URL Parsing Vulnerabilities

Description: This week we'll begin with another in our series of Log4j updates which includes among a few other bits of news, an instance of a real-world vulnerability, and the FTC's somewhat surprising and aggressive message. We'll chronicle the Chrome browser's first largish update of 2022 and also note the gratifying 2021 growth of the privacy-centric Brave browser. WordPress needs updating, but this time not an add-on but WordPress itself. We're going to then answer the age-old question posed during last Wednesday's Windows Weekly podcast: "What exactly is a Pluton, and how many can dance on the head of a pin?"

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-853.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-853-lq.mp3

And finally, after a quick sci-fi reading recommendation and a very brief touch on my ongoing SpinRite work, we're going to take a gratifyingly deep dive into the unfortunate vagaries of our industry's URL parsing libraries to see just how much trouble we're in as a result of no two of them parsing URLs in exactly the same way.

SHOW TEASE: It's time for Security Now!. Steve Gibson is here. He's going to explain/answer Paul Thurrott's question, what exactly is a Pluton? We'll talk about success for the Brave browser, more Log4j vulnerabilities, and then a very interesting deep dive into the problems with URL parsing and why so many programs get it wrong. It's all coming up next on Security Now!.

Leo Laporte: This is Security Now! with Steve Gibson, Episode 853, recorded Tuesday, January 11th, 2022: URL Parsing Vulnerabilities.

It's time for Security Now!, the show where we cover the latest news from the world of security, thanks to our professional security analyst Steve Gibson of GRC.com. He's right here. I like the eyebrows.

Steve Gibson: If I were to nod and synchronize my eyebrows, they could stay motionless while my head moved. I'll have to work on that.

Leo: And there you see on display Steve's amazing analytic abilities. It's just how he thinks, ladies and gentlemen. Well, hi, Steve.

Steve: Yo.

Leo: Yo. Let's get going here.

Steve: Yo.

Leo: Yes.

Steve: Well, so it's 011122.

Leo: Yes.

Steve: Security Now! 853. A team of four researchers, security researchers in the northeast, Massachusetts and New York, I think, took it upon themselves to look at another aspect of our libraries that we're all using and discovered some really, really interesting problems which are right now affecting everyone. And anyway, so a really interesting topic, I think, one of our deep dives. Those who do not have the advantage of show notes in front of them are going to have to - I was going to say close your eyes, but not if you're commuting. But if you're being commuted, then that would be good.

Leo: The governor will call any minute now, I'm sure.

Steve: Then you'll have to picture some of these. But I'll be gentle. Anyway, I think a really interesting topic of URL parsing vulnerabilities. But this week we're going to begin with another in our series of Log4j updates, which includes among a few other bits of news an instance of a real-world vulnerability that has popped up and the FTC's, the U.S. Federal Trade Commission's somewhat surprising and aggressive message relative to Log4j.

We'll chronicle the Chrome browser's first largest update of 2022 and also note the gratifying 2021 growth of the privacy-centric Brave browser. WordPress needs updating, but this time not one of those pesky add-ons, but WordPress itself. We're going to then answer the age-old question posed during last Wednesday's Windows Weekly podcast: What exactly is a Pluton, and how many can dance on the head of a pin?

And finally, after a quick sci-fi reading recommendation and a very brief touch on my ongoing SpinRite work, we're going to take, as I said, a gratifying deep dive into the unfortunate vagaries of our industry's URL parsing libraries to see just how much trouble we're in as a result of no two of them parsing URLs in exactly the same way.

Leo: Interesting, fascinating, might I say engaging content, as always, coming up on Security Now!. And I will, for those of you watching video, I'll display the images. But you'll just, the rest of you, use your imaginations. And then where do people get the show notes, actually? I should - because people may want to after the fact see them. You have them on your website; yes?

Steve: Oh, yeah, GRC.com/securitynow and they're already posted. I did that this morning.

Leo: Good, good.

Steve: So the link is there.

Leo: Let's get that Picture of the Week. You ready?

Steve: Ah, well, this is another one of those oldies but goodies. Well, not really old because it's 2021. But they never get tired. Someone tweeted this to me, Dave Hart. He said: "Thought you'd enjoy this screen at OR Tambo Airport in Johannesburg, South Africa."

Leo: Oh, wow.

Steve: And of course this is - and I love it because it says "2021 Year of Security Culture" it's boasting about. And it's got all kinds of extra logos and bureaucratic mumbo-jumbo. And right in the middle of the whole thing, otherwise very impressive presentation, is a blue box, says "Your Windows license will expire soon."

Leo: Whoops.

Steve: It's like, okay, what?

Leo: Whoops.

Steve: So they're running an unlicensed version of Windows, or what?

Leo: That's secure, sure.

Steve: I wasn't aware that Windows licenses expired. I mean, I've never had a license expire. Is that some...

Leo: No, yeah, yeah, yeah, this is what you get if you first install Windows without giving it an authentication code.

Steve: Ah, right, right, right.

Leo: You can run it for 30, 60, or 90 days. But eventually it says oh, no, you've got to - now you've got to enter your - you've got to activate.

Steve: Yeah. Year of security culture, not so much.

Leo: Whoopsies.

Steve: So anyway, yeah.

Leo: Well, we got it on the Internet. We thought it would work for forever. It must...

Steve: If only there was a - it says "Go to Settings." If only you could press the Settings button. But no.

Leo: No, it's in kiosk mode.

Steve: I don't think that's going to happen. So anyway. Just another fun little bit of flotsam.

Leo: Awesome.

Steve: Okay. Our Log4j update. The U.S. CISA has stated that the Log4Shell exploit of the Log4j vulnerability has not resulted in "significant" government intrusions yet. CISA said: "We're not seeing confirmed compromises of federal agencies, including critical infrastructure. We are seeing widespread scanning by malicious actors. We're seeing some prevalence of what we would call low-level activities like installation of cryptomining malware, but we're not seeing destructive attacks or attacks attributed to advanced persistent threats." So it's like, okay, don't relax, everybody, but so far it's not the end of the world.

Eric Goldstein, who's the CISA's Executive Assistant Director for Cybersecurity, said that there would be a "long tail remediation" because of how widespread the issue is. And of course that's what we've predicted, too, because of this problem of the Java library inheritance tree. He said CISA estimates that hundreds of millions of devices are impacted. Easterly said - oh, Easterly, that's somebody else - anyway, that CISA is aware of reports of attacks affecting foreign government agencies, including the Belgian Defense Ministry - and I did pass over a story about that - as well as reports from cybersecurity firms that nation-state adversaries are developing attacks using Log4Shell, but said CISA cannot independently confirm those reports at this time. So stay tuned.

The overall security group posture is that there's scanning being done and implanting occurring, like a race to get systems' networks penetrated. The exploitation of those penetrations is on a backburner because everyone knows sooner or later those vulnerabilities that are creating the openings are going to close. So it's frightening because it means the bad guys are, like, acting in a smart fashion, in a means to optimize their long-term benefit for short-term forestalling the glory for just getting into systems. So anyway, I think we're going to have an interesting 2022.

