SECURITY NOW!

Transcript of Episode #944

Abusing HTTP/2 Rapid Reset

Description: How have ValiDrive's first 10 days of life been going, and what more have we learned about the world of fraudulently fake USB thumb drives? Should Passkeys be readily exportable, or are they better off being kept hidden and inaccessible? Why can't a web browser be written from scratch? Can Security Now! listeners have SpinRite v6.1 early? Like, now? What was that app for filling a drive with crypto noise, and what's my favorite iOS OTP app? And couldn't Google Docs HTML exported links be being redirected for user privacy? After we address those terrific questions posed by our listeners, we're going to take a look at the surprise emergence of a potent new HTTP/2-specific DDoS attack. Is it exploiting a zero-day vulnerability as Cloudflare claims, or is that just deflection?

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-944.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-944-lq.mp3

SHOW TEASE: It's time for Security Now!. Steve Gibson is here, and we have a very big show for you. Lots of information. But most importantly we're going to talk about the largest DDoS attack of all time, how it happened, why it happened, and what companies and, more importantly, server makers can do to stop it. That's next on Security Now!.

Leo Laporte: This is Security Now! with Steve Gibson, Episode 944, recorded October 17th, 2023: Abusing HTTP/2 Rapid Reset.

It's time for Security Now!, the show where we cover your security and safety and privacy and all that good stuff online, sometimes offline as well, with this man. Now, I usually point over my left shoulder, but he's over to my right now. No, my left. Here he is, Steven Gibson of GRC.com. For those that don't watch the video, I am at my mom's house doing the show from there instead of the usual studio. So we're a double box, you and I.

Steve Gibson: Yes, we are. And for anyone who's wondering, wait, wait, wait, wait, wait, wait, wait, wait, wait, what happened to Part 2 of the Top 10 most significant cybersecurity misconfigurations that was promised for this week, well, I did say that unless something even bigger and more important and timely came along, that's what we would be doing. And wouldn't you know it, that's what happened. There is a serious new way of abusing the most - I was going to say the most recent HTTP protocol, but that's actually HTTP/3. This is HTTP - which hasn't been well - has not been widely deployed. HTTP/2 is, and it's been giving the big cloud providers a run for their money, and for a while now. I mean, so it is a serious new form of DDoS attack against HTTP/2-capable servers.

Leo: We had that story on Sunday on TWiT, that the Google had suffered the largest DDoS of all time.

Steve: By hugeness. I mean, like...

Leo: And I think that was HTTP/2; right?

Steve: That was that story. So we're going to cover a bunch of our listeners' feedback and questions, which brings us some interesting topics such as how have ValiDrive's first 10 days of life been going, and what more have we learned about the world of fraudulently fake USB thumb drives? Should Passkeys be readily exportable, or are they better off being kept hidden and inaccessible? Why can't a web browser be written from scratch? Can Security Now! listeners have SpinRite 6.1 early, like now?

Leo: Ooh, what?

Steve: Uh-huh. What was that app for filling a drive with crypto noise, and what's my favorite iOS one-time password app? And couldn't Google Docs HTML exported links being redirecting be a user privacy feature? After we address those terrific questions and a few more posed by our listeners, we're going to take a look, as I said, at the surprise emergence of a potent new HTTP/2-specific DDoS attack. Is it exploiting a zero-day vulnerability as Cloudflare claims, or is that just deflection?

Leo: Oh, that's interesting. Cloudflare, one of the biggest DDoS mitigation companies in the business.

Steve: Right.

Leo: So they would have some inside information.

Steve: They got caught with their DDoS down, baby.

Leo: They might have some incentive for dissembling.

Steve: Uh-huh.

Leo: Oh, interesting stuff. And now our guy on the job for security, and I guess you're probably the number one place people send silly pictures these days.

Steve: I think we've pretty much cornered the market on that. This one...

Leo: So good.

Steve: So what we have is at least a four-story building because we can see four stories' worth of it, although the picture's cropped a little bit. And you know how code requires that you have often two exits from like a domicile. In case a fire broke out near the front door, you need to have like a back way out; right? Like the traditional fire escape. Well, that appears to be what we have here because we see a line of back doors. You know, there's no numbers on the doors. They don't look like front doors. They look like, you know, backdoors.

Leo: They're escape doors, yeah.

Steve: Yes. Unfortunately, if you were to open the door and run out, that would be bad.

Leo: Wiley Coyote off the cliff.

Steve: Because, like, for a reason that's difficult to explain, the stairs are shifted off to the side. So there's like a spiral staircase where each of the landings of the spiral would correspond with the height at the bottom of the door, like you'd expect, except that they're about, what, six feet off to the side? And, you know, Leo, what occurred to me is that they're lined up with the windows because there's also windows off to the side. And anyway, I gave this picture the title "Slavish devotion to the specifications." As if, like, maybe somebody was looking at the plans and got the windows and the doors exchanged.

Leo: I can see the memo.

Steve: And they didn't stop to ask, why are we putting the stairs over here where there's no doors?

Leo: I can see the memo to tenants saying, now, just a reminder, when you're at the fire escape, take a sharp right; okay? Very, very important. You know, somebody in the chat room said, and this might actually be right, that maybe they're moving - they're prefabbed, and they're moving those into place. And some smart photographer got the shot before they actually got moved over.

Steve: Oh, that's good.

Leo: That would make sense. They do prefab those things, I know.

Steve: That would explain it. So they, like, tilted it up, and it's against the wall.

Leo: Yeah, and then slide it over.

Steve: And now they're going to move it over and then anchor it to something.

Leo: At least I hope that's the case because, if they built it like that, somebody ought to get fired.

Steve: Yeah, I don't think that would pass the code inspection. Let's see. How's your exit? Well, don't walk through it.

Leo: Oh, that's easy. When the inspector wants to see the exit, you just show them the door.

Steve: That's right.

Leo: See through there.

Steve: See you later, that's right. Okay. So I wanted to follow up a bit on ValiDrive. We're 10 days out from its release, and we're currently seeing a seven-day moving average, which is what my server produces, of 2,443 downloads per day, with a total of 27,603, total when I looked this morning.

Leo: That's amazing.

Steve: So that download rate currently places it at the top among GRC's freeware goodies. It has managed to push GRC's always number one, which is the DNS Benchmark, that's currently being downloaded 1,867 times per day, it's pushed it to the number two spot.

Leo: By the way, that's been out for years. That's pretty impressive.

Steve: I was going to say, yeah, the DNS Benchmark has demonstrated somewhat astonishing endurance. It's got more than 8.5 million downloads since its release.

Leo: Wow. Well, I still recommend it all the time. I mean, it's really...

Steve: Well, there isn't really an alternative. It is the right one. And it does run under WINE, so Linux people are able to run it, too.

Leo: Well, that's one problem I have with ValiDrive. You really have to do it on Windows; right?

Steve: Yes, you do. And that's one of the questions that we'll be addressing here in just a second. But anyway, so I think it's fair to say that the DNS Benchmark has become a staple on the Internet. And I expect that ValiDrive may have similar long-term legs, although I would expect at probably a lesser average rate once the news of its existence has settled down. Anyway...

Leo: I just hope that the people selling crappy hard drives on Amazon don't figure out where you live because you're kind of cutting into their business there, Steve.

Steve: Well, so, for example, I got some feedback from someone whose handle is qqq1. I don't know why. It's easy to type over there on the left-hand side of the keyboard.

Leo: It is, actually. That's smart.

Steve: Anyway, he said: "I knew these would be trash," he said, "'free gifts' with stuff we order. We have a large pile of them." And then he sent me in his tweet a screenshot of the ValiDrive results. So the drive's declared size is 32GB. The valid size is 512MB. And so you can see in the graphic there's like a little strip of green where there's actual storage, and then just a sea of red where there's nothing.

