Inside the NetUSB Hack

Description: This week we briefly touch on the ongoing Log4j background noise. We look at the result of the insurance industry's pushback against ransomware coverage and at the resulting changing cyber-insurance landscape. We look at another WordPress add-on problem and a supply-chain attack on a very popular add-on provider. We also wonder whether WordPress still makes sense in 2022. We cover the EU's quite welcome major bug bounty funding, and Kaspersky's discovery of a very difficult to root out UEFI bootkit. We'll share some interesting questions and topics suggested by our listeners. Then we're going to take another of our recent technical deep dives to examine the precise cause of that pervasive NetUSB flaw. It's really fun and completely understandable!

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-855.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-855-lq.mp3

SHOW TEASE: It's time for Security Now!. Steve Gibson is here. Lots to talk about. We're going to briefly touch on Log4j and the background noise. No big severe exploits that we know of yet. But I guess that's only a matter of time. A look at the insurance industry's pushback against ransomware coverage. The EU's major bug bounty funding for a lot of open source projects. And then Steve's going to look at a NetUSB flaw that involved a fairly simple programming error. You might want to listen so you don't make the same mistake. It's all coming up next on Security Now!.

Leo Laporte: This is Security Now! with Steve Gibson, Episode 855, recorded Tuesday, January 25th, 2022: Inside the NetUSB Hack.

Yes, once again it's time for Security Now!. You've been waiting all week for this guy right here, Steve Gibson, GRC.com. Hello, Steve.

Steve Gibson: Hey, buddy. Great to be with you for the last podcast of January. Where did the month go? Next one will be February 1st. We're going to have one of those earliest possible second Tuesdays of the month, two weeks from today. It'll be the 8th for another Patch Tuesday adventure.

Leo: They come so fast, don't they.

Steve: Oh, my. I know.

Leo: Seems like we just ate one.

Steve: They just keep giving so much joy to the world. So this is Security Now! Episode 855 for the 25th of January titled "Inside the NetUSB Hack." And you asked me before we began recording, is this the first time we've talked about NetUSB? And it's funny that you should ask because in my doing some additional digging, I realized there had been a previous event with exactly the same kernel driver in Linux modems, and we titled a podcast, I think it was #508, NetUSB. So indeed, this has been an ongoing problem. I mean, it's been quiet for a long time. But, okay.

The details of what the coder did is so interesting. And it's not super crazy, like no one's going to be able to understand this. And I just thought, you know, when I dug into it a little bit more, I thought, okay, I just have to share this with our listeners because it's a perfect example of how a mistake can get made, that you can look at the code, everything looks fine, the structure, the design, perfect. But due to a little side effect which, you know, I mean, there were some things they could have done that would have prevented this from happening. But it's just it's textbook classic mistake. And so I thought, okay, that's just perfect for the podcast.

Leo: Good, good, good.

Steve: So we're going to get to that. But first we're going to briefly touch on the ongoing, I would call it Log4j background noise at this point. Nothing cataclysmic happened in the last week, but there's little percolatings. We're also going to look at the result of the insurance industry's pushback against ransomware, their insurance coverage of ransomware. It's what we've been expecting. And also the resulting sort of changing cyber-insurance landscape that is emerging as a result. We're going to look at another WordPress add-on problem and a supply chain attack on a very popular add-on WordPress provider. We're also wondering, as a result of this, whether WordPress still makes sense in the current day and age, in 2022.

Leo: I am going to question you on that because these add-ons aren't official WordPress add-ons. They're third-party add-ons. Aren't they?

Steve: Correct. But WordPress is built to support an add-on architecture. The idea is that they deliberately, the WordPress people deliberately keep it rather feature sparse because the whole idea is to encourage an add-on ecosystem. And I think that's the crux of the problem. It was '03 it began, 19 years ago. And it may have made sense back then. I don't think that that model works in today's world. Anyway, we'll talk about that. We've also got the EU's quite welcome major bug bounty funding that they just announced last week. Kaspersky's discovery of a very difficult to, well, root out UEFI, you'd really call it a bootkit rather than rootkit. They found something that's interesting.

Also we've got, as I look through my recent Twitter transactions, there were some interesting questions and some topics suggested by our listeners. And then we're going to have a really interesting and really, if people pay attention, everybody will understand this and kind of have this cool like aha moment where we really take a classic look inside a hack and that something that the designer of the NetUSB code missed, and it's probably been in there forever.

Leo: Fun. Can't wait. Always enjoy that. Know what not to do.

Steve: Yeah.

Leo: Great. We will get to the heart of the subject. In fact, our Picture of the Week, which is a fun one.

Steve: So our Picture of the Week should strike fear in the heart of kids who have probably done this; and, on another level, coders who have done this. What we're looking at is a large container with a divider separating into two halves. On one half we have it filled with Mentos. And on the other half it's filled with Coke. Coca-Cola. And so then there's a hand reaching down from off-screen, getting ready to lift and remove this divider which is currently, thankfully, separating the Mentos from the Coca-Cola. Because we know what happens when you pull that divider out, and the Coca-Cola is able to mix with the Mentos. You get an explosion. And so this picture is titled "When you're gonna merge two branches after a long time."

Leo: Oh, it's a git joke. I get it.

Steve: It's a git joke, exactly. For those who have not messed with source-level control, one of the things that can happen is with source code, a project with multiple files and such, is you can fork the source into a separate branch. And then independently work on various aspects of the code. But there are times when you may want to merge the different changes back into a single branch, essentially, back into a single set of files.

Well, the software to do this has been maturing over time, and it basically does what you'd expect it would do. It looks at, given two different files, it looks at both files and, like, follows them along and waits until it sees that, wait, one of the files has changed in a certain way compared to the other. And so it notices there's been a change, and then it continues looking down until it sort of like finds where they resynchronize again. And so it's able to identify the location where that, like, which region of the source code is different from the other. And then all things being correct...

Leo: Just push the button, and bazinga.

Steve: You hold your breath, and you smash them back together again.

Leo: The beauty of git is you can always get back to the previous setup, thank god.

Steve: Exactly. You're able to roll back in time. But anyway, but the idea is, if there were not many changes between the files being merged, your chances of creating a sane recombination are pretty good. And this says when you are going to merge two branches after a long time.

Leo: There are big differences, yeah.

Steve: Meaning lots of differences have occurred. It's like, I don't know if we're going to - oh, especially if you get conflicting differences of the same region between the two pieces. It's like, okay, now what are you going to do?

Leo: A lot of text on the screen, let's put it that way. A veritable fountain of Mentos. You're kind of new to source code version control. This is, since you set up that GitLab, how are you liking it?

Steve: Well, I'm not using it for that at all.

Leo: Oh. Oh. Just for bug reporting.

Steve: It's just for issue tracking, yes.

Leo: Yeah, yeah.