Meanwhile, Matt Keller, the vice president of Federal Services at GuidePoint Security, told Threatpost that many agencies are unable to patch the problems arising from Log4j due to network-connected end-of-life and end-of-service systems. In other words, mission-critical systems that are still in service, but which cannot be updated because they've passed out of their maintenance life. There aren't going to be updates coming from them, like, oh, yes, you've got Windows 7? Too bad.

Anyway, Matt said that federal agencies are relying upon command-line scripts to locate affected systems. They're also constructing tiger teams as a means of tearing into the massive workload that has resulted, tiger teams being specialized cross-functional teams brought together specifically to solve or investigate one particular problem or critical issue. Between technology issues which may prove to be intransigent, travel restrictions, and shipping delays involved in replacing these systems, Keller predicts that agencies are months away from being able to address Log4j, despite the fact that the CISA said, "No Christmas for you until we've got this fixed. You're not going to go home. You're just going to stay here." Well, everybody left. And so it's going to still be a few months before things calm down.

An example of a specific and unfortunately typical Log4j-based vulnerability has been found in a popular open source Java-based, of course, database engine library. That Maven Java repository which Google had plumbed and we talked about before, indicates that this popular, it's called the "H2 database engine," is being used by 6,807 individual, what they describe as "artifacts," meaning, you know, up-tree dependencies, or dependents, rather. We've talked about the tree of dependencies that one gets when libraries are built from libraries which are built from libraries and so on.

In this case, this HS database appears somewhere underneath those 6,807 different high-level Java "things." And since this is an embedded database engine, where you typically don't even see it, right, it's just like your thing, like, has a database that comes from somewhere. Well, this is where it comes from. But you never really see the H2 embed. So it might not even be clear to those using it that accessible Log4j vulnerabilities are buried underneath there. This particular issue being tracked is CVE-2021-42392 and has been described as "the first critical issue published since Log4Shell, on a component other than Log4j, that exploits the same root cause of the Log4Shell vulnerability, namely that JNDI remote class loading."

Shachar Menashe, the senior director of JFrog security research, said: "Similar to the Log4Shell vulnerability uncovered in early December, attacker-controlled URLs that propagate into JNDI lookups can allow unauthenticated remote code execution, giving attackers sole control over the operation of another person or organization's systems." So basically that's what we know of as the Log4j problem in a nutshell. He added that: "The H2 database is used by many third-party frameworks including Spring Boot, Play Framework, and JHipster. While this vulnerability is not as widespread as Log4Shell, it can still have a dramatic impact on developers and production systems if not addressed." So, yeah, 6,807 of them.

The flaw affects H2 database versions 1.1.100, so that sounds like almost at the start, up through 2.0.204. And it's been addressed in .206, which was first made available last Wednesday, January 5th. And I should note that the JFrog folks, who I was unaware had their eye on this, we'd run across them a couple times before relative to Java stuff. They've got a terrific page of Log4j and Log4Shell resources, including a remediation cookbook, free scanning tools to see if you've got the problem, and what they call a "survival guide."

I have a link to their page in the show notes, but you can just put "JFrog" into any search engine and click, after you go to their page, click the banner link that they've got at the top of the homepage, and that'll take you to their very comprehensive Log4j/Log4Shell resources. Yeah, it's right there at the top, Leo. Yeah, yeah. Is it that little black bar? Yeah, there it is. Yup, and there you are at the resource page. So definitely a useful resource for any of our listeners or their friends who are part of that month-long remediation battle for getting this thing under control.

Okay, now, this sort of surprised me. A week ago, on the 4th of January, while we were doing this podcast, the U.S. Federal Trade Commission posted a blog warning, essentially

warning companies to remediate the Log4j security vulnerability. The blog's title, their own, the FTC's own blog title is: "FTC warns companies to remediate Log4j security vulnerability." And in its posting it directly threatens companies with legal action, likening it to the Equifax negligence. So here's what the FTC posted.

They said: "Log4j is a ubiquitous piece of software used to record activities in a wide range of systems found in consumer-facing products and services." You know, that's the FTC's charter. They said: "Recently, a serious vulnerability in the popular Java logging package, Log4j" - and then they cite the CVE 44228 - "was disclosed, posing a severe risk to millions of consumer products to enterprise software and web applications. This vulnerability is being widely exploited by a growing set of attackers."

They said: "When vulnerabilities are discovered and exploited, it risks a loss or breach of personal information, financial loss, and other irreversible harms. The duty to take reasonable steps to mitigate known software vulnerabilities implicates laws including, among others, the Federal Trade Commission Act and the Gramm Leach Bliley Act. It is crucial that companies and their vendors relying on Log4j act now in order to reduce the likelihood of harm to consumers, and to avoid FTC legal action."

They said: "According to the complaint in Equifax, a failure to patch a known vulnerability irreversibly exposed the personal information of 147 million consumers. Equifax agreed to pay $700 million to settle actions by the Federal Trade Commission, the Consumer Financial Protection Bureau, and all 50 states. The FTC intends to use its full legal authority to pursue companies that fail to take reasonable steps to protect consumer data from exposure as a result of Log4j, or similar known vulnerabilities in the future." So, yikes. That's a newly aggressive tone from the government side that sort of feels like it's getting a little tired of having companies say, well, it's not our fault. We had to go home for Christmas. You know? Christmas happens.

The FTC added an interesting acknowledgement, perhaps in response to anyone wanting to hold the Log4j authors accountable, which you could imagine would be what some pencil-neck C-Suite guy would say: "Hey, it's not our fault. It's their software." So the FTC closed their blog posting by writing: "The Log4j vulnerability is part of a broader set of structural issues. It is one of thousands of unheralded but critically important open source services that are used across a near-innumerable variety of Internet companies. These projects are often created and maintained by volunteers, who don't always have adequate resources and personnel for incident response and proactive maintenance, even as their projects are critical to the Internet economy." I thought this was really interesting, that this was in this otherwise very aggressive posting.

They finished: "This overall dynamic is something the FTC will consider as we work to address the root issues that endanger user security." In other words, it sounds like they're saying, look, you know, we recognize that this is a thankless job that volunteers are doing for the benefit of all. And yeah, they make mistakes, but we're not going to hold them responsible. We're going to, you know, when vulnerabilities are disclosed and widely publicized, and you're told that you can't go home for Christmas, if you do that anyway, and consumers are harmed, you're not going to be able to blame the author of the software. So wow.

In browser news, Chrome fixed 37 known problems last week. I'll note in passing that last Wednesday Chrome received its first update of 2022, which moved Chrome to 97.0.4692.71. The update did not fix, refreshingly, any disclosed zero-day vulnerabilities. Far as we know there weren't any. But it did address 37 security-related issues, one of which is rated critical in severity and could be exploited to pass arbitrary code and gain control over a victim's system. So that baddy is another of the apparently ubiquitous use-after-free bugs, this one having resided in Chrome's storage component. If exploited, it could have had devastating effects ranging from corruption of valid data to the

execution of malicious code on a compromised machine. So it's a good thing that Chrome auto-updates. Everybody's is fixed. I checked mine because my Chrome, as we know, is famously sluggish in updating itself. But I was happy to see that I was also at .71. So, good.