Leo: And the point here that people who didn't hear your original discussion of this and why you did it, is it's really easy to have the firmware report a larger size than is actually available.

Steve: Exactly. Exactly.

Leo: So you can make a cheap 500MB drive and have the firmware say, oh, no, this is 32GB. And the sad thing is a lot of software won't even question it and will just write to these non-existing sectors.

Steve: No software will.

Leo: Oh, wow.

Steve: I mean, it will accept a format because formats no longer actually go out and read the data. A long format does. But even a long format won't catch it because the drive reports no error when you read from this nonexistent memory.

So, okay. It pretends to be a 32GB drive, but it only contains half a gig of nonvolatile storage. So what I was wondering was okay, it's a free gift drive. Why lie about its size?

Leo: Why lie?

Steve: It's free.

Leo: Well, it's really a gift horse, and you should look it in the mouth.

Steve: Yes. So as we know, the drive's electronics proactively lie, as you said, about the amount of storage that it contains. It cannot be a mistake; right? I mean, this is deliberate.

Leo: Yeah.

Steve: It has to be deliberate fraud. So then I had the question, you know, but a fraud perpetrated upon whom?

Leo: Right.

Steve: And in thinking about this further, in the case of such freebie giveaway drives...

Leo: Oh.

Steve: ...it occurred to me that the first target of this fraud is not those of us who receive these freebies.

Leo: Oh, yeah.

Steve: We're the victims, but we're the secondary victims.

Leo: Right.

Steve: The primary victim is the entity, whomever it was, who purchased these as gifts.

Leo: Right.

Steve: You know, in this instance above, they believed they were purchasing probably a huge quantity of 32GB drives, you know, doubtless at a bargain quantity discount, to be used as appealing 32GB giveaway drives. So the fraud was actually perpetrated upon them by the people from whom they purchased these drives. No one wants a free 512MB drive anymore. It's like, what am I going to do with this? You know?

Leo: Thanks. Thanks a lot.

Steve: I can only store, like, one photo on it, you know...

Leo: Yeah, exactly.

Steve: ...these days. So anyway, our takeaway is to be very cautious about the actual use of any drives that you buy for a bargain at the checkout stand as you're moving your groceries along, and there's a little, you know, like a fishbowl of drives, and you can have one for a buck. Or a drive that you may have ever received as a thoughtful thank you gift thrown in with something else.

Leo: For a long time, I said drives are getting so cheap you're going to get them in boxes of Captain Crunch. That actually probably is true, but it won't be the drive you think you're getting.

Steve: Exactly.

Leo: Wow.

Steve: It'll be marked 32GB. It'll format as 32GB. And it'll look like it's storing your 32GB of files. But they're not there when it comes time to read them off.

Leo: So it writes without error, and then it's only when you try to read it and there's nothing there that you go, oh.

Steve: Right.

Leo: And even then you might not suspect. You might just think, oh, I made a mistake.

Steve: No. It will be a - so in order for ValiDrive to show - because I have read error and write error colors in the map. In order for it to show red, it had to - ValiDrive wrote and read with no error. So no error was returned. The drive accepted the data from the operating system, said thanks, I got it. And then when the operating system said give it to me back, it said here you go. Unfortunately...

Leo: Well, how does ValiDrive know that it didn't work?

Steve: Because it uses a crypto random pattern in order to - and puts that out, and then verifies it when it asks for it back.

Leo: So the drive will say here, but what you get is nothing.

Steve: You get zero.

Leo: Or some random bits.

Steve: Yep.

Leo: That don't match the bits you wrote.

Steve: It's typically all zeroes or all ones.

Leo: Okay. Okay.

Steve: And the problem is it'll give it - so the drive will give you the file back.

Leo: You won't get an error. You won't get an error.

Steve: You know, try opening that in your image viewer, and it's not going to be happy. But this is not a valid file. So anyway...

Leo: It's amazing how widespread this is. Last week you talked about many drives on Amazon you bought that almost all of them...

Steve: I bought a dozen. They were all bad.

Leo: Yeah.

Steve: Every single one of them.

Leo: This is terrifying. Now, you said you're going to talk about other operating systems, so I'll let you go on because I would like - I don't have a PC running Windows.

Steve: Well, so Alain said, "Hi, Steve. Thank you for your work on ValiDrive. I'd really like to test a few USB sticks with it, but I only have access to Linux systems. Is it possible to get this running on Ubuntu or some other distro?" Okay, so Alain and our listeners, I understand. The DNS Benchmark that I wrote was deliberately WINE compatible. I went out of my way to make sure that I wasn't doing anything there that WINE did not support on its Windows compatibility layer. So, and I spent some time on that.

But ValiDrive's UI, which uses dynamic USB insert and removal events to determine which drive the user wishes to test, uses features that are outside of WINE's Windows compatibility set. So if it was crucial for ValiDrive to run under WINE, I could probably arrange to somehow make that happen. But I've already spent more time on it than I planned to, and I'm chomping at the bit to get back to finishing SpinRite. If it turns out to be seeing strong long-term use, I would be inclined, once SpinRite 6.1 is finished, to return to ValiDrive and give it a significant, you know, v2.0 update.

There are some other things I could do that, for the sake of expediency and not spending another few months on it, I said no, no, no, no, no, let's not go there. There are many people in the newsgroup who are saying, hey, how about this, and how about this, and how about that?

Leo: I'm sure somebody's asked this, but the one thing I would say would be useful, if you just wrote a paragraph on the methodology. You don't have to say how to do it, but just what works to verify. You know, we're going to write this script, you know, whatever it is. And then maybe somebody can duplicate it on another platform. You wouldn't mind that; right? You're not making money on it.

Steve: No, I wouldn't mind that.

Leo: I think that what we need to do is understand. Because there are ways you could do this that would not be a valid test. Like for instance saying, "Can I read it? Oh, I can read it, it's good."

Steve: Oh, yeah. Yeah, yeah, yeah.

Leo: So you went the extra mile. So what is it that we need to do to really have a valid test? You've got all this data now, as well, and what works and what doesn't work.

Steve: One thing I should mention is that it does run, for example, under Parallels VM, even on an ARM Mac.

Leo: Oh, that's good to know.

Steve: Yes.

Leo: Oh, well, I would do that.

Steve: Yes.

Leo: Oh, okay.

Steve: It absolutely will.

Leo: Oh, well, then I'm set.

Steve: Yeah. There is something known as Safe Emulation Mode which, at least on an ARM Mac....

Leo: Which is what I have.

Steve: ...we have to turn on in order to get it to run.

Leo: So you turn on Safe Emulation Mode. I thought it was really masking the reads and writes. But of course it can't be because you have to read and write data to a drive. You have to get out of the way and let that happen.

Steve: Right.

Leo: So that makes sense. So if I run Windows on ARM in emulation mode on my Mac, using Parallels...

Steve: It'll work.

Leo: It will still be able to run it. Oh.

Steve: Yup. We've got people who are running it all the time.

Leo: That's good to know. That's great.

Steve: Yup. Okay. So some feedback from our listeners. Marko Simo said: "Just visiting X to contact you." Oh, and he's Marko Simo / @markos@twit.social, by the way.

Leo: Yeah. A good Mastodon user. Yeah.

Steve: So he said: "Regarding Passkeys and their lack of easy exportability, QR codes, et cetera, have you ever thought that those exact features would make Passkeys vulnerable for phishing and other social engineering attacks? They're hidden from normal users for a very good reason. Someone should now figure out what could be a phishing-resistant way to move them across ecosystems. Or we could all start using SQRL." Well.