Steve: Yeah. So I have long used something called FileBack PC, which is no longer being produced. It allows...

Leo: That means it's good, of course. I mean...

Steve: Yeah, because it still works and, I mean, it's back almost from the DOS days. It's got a very sophisticated set of, like, incremental backup management where, as I'm just working on my source code, not thinking about it, across multiple files, it's watching me do that and taking snapshots of my work periodically as things change. And, I mean, and it's been super handy many times. Now what I'm doing is, whenever I'm releasing a public build of SpinRite, I have just a batch file that zips all of my source into an archive and sort of, you know, like saves it at that point. And that way, if somebody later says, hey, you know, everything was working until this version, I'm able to go, oh, and go back and take a look at what it was.

I also have a really cool Windows utility called Beyond Compare, which is a side-by-side, very smart Windows-based source comparison tool that I use often when I'm trying to understand what the differences are between two different files. So anyway.

Leo: Nice.

Steve: And I'm using Sync. I've talked a lot about Sync.com. They do like absolute version tracking, like every single time I save my source creates a checkpoint at Sync.com for that file. And again, it's come in handy where I've wanted to roll back and grab a particular point in time. Sync.com has them all. So I've become a big fan of that. And in fact I'm not using FileBack PC any longer because Sync.com just does everything and is also backing it up to the cloud and encrypting it and so forth. And it gives me multiple site access, too. So I'm able to get it from both locations.

Leo: Yeah, I turn on versioning and sync, too, because that is a nice feature. Even not for coding, yeah.

Steve: Okay. So some Log4j news. I didn't want to skip over anything important regarding Log4j this week. And the good news is there was no clear Log4j-related disasters reported during the past week. There was plenty of Log4j stirring, however. Microsoft reported that the SolarWinds, we remember them from last year, their Serv-U servers were under a Log4j attack in a failed attempt at leveraging their support of the LDAP protocol. As a consequence of the details, it wasn't actually possible to exploit them. But the attacks failed. Still they updated their Serv-U servers just to, like, in the interest of not being responsible for any more damage to the industry. And Akamai reported seeing attacks aimed at Zyxel's networking gear.

In response, Zyxel has updated one of their products and posted, they said: "Zyxel is aware of remote code execution vulnerabilities in Apache Log4j and confirms that among all its product lines, only NetAtlas Element Management System (EMS)" - whatever that is - "is affected." They said: "Users are advised to install the applicable updates for optimal protection." So Zyxel did have something that was needing patching, and they've done that. So it's mostly just kind of like that during the week. No huge drama. Just stirrings and rumblings and general indications of ongoing, largely behind-the-scenes work by many, I mean, everyone is scrambling to remediate that surprising discovery of such a widespread vulnerability.

The way "The Record" summed things up was to write: "While news cycles move fast from topic to topic, the situation around the Log4Shell exploit has not changed since last month, and the vulnerability is still heavily targeted and abused by threat actors seeking to enter corporate networks. At the time of writing, there have been reports about threat actors such as ransomware gangs, nation-state cyberespionage groups, cryptomining gangs, initial access brokers, and DDoS botnets, all of which are now using the vulnerability in their operations."

So, you know, all of the usual suspects jumped on this quickly, and all of their various operations acquired yet another means for remotely penetrating networks. We know what the shape of the patch curve looks like. I'm sure 90% of the problems that were publicly present are resolved by this time. It's been nearly two months since early December when this all began. Yet there will be that remaining 10%, probably never get patched. And I'm pretty sure they're probably all well infested by this point.

Okay. So who pays for ransomware attack recovery? I just personally find the topic quite interesting since one of the targeting parameters which we know from some dialogues that were shared, we know that the parameter used by attackers has been their victims' ability to pay. Right? The whole point is getting paid. And so it doesn't make any sense to pay to attack somebody who has no money or who isn't going to be able to pay. And since in the case of many public and private targets, their cyber-insurance has been the actual source of ransomware payment and then post-attack recovery funding, insurance sort of is in this cross-hairs.

Back four years ago, in June of 2017, and I remember we talked about this at the time, Leo, the pharmaceutical giant Merck was hit by the NotPetya ransomware in what was really a surprisingly devastating cyberattack. Four years ago, so that was before the current surge, and NotPetya was one of the early ones. And the attack was apparently quite devastating, with Merck claiming that the data on more than 40,000 Merck computers was destroyed. At the time, Merck estimated the damage from this one attack at $1.4 billion, a quite sizeable loss resulting from production outage, costs to hire IT experts, and costs of buying new equipment to replace apparently all affected systems.

And as I said, we discussed it at the time, and I remember you, Leo, wondering about the size of that number, you know, it's like, whoa. First of all, 40,000 computers? What, are you going to replace every single one? I mean, it sort of seemed like a, I don't want to say a scam, but like they were going to get a nice ride out of their insurance policy if they could. You know. And certainly it seems a bit steep. In any event, Merck was quite well insured, carrying a $1.75 billion "all risk" insurance policy.

Leo: It probably didn't say ransomware then because it was so uncommon; right? It was just...

Steve: Correct, correct.

Leo: Yeah.

Steve: But there was some cyber coverage. It included coverage for software-related data loss events. So at that point, sort of generic. Merck's insurer, Ace American, wasn't happy about the idea of paying out $1.4 billion.

Leo: No, really?

Steve: Oh. You'd like all new computers? You'd like 40,000 new PCs? Is that right? Oh. So they refused to cover the losses, citing that the NotPetya attack was part of Russian hostilities against Ukraine.

Leo: Yeah, I remember this, yeah, yeah.

Steve: Uh-huh. And as a result was subject to the "Acts of War," again in quotes, that is an exclusion clause in insurance policy contracts, the Acts of War exclusion clause which is standard boilerplate present in most, as I said, insurance contracts. So Merck sued Ace American in November of 2019, arguing in court that the attack was not an "official state action," hence the Acts of War clause should not apply. Merck's attorneys argued that the exclusion clause contained language that limited the Acts of War to official government agencies and did not specifically mention cyber-related events. And as a result, the Act of War exclusion clause should not apply to their customer.

Okay. This popped back into the news last week after a court in New Jersey agreed with what seemed rather clear-cut language in the agreement, ruling in favor of Merck. Judge Thomas J. Walsh wrote in an opinion justifying his ruling, he said: "Given the plain meaning of the language in the exclusion, together with the foregoing examination of the applicable case law, the court unhesitatingly finds that the exclusion does not apply." The judge argued that despite knowing that cyberattacks can be acts of war, Ace American did not move to update the language in its exclusion clauses. He said: "Certainly they had the ability to do so. Having failed to change the policy language, Merck had every right to anticipate that the exclusion policy applied only to traditional forms of warfare."