Twenty-four out of the total of 37 flaws were reported to Google by external researchers, and they consider their Project Zero initiative to be external, I guess, to the Chromium Project itself because of course as we know Project Zero looks at everything and has helped lots of other non-Google, even Google-competing projects. The other lucky 13 flaws were flagged as part of Google's own continuous internal security work. Of the 24 externally reported bugs, 10 were rated high severity, 10 were given medium rating, and the rest were low.

Oh, and the privacy-first Brave browser, I thought it was interesting. Following up on last week's discussion of DuckDuckGo's continuing dramatic exponential growth, I wanted to also note that the Brave browser's 2021 usage more than doubled from its previous year. Brave began 2021 with 24.1 million users, those monthly million users, and ended the year with 50.2 million, for a 220% growth during 2021, up 2.2%, or 2.2 times. And its growth is also exponential since Brave has been more than doubling their user base year after year now for the past five years.

And for those who haven't been following the Brave alternative closely, it's also based upon the now quite common Chromium platform and thus offers nearly all of the features available in the other Chromium browsers, including Chrome and Edge. But Brave separates itself from them by explicitly not tracking searches or sharing any personal or identifying data with third-party companies like Google or Microsoft. In addition to its monthly active users, which is what I just quoted, jumping now to above 50 million, Brave is also seeing 15.5 million active users daily, and its mobile browser has received more than 10 million downloads. So anyway, it's doing great. And I know, Leo, I've heard you talking about Brave. I don't know...

Leo: They were an advertiser for a while, yeah. The only thing I don't like about Brave is there's this kind of a tenuous attachment to crypto, its own crypto currency. And I always worry about that these days, you know. You saw probably Norton 360 now sells a cryptominer.

Steve: Oh, my lord.

Leo: You have a option not to, I guess. But still it's weird. And then Avira, which they bought, which is a free antivirus that was very popular, also is now doing that. So I get nervous when people get in the cryptosphere, like, I don't, you know, I don't know. I don't know. But I like Brave a lot. If you want to use - the reason I'm less bullish on Brave is everybody now makes a Chromium variant, you know, besides Edge. DuckDuckGo's will be Chromium when they do their browser. That's going to be one of interest. I think Brave is great because it's very privacy forward. I mean, I like Brave. I just don't - I still use Firefox, weirdly enough.

Steve: I do, too. It's where all my tabs are that I open when I'm pulling the show together.

Since so much of the web is actually just styled WordPress PHP, I thought it was worth noting that the WordPress core code itself has just received an update that might be important. Well, is, depending upon your configuration.

As we all know, by far WordPress's biggest security headaches arise because WordPress is, by design, a user-extensible platform, and those who are extending it don't necessarily need to be highly skilled in PHP coding or security before offering well-intentioned though horribly insecure add-ons, many of which go on to become highly popular before someone who is highly skilled in PHP and security finally gets around to taking a look at what everyone is using and then immediately sounds the alarm. Consequently, this podcast is routinely passing along the news that this or that highly used WordPress add-on needs to be updated with the result of professional oversight which it finally received.

But not today. Today, WordPress itself is in need of some tender loving care. Yesterday's just-released v5.8.3 is not ultra-critical, but is probably worth getting around to for anyone who's not using WordPress's automatic updating mechanisms, which for the couple years that I was using WordPress I certainly had turned on, being a believer in the need for that. And it saved me a couple times. The update to this 5.8.3 eliminates four vulnerabilities, three rated as highly important.

There's CVE-2022-21661, which is a high-severity SQL injection via WP_Query. It's exploitable via plugins and themes that use WP_Query. It can and should be applied to versions all the way back to 3.7.37, so that problem's been around for quite a while.

We've got the CVE ending in 21662, also high severity, also with a CVSS of 8.0, so not to be taken lightly. That one's a cross-site scripting vulnerability allowing authors, which is to say lower privilege users, to add a malicious backdoor - that's not good - or take over a site by abusing post slugs. The fix also covers WordPress versions all the way back to 3.7.37.

The next one is CVE 21664. That's the third high-severity flaw, although that CVSS is down to 7.4. Still ought to get your attention. It's the second SQL injection, this time via WP_Meta_Query core class. And doesn't go back quite as far, so it was introduced in 4.1.34.

And then lastly the 21663 is a medium severity, down with a CVSS of 6.6, an object injection issue that can only be exploited if an attacker has compromised the admin account. So not bad, but you get to fix that one when you're fixing all three high-severity problems.

So there have been no reports of any of these ever being seen exploited in the wild. But WordPress being PHP means that anyone who's inquisitive will be able to quickly determine what the flaws have long been because you just do a dif on the pre-release PHP and the post-release PHP, and you see what they changed. So they could be used in attacks by somebody so inclined. So again, while not super critical, definitely worth doing. And of course CVSSes with an 8 should not be left in place, if given a choice.

Okay, Leo. What exactly is a "Pluton"?

Leo: We asked this on Wednesday on Windows Weekly.

Steve: Yes.

Leo: What the hell is that?

Steve: There was some discussion. I had you guys running in the background while I was working on SpinRite. Pluton is Microsoft's wonderfully named CPU-integrated TPM technology. Now, the press is deeply confused, and you guys had every right to be confused, about what exactly Pluton is, thanks to Microsoft's own original horribly titled announcement of Pluton back on November 17th of 2020. The announcement's title was: "Meet the Microsoft Pluton processor, the security chip designed for the future of Windows PCs." That's great. But only after making the significant correction that what Pluton is, is specifically not a security chip. It's got nothing to do...

Leo: There's no chip.

Steve: ...with security chips.

Leo: Yeah, okay.

Steve: You know. And that's Pluton's entire point. Pluton - and yes, I do love saying the word - is Microsoft's silicon design for an on-chip, on-CPU, integrated TPM-equivalent technology. Microsoft designed the spec and the silicon and has arranged for Intel, AMD and Qualcomm to integrate this core technology directly into their cores. So the problem with any external physically separate Trusted Platform Module is specifically that it's external and physically separate. That means that its lines of communication between it and the system's processors is physically accessible on the motherboard's signal traces.

Now, everyone has always known that. Right? I mean, the problem is TPMs were never designed to protect against physical attacks. The idea was that a TPM would be a much better place to store encryption keys than either in firmware somewhere or on the disk, where they could be retrieved by malware. The TPM was designed as a secure enclave subprocessor where these secret keys could be used without them ever being disclosed. You'd give the TPM a hash of some blob that needed signing, and the TPM would use one of its internal private keys to encrypt the hash without ever exposing any of its keys to the outside world.

But other applications for which the TPM was also used were less secure and securable. For example, when the TPM is used to hold BitLocker keys, the unlocked BitLocker key must eventually become external to the TPM in order for the system to use it to decrypt the drive while it's in use. Of course, Microsoft makes the Xbox, and they've been annoyed through the years by having their precious Xbox's security subverted over and over and over by hobbyist owner hackers who, Leo, had the audacity to think that they actually had the right to mess with the hardware that they purchased and owned. Imagine that. Can't have any of that So those days are numbered. Pluton moves the TPM on-chip, and in doing so it elevates the security provided by the TPM to also include total protection from physical attack. There will be nothing to attack.

