Libssh's Big Whoopsie!

Description: This week a widely used embedded OS (FreeRTOS) is in the doghouse, as are at least eight D-Link routers which have serious problems, most of which D-Link has stated will never be patched. We look at five new problems in Drupal 7 and 8, two of which are rated critical; trouble with Live Networks RTSP streaming server; still more trouble with the now-infamous Windows 10 Build 1809 feature update; and a longstanding zero-day in the widely used and most popular plugin for jQuery. We then discuss what can only be described as an embarrassing mistake in the open source libssh library, concluding by examining a fun recent hack and posing its solution to our audience as our Security Now! Puzzler of the Week.

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-686.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-686-lq.mp3

SHOW TEASE: It's time for Security Now!. Steve Gibson is here. And as usual it's a roundup of significant flaws, one in a free real-time operating system, another in a well-known router - actually many well-known routers - and an exploit that uses jQuery's File Upload plugin, plus a red alert for Drupal users. It's all coming up next on Security Now!.

Leo Laporte: This is Security Now! with Steve Gibson, Episode 686, recorded Tuesday, October 23rd, 2018: Libssh's Big Whoopsie!

It's time for Security Now!, the show where we cover your security and privacy and how things work online with Mr. Steven "Tiberius" Gibson. Hello, Steve.

Steve Gibson: Leo, great to be with you again. Trying to get my fingers in the proper Spock Vulcan salute. But I think it's...

Leo: It's not easy, is it.

Steve: Yeah, I think I'm a little slow on the uptake today.

Leo: I can do it on my right hand. And I'm a lefty. I can't, for some reason, do it on my left hand.

Steve: I think I've told the story before, but the best man at my wedding, he had way too much information on me, and I was terrified that he was going to embarrass me horribly...

Leo: Oh, yeah. That's his job.

Steve: ...when he got up to give a toast. And I said, "Gary, do not, do not embarrass me." Anyway, so being the clever person that he is, he got some small red rubber bands because he needed them in order to get his fingers to do the salute unaided. And all he did was he stood up and he said, "Gibson told me I wasn't allowed to embarrass him. So I'll just say 'Live long and prosper.'"

Leo: Perfect, perfect.

Steve: I said, "Thank you, thank you." So we've got a great podcast. The title is "Lib" - or you say "lib," I say "libe," potato, potahto - "Libssh's Big Whoopsie!" This was just so fun that it immediately became the topic of the episode because there's a mistake in a fortunately not widely used - but still, I mean, it's around - SSH library, an open source library. But the mistake is just so, as they would say somewhere in the U.K., "gobsmacking," that we just have to take a look at it because it's just too fun. Also, in covering another story that we will also conclude with, the hack was so fun that I thought, this would make a great Puzzler of the Week for our audience.

Leo: Ah.

Steve: So we haven't done this as often as we might.

Leo: I would love to do it every week. It's fun, yeah.

Steve: I know. It just never occurs to me.

Leo: Right.

Steve: But sometimes the setup is perfect. And so we will examine a really fun hack, and then I will propose to our audience to, in the intervening week, find a solution in your head. With everything that we've done on the podcast, all of the tools are available to our listeners who have been following along for a while. There is a - well, anyway, now I'm stepping on the story. But there's a super elegant solution. And it'll be fun to see, just this little self test, who can come up with it, so.

But in the meantime, we're going to talk about a mistake found in a widely used embedded OS, actually it's the number one embedded OS, FreeRTOS. RTOS stands for "real-time operating system," free as in free. Anyway, they're in the doghouse. And interestingly, Amazon has picked up FreeRTOS and decided to run with it and support it and sort of be its benefactor. But I'm very impressed with the things that I've seen that Amazon has done. We'll talk about the problems and their solutions.

Also there are at least eight D-Link routers which have serious problems, most of which D-Link has stated will never be patched. And so there's some interesting takeaway lessons for that. We're also going to look at new problems in Drupal 7 and 8. And if you're using Drupal 7 or 8, and you didn't update in the last few days, go do it now. Stop listening. Hit pause.

There are five problems. Two of them are remote code execution, which affect virtually all of Drupal 7 and 8. So this is another bad security problem that has hit Drupal. We talked about them not too long ago because I remember I was - you guys use Drupal, Leo, and your guys were already up on the news and had fixed it before we even got to it on the podcast.

Leo: Oh, yeah. Oh, yeah. We have a good team on that, yeah.

Steve: There's also trouble with Live Networks RTSP streaming server that I'll just mention as sort of a public service for those who might be using it. Still more trouble with the now infamous Windows 10 Build 1809 feature update which we need to touch on. There's a longstanding zero-day in the widely used, most popular plugin for jQuery. And it looks like it's for eight years this thing has been there. And it's not even unknown to bad guys. Somehow the security industry missed it. Anyway, we'll talk about that.

And then we're going to talk about what can only be described as a consequence, I mean, it's where I got the title "Libssh's Big Whoopsie." It's just it's too wonderful a mistake. So I think - oh, and we have a great Picture of the Week. So I think another great podcast for our listeners.

Leo: Busy, busy, busy.

Steve: This number 686th podcast.

Leo: And I am not complaining that we don't have puzzlers every week because you put so much - I don't know how much people realize how much work you put in. If you've ever read the show notes, you know that this is a book Steve writes every week. So having a puzzler in addition to all the other stuff you do, I don't know, that's asking an awful lot. So we're happy, Steve.

Steve: Well, this one just dropped into my lap. And I thought, okay.

Leo: Yeah, free puzzler.

Steve: This is just too fun. This week's picture, I got a kick out of it. It had been in my queue for a while. I don't know where it came from. I'm sure somebody tweeted it to me, and I said, oh, that's just too funny. Sort of a modest prompt for an account creation, this dialogue says: "Pick a password." And then it says: "Don't reuse your bank password. We didn't spend a lot on security for this app." So just caveat emptor, you know. Just be careful. We wanted to get this out the door. We asked Moe if he would make sure that it was secure, but we're not too sure about it, so don't use your bank password.

Leo: That's hysterical.

Steve: Okay. So FreeRTOS is the number one most popular real-time operating system currently. On the first page, well, the second page of the show notes I've got a graph, or a chart, showing that the result of a questionnaire that was of 568 people asked last year what of all the various operating systems were they looking at. And, I mean, everything is there. QNX is down there at 3%. Wind River Linux, that's another popular one at 5%. TI has an RTOS, Texas Instruments, at 8%. The larger OSes, Debian is at 12%, Ubuntu at 11%.

Leo: Were they asking, though, for real-time operating systems, or just generally what operating system?

Steve: Just operating systems because I wouldn't call Debian...

Leo: Debian is not a real-time...

Steve: No.

Leo: Can you characterize a real-time operating system for people who are wondering what that is?

Steve: Yes. And so that's a great question. Essentially the idea is it's what you would want if you were literally going to be a light bulb and not a console with a file system and all that. We know, for example, that our routers have Linux in them because they've got file systems and all kinds of modules and all kinds of stuff. But imagine if you are an appliance, if you're a device, if you're a glucose meter or a parking meter or a light bulb or something that wants to use software, but wants to be more like an appliance than a general purpose thing.

So you need a few things. You have to have a processor. And it doesn't really make sense for the things that are always going to be done again to be done again, like memory management or thread management. You might want to have like several different execution threads. We've talked about execution threads a lot. They're an abstraction of a single-threaded processor where you have a scheduler that jumps the processor around between different things so that everything sort of seems to be going at once, because none of them needs all of the processor, so they can share it. So a real-time operating system will have a scheduler.

There's also typically a common memory pool. And so this real-time operating system will have a memory manager which allows the threads to say I need to use some memory for a minute or two, which it then gives back to the operating system. So the operating system doles out memory and recovers it after it's been freed. Oh, and you often have interthread communications. Sometimes you might have a couple threads that all may need access to a shared resource like the LCD on the parking meter. And so you can't have two threads using it at the same time. So there's something known as a "mutex," a mutual exclusion event object, where one thread says give me access to the LCD. And then any other thread that also wants it has to wait until the first one is through.