So that set some additional case law. And although the case wasn't mainstream news, the insurance industry has been watching it closely, and it has had a large impact and has been having a large impact on the cyber-insurance business, with several major insurers updating the language of their Acts of War exclusion clauses, the latest being

Lloyd's, which updated their language just a few days before the court's ruling. So the Merck case is attention-grabbing due to the amount of money involved, certainly. But there have been a great many lower profile lawsuits brought by ransomware victims in the last couple years, and those cases have also largely been won. The insurance companies are not happy, and they're responding. Which brings us to the other thing I wanted to talk about, the rising cost of cyber-insurance.

The Suburban School Cooperative Insurance Program (SSCIP) is an insurance pool designed to allow school districts to join together, sort of in a collective bargaining way, to negotiate better insurance rates and lower management fees. One of the school districts participating in this co-op program is the Bloomington School District 87 in Chicago, Illinois. That school district recently published its cyber-insurance renewal details, which caught some of the tech press's eye. They reported that their cost jumped in one year from what it had been, $6,661 in 2021, to $22,229 for 2022, 334% of the previous year. And in fact the cooperative was apparently fortunate to even get that. In their memo, the district noted that: "In light of events that have negatively impacted the cyber-insurance market, SSCIP was unable to initially find the required coverage for the group. After a small delay, the cooperative was ultimately able to secure an insurer willing to accept the risks of the pool."

Emsisoft recently published a summary of ransomware attacks in 2021. I mean, that's just a report begging to be written. Wouldn't be very difficult; right? Just play back this podcast for the year of 2021. They wrote: "Predictably, 2021 was largely a replay of the previous two years, with the U.S. public sector again experiencing a barrage of financially motivated ransomware attacks. The attacks impacted a total of 2,323 local governments, schools, and healthcare providers." The breakdown was 77 state and municipal governments and agencies, 1,043 schools, and 1,203 healthcare providers.

An interesting bit of the memo published by District 87, talking about their cyber-insurance coverage, indicated that the districts participating in the cooperative insurance policy would be required to do more than just make their payment premiums or premium payments. To qualify for this coverage, the districts would also be required to fully implement multifactor authentication for all logins across all of their accounts. And interestingly, until that was done, the insurance coverage limits would remain low. So the insurer is saying, okay, here's the price. We're not going to, like, give you full coverage until you can assert that you have taken the steps of implementing multifactor login authentication for every login you have.

Once that's done, and they're expecting to be able to do that by the end of March, then you get the full coverage that you're paying for. Which I thought was interesting. As we know, in the Log4j world, adding multifactor authentication to logins won't help, right, since the bad guys are crawling in through the backdoor rather than guessing the authentication protecting the front door. But adding multifactor authentication where it's been absent before, which was probably everywhere, would certainly remove the lowest hanging fruit. And I would argue that certainly provides significant protection. Overall, though, as we expected to see, the cost of cyber-insurance coverage is predictably increasing significantly across the board for anyone who needs it. Yikes.

And we have to talk about, and this leads us to some discussion again, about WordPress. The guys at WordFence have uncovered another very critical - this one's a CVSS of 8.3 - flaw in a WordPress add-on named "WP HTML Mail." WP HTML Mail is an email template designer used for designing custom email. Unfortunately, its use is currently exposing more than 20,000 WordPress sites that use it to malicious code injection, phishing scams, and more.

The root of the problem is a faulty configuration, well, which is putting it kindly, in the REST-API routes used to update the template and change settings. I say it's putting it

kindly because there is no authentication at all required to access the REST-API endpoint. Consequently, any user, I mean, like anyone on the Internet, has access to execute the REST-API endpoint to save or retrieve an email's theme settings. This would allow the injection of malicious JavaScript into the mail template that would execute anytime a site admin accessed the HTML mail editor. Okay. So threat actors could add new users with admin credentials, inject backdoors, implement site redirects, and use legitimate site templates to send phishing emails, among many other things, basically a complete site takeover.

WordFence said that: "Combined with the fact that the vulnerability can be exploited by attackers with no privileges on a vulnerable site, this means that there is a high chance that unauthenticated attackers could gain" - yeah, high chance - "could gain administrative user access on sites running the vulnerable version of the plugin when successfully exploited." As I said, the plugin is installed on more than 20,000 sites and is compatible with other plugins run by WordPress sites with large followings like the ecommerce platform WooCommerce, online form builder Ninja Forms, and community builder plugin BuddyPress.

WordFence noted in their disclosure: "We recommend that WordPress site owners immediately verify that their site has been updated to the latest patched version available, which is version 3.1 at the time of this publication." And for what it's worth, this latest disclosure comes just a week after the firm known as Risk Based Security published their findings that the number of WordPress plugin vulnerabilities exploded by triple digits in 2021. And all of our listeners know it's a constant topic here just because this is such a problem. And as we mentioned just last week, remember, three different WordPress plugins, all written by the same author, were reported with the same bug, exposing 84,000 sites running ecommerce add-ons to full site takeovers.

So is anyone here noticing a worrisome trend with WordPress? I'm certain that the developers of these plugins have the best of intentions. But as we're going to see in detail at the end of today's podcast, writing secure code is surprisingly difficult. Now, I don't excuse somebody who doesn't even add authentication to the REST-API used for configuring his plugin and the templates that it uses. That's inexcusable. I mean, that's just, like, no security whatsoever. And the problem is most developers stop their work the moment their code starts to work. But that often means that the security of that code, as a vital aspect of it, is barely, if ever, considered.

Okay. So on top of all that we had a supply chain attack on a popular WordPress add-on provider. You know, it's bad enough when plugin authors who are untrained in security coding make mistakes. That's not good. But when deliberately malicious actors are able to get their malicious code into widespread and popular add-ons, things get much worse. In this case, threat actors were able to compromise 40 themes and 53 plugins all belonging to AccessPress, a developer of very popular WordPress add-ons which are used by over - sit down - 360,000 active WordPress sites. There's like 800 and some thousand, more than that, but this is more than a third of all WordPress sites used themes and plugins from AccessPress. So, yeah.

The guys at Jetpack, who are professional WordPress developers, explained their discovery. They said: "While investigating a compromised site, we discovered some malicious code in a theme by AccessPress Themes, a vendor with a large number of popular themes and plugins. Upon further investigation, we found that all the themes and most plugins from the vendor contained this suspicious code, but only if downloaded from their website. The same extensions were fine if downloaded or installed directly from the WordPress.org directory." Meaning that they'd not been contaminated by this supply chain attack.

"Due to the way the extensions were compromised," they wrote, "we suspected an external attacker had breached the website of AccessPress in an attempt to use their extensions to infect further sites. We contacted the vendor immediately, but at first we did not receive a response. After escalating it to the WordPress.org plugin team, our suspicions were confirmed. AccessPress websites were breached in the first half of September 2021, and every one of their extensions available for download on their site was injected with a backdoor.

