users to sync their PC local clipboard history to their Microsoft accounts. Fortunately, the abuse-prone feature is disabled by default, but when enabled, it allows Windows users to access the cloud clipboard by pressing the Windows+V shortcut. This grants users access to clipboard data from all their other devices. And more than that, the service also maintains a clipboard history, allowing users to go through past items they copied or cut and then re-paste the same data in new contexts.

What recently came to light, and what Firefox fixed in last month's Firefox 94, was 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. Is that a bug or a feature? Either way, Mozilla decided it was not good.

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 (about:logins) will no longer be stored in the Windows Cloud Clipboard feature, but instead will be stored only locally, in a separate clipboard section.

Mozilla said it considered this behavior dangerous [yeah!] 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. 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 user credentials.

For what it's worth, no Chromium-based browsers include this protection, and usernames and passwords from Chrome were synced to Microsoft cloud servers—so users should be aware of who might access their passwords if they have Cloud Clipboard enabled and are using a non-Firefox browser on Windows.

Security News

Weaknesses 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 Telecommunications."*

"Handover" is the official name for what we normally call cellular inter-tower handoff. It's the process by which a phone call or data session is transferred from one cell site base station to another without losing connectivity during the transmission. And, of course, it's crucial to establishing and maintaining cellular communications when the user is on the move.

Here's how this works in a bit more detail: the user's equipment sends its received signal strength measurements to the network so that the network can determine whether a handoff is necessary and, if so, the network facilitates a switch between base stations when a more suitable target station is available. Although these signal readings are cryptographically protected, the content of these reports is not verified. This allows an attacker to spoof reports and to thereby force a device to move to a cell site operated by an attacker. An attack arises because these unverified signal strength reports cannot be detected.

So, essentially, a long long established protocol, which was never properly designed to be spoof proof has, indeed, been proven not to be. This doesn't create any new huge 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 a stronger signal. This new research dramatically reduces this requirement so that any user within range can have their traffic rerouted through a man-in-the-middle.

In an experimental setup, the researchers found all test devices, including OnePlus 6, Apple iPhone 5, Samsung S10 5G, and Huawei Pro P40 5G, ALL susceptible to DoS and MitM attacks. Their findings were presented at the Annual Computer Security Applications Conference (ACSAC) held earlier this month.

Cross Wi-Fi / Bluetooth leakage

A team of German and Italian researchers have significantly updated their research regarding a class of what they call "Coexistence Attacks" against Wi-Fi and Bluetooth Chips. Their research paper is titled: "Attacks on Wireless Coexistence: Exploiting Cross-Technology Performance Features for Inter-Chip Privilege Escalation" https://arxiv.org/pdf/2112.05719.pdf

The paper's Abstract explains:

> *Modern mobile devices feature multiple wireless technologies, such as Bluetooth, Wi-Fi, and LTE. Each of them is implemented within 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 shared resources, essential to maximizing networking performance. Today's hardwired coexistence interfaces hinder clear security boundaries and separation between chips and chip components. This paper shows practical coexistence attacks on Broadcom, Cypress, and Silicon Labs chips deployed in billions of devices. For example, we demonstrate that a Bluetooth chip can directly extract network passwords and manipulate*

traffic on a Wi-Fi 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.

To put some additional context to this, I want to share their paper's introduction, which I think everyone will find interesting and, unfortunately, worrisome...

Wireless communication is enabled by Systems on a Chip (SoC), implementing technologies such as Wi-Fi, 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 SoC vulnerabilities multiple months before public disclosure.

Mobile device vendors account for potentially insecure wireless SoCs by isolating them from the Operating System (OS) 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 SoC are rather complex and need to find a bypass for each mitigation.

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 mechanisms they use to arbitrate their access to the resources they share, i.e., 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 can share resources like processors or memory.

This paper demonstrates lateral privilege escalations from a Bluetooth chip to code execution on a Wi-Fi chip. The Wi-Fi chip encrypts network traffic and holds the current Wi-Fi credentials, thereby providing the attacker with further information. Moreover, an attacker can execute code on a Wi-Fi chip even if it is not connected to a wireless network. In the opposite direction, we observe Bluetooth packet types from a Wi-Fi chip. This allows determining keystroke timings on Bluetooth keyboards, which can allow reconstructing texts entered on the keyboard.