Well, so the point is that that's sort of a common resource. And again, the RTOS, the real-time operating system, is the one that manages those things. So these threads of execution are clients of the operating system; but it's not an OS, as I've said, like we're used to - Debian, Windows, Mac or anything. It's just it's like the bare necessities to have a processor, the processor hardware, being able to appear to be doing lots of things at once. Like, for example, the LCD is being refreshed while the buttons on the parking meter are being scanned, while the credit card slot is being checked. And maybe it's got WiFi into the city's network. All of those things are sort of - they're not heavyweight processes, but they all have to kind of happen at once. And so the RTOS does that.

So anyway, the point is that in this chart FreeRTOS, this particular one, is 28%. It is the number one operating system where this group of people, of engineers, who were asked what they're looking at, this was the one that they were aiming for. So as a consequence of its popularity, last Thursday's blog posting by Zimperium Labs, also known as zLabs - and we've talked about Zimperium in the past. They've been a source of interesting discoveries on the security side. The blog posting was titled "FreeRTOS TCP/IP Stack Vulnerabilities Put a Wide Range of Devices at Risk of Compromise: From Smart Homes to Critical Infrastructure Systems." And of course this is not good news to anybody who's using FreeRTOS. And it's been around for a long time, so like 14 years, and it supports over 40, four zero, different hardware platforms.

Leo: Wow.

Steve: I mean, every processor you could imagine. Because it's a C library that is easily transportable. If you've got a C compiler for a piece of hardware, you can port FreeRTOS onto it, and you only need a little bit of assembly language to deal with some things that C can't do because it needs to get right down to the bare metal. But anyway, so it does have a TCP/IP stack. And it turns out that there are some problems with it. It's now at v10.0.1, and everything up to and including 10.0.1 are vulnerable.

Leo: Oy.

Steve: Yeah.

Leo: And the kinds of systems this is in may not be easily updated.

Steve: Oh, that's the problem, exactly. That is, yes, that is exactly - and we know this is classic IoT stuff that may not be, as you said, subject to update. Yet if it's using the TCP/IP stack which comes along with FreeRTOS, it's got problems. So this is, I mean, there's a lot of concern in the community. When Amazon took it over, they reset the numbering and named their fork, essentially, of it "AWS FreeRTOS." And it, too, was vulnerable until fixed, up to v1.3.1. And then there's a commercial version that is sort of like a - it's a functionally identical, so you wouldn't know you weren't using it, but it's not open source.

An outfit called Wittenstein High Integrity Systems (WHIS), they call theirs OpenRTOS and SafeRTOS, although it isn't. And they're vulnerable, too. So what Zimperium said in their blog was, they said: "As part of our ongoing IoT platform research, zLabs recently analyzed some of the leading operating systems in the IoT market, including FreeRTOS. FreeRTOS," they wrote, "is a market leader in the IoT and embedded platforms market, being ported to over 40 different hardware platforms over the last 14 years. In

November of 2017," so one year ago nearly, "Amazon Web Services took stewardship for the FreeRTOS kernel and its components."

They write: "AWS FreeRTOS aims to provide a fully enabled IoT platform for microcontrollers, by bundling the FreeRTOS kernel together with the FreeRTOS TCP/IP stack." However, I was impressed with what I discovered as I dug into this a little bit because these guys said "including modules for secure connectivity, over-the-air updates [yay], code signing, AWS cloud support, and more." They said: "With the infrastructure that AWS provides, and the AWS FreeRTOS platform, developers can focus solely on innovation, thus reducing development time and costs." Sounds a little bit like an AWS ad, but it's not. This was from the Zimperium guys.

And then they go on to explain what I already did about the commercial version from Wittenstein High Integrity Systems that also has these problems. Anyway, they said: "During our research we" - meaning Zimperium Labs - "discovered multiple vulnerabilities within FreeRTOS's TCP/IP stack and the AWS security connectivity modules. The same vulnerabilities are present in the Wittenstein Connect TCP/IP component for OpenRTOS and SafeRTOS. These vulnerabilities" - and, yes, they are as bad as they get - "allow an attacker to crash the device, leak information from the device's memory, remotely execute code on it, thus completely compromising it."

They said: "We disclosed these vulnerabilities to Amazon and collaborated, and continue to do so, with them to produce patches to the vulnerabilities we detected. The patches were deployed for AWS FreeRTOS versions 1.3.2 and onward. We also received confirmation from Wittenstein that they were exposed to the same vulnerabilities, and those were patched together with Amazon."

They wrote: "Since this is an open source project, we will wait 30 days" - so that counter has begun - "before publishing technical details about our findings, to allow smaller vendors to patch the vulnerabilities."

Zimperium's Ori Karliner, who conducted the research, discovered four critical remote code execution vulnerabilities, one denial of service, seven information leaks, and a partridge in - no, no, and one other which was unspecified. So anyway, we don't have details. But I just wanted to sort of put the word out. This is, as you immediately reacted to correctly, Leo, I mean, this is big. This is, I mean, this is like the microkernel hardware platform of choice. Not all devices will be affected that use it because they won't be connected. They won't have TCP/IP. For example, the blood glucose meter probably doesn't have TCP/IP support.

Leo: Whew, yeah.

Steve: And hopefully the pacemaker doesn't. It uses something other than a WiFi link. It's like electromagnetic encoding or something with a magnet placed over the person's chest to talk to it. But, I mean, FreeRTOS is probably what's in these things because it's what you use. It's very small, so it doesn't - it leaves maximum space for the application to use most of it and provides the same set of services that the users, the developers, the designers would have to implement anyway. So why not use it?

The good news is it is getting fixed. The bad news is not everybody will be as responsible as Amazon has been. I mean, the fact that, I mean, I immediately dug into this, wondering what Amazon had done. And they said in their material: "Amazon FreeRTOS consists of the following components: a microcontroller operating system based on the FreeRTOS kernel; Amazon FreeRTOS libraries for connectivity, security, and over-the-air

updates; a configuration wizard that allows you to download a zip file that contains everything you need to get started with Amazon FreeRTOS; and over-the-air updates."

So they're, like, part of it is that they're offering this as a cloud service, the idea being, though, that they've made it like the default is that your device would check in with the Amazon cloud, probably not for free, but for a reasonable price. In fact, as I'm saying this, I'm sure we've talked about this recently - well, not too recently, but like a while ago - that Amazon was going to be doing this; and that while, yes, you didn't have to use Amazon's service, you could use your own, if the price is right, why not just fall back on AWS and let them keep your device up to date? So props to Amazon for doing this.

And the question is what non-AWS-based recent systems, because this only began with Amazon not quite a year ago, do use this always buggy TCP/IP stack with FreeRTOS and WiFi, I mean, you can imagine home connectivity IoT things, webcams and baby cams and baby monitors and security systems, that are using WiFi connectivity. If they're WiFi, they're TCP/IP; and, more likely than not, they're using a now known buggy version of FreeRTOS. So this may not be the last time we're talking about this on the podcast. Anyway...

Leo: So sad.

Steve: Again, props to Amazon. This is what we need. It's got to be in there from the beginning so that it's like a no-brainer for the developer who says, "Oh, yeah, I'd like to have the kernel updated if any problems are found in the future, thank you very much." And why not? If you have a TCP/IP stack, that means you could be on the 'Net, which means you can ping Amazon to see if they've got any news for you.

Okay. Now, unfortunately, the flipside of doing it correctly is how not to do IoT deployment correctly. And for that we have D-Link in the doghouse this week. Once again, a trio of vulnerabilities can be combined to result in a complete takeover of at least eight D-Link routers. A Polish researcher at the Silesian University of Technology in Poland discovered and reported responsibly to D-Link that eight routers that he found and tested all had, I mean, like really bad vulnerabilities. However, D-Link informed him that they would only be fixing two of the eight, that is, the DWR-116 and the DWR-111.