"Once we had established a channel for communicating with the vendor, we shared our detailed findings with them. They immediately removed the offending extensions from their website. Most of the plugins have been updated," they concluded. So anyway, I won't take up any more of our time on this specifically. But I think that our constant Extinction Level Event reports about WordPress ought to give us some pause.

Leo: Here's what I emphasize. They're not about WordPress. They're about WordPress plugins. WordPress base code has always been secure.

Steve: Yes, that's exactly true. I think it's becoming clear that WordPress, with its add-on ecosystem, made a lot more sense when it was initially released 19 years ago, back in 2003, than it does today. And I agree with you, Leo. There's no problem at all with WordPress itself. So one thing people...

Leo: Just be careful about what extensions you use, I guess. I mean, you know...

Steve: Exactly.

Leo: The thing is that it is totally dominant. Half a billion sites are on WordPress.

Steve: Yes.

Leo: I mean, it's by far number one. I think it's almost half of the web.

Steve: Yes. It is a...

Leo: It makes it a target, for sure.

Steve: It's 39.5% of all Internet websites. So, like, 40% of the Internet. And as a CMS, a Content Management System, 62% of all CMS is WordPress. And the number two in like a distant third is Shopify with 3.2%. So I completely agree with you, Leo. WordPress itself, the core, is solid. Those guys are professional developers. The problem, though, is that there's this, I mean, they...

Leo: It's too easy maybe to make an add-on. Maybe that's the problem.

Steve: That and, you know, because I was running WordPress for a while. I had a blog there. And what annoyed me was WordPress lacked really useful and important features.

Leo: Right.

Steve: And they know it. They don't add features to their core product because they believe, and I think this is historically true, but not safe today, that, I mean, they're trying to encourage an add-on ecosystem. The problem is it's just PHP. And anybody can create an add-on.

Leo: It's just too easy, yeah.

Steve: Yes. And we know that security is really hard.

Leo: Yeah.

Steve: And so basically what we end up with is just this constant problem, I mean, the WordFence guys, who have staked out WordPress security, they're busy because, you know, everywhere they look there's like this catastrophic tens of thousands of sites vulnerable to some add-on that, like, oh, look, how cool. You know what, they're all free. And, but, you know, they're a security catastrophe.

Leo: Yeah. They have, and I wonder how much they vet the plugins, I mean, you can get them officially from, as you pointed out, WordPress.org.

Steve: Yup.

Leo: And I wonder how much security they do there. I mean...

Steve: Clearly they have no control over what, you know...

Leo: Any third-party can - yeah, yeah, yeah.

Steve: Yeah. So in a nice little bit of happy news, sort of following on what we were talking about last week about the need for providing more funding to do more of what we're already doing, the European Union has announced that it plans to fund some significant bug bounty programs. There are five open source projects that are heavily used by public services across the EU. Those are going to receive bug bounty funding.

Those five projects are LibreOffice, which as we know is a very good free open source alternative to Microsoft Office, with which it's often compared; Mastodon, which is the web-based system for hosting private social networks. And Leo, I know you talk about Mastodon often.

Leo: We have a Mastodon, twit.social, yup.

Steve: Yeah. Odoo, which is an enterprise (ERP) you know, enterprise resource planning application, apparently very popular; CryptPad, an app for exchanging encrypted messages; and LEOS, L-E-O-S, which is software designed to help with drafting of legislation.

So the bug bounty program will run throughout this year, through 2022, on the Intigriti bug bounty platform, and the EU will provide a rewards pool of up to 200,000 euros, which is about $225,000 U.S. Intigriti, whom we've never mentioned before, describes itself as "Europe's #1 ethical hacking and bug bounty platform." They're I-N-T-I-G-R-I-T-I dot com. So they're like HackerOne. They have more than 300 active programs, more than 40,000 researchers hunting bugs with them, and more than 3 million euros have been paid to date. So that's sort of the European equivalent.

Bug hunters will be eligible to earn as much as 5,000 euros, about $5,600, for finding and reporting "exceptional vulnerabilities," and are entitled to a 20% bonus on top of whatever they're awarded for the vulnerability if they provide a fix for it along with their report. And, you know, having covered the recent problems with Microsoft patching against proofs of concept, rather than actually repairing the underlying problems, I think that this idea of also getting the fix from the same researcher who discovers the problem makes a great deal of sense. Who better to understand the problem that they found than the researcher to suggest the proper way to fix it, and do it completely?

Now, admittedly, it's more feasible with open source projects than it is with closed-source environment like Windows. But I just think that's a very sane perk to offer, you know, and who wouldn't go for it? If you found the problem, you can certainly like, okay, hey, guys, you know, this should have been declared as a long instead of an int, you know, kind of thing. So, I mean, not hard to figure that one out.

This new program was announced last week, and it's sponsored by the European Commission Open Source Programme Office (EC-OSPO), which is pretty new. It was founded two years ago, in 2020. It's the successor to one that we talked about before, FOSSA. The EU-FOSSA was the Free and Open Source Software Auditing project, E-U-F-O-S-S-A. And that was the organization through which the EU had previously funded two other bug bounty program initiatives for open source software, first in 2017 and then in 2018.

Back in 2017 the EU funded bug reports for VLC Player, my own preferred standalone video player. And in 2018 this EU-FOSSA group sponsored bug reports for 14 projects: 7-zip, Apache Kafka, Apache Tomcat, Digital Signature Services, Drupal, Filezilla, FLUX TL, the GNU C Library, KeePass, midPoint, Notepad++, PuTTY, the Symfony PHP framework, and again the VLC Media Player and WSO2. So they've done lots of good stuff in the past, and it looks like they're going to continue to be funding things that they see being necessary and of interest. Oh, and I should mention that the same program also funded security audits for the Apache HTTPD web server and KeePass, the password manager.

So a huge yay to all of this. It would be great to see more of this same thing and more in coming years. And as I said last week, the industry I think is already doing many things that make a great deal of sense. We just need more of what we're already doing. And some of that needs some money. So tip of the hat to the EU, and it would be great if the U.S. White House and admin would arrange to do something to provide similar funding.

In the fun name of the week we have MoonBounce. It's the name given to an EFI bootkit. UEFI was designed from the beginning deliberately to be a secure and securable boot

platform foundation for next-gen firmware for motherboards. Unfortunately, every day it's looking less and less like it's going to meet that mark. Last Thursday researchers at Kaspersky Labs disclosed their discovery of yet another novel UEFI bootkit which can infect a computer's UEFI firmware.

Okay. So what makes MoonBounce, as Kaspersky named it, special is that unlike some previous pre-boot malware, this one doesn't hide inside the hard drive's EFI System Partition, the so-called ESP. That's where most UEFI malware tucks itself. Instead, it arranges to infect the so-called "SPI memory" that's found on the motherboard. SPI is the Serial Peripheral Interface. And it's in fact, it's the way you could have a flash memory which is this little itty-bitty chip with, like, six leads on it. You just need to give it power and serial data and clock, and it's able to do the rest. So you'd hardly even know it was there. And that gets loaded into the chip at boot time in order to provide the processor that's running on the baseboard of its firmware.