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. For instance, Bluetooth?Wi-Fi 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.

Sci-Fi

"The Matrix Resurrections" aka "The Matrix 4"

Just a reminder for those die hard Matrix fans, among whom I number, that being a Warner Brothers release, the 4th and almost assuredly final movie in "The Matrix" series co-releases tomorrow morning, December 22nd, in theatres and streaming on HBO Max. So, despite the fact that IMDB currently pegs it at a 6.8, which falls beneath my normal IMDB 7.0 ratings cutoff, I'm quite certain that Lorrie and I, having received our vaccine boosters at 9am tomorrow morning, will be snuggled into our couch with popcorn at home for two and a half hours of eye candy and special effects enjoyment. And, no... I'm not expecting much and I don't expect to be wowed or surprised. Movies aren't much anymore.

SpinRite

Last week, as planned, 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 — hundreds and hundreds of the little buggers — download, install themselves and set up shop. 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 there. And what choice do you have? It's free. Do you want it, or not? What are ya gonna do? Write it yourself?? (Actually, there was a comment in 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 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 fully know yet, or trust it yet — trust is earned, after all. But now that we have it, I think it makes sense to use it. Our threaded-conversation textual newsgroups are perfect for what they are. They're fast and simple for casual use. And they allow us to move with great speed. They've also served us well up to now. But they're not ideal when we get stuck on a problem. We already have the web forums, which are targeted at end users. And GitLab is clearly not aimed at the uninitiated end user. Initially, there's an awful lot of "where's the button I'm supposed to push?" 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 with this entire process, and that's fine. They swing by every week or two to give the latest release a try. And they generally just post that it worked again like they all have. And I don't want to mess that up or burden them with the need to go somewhere else. So we'll still have the newsgroups for quick posting of go/no-go notes about 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.

And speaking of GRC's web forums and GitLab, last week I tweeted a shout-out to the free service whose performance has stunned me. What I tweeted was:

> *A shout out to @StopForumSpam. GRC's forums were drowning in forum spam, because forum spammers are people, typically in the Eastern bloc, who create temp GMAIL accounts and manually bypass all CAPTCHA challenges. But after adding StopForumSpam -- not a SINGLE fake registration!*

About six months ago, I looked high and low for some solution, and "StopForumSpam" was it. Since then it's been a true godsend for GRC's forums. It's built into our XenForo forum software, and before I allow open public registration for GRC's GitLab instance I may try to find some means for incorporating a query to the StopForumSpam API before approval. If anyone knows GitLab or Ruby and is interested, merging a call to the StopForumSpam API into the registration process would be a HUGE win for the world! It turned out that the guy behind StopForumSpam is a longtime SpinRite user and has listened to Security Now since episode #1. So I sincerely hope that his ears are burning, because the service he's providing is truly valuable. Forum spam is a VERY real problem.

And as for SpinRite, I think a pattern is beginning to emerge. All my new SpinRite code generally works perfectly on all newer hardware. Pretty much anything I do just works right off the bat there. But it's on older hardware — typically very old hardware — where early and probably slightly incompatible operation was originally being hidden behind those older systems' BIOSes. But now that SpinRite is bypassing their BIOSes, we're encountering some gotchas which is what we're in the process of addressing.

My most recent eBay purchases of this old hardware arrived by the end of last week. I believe that I have one of everything that's been reported as causing trouble — three older systems. And after updating the BIOS of the first of those motherboards, I've recreated the problem that two of our testers encountered. I can't wait to find out what's going on!

It's a Log4j Christmas

Last week, 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. If last week's podcast served to introduce the problem, this week we're able to gain some perspective on its consequences.

As we know, Log4j earned that very rare CVSS score of 10.0. But I'm put in mind of the 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 ever there was a case for a CVSS score of 11 out of 10, Log4j is as good a case as any.

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 (ASF) published an updated fix for the Log4j vulnerability after it was discovered that the previous patch was incomplete in certain non-default configurations.

It was technically a second but related vulnerability, so it was given its own CVE of 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 out of 10 because it was initially believed to only have DoS consequences — meaning that bad guys could cause something to crash but apparently couldn't do any worse. 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 leak and remote code execution in some environments, and local code execution in all environments. Whoopsie!