However, I went to D-Link - and this was responsibly disclosed some time ago. The timeline is really disturbing. On May 9th he notified D-Link. On June 6th he asked the vendor, having never heard anything, what was going on. On the 22nd of June, he received a reply that a patch will be released - okay, June 22nd - that a patch will be released for the DWR-116 and the 111, but the other devices were EOL. An announcement would be released. Okay? Then still nothing happened. September 9th, still no reply from the vendor about the patches or announcement. He wrote: "I have warned the vendor that if I will not get a reply in a month, I will publish the disclosure." And on October 12th he did so.

So they had plenty of time. I went to the firmware pages for those two routers out of curiosity, the ones they said they would update, and they were still offering the vulnerable firmware yesterday. So just something to be aware of. I would argue that the height of concern, I mean, on one hand, I guess you would have to suggest that, okay, routers have a right to be EOLed, end-of-lifed; and how long should a router vendor be expected to maintain firmware for a really old router? The problem is this looks like it's all the same firmware - the DWR-116, the DIR-140L, the DIR-640L, DWR-512, 712, 912, 921, and the 111. All of those eight routers are apparently using the same firmware, which would argue that, if they just fixed it - just fixed it - they could make that firmware available to the whole family, for people who did want to maintain security.

Okay. So what are the vulnerabilities? We've got, as I said, a trio of them. The first one - and this is publicly posted now so everybody knows this. The first one allows remote attackers - right, remote - to read arbitrary files via the classic forward slash dot dot, which we've talked about before. Dot dot, as we know, is the go back a level in the directory hierarchy, known as the "directory traversal attack," where you go /.., /.., /.., each one taking you up a level in the directory hierarchy until you get to the root. Then you move back down to the directory you want. He discovered that this works after a GET command to a /uir, which was some resource on the router. So he then posted a proof of concept where he issues a curl to http://routerip/uir// - that's the double forward slash that also performs the backup - and then etc, you know, et cetera, /passwd. Yes, that's the passwords file.

Leo: Is it only on the router? It's not reaching into the network, though.

Steve: No, it's only on the router.

Leo: Any file on the router, okay.

Steve: Well, at this point it's only...

Leo: Oh, because now you have the passwords.

Steve: Now you have the password file. And he writes: "The vulnerability can be used to retrieve administrative passwords using the other disclosed vulnerability. This vulnerability was reported previously" - get this, previously reported - "by Patryk Bogdan in a 2017 numbered CVE, but he reported it as fixed in a specific release. But unfortunately, it is still present in newer releases." So they had some sort of regression where they apparently briefly patched it, but then it became unpatched in subsequent releases.

He says: "The vulnerability is also present in other D-Link routers and can be exploited not only, as the original author stated, by double dot, but also absolutely using double slash." So that suggests that the double slash immediately takes you to the root, and then you do etc/passwd in order to get the password file.

Leo: Yeah. And fortunately, that's encrypted; right?

Steve: Okay. Vulnerability. You've been reading ahead, Leo.

Leo: No, I haven't. It's obvious this is worthless if it is.

Steve: It's so painful. Vulnerability number two: Password stored in plaintext in several series of D-Link routers. And it's so bad that he even redacted the file from his own public vulnerability disclosure, after waiting half a year. He says: "Note: I have redacted the filename in the description to XXX because the vendor leaves some end-of-life routers" - and even those that aren't, as I verified yesterday - "unpatched, and the

attack is too simple." So in other words, this is so awful that even the researcher was unwilling to disclose it fully.

So he wrote: "The administrative password is stored in plaintext in the /tmp/" - and here's where he redacted - "XXX/0 file." Now, of course, anybody can reverse-engineer any of the D-Link router firmware, get the name of that file, and now you know where to find the plaintext admin password. He says: "An attacker having a directory traversal can easily obtain full router access." Right? Because it's under /tmp/something/0. Well, the first vulnerability gives you directory traversal.

And so now he has, again, a proof of concept: $ curl http://routerip/uir//tmp/XXX/0. He says: "This command returns a binary config file which contains admin username and password, as well as many other router configuration settings." Meaning that it's binary, but they're right there unencrypted in ASCII, standing out. He says: "By using the directory traversal vulnerability, it is possible to read the file without authentication."

And, finally, as if that wasn't enough, vulnerability number three: "Shell command injection in httpd server of several series of D-Link routers." So "An unauthenticated attacker may execute arbitrary code by injecting the shell command into the chkisg.htm page Sip parameter. This allows for full control over the device internals." And proof of concept: "Log into the router. Request the following URL after login." And there he provides - now, remember, logging in is not hard because we can now be admin. Username and password, we can get that using the first two vulnerabilities. Now that we have those, we use this - and he provides the full HTML query which gives you a full shell command injection.

And so he finishes, saying exploiting all three together - "taking all three together it is easy to gain full router control including arbitrary code execution." And then he gives us a link to a description with a video. So where we are today is that, as far as we know, I mean, every router, every D-Link router he checked was using the same firmware with the same vulnerabilities. They probably only have the firmware which they have sprayed across who knows how many D-Link routers through time. We know old ones. They're not going to fix them. And even the ones that are still being maintained, they said to him on September 9th that - oh, no, June 22nd was the word back, yeah, we'll fix two of them, but not the other six. Well, those two still haven't been patched.

So I think the takeaway here is that old routers are bad. I do accept the idea that a vendor shouldn't be responsible forever. And I was tempted to go back and figure out like when these various routers were first released. But I thought, okay, I do want to get SQRL finished someday. So I don't know how old they are, but I do know that every D-Link user is now in danger, and there is no recourse currently from D-Link to get yourself out of danger. This is now public. As we know, I mean, we've been talking about problems with routers. It's going to take minutes for Mirai botnet to add this to their collection of things to try as they scan for routers on the Internet. You don't want to be using a D-Link router is what it comes down to.

And unfortunately, that message will not get - it will not ever get out to most of the users of D-Link routers in the world. All of our listeners now know and need to be concerned, unfortunately, whether their particular model of D-Link router is vulnerable. There is absolutely no reason to believe at the moment that it isn't because every D-Link router tested of eight have been, including current ones and especially old ones for which D-Link is never going to provide a firmware update. So I think at this point we need to consider routers as a commodity which ages out of use for reason of the vendor no longer maintaining it.

And certainly in this case, I mean, as we said, routers are the attack target of the year, maybe of the decade, because they're all sitting on the Internet. They've got networks

behind them that may have juicy tidbits on them. And even if not, facing outward, botnets are grabbing them up and using them for attacks and reflecting traffic and more. So I just - any listener, I mean, again, if I had the spare cycles I would add a test to GRC to check for this particular problem. If you're a D-Link router user, certainly if you have any of the D-Link routers I mentioned and that are written in the show notes, there's nothing you can do, as far as I know, except make sure that nothing is exposed publicly. Apparently this does require a public server access for someone outside to get to your router. So if that could be turned off so that there's no WAN-side admin...

Leo: If you turn off the WAN. Okay, good. All right. That's good.

Steve: Yes. Presumably that will protect you.

Leo: Unless there's a bug there, too. I mean, who knows; right?

Steve: There was no mention of mitigation in anything that I found. So I can't say one way or the other. But I do think that at this point we have to take the position that an unsupported router, obviously one where there's known vulnerabilities the manufacturer has said they have no interest in fixing, even though it appears they could just fix it once and make the same firmware available, they just don't care. So at that point you just have to say, okay, they're not that expensive. It's worth saying, you know, it's worth having a garage sale and sticking it out there on the table and say to your neighbor, well, good luck with this. I've got a new one.

Leo: Here. Here's a broken router. Good luck.