So consequently, as a consequence of the fact that it's actually in this little flash chip, unlike other bootkits, defenders who are trying to prevent this stuff from getting them cannot shake this one off by reinstalling the operating system or even by replacing the hard drive. This pernicious bootkit will continue to remain on the infected device until the motherboard's SPI memory is re-flashed to clean it out, or the motherboard's replaced. And according to Kaspersky, MoonBounce is not even the first UEFI bootkit they've seen that is able to infect and live inside SPI memory. It's actually the third. There was the first one was known as LoJax, L-O-J-A-X, and then the second was MosaicRegressor.

And sadly, all indications are that UEFI bootkits are proliferating. Recent months have uncovered something known as ESPectre, E-S-P-E-C-T-R-E, ESP of course as in EFI System Partition. So that's one that does live on the hard drive in the ESP partition. And FinSpy also has a UEFI bootkit, and there are others. Kaspersky commented in their report that what was once considered unachievable following the rollout of the UEFI standard, meaning it was unachievable to infect the EFI firmware, that's now becoming the norm as opposed to the exception. Oh, and I also almost forgot to mention that MoonBounce has been directly attributed, that is, the creation of it, and actually the deployment where it was found to the Chinese government's state-sponsored hacking group, APT41. So that's where that bad guy came from.

The good news is it's not being seen like sprayed or in some - anything that anybody's downloading. It is being used in very highly targeted attacks. So in this case there is some specific entity that the Chinese government or some branch of the government, whoever controls APT41 said we need to get inside these people. And probably some research was done to find out what kind of hardware they're using. And then something was designed specifically to get into that hardware. On the other hand, we know that these things always start off a little more generically than they wind up. I mean, they start off much more specifically in the beginning and then tend to become more generic over time, just as UEFI bootkits are no longer like a shock to anyone. It's like, oh, yeah, okay. What are we going to call this one? How about MoonBounce? No one's done that yet. It's like, okay.

Okay, some feedback from our listeners. I got a note asking, which I saw this morning, it said: "Morning, Steve. What do you use to run pfSense on? I'm looking at switching to that from a UniFi security gateway." Okay. So we know that I run and like pfSense. Actually it's sitting up above me. It was right next to the cable modem which I rebooted. Our listeners don't know, but we had a glitch just before we began recording the podcast. My 'Net dropped offline, and I had to restart the cable modem. Right next to it is a little Netgate SG-1100. And I'm always conscious of the fact that those are my initials, so there's no relation. Netgate SG-1100. It is a cute little prepackaged $189 three-interface router which comes preloaded with pfSense.

So you get it from Netgate. I've got a link in the show notes, or just put Netgate.com into Google and you'll find it. It's got a WAN port, a LAN port, and an OPT port. And it does everything you could want because it's got pfSense in there. One thing to be aware of with these little routers is it's one thing for them to say, oh, we've got GigE ports; right? We've got gig Ethernet ports. But it matters how much processor is in there. The original little box was the SG-1000, which had much less switching power than the SG-1100.

So this is the upgraded version. They rate its routing capability at 880 mbps. Firewall, which is going to slow it down because it's going to have to do some inspecting and filtering, that brings its routing to 656 mbps. And it's able to host an IPSec VPN at 74.2 mbps. So respectable performance. I think I pay Cox for 300Mb downstream, and so this little box, if I run any kind of a speed test on my network I'm seeing that fully delivered bandwidth, which means it's getting all the way through the router down to my machine.

I have a different device in my other location, though, that I just kind of wanted to put on the map for people because I like it a lot. It's from a company called Protectli, P-R-O-T-E-C-T-L-I. And it's Protectli.com. Basically their business is to produce little turnkey boxes, not preloaded with software, but very software agnostic. They talk about there is actually a fork of pfSense called OPNsense, and it runs Linux, and it runs a bunch of third-party packages. You can get it in 2-, 4-, and 6-port models. And I should just mention, I skipped over the fact that that three ports is the key because as we've talked about before, unless your WiFi access point or WiFi router supports isolated guest networks, then you need to provide that yourself. Or if you want to have a wired isolated guest network, as is possible, then you need a third interface, not just a switch on your router. You need three separate interfaces so you can give them different firewall rules.

So anyway, Protectli has 2-, 4-, and 6-port models, and you're also able to pick the speed of the processor, the amount of RAM, the amount of storage. Basically it's a little barebones, as it's called, you know, it's got the processor, but then you provide the RAM and mass storage, or you can get it from them, little fanless box that is just perfect for making little Internet appliances. So anyway, that's how I'm running Netgate, or running pfSense, rather, Netgate here and a Protectli box at my other location.

And just following up on our discussion last week, Leo, about refilling SodaStream cans, Peter Crocker, he was listening to our conversation. He said: "The big tanks need to be tested every five years in a hydrostatic test." He said: "Basically a requirement of any pressure cylinder like welding gas, scuba tank, or CO2." He said: "Then they stamp it with a new date. Most places," he says, "send it away for the test." And he said: "In terms of filling the big tanks, they need to be cold and filled slowly, so often not a service you wait for, hence some exchanging if you want it right away."

And I did mention that one of the things that I noticed when my big tank was last refilled, this is on my mind because it was only a couple weeks ago, is he opened the tap and expelled the CO2. Now, you might at first thing, whoa, wait a minute, he's wasting gas; right? He's like letting it out. No. He's running like the basis of an air conditioner. He's expanding the gas which is pulling heat out of the cylinder. And it ends up cooling it off, which is exactly as Pete says, the way you want to refill it.

So when I was talking about one of my tips for refilling the SodaStream canisters, what we do is keep them frozen until I screw them onto the big tank and then refill. Similarly, you'd like to have any tank receiving CO2 be as cold as it can get it. So they do what they can to cool off the recipient tank actually by blowing the CO2 out of it, which brings its temperature way down.

Paul Walker, he said: "Hey, Steve, have you considered updating ShieldsUP! so it can scan the full port range at once? Just been listening to Episode 854 where you talk about

port 20005. If we could scan the entire range, it might help preempt things like this." And Paul, I completely agree. I remember, and Leo, you and I were talking about this at the time, remember I used to be connected to the Internet by a pair of T1s.

Leo: Yeah, yeah. And that was when it was cool. Yeah, wow.

Steve: And that was when it was like, that was like, whoa. You've got T1s? Hey. The problem was they were, what was it, 1.54Mb, I think, each. So I had them paired so I could get like a little over 3Mb of total upstream - I guess they were symmetric, so it was bidirectional. Yeah, it was 1.54Mb in both directions. And GRC was here, like in this room that I'm talking from right now. That is, the GRC server was sitting on a table to my side with a UPS next to it. You know, those were the days. The problem was...