Basically what this means is that our next-generation processors from everybody - Intel, AMD, Qualcomm - they will just have that TPM stuff on chip, built in, not exposed. And so, for example, in the case of BitLocker, the BitLocker keys being used to decrypt the drive on the fly, they will never lose - they will never leave the silicon. They will never be exposed. So it definitely increases security. Unfortunately, it definitely decreases our ability to do things that we like to do with our Xboxes. So anyway, that's Pluton. It is, again, why Microsoft could announce it as a security chip when why they did it was so that it wouldn't be one is beyond me. But then, you know, Microsoft.

Leo: So it's what we thought, which is it's software.

Steve: No, it's not. It is silicon. It is firmware that they designed that they've convinced Intel, AMD, and Qualcomm to make some room for on their silicon dies moving forward. So it's a processor. It's like it's a Microsoft-designed security processor that instead of being external, thus exposed, is on the chip.

Leo: Okay.

Steve: And so you can't get to the communications between it and the other cores on the chip because they're all on the same chip. You'd have to literally pop the lid and get really NSA-ish.

Leo: Okay.

Steve: But it's not something that Johnny can do in the garage in order to hack his Xbox any longer.

Two quick notes. I am very much enjoying - I need to tell all of the Security Now! listeners who recommended this - the first of Dennis Taylor's three Bobiverse novels, and I'm pretty certain that I'm going to enjoy the other two as well. It's a trilogy. And I'll probably be wishing for more. Now, I do need to provide a caveat. They're an entirely different style of writing than either Ryk Brown or Peter Hamilton. Whereas both Ryk and Peter are consummate and detailed world builders who spend a great deal of time carefully crafting their characters and then placing them into a fully articulated alternate reality, Dennis by comparison just gets right to it.

When you pick up a Peter F. Hamilton novel, you have to be in the proper mood and in no hurry to reach the end because the end will not be in sight for quite some time, as you and I, Leo, have often commented. And similarly, each of Ryk Brown's Frontiers Saga story arcs spans 15 moderate size, I guess what I would call "chapter novels." So again, it's all about the journey with those guys. But Dennis Taylor's Bobiverse novels are fast and breezy. They're also a quick read. I just checked, and I was surprised to see that I was already 80% through the first one. And actually that's before I started waiting for MacBreak Weekly to end. Now I'm at 85%. So...

Leo: That's why we do long shows. There you go. Good for you.

Steve: It feels like, to me, like we just got started on this book, and it's almost over. On the other hand, Dennis wastes no time. I love Peter F. Hamilton's work. It's definitely right up there among the best science fiction I've ever found. But Peter would still be talking about the shape of the rivets on the hull and how the micro ablation created by the thin gases of deep interstellar space while moving at near light speed would tend to give them more of an oval shape and flatten them over time.

Leo: Yeah, no. Bob doesn't care about that.

Steve: No. Interesting factoid, okay, but not crucial to the plot.

Leo: Although you and I kind of like that stuff. Love that, yeah.

Steve: I do. I do. And in fact I just told my nephew, who is completely loving Ryk Brown's stuff, he's also in the middle of - I'm blanking on the book, the one that we love, "The Martian" guy.

Leo: "Salvation"? Oh, oh, oh, the new one, "Project Hail Mary," yes.

Steve: Yes. He's loving "Project Hail Mary."

Leo: Yeah.

Steve: So I told him about the Bobiverse.

Leo: Yeah. Andy Weir also likes ablated spheroids and things like that.

Steve: Yes, exactly. So but I did, I also told Evan that it was going to be crucial for him to eventually move to Hamilton. He's ever read any Peter F. Hamilton. He's definitely a sci-fi...

Leo: "Fallen Dragon." We all agree. Start with "Fallen Dragon."

Steve: Yes. And I said to them, I said start with "Fallen Dragon," and then "Pandora's Star," followed by "Judas Unchained." And boy, do you have a long - no hurry. We're not in any hurry. But I don't know, you're probably as visual as I am, Leo. I have those worlds in my head now.

Leo: Oh, yeah. That's why I like, honestly, I prefer books to TV and movie sci-fi because you can imagine something so much richer than they could ever put onscreen.

Steve: Well, and we've lamented the fact that there's just, I mean, except for "The Expanse," which I'm waiting for to get done so I can cruise through the final season.

Leo: You like to binge it. You don't want to watch it week by week. You're a binger.

Steve: Well, but we used to have, like, back in the day, Jean-Luc was flying around. Like for, what was it, seven years? There's no good sci-fi now. I don't, you know...

Leo: "Foundation" was terrible. "Invasion" was terrible.

Steve: Yeah.

Leo: "Wheel of Time," which is fantasy, not sci-fi, is worse than terrible. I'm so frustrated. So, yeah, yeah. I think I'm going to stick with the books. In fact, honestly, I still haven't seen "The Expanse" despite you and many other people recommending it. But I think I'm going to do the James S.A. Corey novels.

Steve: I did first, yes. Mark Thompson turned me on to them. He told me that "The Expanse" was in production. I read the novels first. It's always the case, oh my god, especially the first episode or two of "The Expanse." You're like...

Leo: What the hell?

Steve: What the heck is going on?

Leo: Stacey says, or was it Stacey? No, no, it was Amy Webb on Sunday said "Turn on the subtitles for sure" because that spacer lingo, it's hard to follow.

Steve: Yeah, yeah, yeah, yeah, yeah. The good news is I tried to get Lorrie into "Firefly," and she looked at it and had the same sort of feeling, like what? Except that we're now watching - she never watched "Castle." And I'm a Nathan Fillion fan.

Leo: I love him, too. And he's great in "Firefly."

Steve: I just think he is so good. And so...

Leo: "Firefly," though, was never a book; right? That was just a TV show, then later movie.

Steve: Correct. And Fox cancelled it. Like it was barely...

Leo: I know. They ruined it.

Steve: It didn't even reach adolescence, and they killed it in its crib.

Leo: But that's to me the exception because I loved that show. That was a great show.

Steve: Yeah. Well, and what's so fun is that she's now fallen in love with Nathan Fillion.

Leo: Yeah. He's great.

Steve: So after we get through with "Castle," that went for like eight years and is just, well, it's good writing, then I'm going to be able - she will then be able to do "Firefly" and then of course "Serenity," the movie that was made from it because fans just demanded more.

Leo: Yeah. Wasn't it crowd - I think the movie "Serenity" was crowd-funded at first. And then they finally, they said, all right, we'll make it. All right. I think that they raised money to...

Steve: Although we might be confusing it with "The Expanse" because remember what was crowd-funded was flying an airplane around the Amazon headquarters in order for Bezos to finally say, okay, okay, fine.

Leo: Okay. We'll green-light it.

Steve: And he really liked the series, too. So he decided to pick it up. And I have to say the last, the ones that Amazon did had a higher level of production.

Leo: Oh, okay. See, I haven't even gotten that far. I've only watched the first five or six episodes of the first season.

Steve: It really is good.

Leo: All right. I'm going to do the books, and then I'll come back, yeah.

Steve: For what it's worth, on the Bobiverse trilogy, 88% of the reviewers on Amazon gave it five stars. The other 12% gave it four. Nobody gave it fewer than four. They're available through Kindle Unlimited. So if you're a reader, and you're a Kindle Unlimited person, they won't cost you anything. And again, it's a different kind of style, but sometimes it's just fun to get on with it. I mean, I've noticed that television shows where a lot happens in an hour, you feel like you got something for your time, instead of like stretching it out for like no good reason.