So this second discovery affected all versions of Log4j from 2.0-beta9 through 2.12.1 and 2.13.0 through 2.15.0 — and 2.15 was the big fix that the Log4j project maintainers had just shipped the week before. As a result of the incomplete patch for the original CVE-2021-44228, it was possible to craft malicious input data using a JNDI Lookup pattern resulting in a denial-of-service (DoS) attack and later more.

So this brought us to v2.16.0, for Java 8, and that newer release essentially removes support for message lookups and disables JNDI by default, thank god. Users who need a solution for Java 7 were, last week, recommended to upgrade to release 2.12.2 as soon as it's available.

The Apache Software Foundation's Ralph Goers said: *"Dealing with CVE-2021-44228 has shown that JNDI has significant security issues. 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."*

And since last week the line-up of well-known affected organizations has, of course, grown. A list compiled early last week, in alphabetical order, of only the biggest and most well-known read:

"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."

Also as of early last week, ten different attacking groups have been identified, and roughly 44% of corporate networks globally have been under attack, and the US CISA Cybersecurity and Infrastructure Security Agency (CISA) has given all federal agencies the deadline of December 24th to fully incorporate all patches for the vulnerability. No one gets to have Christmas until every agency's Log4j vulnerabilities have been fully remediated.

Sopho'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 is a lull before the storm in terms of more nefarious activity from the Log4Shell 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."*

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 which enables adversaries to exploit servers locally by using a JavaScript 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."* He said: *"At this early stage there is no proof of active exploitation. But this vector significantly expands the attack surface and can impact services even running as localhost which were not exposed to any network."*

Although this latest discovery can be resolved by updating all local development and internet-facing environments to Log4j 2.16.0, 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 which was fixed by version 2.15.0.

- Then CVE-2021-45046 with its score upgraded from 3.7 to 9.0 which is fixed by 2.16.0.

- And finally, CVE-2021-45105 with its CVSS of 7.5 which is fixed by 2.17.0.

Jake Williams, the CTO and co-founder of the incident response firm BreachQuest said: *"We shouldn't be surprised that additional vulnerabilities were discovered in Log4j given the additional specific focus on the library. Similar to Log4j, this summer the original PrintNightmare vulnerability disclosure led to the discovery of multiple additional distinct vulnerabilities. The discovery of additional vulnerabilities in Log4j shouldn't cause concern about the security of log4j itself. If anything, Log4j is more secure because of the additional attention paid by researchers."*

Okay. So not too surprisingly, additional problems were found with the original vulnerability and

its initial fixes turned out to need fixing. So last week there was some additional scurrying. The main takeaway is that while it's not as urgent, updating again to 2.17.0 would probably be a good idea.

So, what have we learned about the extent of Log4j's usage which needs to be found and fixed?

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 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 (CVE-2021-45046).

Members of the Google Open Source Insights Team said that when a major Java security flaw is found it generally affects 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 the Maven Central's total of ~440,000. This was an impact that the team members characterized as "enormous."

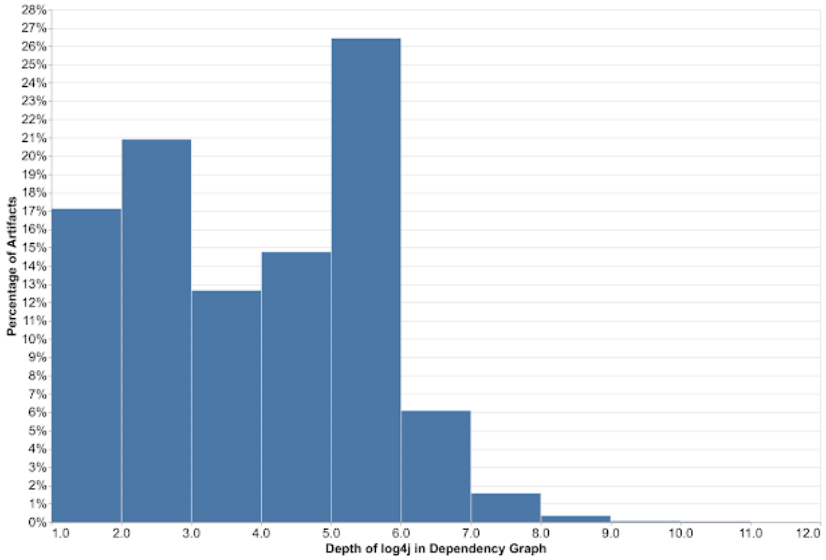
And despite the effort and focus, fixing everything is a monumental task. Google published their report at the end of last week on the 17th where, at that point, 4,620 out of the total 35,863 packages had been updated. But we've seen that these things tend to taper off exponentially with the most work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

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 from other Java libraries, which 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 (though 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 use have updated their own libraries.