Leo: Those were the days. 3Mb, ooph.

Steve: That's right. The problem was I was worried about saturating my own bandwidth with TCP SYN packets because if I couldn't get them out, then they wouldn't bounce off of the far end's IP whatever it was, router or machine or whatever, and then get SYN ACKs back to me to sense that ports were open. So the point is I could be producing false positive stealths or false positive closes, that is, not seeing a SYN ACK return if the problem was at my end rather than at their end. So I deliberately, I did a couple things. I deliberately limited the number of ports that I was scanning, and I throttled them out so they wouldn't be bunched up.

Anyway, the point is that was then. I've got lots more bandwidth now. I'm at Level 3. I've got a big fat pipe connecting my systems. So I absolutely could do that. The only thing I am lacking, famously, is the time to do it. But if at some point in the future I get to a point where SpinRite is essentially done once again - although we know I've got a lot of work to do before I get to that point. I'll get there.

When I get there, one of the things I would love to do is basically do a comprehensive port scan, every single port at a given IP. It might take a while; but I agree, it would be really useful to be able to do that. And I need to do that. I also need to update everything to IPv6. I get a lot of requests for the DNS Benchmark to be running IPv6, not just IPv4. And that's still the most downloaded thing we've got is that benchmark, like 3,000 downloads a day, like endlessly.

And lastly, Dylan Anthony said: "According to the author of cURL, it's not an RCE. So you're not the only one who couldn't figure out why Microsoft categorized it as such last week." Remember I was talking about the Patch Tuesday there were two open source projects, Libarchive and cURL. And Microsoft had them both categorized as remote code execution. And I said, no, you know, I could see a man-in-the-middle attack because this thing was about the kludge of using STARTTLS to bring up a secure connection when you were creating a cURL link to an email server that wouldn't be initially secure. And it just seemed like, nah, I couldn't see how that was remote code execution. And indeed it isn't one.

So I did some more examination of the NetUSB hack which reportedly, as we know, affects many millions of Internet-connected routers. Last week when we first talked about it, I skipped over the techie details because there was so much else to talk about rather than just the nitty-gritty of the attack's actual mechanics. But after spending some time looking into what went wrong with the code, I realized that not only would a full explanation of the flaw be well within our listeners' ability to grasp, but that it would

serve as another very valuable example of precisely the way things go wrong with software and Internet security. So what better thing to talk about on this podcast?

One thing I should note is that this flaw is incredibly widespread. The tiny list of router vendors that I shared last week from the guys who found this flaw should not provide anyone with any solace. If your router has a USB connection port, you really do need to make absolutely certain that it's not listening on its WAN interface port 20005.

And as you said, Leo, at the top of the show, your memory was exactly right. This is not the first time that KCodes Technology, the Taiwanese developer and licensor of the NetUSB technology, has had trouble. This Security Now! podcast Episode 509, recorded May 26th, 2015, nearly seven years ago, was titled "The NetUSB Bug." You know, it should have been Part 1 because here we are again. And that bug was a biggie, too. And perhaps because, I don't know, maybe because we were all seven years less jaded then by the endless parade of vulnerabilities that we've been subjected to than we are today, there was more attention paid to the breadth and scope of the trouble than there seems to be now. It feels to me as though maybe today's tech press got one headlined page out of the story, then went back to wondering what NFTs are and, you know, what they should have to say about them.

That earlier vulnerability, which was found in the same NetUSB kernel module as the current one, was discovered by a group called SEC Consult. At the time they wrote: "NetUSB suffers from a remotely exploitable kernel stack buffer overflow. Because of insufficient input validation, an overly long computer name can be used to overflow the computer name" - I know, Leo. I know.

Leo: Okay.

Steve: "...can be used to overflow the 'computer name' kernel stack buffer. This results in memory corruption which can be turned into arbitrary remote code execution. Furthermore, a more detailed summary of this advisory has been published at our blog." And then I've got the link for anyone who's curious. So, yeah. Insufficient input validation. When you don't look at how long the name is, and you just stick it in the buffer.

Leo: I don't know why anybody would name their computer more than five letters.

Steve: No.

Leo: It just doesn't seem sensible.

Steve: Too much to type. And at the time these guys provided a benign proof of concept which simply crashed the targeted router. In their disclosure under vulnerable and tested versions they wrote, they said: "The vulnerability has been verified to exist in most recent firmware versions of the following devices: TP-Link TL-WDR4300 V1 and the TP-Link WR1043ND v2, and the Netgear WNDR4500." Just those three. But then they added that they had identified NetUSB as being present in the most recent firmware version of the following products, noting that the list they were providing was not necessarily complete. Okay. And everyone will thank me for not reading that list because there are truly too many for me to read. So I shortened the list for the podcast. But I wanted to give everyone a feel for it.

In the list was the D-Link DIR-615C, then 42 different Netgear router models, 40 different TP-LINK models, 14 TrendNet models, and four Zyxel routers. And they added that: "Based on information embedded in KCodes drivers," they said, "we believe the following vendors are affected: Allnet, Ambir Technology, AMIT, Asante, Atlantis, Corega, Digitus, D-Link, Edimax, Encore Electronics, Engenius, Etop, Hardlink, Hawking, IOGEAR, LevelOne, Longshine, Netgear, PCI, PROLiNK, Sitecom, Taifa, TP-LINK, TrendNet, Western Digital, and Zyxel."

In other words, this one company appears to have cornered the market on, and they do claim to have patents on, extending USB links across consumer WiFi to router USB ports. And although that was seven years ago, there is no reason to believe that any router using USB extension today is not using KCodes' troublesome technology. In fact, there is a project called USB/IP which DD-WRT uses. It's the only one I know of that is doing USB/IP and not using this KCodes technology. So again, I mean, all of those companies just license the stuff from KCodes, and thus they all have this problem.

SEC Consult's original write-up provided a full responsible disclosure timeline, and I looked at it. It would be charitable to say that KCodes Technology was unresponsive, that is, the company, when SEC Consult tried to contact them seven years ago and said, hey, guys, you've got a problem here with long computer names that is really bad. Silence.

So SEC Consult finally disclosed what they had found to the CERT Coordination Center back then and directly to several of the major, most seriously affected router vendors. After speaking with a few of the gazillion vendors, they wrote: "Sometimes NetUSB can be disabled via the web interface, but at least on Netgear devices this does not mitigate the vulnerability. Netgear told us that there is no workaround available. The TCP port cannot be firewalled, nor is there a way to disable the service on their devices."