Leo: Yeah, yeah, yeah.

Steve: Anyway, I am continuing to move forward nicely with SpinRite. As I have been, I'm focusing upon all of the outlier machines our various testers have managed to throw at it. And since I'm always working to find a generally applicable generic solution, rather than doing any special casing, so far I have no special casing code in SpinRite, it's getting generally more robust with each success because it'll be able to take things in stride that it's never seen before, which are also weirdos.

Leo: I think it's time to talk about the subject of the day, URL Parsing.

Steve: Indeed. And I just sent you a text with the link to the SodaStream refill adapter.

Leo: We were talking before the show about why never, never to buy SodaStream cartridges. They're overpriced. Steve has of course a Rube Goldberg invention for making it possible for all of us.

Steve: $17 for the little brass gidget that allows you to refill your own SodaStream cartridges. So definitely a win.

Okay. URL Parsing Vulnerabilities. As I mentioned at the top of the show, this week's topic centers around the work of four security researchers, two from SYNK spelled - and I guess all the normal spellings are gone, so SYNK spells their name S-Y-N-K. And two from Claroty spelled C-L-A-R-O-T-Y.

Leo: What the what? Okay.

Steve: I know. They're both in the Northeast U.S., as I mentioned. They decided to take a closer look at another set of wildly popular and widely used libraries. Naturally, I mean, URL parsing, come on. Like everything needs to do that. Which as a consequence of that would inherently have broad exposure to external attack. The title of this week's podcast, URL Parsing Vulnerabilities, discloses their wisely chosen, in retrospect, research target and suggests what they did indeed find. I have a link in the show notes to their 15-page PDF for anyone who wants to dig in deeper than I do, although we're going to be digging in plenty deep.

What they said was, to set the stage: "The Uniform Resource Locator (URL) is integral to our lives online because we use it for surfing the web, accessing files, and joining video chats. If you click on a URL or type it into a browser, you're requesting a resource hosted somewhere online. As a result, some devices such as our browsers, applications, and servers must receive our URL, parse it into its Uniform Resource Identifier (URI) components, for example the hostname, the path, and so forth, and fetch the requested resource.

"The syntax of URLs is complex, and although different libraries can parse them accurately, it is plausible for the same URL to be parsed differently by different libraries. The confusion in URL parsing can cause unexpected behavior in the software, like a web application, and could be exploited by threat actors to cause denial-of-service conditions, information leaks, or possibly conduct remote code execution attacks.

"In Team82's joint research with SYNK, we examined," they wrote, "16 URL parsing libraries, written in a variety of programming languages, and noticed some inconsistencies" - which is putting it mildly, as we'll see - "with how each chooses to parse a given URL into its basic components. We categorized the types of inconsistencies into five categories and searched for problematic code flows in web applications and open source libraries that exposed, indeed, a number of vulnerabilities.

"We learned that most of the eight vulnerabilities we found" - and by the way, they all CVEs assigned, I mean, these are real problems - "largely occurred for two reasons. First, multiple parsers in use." They said: "Whether by design or an oversight, developers sometimes use more than one URL parsing library in projects. Because some libraries may parse the same URL differently, vulnerabilities can be introduced into code. The second, specification incompatibility." They said: "Different parsing libraries are written according to different RFCs or URL specifications, which creates inconsistencies by design. This also leads to vulnerabilities because developers may not be familiar with the

differences between URL specifications and their implications, for example, what should be checked or sanitized."

I thought that the first case, the probably inadvertent use of different parsers, was really interesting, where developers might make the reasonable, but ultimately incorrect assumption that different areas of their code would decompose input URLs the same way. And I just, I invented a typical instance of how that could go wrong. It's not one that we'll be talking about in a second. But for example, imagine that upon decomposing a URL into its various pieces, those pieces were sized and storage was allocated to fit, but the code wasn't yet ready to fill those buffers.

Then later, when the code was ready, the same URL was again parsed with the URL's component data finally being copied into the previously allocated storage. The programmer could assume that since the same URL was being parsed both initially and later, the component pieces would naturally be the same size. But if different URL parser libraries were used, and they interpreted the URL's format in slightly different ways, the allocation might not fit its data, and a buffer overrun might occur, for example.

Their second issue, which they termed "specification incompatibility," is a variation of a fundamental programming challenge that I've spoken of from time to time and also recently. My term for it is "weak definitions." If a coder is not absolutely certain what something is, for example a variable represented by a name, that coder, or god help us, some future other coder might come along and use that variable differently because its purpose wasn't made absolutely clear and obvious by its name. You misname something, you're going to come back later and forget that you, like, while you were coding, you ended up using it differently than you originally thought you were going to because of the name you gave it, which is no longer correct. And I find myself often going back and fixing names of things when I go, ooh, right, that's not really what it's doing any longer. I'd better fix that now.

So in fact we have another example of the URL parsing duality dilemma right in our own backyard with Log4j. During our final podcast of 2021, we talked about how the Log4j trouble was that a URL of the form jndi:, and then using ldap as the scheme, so ldap:// and then, for example, evilhost.com/a was being parsed from within a log message. Log4j, upon seeing that URL, would dutifully run out onto the Internet to fetch whatever from wherever. So, easy to solve this problem; right? Just create an access whitelist of allowable hosts for that JNDI URL lookup and default the whitelist to only containing a single entry for localhost, which was probably the only valid source for JNDI material to come from anyway; right? Problem solved. Nope.

Leo: No?

Steve: Nope. Not long, and that was the first fix that was offered to this problem. After that first fix was added, a bypass was found, and it was even awarded a CVE number, 45046, which once again allowed remote JNDI lookup and allowed the vulnerability to be exploited in order to achieve remote code execution. So how was the clean and simple valid host targets whitelist bypassed?

Leo: Yeah, how?

Steve: The new hack to bypass the whitelist used a URL of the following form. And this actually worked. It used ldap://127.0.0.1.

Leo: Localhost, yeah.

Steve: Localhost. #.evilhost.com:1389/a. Believe it or not, tucked inside the Log4j code are two different URL parsers. I kid you not. One parser is only used for validating the URL, whereas another is used for fetching it. In order to validate that the URL's host was allowed, Java's built-in URI class is used. Java's built-in URI class parsed the URL, extracted the URL's host (127.0.0.1) and checked if the host is inside the whitelisted set of allowed hosts. And indeed, if we parse the URL...

Leo: Yeah, yeah, fine.

Steve: Yeah. No, no problem.

Leo: Now you take it from here.

Steve: Right. However, it was discovered that when the JNDI lookup process actually goes to fetch this URL, it does not fetch it from 127.0.0.1. Instead, it parses URLs differently. It makes a request to 127.0.0.1#.evilhost.com in other words, the subdomain of evilhost.com. Again, 127.0.0.1#.evilhost.com, which is exactly what the bad guys figured out they could do. So after being initially shut down by the update, which added the whitelist, because after all we wouldn't want to simply remove some dangerous and completely unneeded functionality from Log4j, oh, heavens no.

Leo: Oh, heaven forfend, yeah.

Steve: Uh-huh. The bad guys simply tweaked their evilhost.com server to reply to subdomain queries, and they were back up and hacking. And as we know, that dangerous and completely unneeded functionality was finally disabled by default after, what, the fifth try at fixing the Log4j vulnerability.