Leo: Because you solved this already.

Steve: I did. So I completely agree with Marko's observation. It is definitely a double-edged sword to make the Passkeys less mysterious. And I have to say, Leo, your comment about the lackluster success, to put it, well, to phrase it oddly, of Passkeys heartened me a little bit. You know, not because I don't want the world to have a public key-based network authentication system, but because the reason my SQRL system has gone nowhere wasn't just me. Of course, that is the reason it's never going to go anywhere. But at least Passkeys hasn't yet taken the world by storm. Or if it had, I would have at least expected a little bit more from SQRL. And I use the word "yet," you know, hasn't "yet" taken the world by storm since I really do fully expect that this FIDO2-based Passkeys public key system will someday supplant the use of secret passwords. But probably not in my lifetime. Seriously. Not in my lifetime.

Leo: Okay.

Steve: All of the evidence suggests - and I'm feeling great right now, so I'm planning to be around for a while.

Leo: You know, I think [crosstalk] take off in the next 10 years, it's going to be something else; don't you think? You think it's going to be around?

Steve: No. I think they did it right. I think, and my god, if it takes 10 years to get this thing going, we don't have another 10 years to go for Plan B. So, you know, seriously, all the evidence suggests that implementing this sort of sweeping change really will be that slow. But, you know, the reason I put all that time into SQRL was I said, well, no one had done it. And if no one does, then it's never going to happen. Well, my system's never going to happen anyway, but at least a public key system option now exists. And the problem is it just, from the users' standpoint, there's nothing that's obviously better about it. As you said, Leo, I mean, we already have password managers. This problem has been solved.

Leo: Yeah, yeah.

Steve: You know? It's not solved well, you know, and we're kind of limping along with a few arrows in our back, some of us. But at least it's obvious how it works.

Leo: Yeah, everybody's already gone the extra mile to get that working.

Steve: Yup.

Leo: And if they haven't, that's even going to be harder to get them to switch over because they're just entering their pet's name and their birthday on every site. And they say, well, what's wrong, you know, there's no - they don't see a problem.

Steve: Right.

Leo: So what are you fixing?

Steve: Yeah. Yeah, so Michael Moloney, he tweeted: "Aren't Passkeys meant to be a device verification like SSH keys? While you could copy them between devices, best security is to generate them on the device you need to authorize, so if that device gets stolen or hacked, you can revoke the key for that device. Also, if the device is hacked, logs are way more useful if it shows 'Paul's iPhone' rather than 'Paul's key,' and then he has to redo every Passkey device." He said: "Aside from backups, I hope Passkeys stay nontransferable. When you put your PIN or fingerprint in your phone, you're basically unlocking all the sites this Passkey has access to, same as when you sign into your PC with your password that has all your SSH keys."

Okay. So that's true, and I think it's a good point. This would be roughly equivalent to having multiple separate username/password pairs for logging into one's bank account with a different username/password pair stored in each device. Then, if a device was compromised, the usernames and passwords that were being used by the compromised device could be independently disavowed and disabled. Now, while that's good in theory, that really raises the level of management complexity to a whole new level. You know? Yes, there's a lot of power in that approach, but also a huge amount of responsibility for keeping track of who's on first. And, you know, okay, yes. A crazy power user could do it. But we can't even get anybody to use a Passkey, let alone manage them all separately.

Okay. Brian M. Franklin said: "@SGgrc Maybe it IS still possible to write a new web browser, with even better features." And then he quoted a thread in X, which I'm now calling it, where from someone named AKHIL, who tweeted: "Here's the multiplayer browser engine I've been working on in action." And I have to say I love the name because, because it's multiplayer, he named it the BRAID browser, as in, you know, B-R-A-I-D, like something braided, which I think is a very cool name for it. Anyone who's curious can go to braidbrowser.com, B-R-A-I-D-B-R-O-W-S-E-R dot com.

Leo: I wish it didn't sound so much like Brave.

Steve: Yes.

Leo: So Braid like in hair.

Steve: There is a collision there.

Leo: Yeah, yeah.

Steve: Okay. So, okay. So here we've got a guy who wrote a browser. Well, there's browsers, and then there's browsers. I love the idea of the creation of hobby browsers. You know, it's a modern take on the idea of a hobby operating system. And a bunch of hobby operating systems have been created. In fact, I carefully considered basing SpinRite's future upon a few of them. But lack of support for booting on UEFI-only systems ultimately led to me to the embedded RTOS-32 OS that I've talked about before.

Similarly, while I think it's possible to create a functional outline of a modern browser, the task, as I've said, has grown in size such that it's similar to that of creating a fully functional operating system. You know, yeah, there's hobby browsers, and then there's production-scale Chromium, Firefox and Safari, and that's it. So anyway, the guy, I don't know what this thing actually does. I played the video that was there in X. And there were two cursors moving around separately.

Leo: Ooh. What a cool feature.

Steve: I don't, what, if you're ambidextrous maybe?

Leo: I don't know what...

Steve: Maybe you keep one cursor over on the left and one on the right? Then you don't have to go so far? I don't know. Anyway, whatever it is.

Leo: That's not a thing a hobbyist browser would do.

Steve: That's right. Because that's what we really need. We need more cursors on our browser.

Leo: I'd be very careful because really the browser is the primary vector for malware into your machine.

Steve: I know.

Leo: And it's really easy to make a mistake in a browser. You know, maybe people are using Chromium as their base for all the rendering.

Steve: This guy claims it's a million lines of C++. And you know, Leo, every one of them is perfect. Every one of the million lines.

Leo: That scares me. That scares me.

Steve: Yeah.

Leo: There's a new browser called Arc, A-R-C, from Arc the browser company. But it's Chromium, and then they're doing the GUI on top of it, the chrome, as it were. And that's a little more doable. I don't know if I'd trust an individual to write a browser. That's hard.

Steve: We've got Mozilla. We've got Chromium. We've got Safari.

Leo: Yeah, yeah.

Steve: I think that's it. Techgifthorse tweeted: "Hi, Steve. long-time listener, first-time DMer. Regarding Episode 943" - the last week - "where you introduced that Google was embedding tracking links in Google Docs exports, I recently noticed that Google is also doing this in Calendar invites. No privacy. I'm not sure if they were trialing it in Google Docs HTML exports, and now it is expanding, but I thought it was noteworthy. I also wanted to say that you and Leo have inspired me to start my master's in cybersecurity."

Leo: Oh, that's great. Oh, that's great.

Steve: "So please keep the pods coming and keep up the great work. Can't wait for your forthcoming email option so I can delete this bogus Twitter account. Sincerely, Sean."

Leo: People are going over to Twitter just to talk to you.

Steve: Yeah, they are. And, I mean, I've seen enough of that. And that was why, when I said I'm leaving Twitter, what I meant really was I'm not going to make everyone use Twitter as a means of contacting me.

Leo: Yeah, that's the problem, yeah.

Steve: So just as soon as I have SpinRite 6.1 ready for rollout, I'll let everyone here know, of course, to get the official final. That's the super-easy communication path, Leo, that you and I have built with all of our listeners every week. Then I'll begin work on bringing up a new email facility so that I'm able to get the word out to all of 6.0's current owners, and part of that work will be to create a new means for me to send announcements to my Twitter followers and also create a means for them to send me the equivalent of private DMs.

Brett Russell says: "Hi, Steven." Being formal. Ant calls me Mr. Gibson. So, yeah, this is different. This is "Hi, Steven."

Leo: Ant doesn't call me Mr. Laporte, he calls me Leo. But everybody else is Mister, which is well-brought-up Southern boy.

Steve: Here we have "Hi, Steven." It actually is my given name, S-T-E-V-E-N, so very formal.