And of course finally, as is inevitable, more so today than seven years ago, a fully weaponized exploit was published, and it's archived on GitHub. I grabbed a chunk of the boilerplate at the top of the Python script. The author said: "This is a weaponized exploit for the NetUSB kernel vulnerability discovered by SEC Consult." The hacker author says: "I don't like lazy vendors. I've seen some DoS PoCs floating around for this bug," you know, meaning proof of concepts that just crashed the router. He says: "And it's been almost five months. So let's kick it up a notch with an actual proof of concept that yields code execution. So anyway, a remotely exploitable kernel vulnerability. Exciting, eh?" And then he says: "Smash stack, ROP" - meaning return-oriented programming - "decode, stage, spawn userland process. Whoo."

He says: "Currently this is weaponized for one target device," he says, "the one I own." He said: "I was planning on porting OpenWRT, but got sidetracked by the NetUSB stuff in the default firmware image." Anyway, the point was he was going to replace whatever device he had which had the NetUSB vulnerability with OpenWRT instead, and that would be good, but he got a little sidetracked. He said: "Oh, I'm going to weaponize the NetUSB bug before I remove the firmware from my hardware."

Anyway, it's there. It's seven years old. And here we are again today. That was then. But I think it provides some important context for today. I wanted to be certain that everyone understood that many, many more than five or six router vendors were involved, and that KCodes Technology has in the past been anything but helpful and responsible in the way they've acted.

The guys who discovered today's problem understood that the only possible way to get all of the routers whose manufacturers had licensed the common NetUSB code updated would be to contact KCodes Technology. And as we know, in the past that hasn't turned out so well. Although the guys at Sentinel Labs, the people who found what we're talking about today, deliberately stopped short of providing anything beyond a simple denial of

service proof of concept, you know, crash and reboot the router, there is every reason, in fact much more in today's climate than there was seven years ago, to expect that someone is going to weaponize this exploit, if it hasn't already been done by the time we're recording today's podcast. It is too pervasive; and it is, as we're going to see, too simple. There is nothing left to the imagination here.

The Asus router that I use at my other location does not offer NetUSB functions. Asus never has. And even if it did, it's daisy-chained, as I mentioned, behind another little router, the Protectli router running pfSense. So no ports that the Asus might have open would have been exposed to the Internet anyway. I use pfSense to perform port translation in order to get around some annoying Cox port filtering, which they're doing for the benefit of normal people.

Okay. So now that I have everyone's attention, let's take a look inside KCodes' technology to see what they messed up. To initiate a connection with the router, any PC located on the LAN initiates a TCP connection to the router's port 20005. The router's kernel service and server listening for those incoming connections should only be listening on the router's LAN interface. But as we know from last week, the worst aspect of the flaw is that the NetUSB service is bound to 0.0.0.0 on the TCP/IP stack, giving it a presence on both the router's LAN and WAN interfaces, and thus exposing it to the public Internet. It would still be a problem to have this vulnerable service, if only on the LAN, since anyone on the LAN, like in an enterprise environment, where maybe you don't trust everybody, could potentially take over the router without authorization and authentication. But allowing it to happen with anyone, anywhere, meaning on the WAN, definitely takes it up a notch.

Okay. So once the standard three-way TCP connection handshake has occurred, the PC wishing to have access to the USB devices connected to the router sends a signature plus 16 bytes (128-bits) of what they term "Verify Data." It appears that these 16 bytes of so-called "Verify Data" serves as a connection nonce to prevent replay attacks. Right, just the PC gets 16 bytes, 128 bits, of random stuff and says, you know, this is how I'm going to tag this conversation.

Upon receiving the PC's signature with its "Verify Data," the router AES-encrypts that "Verify Data" and returns it along with its own 16 bytes of "Random Data." And, you know, the PC probably decrypts the AES-encrypted "Verify Data" that it sent and got from the client to verify that it matches. So basically each end sends the other a large random nonce which they each encrypt and send back so that the other end can decrypt it and verify it. So they establish a replay-proof verification of the two endpoints.

Assuming that everything works, it then gets ready to take the next step. They've exchanged this nonce data. The router's code then drops into, having established this connection, a command-parsing loop, a so-called "while-loop," to await the PC client's commands. So everything is being driven by information now being sent by the PC that has connected to the router. And of course this could also be anybody anywhere on the Internet in the case of this service being exposed on the WAN. And I mentioned this is a while-loop. There is probably a command that exits that loop. That is, so that the loop will sit there accepting and processing commands from the client that is connected to the router until a hang-up command is received, which drops it out the loop and terminates the connection.

Anyway, this loop waits for and receives a 16-bit command word which causes it to jump to a function which then further handles the needs of each specific command. So that's sort of a simple way of executing or implementing a command-driven protocol. You have a 2-byte 16-bit command which the router will receive, and that causes it then to branch, you know, you would call it a switch function. The command would cause it to call a specific subroutine to process the rest of that command.

The researchers at Sentinel Labs found a problem in the function whose 16-bit command is hexadecimal 805F. So the receipt of that 16 bits as the command word 805F branches the code to a specific routine named "SoftwareBus_dispatch Normal EP Msg Out." And doesn't really matter what that does. That's not germane. Whatever it does, fine. We don't need to know or care. So the client first sends that command. Then it sends 4 bytes, which is of course 32 bits, to be the maximum number of bytes to follow, which will be read from the client. And, okay, that's clean since the client is declaring upfront the number of bytes that it will be sending in this 4-byte 32-bit value. This allows the receiving server, the router, to request an allocation of memory from the underlying operating system. Basically it says I need a buffer which will be used to receive and hold up to that much data received from the client.

And whatever that function is doing, 17 bytes of additional, you know, we could just call it scratchpad working memory space is apparently also required to process the command. So to the size of the memory allocation being requested, the function adds 17 so that an extra 17 bytes of memory will be obtained from the operating system for the function to use. That all seems great.

Okay. So to recap, the client sends the hex command code 805F, which is then followed by 4 bytes to indicate the amount of follow-on data that the client will be sending and that the router should be ready to accept. And upon receiving those 4 bytes it allocates memory to serve as a buffer into which to receive the client's data. And the server asks the operating system for that much plus 17 bytes more, so that it has a little bit of extra memory to use for whatever it's doing. After that, the code calls a data receiving function, giving it a pointer to the buffer which it's received from the operating system's allocation and the number of bytes that the client said would be forthcoming to fill that buffer.

The data receive function will receive data from the client, placing that data into the buffer that's been pre-allocated to contain it, until the specified number of bytes of data has been received. Once the expected count of data is received, the data receive function will return to its caller with the buffer filled with the expected data, and those 17 extra bytes at the end for the command processor to use.

Okay. This is all perfectly reasonable code. You could stare at that code all day long and find nothing to criticize. There's no way for the sender to overflow the receiving buffer, since the sender pre-declares the number of bytes that it will be sending, and that's all that the data receive function will accept. After it's got that many, it says okay, done, and returns to its caller. Upon receiving the 4-byte data size, an allocation of that amount plus 17 extra bytes is requested from the operating system. Upon the successful allocation of a buffer to hold the incoming data, the data receive function is told how many bytes to accept, and it does.