Okay. So what's relevant for us today is that this just actually happened in the very real world, that is, this Log4j thing. We just saw a URL parsing being done differently in two different places, actually causing a real-world problem. Okay. So what did this new team turn up when they really dug into this?

They said: "Throughout our research, we examined 16 URL parsing libraries including: urllib, a Python library; urllib3, also Python; rfc3986, which is the name of a Python RFC, a URL parsing library; httptools in Python; curl lib in cURL; Wget; the Chrome browser itself; Uri, which is a .NET library; URL in caps, which is a Java library; URI, a Java library; parse_url, which is a PHP library; lowercase url, which is a NodeJS library; url-parse, also NodeJS, net/URL, which is written in Go; uri lowercase in Ruby and URI uppercase for Perl."

So, yeah. As I said, URLs are everywhere, and there are 16 different parsing libraries. No two of them do the same thing, despite the fact that they're all trying to do the same thing.

They said: "We found five categories of inconsistencies: scheme confusion; slashes confusion; backslash confusion, which is different from slashes confusion; URL encoded data confusion; and scheme mix-up." They said: "We were able to translate these

inconsistencies into five classes of vulnerabilities: server-side request forgery, cross-site scripting, open redirect, filter bypass, and denial of service. In some cases, these vulnerabilities could be exploited further to achieve a greater impact, including remote code execution." And they finished: "Eventually, based on our research and the code patterns we searched, we discovered eight vulnerabilities in existing web applications and third-party libraries written in different languages used by many popular web applications."

Okay. All eight of those have been assigned, as I said, CVEs because, as we'll see, they're really not good. After digging through all of those libraries and the applications that use them, they found, as I mentioned earlier, five different situations where most, as they put it, of the URL parsers behave unexpectedly. And that's the scheme confusion, slash confusion, backslash confusion, URL encoded data confusion, and scheme mix-up.

When we're not sure what we want or where we're going, it's often difficult to create a specification for that mission, which is fuzzy, beforehand. And probably nowhere has this proven to be more true than for the exacting definition of the format of the URL. The trail of obsoleted and abandoned URL RFCs, the formal specifications of things, speaks volumes. The original URL RFC was 1738, and it was updated by 1806, 2368, 2396, 3986, 6196, 6270, and 8089. And along the way it was obsoleted by 4248 and 4266. Then you have the RFC for the more generic URI. It also updates the original 1738, obsoletes 2732, 2396, and 1808, and is then itself updated by 6874, 7320, and 8820. So imagine being a coder who's trying to decide how to handle every possible curve that somebody might either accidentally or maliciously toss down the throat of your URL interpreter.

And as if all that weren't enough, there's also the WHATWG, which is W-H-A-T-W-G, and their TWUS, which is T-W-U-S. WHATWG, that is, W-H-A-T-W-G, is the Web Hypertext Application Technology Working Group (WHATWG), a community founded by well-meaning individuals, I'm sure, from leading and technology companies, who have tried to create an updated, true-to-form URL specification and URL parsing primitive because the final RFC, eh, we're not - we don't know.

This resulted in the TWUS, the WHATWG URL Specification (TWUS). While it's not very different from the most up-to-date URL RFC, which we left off at 3986, minor differences do exist. For example, while the RFC differentiates between backslashes and forward slashes, where forward slashes are treated as a delimiter inside a path component, and backslashes are a character with no reserved purpose, WHATWG's specification states that backslashes should be converted to slashes, and then be treated the same as. WHATWG's rationale is that this is what most web browsers do. Browsers, it turns out, treat forward and backslashes identically. And we'll see later that they do this on purpose.

So WHATWG feels that what they regard as their so-called "living URL standard" should correspond with common practice, rather than being some stodgy old numbered document that isn't what people are doing anyway. But this "living URL standard" broke compatibility with some existing standards, as we know, and with the contemporary URL parsing libraries that followed. It turns out these interoperability issues remain one of the primary reasons why many maintainers of some parsing libraries have just said, okay, wait. We're sticking with the RFC, that being 3986. Even though they don't do that correctly either, at least their intention was in the right place.

So this brings us to scheme confusion, a confusion involving URLs with missing or malformed schemes. Of this the team wrote: "We noticed how almost any URL parser is confused when the scheme component is omitted. That's because RFC 3986 clearly determines that the only mandatory part of the URL is the scheme" - you have to have

that - "whereas previous RFC releases" - that is to say RFC 2396 and earlier - "don't specify it. Therefore, when it's not present, most parsers get confused."

And the real world behavior they show - this is me talking - is surprising and worrisome. They asked, these guys, the researchers, asked five different popular Python libraries to parse the schemeless input just google.com/abc. Most of the parsers, when given the input google.com/abc, state that the host is empty, right, the host being google.com. Most of the parsers say, eh, you've got no host, while the path they have is google.com/abc, which is wrong, obviously. However, urllib3 correctly states that the host is Google and the path is /abc, while httptools complains that the supplied URL is invalid. Okay, because it's really adhering to the spec; right?

And if you don't have a scheme, RFC 3986 says you're not going anywhere. When supplied with a schemeless URL, almost no URL parser parses the URL correctly because the URL does not follow the RFC specs. But most don't complain, they just guess. And most of them guess differently. cURL's parser dealt with the missing scheme by providing its own, guessing at what was not provided. It got the right result, but should it have guessed? Could there be an abusable downside to such guessing, one might ask?

In their report they then provide a sample bit of Python code to show a real-world example of how a security bypass might occur as a direct result of these parsing differences. And when I first encountered that, I thought, okay, they've constructed an example, synthetic, of how this would happen. They use URL split, which is a function imported from the urllib.parse library in order to do the splitting, and then later they use, where is it, netloc in the parsed URL library in order to perform the fetch. And exactly the problem that we were talking about occurs because in this block of code two different URL parsers are being used.

I learned later that this is actual Python code in a highly used library which is handling these malformed URLs exactly in this wrong way. And it is used everywhere. That's unbelievable. PoolManager is the function in urllib3 which is invoked, and that's like what everybody uses to pull queries across the web.

Okay. So there's that. Another oddity they found was in the handling of an irregular number of slashes. Oh, you're going to love this one, Leo. They called it "slash confusion." Now, this is different than which way the slash is leaning.

Leo: Oh, okay. All right.

Steve: But we get to that next. That's the backslash confusion. Slash confusion. I know, it's unbelievable. How does any of this actually work? The controlling RFC 3986 states that a URL authority - okay, now, the authority is the technical term for the domain or the host, what we all call the domain name or the host in the URL. The actual RFC calls it the "authority." It states that the authority should start after the scheme, separated by a colon and two forward slashes.

Leo: Yeah.

Steve: Right? Http://. How many times have I said that on the podcast? It should persist, that is, the parsing of the authority should persist until either the parser reaches the end of the line or a delimiter is read, these delimiters being either a slash signaling the start of a path component, a question mark signaling the start of a query, or a hashtag signaling the start of a fragment.

So they played around with URLs beginning with a scheme, a colon, and three or more slashes followed by a domain name and path. Apparently, the pattern matching that were being used found the :// and thought, ah hah, here comes the domain name. And when they immediately hit the third slash they thought, ah, and there's the end of the domain name.