Steve: Works great. And the hackers love it.

Leo: Yeah, works great for everyone.

Steve: That's right. Okay. So Drupal. As I said at the top, if you're using Drupal 7 and 8, US-CERT, the United States Computer Emergency Readiness Team, announcement said a remote attacker could exploit some of these vulnerabilities to take control of an affected system. So it really needs to be Drupal 7 and 8 have a problem. There were five problems. Two were critical, and three Drupal's own security team said were moderately critical.

One of the two critical bugs is an injection vulnerability in the default Drupal email backend, which uses PHP's mail function, which is DefaultMailSystem::mail in both Drupal 7 and 8. When using this default mailer to send email, some variables were not being sanitized - get this - for shell arguments. As is common, when untrusted input is not sanitized correctly, remote execution may result. And in this case it does.

The second of the two remote code execution bugs exists in Drupal 8's Contextual Links module. In Drupal these modules supply contextual links that allow privileged users to more easily perform tasks related to regions of the page, thus contextual, without having to navigate to the admin dashboard. However, the Contextual Links module also doesn't sufficiently validate the requested contextual links, which allows an attacker to launch a remote code execution on those links. That is to say that the links themselves are to

code in Drupal which assumes its own variables haven't been tampered with. But you can tamper with them, use the same link target URLs, and execute your own code on that Drupal service.

So then in addition to those two baddies, the Drupal security team acknowledged that there were three other moderately critical ones. They said users of any version of 7 should move to at least 7.6. Users of 8.6.x should move to at least 8.6.2. And users of 8.5.anything or earlier should move to Drupal 8.5.8. And then they noted in their security advisory that minor versions of Drupal 8 prior to 8.5.anything are not supported and do not receive security coverage. So sites running older versions should update to the above 8.5.x release, which is currently 8.5.8. And those older Drupals, 8.5 series, will receive security coverage until May of 2019. So if you're going to jump, it's probably worth jumping to 8.6.2. Just bite the bullet now so that you can continue to get coverage.

Now, although Joomla and Drupal both lag far behind WordPress's nearly 60% domination of the content management system (CMS) market, Joomla and Drupal having 6.6 and 4.6% usage in the CMS market, respectively, even 4.6% of Drupal CMS-driven sites being vulnerable to remote code execution is no laughing matter. So I hope any admins using Drupal are signed up for security updates and are going to take these problems seriously and get this fixed.

Leo: I have to say, I mean, as Drupal users here, we love Drupal, and I've used Drupal since the beginning. You know, it's easy to - any software can have bugs, and Drupal does a good job of keeping it up to date. And they always have said for years, don't use old versions. Keep it up to date. But sometimes the jump is huge. Sometimes it's a big discontinuity, as we mentioned before, between major versions.

Steve: And in fact it's funny you should mention that. There does look like there's some things that they changed so that you do need to dig around in the code a bit. So it's not just a completely seamless jump. They're not happy with some of the functions that they have defined, and they had to change them.

Leo: They changed - yeah. So, yeah, that's always been the problem. I blame PHP. I really do. A lot of this - and I think the RTOS, FreeRTOS, as well - goes back to the choice of language. And I just - people need to use type-safe languages and capture these problems at compile time, not run time. That's just...

Steve: Amen.

Leo: Yeah. We know how to do that, you know? So let's, let's.

Steve: Yes. Declare all your variables and make sure you don't use them until you've declared them.

Leo: Yeah. Things like that, yeah.

Steve: Gee, what a concept.

Leo: Because a lot of this comes to referencing null pointers and things like that, and you can avoid that. And a compiler should catch it anyway. All right. I'll get off my high horse.

Steve: Okay. So this is sort of mostly just a public service announcement for anyone who might be using and have a publicly available real-time streaming protocol (RTSP) media server. There's a company called Live Networks that has a very popular multi-format RTSP media server known as LIVE555. It contains, unfortunately, a critical remote execution bug. Boy, is this becoming a broken record.

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

Steve: Which affects versions prior, all versions prior to last Wednesday's release of 0.93. So that just happened on October 17th. And if you haven't updated, if you're using LIVE555 streaming media server to offer anything publicly, if it's just internal Intranet then you're okay, assuming you can trust all your internal users. But there is a remote execution bug which would allow any publicly exposed version of this media streaming server to be taken over remotely. So anyway, I did not get a sense for how widely used it was. But again, it only takes one, if you're the one who uses it, and somebody's able to scan, find the server, and say, oh, thank you very much, we want to crawl inside your network through this little portal that you've created.

Leo: We don't use it, but I'm well aware of Live. They've been around for a long time. And in fact...

Steve: Yeah, Live Networks is like the real deal.

Leo: Yeah, they're one of the biggies. The only thing I know, though, is I have a real-time streamer, Facebook streamer, that I bought the Mevo cam from them, which probably uses that protocol.

Steve: Yeah, probably.

Leo: Yeah, something to be aware of.

Steve: So we're unable to stop talking about, for better or for worse, the Windows 10 October 2018 update. The infamous Build 1809 has another problem. So as we know - the good news is it's still not rereleased yet. So they found this in some preview build 18234, also known as 19H1. I've not been tracking all this cryptic insider...

Leo: 19H1 is the next one.

Steve: Okay. Okay.

Leo: So 1809 is the one that was supposed to come out now, September 2018. Now you're talking about 1903, basically, which is 19H1, the first half of 2019.

Steve: Ah.

Leo: So that's in the Insider - the Insiders are getting this now.

Steve: Okay.

Leo: You know, there's something wrong with the process at Microsoft. This is actually getting to be problematic. I mean, seriously.

Steve: Yeah. And you know, Leo, we're looking for that new era of enhanced productivity.

Leo: Oh, lord.

Steve: Was that what they were advertising?

Leo: That's something, and the most secure version of Windows ever.

Steve: That's right. I love that one.

Leo: As we well remember.

Steve: Good old XP.

Leo: I've seen a number of articles recently saying the problem really is the way Microsoft does this, which is they've got a code base. They do these short, like six-week sprints to add a feature. So they spend a long time thinking of the features. In six to eight weeks they create the feature. There's no testing at that point. They lay it into a testing version which they then test for a long period of time. This is how they did it when they did three-year-releases, but they're still doing this now for these biannual releases. And it's not an effective testing process. They need a better way of testing before they get them into these beta releases. That may be. There may be a structural problem here. We'll talk about it tomorrow on Windows Weekly, I'm sure.

Steve: So as we know, when the content, those of us who are power Windows users and who understand zip files, when the content of a zip file extraction would cause the overwrite of an existing same named file within the archive, the user - I know, Leo. Are you sitting down? The user should be prompted about the pending overwrite.

Leo: Collisions, yeah.

Steve: That's right, a file naming collision, and asked whether to replace or skip the extraction of the colliding file. It turns out...

Leo: It doesn't.

Steve: ...that Build 1809 is reportedly and reproducibly either overwriting existing zip file content without notification or silently failing and doing nothing. So the good news is this problem has been caught before, well, what I wrote - now I'm not sure. So it's been caught before the full formal re-release of Build 1809. But it was reported as being in 1809. I assumed it was the pre-release people.

Leo: Yes, yeah.

Steve: A recent tweet by an IT staff engineer at Microsoft on the Windows Insider Program Team indicated that this problem has been resolved back on October 6th with the Windows Insider Preview Build 18234, which is 19H1.

Leo: So that means they fix it for the next generation.

Steve: Yeah, well, they clearly have to fix it now.

Leo: Fix it for both, yeah.

Steve: And I know that this is just - I'm kicking this dead horse one last time. But in some recent reporting over on Computer World I noted that Microsoft's forensic analysis revealed that as many as 1,500 instances of Build 1809's pre-release testers had their files deleted and complained without Microsoft noticing.

Leo: Microsoft said it was 0.01%.

Steve: Yeah, well...