Leo: I didn't know that.

Steve: "I realize this is a request out of the blue, but my hard disk is failing on my main machine. I own a copy of SpinRite," and then he provided in the tweet his code, which is his SpinRite serial number. "But the disk is set up as GPT, which 6.0 does not support. Any chance you can give me a pre-release of 6.1 to run, please. There's nothing critical on the drive; but if it dies, it will take some time to bring it all back."

So of course, Brett. And this goes for everyone. If you go to grc.com/prerelease.htm, just P-R-E-R-E-L-E-A-S-E dot htm, what you'll get is a little form. Enter your current SpinRite 6.0 serial number, which is presented whenever SpinRite is run. And I just verified that Brett's code works perfectly for him. So when you do that you'll receive a link to download your own personalized and licensed copy of the latest prerelease of v6.1.

What you'll get at this point is the DOS executable, since I haven't yet built it all into the Windows boot prep thing. That's coming next. So just place that DOS executable, I think it's 95K, onto your SpinRite boot drive and let 'er rip. The current prerelease expires at the end of November, so that's six weeks from now, and I'm sure we'll have updated releases before then. So you just come back and get an update. Or just retard the machine's date, you know, if you want to buy yourself some more time.

At this point I'm 100% certain that SpinRite is safe to use. The reason is I just checked, and we have exactly 800 registered SpinRite prerelease testers who have been using and pounding on all of the SpinRite prereleases. I've taken great pains to be very careful along the way, and no one has ever experienced any data loss at any point. So this thing's ready for the world. I just need to tie up a bunch of loose ends which, you know, any three-year project of this complexity is going to have. So, you know, that's where we are. But, you know, absolutely, all of our listeners, anybody who has SpinRite, you're welcome to go grab 6.1. It's all but done, just a few last things will need to get changed.

Ray Franklin tweeted: "Steve, I've looked through the current show notes for the reference 'wipe drive with encryption.' What's the program to effectively wipe a drive with random data?" And that's VeraCrypt. And I tweeted him the link. It's at veracrypt.fr because it's French. And you could use it as we've talked about, basically to fill your drive with cryptographically secure noise. It's actually encrypting what you have already there. But if you throw away the key, no one's ever going to get it.

Marcus, whose handle is @BadAssB, he said: "Hi, Steve. I just had a quick question. I heard you say that you do not use Authy for two-factor authentication, and I was wanting to know what you do use. Thanks." So being an Apple ecosystem person for mobile, I use an app called OTP Auth, and I love it. It's clean, simple, and is very widely praised. And for anyone who can't find it easily, I've got a link in the show notes.

Leo: I also, just I'll plug for something that's both iOS and Android, which I've been using, also free, also open source: 2FAS, 2FAS. And what's nice about it is it will back up an encrypted blob with your secrets to either Google Drive or iCloud. So it makes, you know, this is an issue for me.

Steve: Nice, nice.

Leo: And one of the reasons we used Authy is so that we can move - because I get a new phone, many new phones all the time. And so the ability to move is very, very important to me.

Steve: And especially be able to move cross-platform.

Leo: Yes, exactly, yeah.

Steve: Nice.

Leo: I just installed both a new Pixel 8 Pro and a new iPhone 14 Pro, and 2FAS works on both. And I was able to import those secrets. And it was really nice. And it looks good. Here, I'll show you, it's pretty. It's a pretty - I don't know about yours, but this one's pretty. And it uses Face ID to...

Steve: Oh, very nice.

Leo: By the way, there's all my two factors. They're useless in 30 seconds. Use them quick, kids.

Steve: This is very pretty.

Leo: It's very pretty. You click it, and it will copy it to the clipboard. It goes red when it's about to expire and then generates a new one.

Steve: Oh, I like that. OAuth does not do that.

Leo: And it can sort alphabetically or in other ways, which is also nice. Yeah, and it's just the I use. But 2FAS. To each his own. The nice thing is you don't have to use a page solution or a solution from Google or Microsoft. You can, you know, this is a standard, TOTP is a standard, and there are lots of people that make software.

Steve: And finally, Henning said: "Dear Steve. When listening to SN-943" - again, last week - "I have a different explanation for why Google is rewriting links in Google Drive documents published as HTML pages: Prevent the leakage of the document's URL through referer information. Though the drive documents are public, they're hidden by their long, obscure URL. But the referer info can reveal the URL in the server logs of the embedded links. The additional step through a second Google service hides this information."

So first of all, Henning, that's a neat thought. That hadn't occurred to me. But these links were present in offline HTML exports of the doc. So wherever the HTML was being served from would be appearing in the referer header. Also, if this was a favor that Google was doing for us to protect our privacy, then their referring link URLs would not have been loaded down with unique tracking parameters in order to see exactly what link it is that someone has clicked on in the future. So, much as I love the idea that they might be protecting us, that doesn't appear to be their intent here.

Leo: There's been a lot of conversation about Google rewriting search queries, as well.

Steve: Drives me nuts. I mean, and they even show you the true link, but then that's not what you get when you click on it.

Leo: Yeah. Yeah. You know, I think that, let's face it, they're an advertising-based business, and the need to monetize through advertising kind of permeates everything, all these free services. And as a result I think, you know, if you want privacy, maybe, I pay 25 bucks a month for advertising free search.

Steve: A month?

Leo: Well, it doesn't have to be a month. No, it is, I think you can get it for five bucks. I pay a little more because they have a browser and stuff. But Kaki is very good results, Google-quality results, but no ads, no targeting, none of that. But they don't have Docs. They don't have Google Drive. They don't have all these other great features. And that's the problem. But all that stuff's free, and you've got to monetize it. I understand.

Steve: Abusing HTTP/2 Rapid Reset. As I said last week, we would be continuing with the second part of our look into the Top 10 Cybersecurity Misconfigurations if something bigger and more important didn't bump it to the following week. Well, something did indeed. So we'll do the second part of the Top 10 Cybersecurity Misconfigurations if something doesn't again bump it into the following week. But today we're going to talk about an interesting surprise.

The headline about this which first caught my attention was Cloudflare's, was on Cloudflare's blog. The headline read: "HTTP/2 zero-day vulnerability results in record-breaking DDoS attacks." So, you know, yeah, what? The idea of there being a newly discovered zero-day vulnerability in a core Internet protocol such as HTTP/2 would be huge news, if it were true. For the rest of the podcast we're going to explore what has happened and decide to what degree this is actually true. And as always, the devil is in the details, and these details are both interesting and important.

So a few lines into their coverage, Cloudflare wrote: "Cloudflare has mitigated a barrage of these attacks in recent months." So my first reaction was to question their use of the term "zero-day" if they've been doing something for several months. You know, but after wondering whether it might be an abuse or an overuse of the term, I suppose it's correct if they first learned of this unsuspected problem through an attack.

You know, we've settled upon the definition of "zero-day" as being the exploitation of any previously unknown vulnerability. Or stated another way, the first indication we have of there being some specific vulnerability is not when we find it through examination or when it's reported by a responsible researcher, but when we're surprised by something happening that we didn't believe to be possible. So, yeah, it would probably be correct for this to be considered a zero-day vulnerability in the v2 HTTP protocol, if indeed it was a vulnerability in the HTTP/2 protocol - which, surprisingly, is not the case.

So it is to Cloudflare's benefit, Leo, exactly as you said at the top of the show, since they are in the DDOS protection business, to characterize this as a surprising zero-day vulnerability in HTTP/2. That deflects the blame to the protocol when it appears that the true culprit is implementation. The HTTP/2 protocol is not to blame because there's a difference between a vulnerability and the abuse of a feature that's exacerbated by Cloudflare's internally decoupled serial processing multi-proxy architecture.