Leo: Oh, my god.

Steve: Yes. Yeah.

Leo: So is it always - are they using regular expressions to do this?

Steve: Yeah, yeah. And in fact if you scroll down in the show notes to the top of page 13, you'll see the regex which is doing this.

Leo: Regex is notoriously difficult and easy to screw up. And this is...

Steve: I know. You have to, I mean, it is, yes, it is super powerful and so easy to have side effects that you don't anticipate.

Leo: Because the Advent of Code uses regex a lot. I've been writing a lot of regex lately. I love regex, and there's a - Jeffrey Friedl's book on "Mastering Regular Expressions" is one of my favorite coding books of all time.

Steve: And boy, Leo, can you get yourself some, I mean, it like Forth is a write-only language.

Leo: Yeah, yeah. Sometimes they call it the problem with the toothpicks or, yeah. And it's just it gets crazy with all the escaping in the back because that's part of the problem is backslash is often used to escape.

Steve: Yup.

Leo: And so sometimes you'll have \\\ to escape a backslash. It gets kind of nutty.

Steve: Yeah, in fact you can see in red...

Leo: There are more modern parsing libraries than regex.

Steve: Oh, Leo, nobody should be using it. You ought to use a careful algorithmic parser in order to take - because, I mean, it is a...

Leo: Like Parsec or something, yeah.

Steve: It is a forward-moving flow with a well-defined structure, you know, follow the proper RFC.

Leo: The stream, yeah.

Steve: Yes, exactly, and treat it that way. And, well, nobody does. Sixteen URL parsers, let's write another one.

Leo: It's the age of the code, too. I mean, oh, god.

Steve: Yeah, I know.

Leo: And it's also I think this is in some ways an open source problem because...

Steve: Yeah, because you don't have to be an expert in order to throw out another URL parser and say, here, Jimmy's URL Parser.

Leo: Yeah. And people, this is also a big problem is people just use libraries and assume they're correct. You should definitely not mix libraries. That's really funny. That's really strange.

Steve: Yup. Okay. So we have the problem that three slashes in a row causes a null host and the authority then to be moved to the beginning of the path. That actually happens. Then we have the backslash confusion, a confusion involving URLs containing a backslash. RFC 3986, the controlling document, clearly specifies that a backslash is an entirely different character from a forward slash and should not be interpreted as one.

Leo: No.

Steve: This means the URL https://google.com and https:\\google.com are different.

Leo: Yes.

Steve: And should be parsed differently.

Leo: Yes.

Steve: And being true to the RFC, most URL parsers do not treat a slash and a backslash interchangeably.

Leo: Well, that's a relief.

Steve: But our web browsers do.

Leo: Oh, no.

Steve: Yes. Every web browser, when a URL having backslashes or even a mixture of backward and forward slashes is used in a URL, Chrome and its brethren are completely happy, treating either as forward slashes.

Leo: So they're assuming maybe that users don't know the difference?

Steve: I think that's it. We might think that this wacky behavior occurs because most browsers follow that WHATWG URL specification which states that backslashes should be treated the same as forward slashes. But it may be more accurate to say that the TWUS spec follows the browsers rather than the other way around.

Leo: Yeah, because people, for years, in fact I just saw that TV ad the other day where they said, instead of "slash," they said "backslash." And just, and nobody caught it. They just read it that way. And I think it's because, well, backslashing was created by, I mean, people got used to it because of Windows' horrific use of it.

Steve: Oh, my god, and hasn't that been a sin. Isn't that a sin that has hurt us?

Leo: Should revisit it.

Steve: Whenever I'm spending a lot of time in Unix, and then I come back to Windows, I'm like, okay, wait a minute.

Leo: Backslashes, yeah.

Steve: Which one do I use?

Leo: Backslashes, oh, god.

Steve: Anyway, this code at the top of page 13 shows what the guy did on the second line where he's parsing, he's using a regex to parse out the authority. He's matching on the - after finding the ://, he then sets up a group in order to grab the authority, and he's pulling together, and you can see it, marked in red, the top of page 13, I think it's a little lower than where you are, or maybe it's above where you are.

Leo: It's above, yeah.

Steve: Yeah.

Leo: All this backslashing is making me nuts.

Steve: You can see two, yeah, top of page 13.

Leo: That's this.

Steve: Really? Oh, no, I'm sorry. You're on their PDF. I'm on my show notes.

Leo: Oh, that's why. There you go.

Steve: Sorry.

Leo: Okay. I'll find it. Go ahead.

Steve: Yeah. Anyway, they marked them in red, two backslashes in a row, which of course is an escape for a backslash, meaning I actually mean one backslash. But it's grouped in there along with a forward slash as being a valid terminator for the authority, that is, for the domain name. Okay. So this means that if an authority contains a backslash, urllib3 would split it out at that point and use only what appears before the backslash as the authority, then concatenating what follows the backslash to the front of the URL's path.

Anyway, the bottom line is it's exactly what we were talking about, this problem that occurred in Log4j. So, for example, and here's where our listeners need to visualize, if you had http://evil.com\@google.com/, if the latest RFC is obeyed, as most parsers do, which specifies that backslash has no special meaning inside a URL authority, the malformed evil.com URL would be parsed as evil.com\@. Now, remember the old and now deprecated URL-embedded username and password syntax. Remember, you used to actually be able to put a username and password in the URL?

Leo: Another bad idea.

Steve: Oh, my god. Really bad idea.

Leo: Oh, my god.

Steve: Oh. What were they thinking?

Leo: Oh, geez.

Steve: Anyway, they first, you know, started saying we really don't think that's a good idea. Then they have formally now said no, that's no longer going to be considered legal. It should never be done. But it's still tolerated because of old URLs out there.

Leo: Right, right.

Steve: So that means that everything in front of the @ sign is considered "userinfo," as it's termed. So that means that something that doesn't treat backslash specially inside the authority, as the RFC says you shouldn't, will end up parsing that with evil.com\@ as userinfo, and so the actual domain will start and be seen as google.com.

Okay. The researchers tested this and found that, yes indeed, this is what most of the RFC-compliant parsers do.

Leo: So this is harmless because it's ignoring evil.com.

Steve: Correct. They do not treat the aberrant backslash as the end of the authority. Since it precedes the @ sign, they treat it as part of the userinfo. But not urllib3, which is heavily used. Urllib3, as we saw in that regular expression, it's right above - it's at the top of page 13, yeah, if you scroll up you'll see it, the second line in that chart, the two red backslashes, that's in there as the terminator for the authority, that is, it and a forward slash, either of those. So they did it on purpose.

Leo: Oh, wow.

Steve: Yup.

Leo: You could see how easy this would be, though, if you look at this regular expression, to make a mistake.

Steve: Yeah. I mean, yeah. And so, okay, it's in there. Its result will not be the same. And here it is, Leo. Since the "requests" Python module, that's where that earlier snippet of code came from, a heavily used Python module "requests" uses urllib3 as its primary parser while still using urllib's urlparse...

Leo: Don't mix them.

Steve: ...and urlsplit for other purposes, yes, it mixes them. So we could easily run afoul of the differential parsing scenario we described earlier. And as I said, I thought when I first saw it, it was a synthetic example. No. It's right out of the Python code, where it comes up with different answers.