So what's the problem? What is it that the clever researchers at Sentinel Labs found? As we know, the sending client specifies its byte count as a 4-byte 32-bit integer. So it's reasonable to store that byte count in a 4-byte 32-bit integer. The authors correctly declare the integer, that is, the original authors of the code, declare the integer as unsigned because a byte count should logically be unsigned; right? You can't have a negative byte count. That doesn't make any sense. So that's fine. The problem arises when those 17 extra bytes are added to the memory allocation. The largest value that a 32-bit unsigned integer can represent is that old familiar number just shy of 4.3 billion, you know, like the same as the number of IPv4 IP addresses, you know, 2^32-1. That's the largest number that will fit in 32 bits.

So what happens if a malicious client connects to this router's port 20005, properly negotiates a handshake, then sends that vulnerable command code 805F to tell it which command it wants to execute, and for the count of the bytes it's going to send, it

declares that maximum value of 4.3 billion? In binary, that's all ones, the largest number that can be represented in 32 bits.

So, okay, the command processing code needs to allocate a buffer from the operating system to contain that many bytes of data. As before, it adds 17 to accommodate its small bit of extra scratchpad working memory. But now, adding 17 to a 32-bit value that is already as large as it can possibly be will cause the addition to overflow, and the value to wrap back around. To make that clear, if we were to add one to the 32-bit value that's all ones, the count would wrap around, as it's called, back to zero. But we add 17, so the count will wrap around to 16. And that 16 is the number of bytes this code then asks the operating system to allocate for us to hold the data that the client is about to send.

So the operating system does that. It allocates 16 bytes, which is what we told it we wanted. We receive a pointer to a 16-byte buffer, which we then hand to the data receive function, asking it to please receive the original byte count of 4.3 billion bytes of client data, which it dutifully begins to do. And what we have is a textbook perfect classic buffer overflow, where the client has absolute and total control over the contents that fills and then overflows the tiny 16 bytes of data buffer.

Without any additional work, that will immediately crash the router, probably causing it to reboot. But with the addition of some skilled hacking, it's quite clear that many, many millions of consumer routers are exposed to a very critical and extremely exploitable remote code execution attack where the attacker is readily able to supply the code that they want to send. Just, I mean, just a perfect example of a simple-to-make mistake. Nothing looked wrong about it. The logic of the flow was well thought out. Let's provide the byte count upfront. We'll get that many bytes plus a little extra that we need from the operating system. We'll hand that byte count to the receive function that will only receive up to that many bytes, so the buffer can't overflow. Once it's got that many, it'll come back to us, and we'll do whatever we want to with it. The problem, of course, is that they hit a wraparound, and 16 bytes were allocated rather than the amount declared plus 17.

So there are several points of failure evidenced in the design of the code. For one thing, though we don't know what command 805F does, it does seem quite unlikely that telling it that it's about to receive 4.3 billion bytes of data would be unreasonable.

Leo: Yeah, no kidding.

Steve: You know, it's like, what? No.

Leo: Get ready. Here come some gigs.

Steve: So for all we know, the most that would ever really normally be sent is some packet of something; right? Maybe a few K, a few Kbytes at a time. Obviously the router's got to have memory to hold it, to allocate a buffer for it, so it's not going to be big. Yet that command performs no sanity checking of any kind of what that forthcoming bytes size is. If it's 4.3 billion, fine. Come on. Bring it on.

We've often talked about this problem which affects the designers of interpreters. They assume, as the designers of this code that we're looking at now must have, that only a valid client would ever be connecting to their server. After all, they probably wrote the other end, the drivers in the PC that are going to be connecting to their server. So they

never bothered to place any sanity-checking limits, even really high-end, like this could never happen, so abort this, or anything. They just accept whatever the client will send.

If the machine had been 64 bits rather than 32 bits, adding 17 to a maximum 4-byte value would not have wrapped around, and the operating system would have been asked to allocate 4.3 gig of RAM, plus 17 bytes, which it would surely have balked at, failed the allocation request, and thus protected the router from any attack. But our little consumer routers are using inexpensive 32-bit MIPS chips, so 32 bits is going to wrap around back to zero and then some.

As a consequence of this pervasive bug, we can expect that a lot of damage will be done to users whose routers have aged out of their service life, or who have manufacturers or users who are not paying attention to what's going on. Fortunately, everyone here is.

Leo: Okay, well, there you go. Sanitize your inputs, kids. I guess that's always the motto.

Steve: Well, or, yes, exactly. Sanitize anything you accept from an untrusted source.

Leo: Or any source, yeah.

Steve: Anybody connecting to you over TCP. It's like, okay, I don't know who that is.

Leo: Interesting. Yeah, that's right, yeah.

Steve: Yeah.

Leo: Somebody said you should be a professor. You teach very well. I agree. You want to share this with your friends, tell them it's called Security Now!, and you can get it in many places. We'll start with Steve's site, GRC.com. He has 16Kb audio versions, 64Kb audio versions. He's got transcripts, too, which emerge a couple of days after the show. That's a great way to read along as you listen or to search and find a part that you're particularly interested in. Say you wanted to find the last NetUSB exploit. You could find that by doing a little search. He also has SpinRite. I wanted to say SpinRite. I mixed SpinRite and ShieldsUP!. He's got ShieldsRite. He's got ShieldsUP!, which is free.

Steve: And SpinUP.

Leo: And SpinUP. SpinRite, which is the world's finest mass storage maintenance and recovery utility. SpinRite is his bread and butter. So go over there and buy a copy. It's not expensive, given what you get. And if you buy it now, you'll get a free upgrade to v6.1, which is in process. You get to participate in the development of it and all that, too. GRC.com. You can leave feedback at GRC.com/feedback. Or better yet, go to his Twitter. His DMs are open. He's @SGgrc. That's another good way to leave him pictures and suggestions and questions and so forth.

We have the show on our website, as well, TWiT.tv/sn. We've got 64Kb audio. We also have video of the show. You can download those. There's a YouTube channel dedicated to the show. You can also subscribe in your favorite podcast client and automatically get every version as soon as it's sent out. Now, people sometimes ask, well, I want all the shows, all 855 of them. We're not going to put 855 shows in an RSS feed. It would be more than 4.3 billion bytes. It would be big. So we just put the most recent 10 shows.

If you want more than that, you can get them from Steve's site, or you can get them from our site. Both of us have every show since Episode 1 available for download there. And there are people have written scripts and so forth that will automatically get them. But, you know, I'll leave that as an exercise for the reader. Thank you, Steve. We'll see you next Tuesday at 1:30 Pacific, 4:30 Eastern, 21:30 UTC.

Steve: February 1st.

Leo: February 1st.

Steve: Thanks, buddy. Bye.