Leo: Oh, this is interesting. Ah.

Steve: That's what made it such a problem. And it applies to the cloud providers because the way they distributed the workflow interacted with this feature. So as I think the evidence will show, and Google was much more forthcoming, as we'll see, this is more about something that Cloudflare's behind-the-scenes request-processing architecture was ill-suited to handle than the exploitation of any previously unknown vulnerability in HTTP/2.

It's a clever and, in retrospect, probably foreseeable and maybe even inevitable leveraging, that is, the attack is, of a feature in HTTP/2. So, you know, I don't mean to jump on Cloudflare too much about this. But characterizing this as a zero-day vulnerability in HTTP/2 is really not accurate or fair.

Okay. So let's back up a bit. Let's start with a very quick background review of layered network design, and then a quick history of HTTP. We refer to HTTP, the Hyper-Text Transfer Protocol, thus HTTP, as an application-level protocol - also known as a Layer 7 protocol because that's as high as the layers go - because after we've finally finished building up layer upon layer of the various underlying enabling protocols, we finally get to HTTP, which was the whole point. And it's there where something finally happens and gets done.

So what do we mean by "layer upon layer"? At the bottom-most level or layer we have an agreement about wires and voltages and clock rates and, like, signal transmission times. And those are used to signal the transmission of individual discrete zeroes and ones. That's typically referred to as the physical layer. You know, that's as physical as it can get. Like what's the voltage on the wire, and what does it mean? Then, with that agreed upon, we describe the groupings of those ones and zeroes to form packets of data which are addressed between physical hardware endpoints on a single local network.

This is then the Ethernet protocol. And those Ethernet packets, in turn, contain IP (Internet Protocol) packets which carry the addresses of endpoints on the global Internet. And inside those IP packets are TCP packets where the TCP protocol is used to manage the numbered, sequential byte-oriented reliable flow of data between those specific virtual ports on the devices specified by their IP addresses on the global network.

And finally, the data that flows back and forth under the watchful eye of the TCP protocol is formatted as specific queries and responses in accordance with the top-level application protocol, HTTP. So that's the layering all the way up from the physical layer to layer 7, the application layer. And over the decades since their initial definitions, all of these protocols have seen some tuning, some tweaking and evolution in their definition as they interacted with the real world. And that's certainly been true for HTTP.

With a bit of editing by me for clarification, here's how Wikipedia briefly sums up HTTP's evolution. They wrote: "Development of HTTP was initiated by Tim Berners-Lee at CERN in 1989 and summarized in a simple document describing the behavior of a client and a server using the first HTTP version, which was 0.9. That version was subsequently developed, eventually becoming the public 1.0.

"Development of early HTTP RFCs began a few years later in a coordinated effort by the IETF" - that's our Internet Engineering Task Force - "and the W3C" - W3 meaning World Wide Web, and C for Consortium - "with work later moving exclusively back to and now at the IETF.

"HTTP/1 - and it's shown as a /1 rather than a v1 because HTTP/1 is actually what's sent over the wire with each query to specify the format of everything that will follow, so you need to specify the version right upfront so that the recipient is able to know how to interpret everything that's going to come. So v1.0 was finalized and fully documented in '96. It quickly evolved into v1.1 a year later in 1997, after which its specifications were updated in '99, 2014, and even just last year in 2022. Its secure variant, HTTPS, is now used" - as we know - "by more than 80% of websites," writes Wikipedia.

Now, HTTP/2, which is the topic for today, published first in 2015, so it's already eight years old, it "provides a more efficient expression" - and we'll be talking about this efficiency in a minute - "of HTTP's semantics 'on the wire.' As of April 2023, HTTP/2 is used by 39% of websites." So as of April this year, just shy of 40% of all websites offer HTTP/2. "And it's supported by almost all web browsers, representing over 97% of web users." So, you know, all the popular browsers do it now. "It's also supported by major web servers over TLS using an Application-Layer Protocol Negotiation (ALPN) extension where TLS 1.2 or newer is required."

And just to round this out for the sake of completion, we actually drop TCP as the transport protocol for HTTP in favor of the faster to set up, yet less feature-rich UDP, when we're using HTTP/3, which is technically out there now, though not yet widely adopted. "HTTP/3, of course, is the successor to HTTP/2. That was published just last year. It's now available from one quarter of all websites," says Wikipedia, "and it's at least partially supported by most web browsers. As I mentioned, unlike all of its TCP-based predecessors, HTTP/3 does not run on top of TCP. Instead it uses QUIC, which runs on top of UDP. Support for HTTP/3 was added to Cloudflare and Google Chrome first, and is also enabled in Firefox. HTTP/3 has lower latency for today's web pages which are pulling information together through many separate widespread sources. If HTTP/3 is enabled on the server, such pages," says Wikipedia, "will load faster than with HTTP/2, which is, in turn, faster than HTTP/1.1, in some cases as much as three times faster than HTTP/1.1."

Okay. So now that we have the lay of the land, let's look at what took Cloudflare and others by surprise a few months back. There's a bit of, I guess it's marketing/bragging going on here from Cloudflare, but mostly a lot of good information, and I'll insert a little bit of editorializing as we go. So Cloudflare wrote: "Earlier today" - and this was just posted exactly one week ago on the 10th of October. They said: "Cloudflare, along with Google and Amazon AWS, disclosed the existence of a novel zero-day vulnerability dubbed the 'HTTP/2 Rapid Reset' attack."

And they wrote, Cloudflare wrote: "This attack exploits a weakness in the HTTP/2 protocol to generate enormous, hyper-volumetric Distributed Denial of Service (DDoS) attacks. Cloudflare," they said, "has mitigated a barrage of these attacks in recent months, including an attack three times larger than any previous attack we've ever observed, which exceeded 201 million requests per second. Since the end of August 2023, so that's just a couple months ago, Cloudflare has mitigated more than 1,100 other attacks with over 10 million requests per second, and 184 attacks that were greater than our previous DDoS record of 71 million RPS (requests per second)."

And again, now these new attacks, 201, so nearly three times those 184 attacks that were greater than their previous DDoS record of 71 million. And I should note that a similar stat from Google about this put it at just shy of 400 million requests per second. I think that's 390 something, as I recall.

So I'll pause here just to note that there are several different ways to measure or characterize DDoS attacks. Traditionally, attacks leveraged bandwidth flooding and saturation. Just pure bandwidth. The incoming network links to a server would be so saturated with packet traffic that a site's upstream routers would be overwhelmed as they were aggregating the traffic being aimed at a destination server, so overwhelmed with bogus packets that legitimate packet traffic stood little chance to get through to the server. It was just a matter of just, you know, a matter of numbers. If you've got a hundred bogus packets for every good one, there's a 1% chance that the good one's going to get through. So essentially the server's services are being denied, and thus the term Denial of Service, of course.

But as we know, some time ago web pages shifted from being simple static dumps of a previously written textual page. For example, my own GRC site remains that way to this day. It's just simple static HTML. But most web services switched over to being the public-facing aspect of a content management system, a CMS of some sort. Under this new website paradigm, any incoming HTTP query is fielded by some sort of code running on the server. That code examines the query details and assembles the page which will be returned to the client on the fly. And this, in turn, typically involves multiple queries to a backend relational database of some kind, typically a SQL server.

The point of this is that under this new content management paradigm, replying to a query for a simple page, what looks like a standard HTML page, consumes server-side compute resources in order to pull the various components of the page together for its return to the client. The other thing is that in the earliest days of this new approach, these on-the-fly page assembly systems were often not very efficient. So each query, especially deliberately complex queries, which, you know, might be requiring a lot of backend database work to pull a complex page together, could be quite costly to answer or to reply to.