Okay. So we've established that a collection of five what we might call "exploitation primitives" exists. It should be clear that when parsing the same data, all URL parsers should be deterministic and should return identical results. When that is not done, the parsers introduce a dangerous uncertainty into the process to create a wide range of potentially exploitable vulnerabilities.

They gave us a bunch. I'll just describe one. It's a very popular product or package called Clearance for Ruby. As a direct consequence of these parsing mistakes, the researchers discovered a dangerous and exploitable Open Redirect vulnerability in the Ruby gem package known as Clearance, and it was assigned a CVE of 23435. Actually 2021-23435. Open redirect vulnerabilities enable powerful phishing and man-in-the-middle attacks by secretly redirecting a user's browser to fraudulent sites. The vulnerabilities occur when a web application accepts a user-controlled input that specifies a URL that the user will be redirected to after a certain action such as a successful login or logout occurs.

In order to protect users from an open redirect attack, web servers carefully validate the given URL and allow only URLs that belong to the same site or to a list of trusted redirection targets to be used. So you can probably see where this is going. Right? If you can't trust the thing that's parsing your verification, we just saw where a whitelist could be broken, then you're in trouble.

To make a long story short, what this thing allows is a bad guy to use these parsing vulnerabilities that are present in the Clearance Ruby gem package to perform successful man-in-the-middle attacks to hijack people's sessions and bring them back after logging into the bad site, even though the victim site where the user logged in has explicit code to prevent that from happening because the danger is well understood. So this just ends up being a huge problem.

In the case of this Clearance package, as I said, it will redirect a user to a logged-on page. In the show notes at the top of page 16 I show their example, which is www dot - actually it's my example - brokensite.com/////evil.com. When that ends up getting parsed by the broken parser which the Clearance package is using, it ends up giving the browser ///evil.com.

So what's a bit surprising is that if the user were using Chrome, Chrome would receive the URL ///evil.com, which hardly seems valid and which we might expect Chrome to balk at. But web browsers, as we were getting to talking about, have evolved to be hyper lenient with what they will accept. After all, users are entering cryptic-looking URLs and aren't perfect. So Chrome's lenient behavior is they consider it a feature, not a bug.

And in fact in the Chromium source, in a big comment block, it says: "The syntax rules of the two slashes that precede the host in a URL are surprisingly complex. They are not required, even if a scheme is included [and then they say] (http:example.com) [no slashes] is treated as valid," which I never knew. I'm not going to bother putting slashes anymore. Chrome ignores them. And it says: "...and are valid even if a scheme is not included." Then they say: "(//example.com is treated as file:///example.com)." He says: "They can even be backslashes." What? Yes, "(http:\\example.com and http [no colon] http\/example.com" - this is Chrome's own notes, this is in a comment block - "are both valid) and there can be any number of them" - it actually says that - "(http:/example.com and http://////example.com are all valid)."

Leo: And this is because users are going to type that. And so we just want to...

Steve: Yes. Yeah, exactly.

Leo: And you can't really blame them for the Log4j problem.

Steve: No.

Leo: No. That's Log4j's problem in parsing its input.

Steve: Correct. Correct.

Leo: Because Chrome's a browser, and it's just trying to do what it thinks you want.

Steve: Yes. Exactly. It's going to try to make sense of what gibberish you have just typed into the URL and go, okay, let's guess here where they want to go.

Leo: Yeah.

Steve: So suffice to say that with all this, we have another example of longstanding, deeply embedded, critically broken code which despite now being identified, will remain in use until all references to it are eventually, if ever, weeded out.

We also have another example, which I did appreciate the fact that the researchers were able to see what was going on, of the power of open source software. The researchers were able to peer into the code to see and understand how it operated and what it does. The fact they didn't like what they found, well, they can fix it, or find the guy who can fix it. The flipside, of course, is that malicious actors have access also.

But overall, "open" seems clearly better, since when well-meaning researchers are able to offer feedback to make bad code better, that happens; whereas bad guys need to operate with whatever they can find, unless they worm their way into being trusted on a project. That's the one case. Otherwise they don't have the ability to make good code bad in the same way that researchers can help to make bad code good. So thus the argument for letting lots of people look at it, and it's just going to kind of float to the surface and be good all on its own. But anyway, URL parsing vulnerabilities exist, and they're all around.

Leo: It's amazing. Just amazing.

Steve: Yeah.

Leo: Wow. This is the root cause of the Log4j problem is just this inability to recognize malformed URLs.

Steve: No, actually, this intersected log - the root cause of Log4j is that you insanely processed a URL in a log output. It's like, what?

Leo: You're right, that's a mistake by itself, yeah, yeah.

Steve: And so what happened was the first fix was to do a whitelist on the domain.

Leo: And then that didn't work, yeah, because of this problem.

Steve: And that didn't work because of the parsing, the URL parsing vulnerability.

Leo: Wow, wow. I bet it's turtles all the way down. I mean, bet, honestly, you solve that, and then you're going to find another one, another one, another one, another one.

Steve: Yup. Yup. As we've said, security is porous. The harder you look, the more you find.

Leo: Yeah. That's amazing. What an interesting topic. I hope everybody subscribes to this show so they can hear this. Obviously you're listening. Thank you. GRC.com is the place to go for Steve's website. That's where SpinRite, the world's best mass storage recovery and maintenance utility, lives. 6.0 is the current version; 6.1 is imminent. If you buy 6.0 now, you'll get 6.1 for free. You can also participate in the development of 6.1.

Steve also has this show there, the 16Kb audio version as well as the 64Kb audio. He's got the show notes, as we mentioned. He also has transcripts. A couple of days after the show we'll have transcripts written by an actual human, so you can actually follow along or search for the topic you're interested in, et cetera, et cetera. This is a show I think people will share with one another just because just that last part alone is really, really interesting.

We have 64Kb audio and video at our site, TWiT.tv/sn. You can always download any show there. There are 853 to choose from. This is the 853rd. There's a YouTube channel with all the shows, dedicated to Security Now!. And of course subscribing is probably the best thing to do, get it automatically the minute it's available. And if your podcast player supports reviews, please tell the world how great Security Now! is. Leave us a five-star review so everybody can discover this show because they don't want to miss this. This is good stuff, Steve.

If you want to watch us do it live, it's a little tricky because MacBreak Weekly is an accordion and can expand or contract completely at random. But nominally we do this show at 1:30 p.m. Pacific, 4:30 Eastern of a Tuesday afternoon. That's 21:30 UTC. And the livestreams, audio and video, are at live.twit.tv.

My suggestion is just have it on all day Tuesday. Start in the morning at 9:00 a.m. Pacific with iOS Today, then MacBreak Weekly at 11:00, Security Now! whenever we get around to it, and All About Android after this. So, you know, it's a big - Tuesday's our biggest day, I think. Lots of good content, all day long. So just have TWiT on live all the time. That's my advice to you. All right, Steve. I'm off to read "The Expanse."

Steve: And to recharge your SodaStream cartridges.

Leo: Oh, yeah, yeah. I've got some questions about that. We'll talk about that off the air. Thank you, Steve. Have a great week.

Steve: Okay, buddy. Ciao.