Leo: What did you just say? What percentage?

Steve: 1,500 by number.

Leo: Oh, okay.

Steve: 1,500.

Leo: That might fit. That might fit. Because a lot of people try these builds. This is a public build.

Steve: They did. They did. And but a lot got bit. And Microsoft said, oh.

Leo: Yeah, that's a lot of people.

Steve: We missed that one.

Leo: Lot of people with bad [crosstalk].

Steve: Also I should just note that, after talking about this for the last few podcasts, in between then and now I updated the machine I'm talking to you on, Leo. This camera that I'm looking at is running Windows 10. It was running Windows 10 Home, which is what came preinstalled on the little Windows 10 box that I just grabbed, just a little turnkey box. Anyway, it's now running Pro because I'm an MSDN subscriber, so it doesn't cost me anything to update.

So I updated to Pro since I definitely decided that I want to begin hanging back from each month's security updates as well as the biannual "feature" update. There's nothing I need that much each month that's worth being bit like this. And I'd rather let them, you know, I think that the security release we would know within a week if it was causing problems. So I've set that to give me two weeks. And I think I set it to 30 days for the feature update because we would know by then if it's something, if you really should put it off further.

So again, and as I mentioned, Windows 10 Home does not give you the option to delay, to defer these. You take them when they make them available. I hope that our listeners consider, after this painful set of October surprises, consider deferring, as I now have, I mean, so much so that I switched to the Pro version just so that I could have that feature. It just seems wrong that Microsoft is being stingy about that. No, we're going to make you have Pro if you want to defer. It's like, my god, okay.

Okay. So the most popular, second only to the jQuery platform itself, the most popular jQuery plugin, which has been around for 10 years, is vulnerable. This is the jQuery File Upload plugin which was released, like I think it was within a week, just like five days before the Apache team changed the way the .htaccess file is handled in Apache. As a consequence, the use of .htaccess, which is, as people know, .htaccess, anyone who's configured Apache, you're able to use that to place that file in a directory to apply access restrictions to that directory. Well, it turns out that 10 years ago, with Apache v2.3.9, the Apache maintainers deliberately disabled support for the .htaccess file, apparently as a performance improvement, because then the server would not need to check for this file every time it accesses a directory.

Leo: I remember when this happened because I used .htaccess frequently, and it broke a lot of stuff, yeah.

Steve: Yes, yes. And also the problem is this left some developers - oh, and the other reason was the Apache people didn't want the local application of .htaccess to interfere

with the server-wide configuration because they had alternative means for providing that protection.

Leo: Yeah, you just use sig file now.

Steve: Right, exactly. Okay. So the story goes that the Messaging Malware Mobile Anti-Abuse Working Group met in Brooklyn, New York two weeks ago, Monday through Thursday. Attending that meeting was Akamai's Larry Cashdollar. That's actually his last name.

Leo: What a great name.

Steve: Cashdollar.

Leo: For a guy running a CDN. Awesome.

Steve: He expected the weather to be nice, so he failed to bring a raincoat, and it rained throughout the week. So Larry was hotel bound. Having therefore nothing else to do - he couldn't walk around, sample the local fare - he decided to poke around at the various add-on packages available for Node.js at NPM, which is the packet manager for Node.js, npmjs.com.

Okay. So I read into the story pretty far, as you can tell. I'll skip the details of how he arrived at what he found. Two days later Larry posted an entry on Packet Storm titled "jQuery-File-Upload 9.22.0 Arbitrary File Upload," with the description "jQuery-File-Upload versions 9.22.0 and below" - meaning all previous - "suffer from an unauthenticated arbitrary file upload vulnerability that allows for" - you guessed it - "remote code execution."

Okay. So first of all, it's always been the case that allowing uploaded files to a server is extremely fraught, I mean, it's inherently fraught with danger. This is not to say that it's not possible to do so safely. But few things should instill more fear in the heart of the responsible web designer than enabling file uploads. How many times have we here on this podcast covered buffer overrun exploits in image renderers? As we've said, interpreters are very difficult to get correct. And there have been JPEGs and GIFs and PNGs, I mean, all these image formats.

Leo: Well, and even JavaScript, to be honest.

Steve: Yes, yes.

Leo: We had that problem. We were bit with malware. We had an old Drupal plugin that allowed somebody to hack our code. And he was able to upload arbitrary code to an executable directory. And it was JavaScript. Or maybe it was, probably was PHP, come to think of it. But, yeah, I mean, [crosstalk] anything you want.

Steve: Yeah. So imagine that you're able to put anything you want anywhere, and then you invoke it from a URL outside...

Leo: And that's the problem with PHP. It's a URL-invokable protocol.

Steve: Yes, yes.

Leo: So stupid.

Steve: So what we have here is much worse.

Leo: Worse?

Steve: Than even that, yes. Due to a presumably, as I mentioned, well-meaning change that the Apache group made back in 2010.

Leo: Well, that's one of the ways they used .htaccess is to keep people from uploading files to directories; right?

Steve: Yes, yes.

Leo: To block a directory.

Steve: So starting with that version of Apache, 2.3.9, the httpd server offered an option that would allow server admins to ignore custom security settings made to individual folders via the .htaccess files. This setting was made for security reasons, was enabled by default, which means, as I've often said, the tyranny of the default, enabled by default, and remained so for all subsequent Apache httpd server releases. So in the process this jQuery file upload, which is the most popular plugin, second only to the platform itself on GitHub, its assumption that it could protect its file uploads using a local .htaccess file was rendered, since 2010, invalid.

So on GitHub this plugin says: "File Upload widget with multiple file selection, drag-and-drop support, progress bar, validation and preview images, audio and video for jQuery. Supports cross-domain, chunked, and resumable file uploads. Works with any server-side platform - Google App Engine, PHP, Python, Ruby on Rails, Java, et cetera - that supports standard HTML form file uploads." In other words, it's been around for years. Not surprisingly, lots of people use it. It's very popular.

So our Larry Cashdollar says in his vulnerability disclosure: "The code in" - and then he cites the URL on GitHub - "doesn't require any validation to upload files to the server. It also does not exclude file types. This allows for remote code execution." As Larry wrote in his description of this discovery, he said: "I started looking through the package's source" - this is during a rainy day in Brooklyn - "and found myself peering at two PHP" - there's your favorite acronym, Leo, or abbreviation - "files under the directory server/php. The files are named upload.php and UploadHandler.php. The upload.php file calls the main file UploadHandler.php where all of the file upload code resides."

He says: "I also saw that all files were uploaded to the files/directory in the web server's root path. I wrote a quick command line test with curl, and a simple PHP shell file confirmed that I could upload a web shell and run commands on the server."

Leo: Yikes.

Steve: And it's literally one line, and he used example.com just for the safety of posting, where he provides shell.php. And shell.php is a simple PHP program that just launches the system command shell into the HTTP response. He says: "A browser connection to the test web server with cmd=id returned the userID of the web server's running process." He said: "I suspected this vulnerability had not gone unnoticed." And get this, Leo. "A quick Google search confirmed that other projects that used this code or possibly code derived from it were vulnerable. There are a" - get this - "a few YouTube videos demonstrating the attack..."

Leo: Is there anything YouTube can't do?

Steve: "...for similar software packages." Okay. So this is of extra concern because the jQuery File Upload bug is not some obscure widget. It is an extremely capable, as we noted, it is extremely capable and an extremely popular add-on - get this - having been forked on GitHub 7,828 times to create descendant projects of that base package which are widely spread throughout the industry, deployed on websites far and wide. So that means right now, once again, this is public, and all PHP-based sites which chose to use this jQuery file upload in its 7,828 variations are currently subject to any attacker uploading any file of their choosing, executable, and running it on that hosting server. Since discovering this critical vulnerability, Larry's been busy. He's examined 1,000 out of the 7,828 forks of the plugin. Every one of them was also exploitable.