To put some numbers to this, 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.

When Google was asked why fixing the JVM ecosystem was so hard, Google's open-source team explained that by far most JAVA modules depend upon Log4j indirectly. And the deeper the vulnerability is down a dependency chain, the more steps are required for it to be fixed since the bottom-most 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. Then the module above that one needs to be rebuilt, and so on.

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 use of vulnerable Log4j instances:



You'd expect to see a more or less decreasing incidence statistically. 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% of all dependencies occur at a depth of 4. But then, for some reason, that almost doubles to about 26.5% of all dependencies occurring at a depth of 5 — more than one in four. And then, as if to make up for that, it drops off dramatically to 6% at a depth of 6.

But this means that if the root Log4j dependency is 5 dependency layers down, then all five of those packages must each be updated, and they can only be updated from the bottom up. So everyone in the chain must wait their turn until the package(s) beneath them have been updated. And note that since dependency trees branch outward as they descend, it might also be that any of those packages could have multiple dependencies which are each affected by Log4j, and they would all need to be updated before the dependent package could be updated... which would then finally allow the packages that depend upon it to consider updating.

It's pretty clear why this disaster is going to take some time entirely to wash out of Java's code base.

How much time? ... is difficult to estimate at this early stage since 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 affecting Maven packages to get a sense of how quickly other vulnerabilities have been fully addressed. Are you ready for this? Less than half (48%) of Java packages affected by a previous vulnerability have been fixed. So Google says that we might be in for a long wait, likely on a sale of years.
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 and is for Log4j. After all, in less than a week, 4,620 affected libraries (~13%) have been fixed. How quickly that pace slackens only time will reveal.

What's happening on the attack front?
Bitdefender, the Romanian A/V maker says 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 11, and were being carried out using a new strain of ransomware named Khonsari. (Khonsari means "bloodshed" in Persian.) The Khonsari ransomware is coded in .NET and can only target Windows systems and it hasn't impressed researchers. The MalwareHunterTeam and Michael Gillespie have both described it as low-effort "skidware" assembled from publicly available code. 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 has, so far, been discovered in it.

A group that doesn't produce "low-effort skipware" is the professional Conti ransomware gang. They were the second ransomware gang to immediately jump on the vulnerabilities in Log4j with scans and attacks beginning Monday, December 13, just 4 days following the December 9th public disclosure of the vulnerabilities. The Conti gang are specifically targeting VMWare vCenter servers, which are known to be vulnerable to Log4Shell attacks. When exploited, they gain access to the server and move laterally across enterprise networks.

Besides the threat from Khonsari, Bitdefender also reported on other malware gangs have been using the 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 ten different groups abusing this vulnerability as well. And Check Point has reported that it has seen at least 60 (six zero) variations of the Log4Shell exploit so far, many as a result of bad guys working to evade detections and those first quick mitigations which were put in place after the exploit first became public.

And, of course, all of the serious players jump on this immediately. These are the APT — Advanced Persistent Threat — groups based in China, Iran, North Korea, and Turkey. The strategy appears to be to operationalize the vulnerability by discovering and establishing some foothold in as many systems as possible, as quickly as possible, with the intent to come back later to see what fish were caught in the net. By these serious actors, over 1.8 million attempts to exploit the Log4j vulnerability have been recorded as of early last week. Exploitation of this thing just took off like a rocket.

Initial Access Brokers (IABs) are those miscreants who first penetrate systems then sell their illicit access to others for exploitation. Microsoft Threat Intelligence Center (MSTIC) said it had observed IAB's immediately leveraging the flaws in Log4j to obtain initial access to target networks that were then sold to other ransomware affiliates. 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 swathe of industries exposed to remote exploitation, including electric power, water, food and beverage, manufacturing, transportation, and more.”

"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 in 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.