Naturally, it didn't take the bad guys long to figure this out, after which they switched their attack tactics from flooding a server's raw bandwidth pipeline with what could well just be noise, it didn't really matter what the packets, you know, contained, to loading the server down with actual valid queries, each of which could be quite expensive for it to answer. Thus we went from thinking of DDoS attacks in terms of bits per second to requests per second because it was the request that was expensive to handle, not just a bandwidth flood.

Now, I take issue with Cloudflare's characterization of this as an attack which "exploits a weakness in the HTTP/2 protocol." That's not what has been happening. As we'll see, these new attacks exploit some new features of HTTP/2 which were added to make the protocol much faster overall. But it turns out that these new features are subject to abuse. And if a server's query-processing chain is not really explicitly designed to handle these new features, it can be forced into collapse by attackers. This is where we'd say "it's not a bug, it's a feature," and we'd mean it.

So Cloudflare continues. They said: "This zero-day" - which, okay, it was a surprise - "provided threat actors with a critical new tool in their Swiss Army knife of vulnerabilities to exploit and attack their victims, at a magnitude that has never been seen before. While at times complex and challenging to combat, these attacks allowed Cloudflare the opportunity" - oh, it's an opportunity - "to develop purpose-built technology to mitigate the effects of the zero-day vulnerability." In other words, whoops. As I said, they got caught with their DDoS down and needed to, you know, up their technology in order to deal with this. And again, we're going to see, and I'll shortly provide ample evidence to back it up, this is not really a vulnerability, and I am a little disappointed in Cloudflare's approach to this. We all know that I'm a big fan of Cloudflare, but I believe they took the wrong tack on this one.

So they said: "In late August of this year, our team at Cloudflare noticed a new zero-day vulnerability, developed by an unknown threat actor, that exploits the standard HTTP/2 protocol, a fundamental protocol that is critical to how the Internet and all websites work," meaning HTTP, and now 2 is a lot better. "This novel zero-day vulnerability attack, dubbed Rapid Reset, leverages HTTP/2's stream cancellation feature by sending a request, then immediately canceling it, over and over.

"By automating this trivial 'request, cancel, request, cancel' pattern at scale, threat actors are able to create a denial of service and take down any server or application running the standard implementation of HTTP/2. Furthermore, one crucial thing to note about the record-breaking attack is that it involved" - and pay attention here - "it involved" - that is, this attack that was bringing down Cloudflare's very strong preexisting DDoS defenses, they said - "consisted of roughly 20,000 machines in a modestly sized botnet." They said: "Cloudflare regularly detects botnets that are orders of magnitude larger than this, comprising hundreds of thousands and even millions of machines. For a relatively small botnet to output such a large volume of requests, with the potential to incapacitate nearly any server or application supporting HTTP/2, underscores how menacing this vulnerability is for unprotected networks."

Leo: So I had assumed that this was some sort of massive amplification attack because previous attacks have had to have massive botnets in order to do this.

Steve: Right. Right.

Leo: Why is it that this HTTP/2 cancel request, rapid reset request, can't be more easily blocked? If it's only 20,000 IP addresses it's coming from, how come you can't just mitigate it? I don't understand.

Steve: That actually is the strategy. That's the only strategy. And I actually, I wind up saying exactly that.

Leo: Okay, good.

Steve: The only strategy that I think makes sense.

Leo: I don't want to jump ahead.

Steve: You jumped ahead.

Leo: Okay.

Steve: Because thank god TCP cannot have its source address spoofed; right? You need to have a full communication link that is confirmed on each end so you do know the IP address of every bot that's attacking you.

Leo: So previous attacks were much simpler. There were often SYN/ACK floods where a bot would say Hello, and then your server would have to say Hello to start the conversation. At which point that server, that client would go, never mind, Hello.

Steve: And that's actually a good analogy to this attack. But it's up to level where it's not just a SYN which is initiating a TCP connection. It's actually a request that is initiating backend work.

Leo: So there's a lot of work because it's reset, you're asking it to reset the thread. So it's more than just hello, hello, are you there.

Steve: It's more than just a single packet being received.

Leo: Yeah.

Steve: Okay. So I'm going to make three points: First, I hope it still astonishes us when Cloudflare writes: "Cloudflare regularly detects botnets that are orders of magnitude larger than 20,000, comprising hundreds of thousands and even millions of machines." Just think about that. This kind of power in the hands of miscreants is fundamentally destabilizing. Because the Internet is so powerful, and because it mostly works so well, we've allowed ourselves to gradually grow to become utterly dependent upon its presence. But last week we noted Microsoft was observing around 1700 DDoS attacks per day. So there are websites that are occasionally denied the sort of reliability that the

Internet has to offer because someone somewhere just says no, you know, we want you off the 'Net. We're going to, you know, we're not happy with something you just did, so we're going to punish you.

Leo: Well, people might remember, back in the day Steve had this happen to him.

Steve: Yup.

Leo: It was going on during the show.

Steve: Yup.

Leo: And it was just, you know, it's just vandalism of a kind.

Steve: Well, it turned out it was because I had those DDoS attack reports on my site. And so they thought I was bragging in some way. And in fact I was not.

Leo: You were just doing the news.

Steve: And when I removed those pages, the attacks stopped.

Leo: Oh, that's very interesting.

Steve: You know, it wasn't me saying nanny nanny nanny, you can't get me. It was like, ow, you know? I just got blasted off the Internet.

Leo: Yeah.

Steve: So the second point I wanted to highlight was that Cloudflare witnessed by far the largest attack they have ever experienced, which was produced by what is unfortunately now considered to be a relatively modest-sized botnet, consisting of only around 20,000 individual clients. Since this sort of attack rides on TCP, the attacker's source IPs cannot be spoofed since TCP requires confirmation of packet round trips before the query can be sent. So Cloudflare has the specific IP of every bot that was attacking it. That's how it knows how many, because it can count.

Leo: Right.

Steve: And the last point is that, since this was a new form of abuse to which HTTP/2 is prone, every one of these bots needed to be updated with a specific module designed to implement this attack against its target. So someone, somewhere, figured this out, wrote the code to do this, then either established a new botnet for this purpose or caused an existing modest-sized updatable botnet to be updated to add this new form of attack to its bag of tricks.

They said, Cloudflare said: "Threat actors used botnets in tandem with the HTTP/2 vulnerability" - again, I'll take issue with that characterization, but we'll get there in a second - "to amplify requests at rates we have never seen before. As a result, our team at Cloudflare experienced" - I love this phrase here - "some intermittent edge instability." Uh-huh. Intermittent edge instability. Right. In other words, we're all about DDoS protection, but this one got us because we had never seen anything like it, and our backend architecture was ill-equipped to handle it at the time. And in fact they do, they did publish a chart where it was like up to 80% of their connections were having trouble because, I mean, this thing really clogged up their network.

Leo: Well, that makes sense since you said 80% of all servers use HTTP/2. It was able to hit them all, basically.

Steve: So their entire customer base, you know, their whole frontend is HTTP/2 equipped.

Leo: So if you're using Cloudflare, it is HTTP/2.

Steve: Yeah. You have HTTP/2 front-facing.

Leo: Ah, interesting. Ah.

Steve: So they said: "While our systems were able to mitigate the overwhelming majority of incoming attacks, the volume overloaded some components of our network, impacting a small number of customers' performance with intermittent 4xx and 5xx errors, all of which were quickly resolved." Right.