Leo: Wow. Of course, because you just copy the code, yeah.

Steve: Exactly. And still worse, it turns out that at least some of the underground hacker community have been aware of this widespread backdoor for years. As ZDNet explains in their coverage under the title "Zero-Day in popular jQuery plugin actively exploited for at least three years," they said: "A fix is out, but the plugin is used in hundreds, if not thousands, of projects." They say: "Patching will take ages."

ZD said: "For at least three years, hackers have abused a zero-day in one of the most popular jQuery plugins to plant web shells and take over vulnerable web servers. The plugin is the second most starred jQuery project on GitHub, after the jQuery framework itself. It is immensely popular, has been forked over 7,800 times, and has been integrated into hundreds, if not thousands, of other projects, such as CMSes, CRMs, Intranet solutions, WordPress plugins, Drupal add-ons, Joomla components, and so on. A vulnerability in this plugin would be devastating, as it could open gaping security holes in a lot of platforms installed in a lot of sensitive places."

They say: "This worst-case scenario is exactly what happened. Earlier this year" - and as we know it was a few weeks ago - "Larry Cashdollar," they write, "a security researcher for Akamai's SIRT (Security Intelligence Response Team), has discovered a vulnerability in the plugin's source code that handles file uploads to PHP servers." And on and on and on.

Cashdollar says that attackers can abuse this vulnerability to upload malicious files on servers such as backdoors and web shells. He said the vulnerability has been exploited in the wild. "I've seen stuff as far back as 2016," he told ZDNet in an interview. And apparently the vulnerability was one of the worst-kept secrets of the hacker scene and appears to have been actively exploited even before 2016. Larry found several YouTube videos containing tutorials on how one could exploit the jQuery File Upload plugin to take over servers. One of the three YouTube videos that Larry found and shared with ZDNet was dated August 2015.

So actually I'll note, I've mentioned this before, but it is for exactly this reason that my own forthcoming PHP-based - as I mentioned, I did go to v7.2 because, since I was setting up a server fresh, why not? The SQRL public forums are hosted on that. But they are running on their own physically separate and network-isolated machine that has no connection to any of the rest of GRC's network because there's just no way to trust a system like that. These sorts of things are going to happen. And while, yes, as long as somebody is wired into security events in the industry, you can keep up with things, I can't have code that I didn't write that was sourced from hundreds of different places in order to glue together a solution. I can't have that on my network.

So yikes. I hope that anybody who is aware of what's going on, knows that they used a descendant of this jQuery File Upload, will recognize that the author had the best of intentions. He worked with Larry. Initially he could not duplicate what Larry was seeing because the author's PHP test server was not configured to ignore the .htaccess file. By default, as we know, for eight years, since 2010, Apache has been. So anyway, the author immediately put file type restrictions on last week's fix of this. But it needs to be fixed comprehensively. Wow.

Oh. And we've talked before about the dangers of lapsed domains. Since domain ownership is valuable, we have systems in place to rigorously protect that ownership. As a consequence, over time, trust is created since domains are rarely successfully hijacked. But what about when a domain that's in use for some purpose is deliberately abandoned, and its name is allowed to lapse? We've talked about the problems of overlapping security certificates in the past where somebody would have a certificate that was still valid for a domain that was reregistered. Lapsing domains is something we see all the time since the Internet, as we know, is a constant churn with domains being abandoned and created.

We sometimes find that a link we haven't visited in a long time now takes us to some weird search engine or a marketing page or something. Advertisers long ago figured that lapsed domains would see some traffic, some non-zero level of traffic. So they began snatching up any that lapsed to camp out their own nonsense there. And in fact that happened to me. I used to - I referred in Podcast 44 to a domain that I had, grcmail.com, which I deliberately allowed to lapse. And if you go to grcmail.com, my uBlock Origin immediately blocks it because there's some horrible marketing junk. Some marketer grabbed that domain name when I allowed it to expire because I didn't want to keep paying for it every month, and I decided I wasn't ever going to use it. And now somebody's camped out there.

But what happens when a supplier of active content, like in this case embedded web page scripting, decides to throw in the towel and no longer host something that they have been providing for years? The Sucuri blog tells the story of that very nicely. I've paraphrased from what they wrote. They said when Twitter announced their new design for Tweet and Follow buttons back in October of 2015, so just about exactly three years ago, marketers across the web developed a mild anxiety, Sucuri wrote. The new design came with a decision to nuke their beloved Tweet Count feature. Social signals can be a huge credibility indicator for visitors and site content. So who doesn't think there's a

psychological relationship between the number of social shares and the credibility of the content that's there? It's social validation, they write, plain and simple.

Naturally, bloggers and website owners with an aversion to change started looking for alternative solutions that offered the same feature. Marketers breathed a sigh of relief when easy-to-use services started popping up to offer Twitter share counts, and one specific one called "New Share Counts" quickly gained traction. It even integrated with other existing social share plugins, they write, like SumoMe, AddThis, and Shareaholic.

Setting up New Share Counts on a site was simple: Navigate to newsharecounts.com, which by the way is gone. Link your Twitter account and website, then add two lines of code to the bottom of every page you want to track shares from. And so there's script type text/JavaScript. And then it says the source for the scripts is //newsharecounts.s3-us-west-2.amazonaws.com/nsc (as abbreviation for new share counts), nsc.js. So what happens is, naturally, any time a visitor pulls a page from a site that has decided to use New Share Counts, the script at the bottom of the page pulls nsc.js from that AWS bucket and runs it. Right?

So this summer, after not quite three years, in July of 2018, newsharecounts.com was abandoned, and its service was discontinued. The service's original provider was about as responsible as one might hope. He posted a notice on his site referring visitors to opensharecount.com or - and apologies to Leo - twitcount.com.

Leo: Ay yi yi.

Steve: Uh-huh.

Leo: I can't win.

Steve: Yes. Okay. So, however, Sucuri, they did some digging. They found that more then 800 websites, more than 800 websites did not get the message. They continued to embed the now discontinued script, which as I mentioned pulls a file, that file nsc.js, from an AWS S3 bucket. That nsc.js script originally loaded another script from newsharecounts.com, so it was a redirect, which after the domain was discontinued stopped functioning.

On October 3rd of this month, three weeks ago tomorrow, the original AWS S3 bucket was canceled. Some very clever and nefarious hacker had apparently been waiting, like literally checking it daily, for just that to happen, since the next day they registered a new AWS S3 bucket under the same name and uploaded a malicious version of the nsc.js script.

Sucuri wrote: "During a recent remediation investigation, our Remediation Team led by Ben Martin noticed malicious redirects being served by websites using the New Share Counts service." In their report, in their blog posting, they repeat that URL of the newsharecounts.s3-us, the idea being that the original one was abandoned, releasing the registration of that AWS bucket. A bad guy grabbed it, much as someone would grab a domain name that had been abandoned, and hosted malicious script there under the same name, nsc.js. I've got a picture of it obfuscated in the show notes.

The Sucuri guys decoded, decrypted this. They said: "Once decoded, this script contains absolutely no reference to Twitter or New Share Count. Instead, this snippet of code adds 10 fake browser history entries for the page that hosts it. An interesting feature of these

history entries is that it prevents the user from choosing the previous page from the back button. When the user loads the page, a malicious event handler is added. This handler waits until the user taps the back button on their device or tries to navigate to a previous page. It then fires an event, which causes the browser to open to the following destination." And they give the URL of it. It's a scam page instead of taking them back to the previous page.

They said: "This behavior is only seen on Android, iOS, and other mobile devices, and only under the condition that the user taps the back button in their browser. Users on other devices won't notice anything suspicious, except for maybe the lack of Twitter share counts that would exist if the original New Share Count script actually functioned."

They said: "At the time of writing, 800-plus sites use this script, some of which are fairly popular, who simply wanted to show off their social signals. Now they are instead maliciously redirecting all of their mobile users to this malicious traffic. Loading third-party scripts and elements," they write, "on your website always opens up the risk of unwanted content being served on your site without consent, especially when they come from an expired or unmaintained service." So, yikes.

And Leo, about a month ago I mentioned that I had received a DM from someone whose - the DM I sort of ran out of time in preparing the show. He provided a lot of information, which was sort of too much, I mean, it was sort of a little bit superfluous. And I didn't have time to get it into shape. But I did this time, and I didn't want it just to go unnoticed. He sent this on Saturday at 8:03 a.m. was the time of his tweet.

The tweet's subject or first line was "SpinRite giving a blast from the past." He said: "Hi, Steve. I've been a long-time listener to Security Now!, and each week someone gets to tell his SpinRite success story. I've been owning a copy of SpinRite for years and occasionally tried it on hard drives when one of my numerous RAID boxes would drop, and SpinRite would perform its magic.

"Two weeks ago on a Sunday, our Active Directory master server dropped off the network, and when I came in on Monday I immediately checked it. It had crashed and would no longer boot. The Active Directory server was using some older SAS" - which is a form of SCSI interface - "drives, in RAID 0, on a controller without BIOS support." Now, what that means is that, when you boot it, SpinRite wouldn't naturally see it.

He says: "So I needed to add the old DOS aspi8xx.sys and symdisk.sys drivers" - any of us old DOS hands around here will remember those days - "to SpinRite's boot config. That allowed SpinRite to see the drive. Running a Level 2 SpinRite scan on the drive found and repaired two errors on the drive, one which the drive fixed on its own and one where SpinRite's Dynastat recovery kicked in to work at repairing the sector."

He said: "Since Dynastat took a while, I left it running overnight. And when I came back in the next morning, SpinRite showed me a green screen, stating that it had successfully completed all its tasks. Just to be double sure, I reran the Level 2 scan, and SpinRite zipped through the drive in 90 minutes. Then I reinstalled the drive into its original server. The SAS controller recognized the drive and reassembled the RAID 0 and virtual drive." He says: "Windows 2012 Server booted right up and worked again without issues. Thanks for a terrific product and for letting me have my very own SpinRite story. Kind regards, Stephan Budach in Hamburg, Germany." And Stephan, thanks for sharing.

And I'll just mention that, moving forward, just for exactly this reason, I'm going to continue to support config.sys drivers. They're not often needed. Anybody with a motherboard made in the last 10 years will be able to use the AHCI chipset which SpinRite 6.x will then be able to talk natively to. But for the sake of infinite backward

compatibility, 6.x moving forward will continue to support config.sys drivers for instances like this.

Leo: What a blast from the past.

Steve: Yeah. Libssh's Big Whoopsie. This was just, as I said, it's just too fun. Okay. So I'm sure most of our listeners are familiar with SSH. For those who are not, it is sort of the logical evolution of the original telnet. Telnet was a service, a server and client telnet protocol, which gave you basically a remote console. You could use a telnet client which looked like a console window, connect to a remote server over TCP, and you got a prompt. Typically you had to log in. It would give you a challenge of a login and then a password. And you'd be logged in as if you were a user sitting in front of the actual console of that same machine.

The problem was telnet didn't itself offer any inherent security. It relied upon a strong username and password. So which today just seems quaint to us, the idea that you would have a telnet port open. And this is why, when I've talked about routers that do, I just put, you know, you put your head in our hands. It's like, really? It's just an invitation for a bad guy to hook up and start guessing usernames and passwords. There's often no monitoring, no limit to how many they can guess, no mitigation against that at all.

So what's happened is, over time, responsible OSes have started refusing to even allow it. I was using telnet in my multilayered security for a much earlier version of FreeBSD. I think it was v4. And we're at 10 or 11 now, or more. And it's funny because, when I recently set up a new Unix machine, I just sort of defaulted to - because, I mean, I've got layered, layered security. There's no way I would ever allow a port 23 to be exposed. And it didn't. But the machine itself, because of the security environment, could have port 23 enabled and other things were connecting to it.

Unix, FreeBSD Unix refused. It was like, what? Are you crazy? No. No. We will not let you have telnet exposed on a public interface. And it's like, yeah, but I know what I'm doing, really. No. We don't care. It's like, okay, fine. So SSH. SSH, whereas telnet ran on port 23, SSH runs on 22. In the acronym SSH is Secure Shell. And so it is a next-generation, much more secure solution. I use it, as I'm sure you do, Leo, with certificates. We've talked about this before.

Leo: Actually, I use an SSH key, but same idea; right?

Steve: Okay, yes, right. So you're using some...

Leo: Public key, yeah.

Steve: ...high-entropy thing. Yes, exactly, a public key that no hacker - it's not a matter of guessing usernames and passwords. You can have those, too. But you also need to have a private key that no hacker's going to have.

Leo: I guess that's the same as a certificate. I use...

Steve: It is. It is a certificate.

Leo: Yeah. I use OpenSSH to generate - or maybe I use PGP to generate a key, and then the public key gets stored on the server, and I have access to the private key. Or is it the other way around? Yeah, public key [crosstalk].

Steve: Yeah, probably the private key - yeah, well...

Leo: No, the private key is - I can't remember.

Steve: Yeah. And actually you're able to key each end. So it's able to do mutual authentication.

Leo: That's right, yes.

Steve: The server has a certificate. Your client has a certificate. And they check each other in order to validate the connection.

Leo: Right.

Steve: So it sort of goes both ways. That way you know, well, that protects you from a man-in-the-middle attack because each end is validated with static certificates. Okay. So that sort of sets the context. It's all about security. I mean, and you need to rely on that; right? Okay. So what happened here with libssh? The design of SSH uses a state machine. A state machine is sort of a logical abstraction. They're very handy. They're a way of simplifying a complex system so that the system is, at any given time, is in a single state. And so like it might be in connection accept state. Then it accepts a connection, and so then that event of accepting a connection moves it from connection accept state to connection accepted state. Okay? And so from there a number of things are possible. It might then be able to receive a request for - like bring up the protocol event.

Anyway, so it's a state machine where at any given time it's only in one state, and different events cause it to move from where it is to where it is next. Okay. So Peter Winter-Smith of the NCC Group, who we've also spoken of before, or the group, discovered in this libssh what can only be described as a deeply embarrassing flaw. This flaw was introduced four years ago, back in 2014, with libssh version 0.6. So 0.6 and above all have an authentication bypass vulnerability in their SSH service. And as we were just saying, authentication is the whole reason you have SSH. So if you can bypass authentication you're back to telnet. And in fact worse because sometimes SSH, because the authentication is so good, it authenticates you to the platform that you're logging into also.

So, okay. During normal SSH protocol, the client requests a user authentication challenge by sending the server a message that's known in the SSH protocol, SSH2_MSG, for message, USERAUTH_REQUEST. So that's the client sending that protocol message, SSH2_MSG_USERAUTH_REQUEST. If all goes well in their back-and-forth handshaking of swapping public keys and private keys, generating nonces, signing nonces with the public key, and then sending it back, and the other end verifies it with their private key, I mean, all this rigamarole to be like ultra-galactically super sure that

you've authenticated both endpoints. Everybody's got privacy and lots of bits in there, entropy, and all that's happening; right?

Finally, if all goes well, the authenticating code at the server end will emit an SSH2_MSG_USERAUTH_SUCCESS message to indicate that the authentication succeeded and the user is now authenticated. But believe it or not, what Peter Winter-Smith discovered, and I can just imagine the look on his face, this buggy for the last four years libssh state machine does not discriminate between the source of the messages. He discovered that a remote connecting client can skip all of that hassle and simply send the SSH2_MSG_USERAUTH_SUCCESS to the server and be immediately authenticated and given full remote shell access on the target system.

Leo: Eliminate the folderol.