"Once we successfully mitigated these issues and halted potential attacks for all customers, our team immediately kicked off a responsible disclosure process." Meaning for several months there has been stuff going on behind the scenes that no one knew about until Tuesday, when this was officially revealed in a coordinated disclosure. They said: "We entered into conversations with industry peers to see how we could work together to help move our mission forward and safeguard the large percentage of the Internet that relies on our network prior to releasing the news of this vulnerability to the general public."

They wrote: "There's no such thing as a 'perfect disclosure.' Thwarting attacks and responding to emerging incidents requires organizations and security teams to live by an 'assume breach' mindset because there will always be another zero-day, new evolving threat actor groups, and never-before-seen novel attacks and techniques. This 'assume breach' mindset is a key foundation towards information sharing and ensuring in instances such as this that the Internet remains safe. While Cloudflare was experiencing and mitigating these attacks, we were also working with industry partners to guarantee that the industry at large could withstand this attack.

"During the process of mitigating this attack, our Cloudflare team developed and purpose-built new technology to stop these attacks and further improve our own mitigations for this and other future attacks at massive scale." In other words, they learned that there was something that was able to get by their DDoS mitigations, and they needed to develop new technology to deal with it.

Leo: Boy. Wouldn't you love to be a fly on the wall in the war room, there when they get these, you know, emergencies?

Steve: Oh, yeah.

Leo: And they all leap into action. I have to say, I mean, whatever they did wrong here, you've got to credit them with really talented people doing a lot of...

Steve: I do. I love Cloudflare.

Leo: Yeah.

Steve: I just wish they weren't calling it a zero-day vulnerability in HTTP/2. It's not. It's a feature. But anyway, we'll get there in a second.

Leo: It is a vulnerability, however, a feature-rich vulnerability.

Steve: It is, yeah, an exploitable feature.

Leo: Feature. There, that's a better phrase. Yeah, they should have said that, yeah.

Steve: An exploitable feature. So they said: "These efforts [theirs] have significantly increased our overall mitigation capabilities and resiliency." So yes, we're now better than ever. They said: "If you're using Cloudflare, we're confident that you are now protected. Our team also alerted web server software partners who are developing patches to ensure this vulnerability cannot be exploited." And they said: "Check their websites for more information. Disclosures," they said, "are never one and done. The lifeblood of Cloudflare is to ensure a better Internet, which stems from instances such as these. When we have the opportunity to work with our industry partners and governments to ensure there are no widespread impacts on the Internet, we are doing our part in increasing the cyber resiliency of every organization, no matter the size or the vertical.

"It may seem odd that Cloudflare was one of the first companies to witness these attacks. Why would threat actors attack a company that has some of the most robust defenses against DDoS attacks in the world?" Well, they've answered their own question, and they're about to. "The reality is that Cloudflare often sees attacks before they are turned on more vulnerable targets. Threat actors need to develop and test their tools before they deploy them in the wild. Threat actors who possess record-shattering attack methods can have an extremely difficult time testing and understanding how large and effective they are because they don't have the infrastructure to absorb the attacks they're launching. Because of the transparency that we share on our network performance, and the measurements of attacks they could glean from our public performance charts, this threat actor was likely targeting us to understand the capabilities of this new exploit." And I think that's exactly, exactly right.

Leo: So we're just going to launch a small botnet against you, just to see what you can do.

Steve: Yeah. Remember in the days when I was under attack, I had two T1s which were each 1.54Mb.

Leo: Right.

Steve: So a 3.4Mb flood would swamp me. I mean, I was a gnat. I wasn't even worth swatting. So, you know, it took nothing to knock me off the 'Net.

Leo: But on the other hand, Cloudflare has terabytes of bandwidth.

Steve: Yes. Yes. It is very difficult to push them down. But this attack did it initially. They said: "But that testing" - meaning the bad guys' testing - "and the ability to see the attack early helps us develop mitigations for the attack that benefit both our customers and the industry as a whole." That is, you know, they shared what they learned. Even so...

Leo: Both sides get information, the bad guys and the good guys, yeah.

Steve: Yup. "Even so, whether it was Log4j, SolarWinds, EternalBlue, WannaCry/NotPetya, Heartbleed, or Shellshock, all of these security incidents have a commonality, a tremendous explosion that ripples across the world and creates an opportunity to completely disrupt any Internet-dependent organization, regardless of the industry or the size. While we wish we could say that Rapid Reset may be different this time, it is not. We're calling all CSOs no matter if you've lived through the decades of security incidents or this is your first day on the job this is the time to ensure you are protected and stand up your cyber incident response team.

"We've kept the information restricted until today to give as many security vendors as possible the opportunity to react. However, at some point, the responsible thing becomes to publicly disclose zero-day threats like this. Today is that day. That means that, after today, threat actors will be largely be aware" - largely, you bet - "largely be aware of the HTTP/2 vulnerability." Again, it's a feature that's exploitable. "And it will inevitably become trivial to exploit and kick off the race between defenders and attackers first to patch versus first to exploit. Organizations should assume that systems will be tested, and take proactive measures to ensure protection."

The guy wrote: "To me, this is reminiscent of a vulnerability like Log4j, due to the many variants that are emerging daily, and will continue to come to fruition in the weeks, months, and years to come. As more researchers and threat actors experiment with the" - and they're calling it again a vulnerability - "we may find different variants with even shorter exploit cycles that contain even more advanced bypasses." I would argue against that, but we'll get to that in a minute.

And they said: "And just like Log4j, managing incidents isn't as simple as 'run the patch, now you're done.' You need to turn incident management, patching, and evolving your security protections into an ongoing process because the patches for each variant of a vulnerability reduce your risk, but they don't eliminate it. We don't mean to be alarmist, but we'll be direct: You must take this seriously. Treat this as a full active incident to ensure nothing happens to your organization."

Okay. So that's a little bit self-serving since Cloudflare has just finished explaining that while they were initially caught off guard by this unanticipated and massive zero-day attack, they're now up to speed and are inviting everyone to come and hide behind their perimeter. On the other hand, they're offering it at no charge. They do offer the service for free.

So they wrap this up by writing: "Cloudflare's mission is to help build a better Internet. If you're concerned with your current state of DDoS protection" - and if you weren't before, you should be now - "we're more than happy to provide you with our DDoS capabilities and resilience for free to mitigate any attempts of a successful DDoS attack. We know the stress that you're facing as we have fought off these attacks for the last 30 days and made our already best-in-class systems even better." Okay. So thank you very much.

Now, even though this first posting of theirs didn't give us any of the meaty technical details that this podcast never shies away from, I wanted to share it since it's what Cloudflare is telling the entire world. And sharing it here is important because I believe, as I've said, that it mischaracterizes HTTP/2 as having surprising and unsuspected vulnerabilities, which is not the case. We're going to get into the meaty details, Leo, after you tell us who's paying for them.

Leo: This is really fascinating. And, you know, the takeaway is, if this was a test with a small botnet...

Steve: Oh, boy.

Leo: ...imagine what the traffic could be if they really targeted somebody. And of course there's some events coming up that are very commonly DDoSed, including the Super Bowl, it's very common to DDoS betting groups, or at least to extort them. That's the other reason. It's not just a test, it's also a note to companies that are getting ready for Black Friday, Prime Day, the Super Bowl betting, that we have the capability to shut you down on those days. And I think that that's very clear message, and I'm sure that the teams, the security teams of those companies are sitting up and taking notice. It's hard to fix a feature.

Steve: Exactly.

Leo: Easier to fix a bug.

Steve: Exactly.

Leo: Wow. All right. I'm just - I am just blown away by this HTTP/2 vulnerability that you say is a feature, this rapid reset is a feature, not a bug.