Steve: So do not request authentication, which you may not be able to provide, after all. Simply send "We've just successfully authenticated."

Leo: You know who I am. Come on.

Steve: And the server says, "Oh, okay, welcome." Okay. So how widespread are these publicly accessible libssh servers? Well, to answer that question, everybody immediately goes to Shodan and does a Shodan scan, which turns up 6,000 candidate currently public libssh-based servers. Closer probing confirms that there are approximately 3,000 servers connected to the Internet that use the library. And roughly 18 to 1,900 of them, that is, 1,800 to 1,900 are currently vulnerable and are using this vulnerable version of the library.

Who uses libssh? Well, Red Hat indicated that Red Hat Enterprise Linux 7 Extras uses it. F5 Networks BIG-IP load balancers use it. KDE uses libssh for its SFTP file transfers. Cisco is examining their many devices since they also chose libssh. GitHub implemented their git ssh server with libssh. But the particular way GitHub uses it protects them from this exploit. There's a remote desktop solution for Linux, known as X2Go, as in X server, X2GO. It uses libssh. Csync, a bidirectional file synchronizer. Remmina, I can't pronounce it, Remmina, R-E-M-M-I-N-A, has a GTK+/Gnome Remote Desktop client which uses it. XBMC is a media player and entertainment hub for digital media, which uses it. And the GNU Gatekeeper, which is a full-featured H.323 gatekeeper uses it.

On the next page of the show notes, Leo, I've got them broken down by organizations. And yes, Verizon Wireless has 602 servers currently vulnerable. Number two is Sprint PCS. It looks like about 325, maybe 330. Then we've got Fastweb, University of Maryland, Telstra Internet, Korea Telecom, Telefonica, Amazon.com. Whoops. They've got about 75. Comcast Business, looks like they have about 45. And another instance of Telefonica with about 40.

And then the green chart is a breakdown by version, showing that 0.6.0 is by far the most used at 1,275, and then it falls off pretty rapidly. But 0.6.3, 0.7.0, it looks like only - no, actually that's, yeah, 0.6.0 and looks like - I can't tell if that's 0.2 or 6.2. But that's the second most. And then so forth. So anyway, since the announcement, there are now at least four proof-of-concept scripts which have been uploaded to GitHub. One is a Python script. There's libssh authentication bypass, that's blacknbunny. Hackerhouse-opensource has put one up. And Vulnhub has put up one for the exploit.

Also the guys at Leap Security posted a Python-based scanner to search for vulnerable versions of libssh. It's there at LeapSecurity.io on their blog. I've got a link to it in the show notes. It's got two modes: a passive mode which determines if systems are vulnerable by banner grabbing, and an active mode which attempts to actually leverage the vulnerability to bypass the server's authentication to confirm its vulnerability.

So needless to say, those 1,800 to 1,900 servers are under attack now. And they need to get fixed because there is a publicly available scanner. There is open source, four different open source proof of concepts presented showing anybody how to do it. And you can have a shell on any of these machines. So it's not as if this thing were OpenSSH, which everybody uses. It's a lesser used libssh library. But it's been used, and boy does it have a mistake. Yeah, we don't really care who says the authentication succeeded. Just claim it, and we'll let you go.

Leo: Wow.

Steve: Okay. And we wrap with this week's Security Now! Puzzler of the Week. And Leo, you need to be on your best behavior because...

Leo: I'm not going to guess. No guessing.

Steve: Well, you can guess, but not until after we stop recording.

Leo: Not out loud, okay.

Steve: Because we want to let our listeners - I'm going to pose this as a design problem. It's just it's perfect. Okay. So the back story is vending machines made by Argenta, a successful Italy-based purveyor of coffee services, are also widely used to dispense soft drinks, dry goods, cigarettes. They're a very popular Italian-based vending machine company. Some of these machines were located on a university campus - you never want to put your high-end vending machines on a university campus, Leo, that's just asking for trouble - where they came to the attention of Matteo Pisani, an Italian hacker who is also the CTO of Remoria VR. Matteo was made curious by the fact that these high-end vending machines support both Bluetooth Low Energy (BLE) and Near Field Communications (NFC) technologies to link to a companion Argenta Wallet smartphone app. You can see where this is headed.

The app was poorly designed, easily reverse engineered, and exploited. The "Wallet," I put that in air quotes, maintained the user's balance, which the vending machines naturally assumed to be valid. Presumably, unlike last week's Picture of the Week where remember the coffee machine was offline for I guess an hour and a half while it updated itself during the middle of the day, these high-tech vending machines lacked a connection to the Internet to enable independent verification of the user's balance. This lack of verification allowed Matteo to easily manipulate the Wallet's database, which he was able to find and decode and reverse engineer and give himself, in the screenshot that was shown, 999 EU of credit, which the machines readily accepted, thus giving him free soft drinks, dry goods, and cigarettes, probably for the rest of his life, except that he chose to responsibly disclose the problem to the company and gave them...

Leo: Wow, that's nice.

Steve: I thought it was, yes, very nice. Okay. So he did report and responsibly disclose this to the parent company, who presumably will work out a solution. So what solution? We've often talked about the impossibility of encrypting a DVD which a consumer's DVD player must be able to decrypt right there in the living room. Therefore the keys to decrypt it must be present in the accessible player for it to work. But the security model here is subtly different. And using only the concepts we have often discussed here on the podcast, there is a wonderfully simple, robust, and utterly uncrackable solution for the design of this system. And Leo has assumed his Thinker pose.

Leo: It sounds a little bit like SQRL.

Steve: Well, I've talked about it. It uses those sorts of technologies. I want everyone to think about this for the next week. Next week I'll describe how I would solve this problem, and everyone can check to see whether they came up with the same simple, elegant, and uncrackable solution - or, who knows, perhaps something even better. So that's the Security Now! Puzzler for the Week. You have vending machines. You've got a smartphone app. The app can be reverse engineered. You can't protect it. What technology could we think of that could be used to create something absolutely elegant that would work and be uncrackable? Think about it for a week.

Leo: In the sense that the wallet, the integrity of the wallet would be assured. Is that what we're trying to do? Or any sense?

Steve: In whatever sense is necessary.

Leo: Okay.

Steve: Just think about crypto things.

Leo: I've invented this thing called a "coin." All right. I like this. So everybody's going to get to work, think about it, and we'll talk about the answer next week on Security Now!. That's awesome.

Steve: And I think people will like it. Or at least they will like my solution, and we'll see if they got the same one because it's right there. And if they didn't, it'll be like, doh.

Leo: And please don't use the word "blockchain" in your answer. That's all I'm going to say. If you can't watch live or be here live, you can always get on-demand versions of the show. Steve's got audio and transcriptions, so you can read along as you listen, at GRC.com. While you're there, pick up a copy of SpinRite, the world's finest hard drive maintenance and recovery utility, as you know.

Steve: It is.

Leo: Lots of freebies there, yeah, including SQRL and all the ongoing development conversations around that. And I recommended - I have a friend who has trouble sleeping, and she's been using the Sleep Formula with great results.

Steve: Oh, yay.

Leo: Yeah. As do I. I can go on and on. But, you know what, it's a great site. Lots of stuff, a real rat hole. Prepare to spend an afternoon.

Steve: That's right, it's a rat hole.

Leo: It's a rat - in a good way. You go fall in, it's a rabbit hole. Maybe that's better. You fall in, and you just don't want to leave: GRC.com. We have it here at our own particular rat's nest, TWiT.tv/sn. We actually have audio and video, although god knows why you'd want to watch us. But if you do - unh-unh. Steve says unh-unh. If you do, you can get it there, TWiT.tv/sn. Or of course, if you subscribe, find your favorite podcast application on your phone or your tablet or whatever and subscribe, you'll get it every Tuesday afternoon, the minute it's ready. Steve, thanks. See you next time on Security Now!.

Steve: Okay, my friend. Till then. Bye.