Steve: So, yes. Now we get to the meaty details. It is described in CVE-2023-44487. The CVE disclosure is a bit more fair, but even it contains a bit of spin and a little bit of misdirection. The description on the CVE states: "The HTTP/2 protocol allows a denial of service (server resource consumption) because request cancellation can reset many streams quickly, as exploited in the wild in August through October of 2023."

Okay, now, Google's summary of the problem is, I think, exactly correct. So here's what Google wrote. They said: "Since late 2021" - so for the last two years almost - "the majority of Layer 7" - meaning HTTP, you know, high-level, application-level - "DDoS attacks we've observed across Google first-party services and Google Cloud projects protected by Cloud Armor have been based on HTTP/2" - meaning so HTTP/2 has become the preferred DDoS attack protocol - "both by number of attacks and by peak request rates. A primary design goal of HTTP/2 was efficiency, and unfortunately" - and here it is - "the features that make HTTP/2 more efficient for legitimate clients can also be used to make DDoS attacks more efficient."

So that's what it is. It's an abuse of the efficiency that HTTP/2 offers. That is precisely the truth. "The features that make HTTP/2 more efficient for legitimate clients can also be used to make DDoS attacks more efficient." So not any surprise zero-day vulnerability, just the clever abuse of a new feature of HTTP/2 that anyone designing HTTP/2-capable servers will need to take into account. Until now, Cloudflare hadn't. They'd been relying upon their conventional DDoS protections, which are doubtless very strong; but this new form of attack was too much even for that.

But that said, I do agree with the creation of a CVE and the raising of alerts about this since, indeed, rapid HTTP/2 stream resets promise to cause some havoc with any HTTP/2-capable web server that's not capable of handling these resets efficiently.

To get some scope of the problem, switching back to Cloudflare for a minute, in their more technical write-up they wrote: "Starting on August 25th, 2023, we started to notice some unusually big HTTP attacks hitting many of our customers. These attacks were detected and mitigated by our automated DDoS system. It was not long, however, before they started to reach record-breaking sizes, and eventually peaked just above 200 million - 200 million - requests per second. This was nearly three times bigger than our previous biggest attack on record. Concerning is the fact that the attacker was able to generate such an attack with a botnet of merely" - and unfortunately we're now using the word "merely" when we talk about 20,000, you know, consumer routers and other things that have been compromised.

Leo: I'll never forget, and this was 20 years ago at TechTV, we had - I can't remember who it was, one of our famous [indiscernible] hackers come in and show us an IRC channel that was being used to control, for command and control of a botnet. And we're sitting here watching this IRC channel. I think this was right before they shut it down. This was law enforcement. But just one after another, a compromised system announcing itself and effectively saying, you know, what is thy will, my master? You know, what do you want me to do? And they just all sit there in this IRC - I don't know if they still use IRC for this. But they just sit there waiting to be commanded.

Steve: Now, maybe you're remembering my write-up because I...

Leo: Maybe it was you. I don't know. I saw this happening, though.

Steve: Yeah.

Leo: I think it was on the set of the TV show.

Steve: Okay.

Leo: And I believe law enforcement brought it in.

Steve: Yeah. And I also saw the same thing because I infiltrated the attackers on their C channel, and I captured a screen of all these bots reporting for duty.

Leo: Yeah. This is 20 years ago. So it's been going on for a long time.

Steve: Oh, yeah.

Leo: Do they still use that kind of command and control? Or do they have a more modern way?

Steve: I've no idea. I've been out of that specific loop.

Leo: Yeah.

Steve: So the Cloudflare guy said: "There are botnets today that are made up of hundreds of thousands or millions" - and remember, this is a 20,000-machine attack. There are botnets with millions of machines. So imagine when those millions of botnets are updated with this attack. I mean, that's the meltdown event. I mean, it's, whoa.

Leo: Yeah. We could see a world [indiscernible] here.

Steve: Given that the entire web - oh, here's another really interesting metric. So this was 201 million RPS, requests per second. They wrote: "Given that the entire web typically sees only between one and three billion requests per second, it's not inconceivable that using this method could focus the entire global web's worth of requests onto a small number of targets."

Leo: Holy cow.

Steve: It's like focusing the sun.

Leo: Oh, god.

Steve: You know, onto Manhattan.

Leo: The magnifying glass with all that power into a pinpoint.

Steve: Yup.

Leo: Your server's going to melt.

Steve: It's going to melt. So just to be clear, since putting this into context is important, this attack by just 20,000 bots generated, on its own, a peak of 201 million requests per second. And the Internet across its entire expansive whole sees between one and three billion requests per second. And whereas this breathtaking attack was produced by only 20,000 bots working in concert, botnets consisting of hundreds of thousands to millions of individual devices are known to exist. And as Cloudflare wrote in their less technical overview, as of last Tuesday October 10th, every bad guy bot operator who may not have been in the loop until now, now knows about it, and it's a trivial attack to create.

What's more, as Wikipedia told us, HTTP/2 is offered as an available protocol by around 39% of the Internet's web servers. We don't currently know what percentage of those web services poorly handle HTTP/2 stream resets. It could be that cloud providers inherently have a greater problem with this due to their more highly distributed architecture. A single standalone HTTP/2 server might now be much more affected by this than by an HTTP/1.1 request flood.

Okay. However, unless and until any vulnerable web servers are updated and/or moved behind some sort of perimeter protection, they're going to be at least in danger of exploitation from this novel server-side resource depletion attack. And what's significant is that this is true even from the great many more lesser bot operators who run much smaller botnets. Once the code to implement this attack has circulated throughout the underground, as I noted above, it's trivial to do this attack. And given how things are likely...

Leo: Out of curiosity, is it just a loop? Is it just, like, a couple of lines of code that says send this reset request, rapid reset request, to this server, and then go to 10?

Steve: That's what it is.

Leo: Is it basically that?

Steve: That's it.

Leo: I just implemented it.

Steve: Yup.

Leo: Good lord.

Steve: Probably in LISP, knowing you.

Leo: Yeah, I'll do it in LISP. What is the command for rapid reset? Is it a straightforward command?

Steve: We're getting there.

Leo: Okay. All right. Sorry. I'm getting excited.

Steve: So also dated last Tuesday the 10th was Cloudflare's posting titled "HTTP/2 Rapid Reset: Deconstructing the record-breaking attack." Their description begins: "This attack was made possible by abusing some features of the HTTP/2 protocol and server implementation details." Now, see, and the techie guys are a little less hype-y; and, you know, they're getting it right. So this is still - this is Cloudflare saying: "This attack was made possible by abusing some features of the HTTP/2 protocol" - not calling it a vulnerability - "and server implementation details."

Leo: Well, I mean, to be fair, it's a feature, but it is also a vulnerability.

Steve: It's not a goodie.

Leo: If you had a soft spot on the top of your head, it's a feature, but I can also knock it.

Steve: It's still a soft spot, yeah.

Leo: It's still a soft spot.

Steve: So they said: "Because the attack abuses an underlying weakness in the HTTP/2 protocol, we believe any vendor that has implemented HTTP/2 will be subject to the attack. This includes every modern web server. We, along with Google and AWS, have disclosed the attack method to web server vendors who we expect will implement patches." Mmm, we'll see. I mean, there's not much you can do.

Leo: That's my question, yeah, yeah.

Steve: In the meantime, yeah, the best defense is using, of course, here they're going to sell themselves, although it is free, a DDoS mitigation service like Cloudflare's in front of any web-facing web or API server.

Okay. Again, what exactly now is an HTTP/2 stream reset? I could write this up myself, but in the interest of saving some time so that I have time to assemble other interesting things this week because actually I did this first, and then I did the user feedback, I'm going to switch to Google's explanation. I'm switching to Google because Cloudflare is pretty much unable to stop blaming this on HTTP/2 and calling it a zero-day vulnerability.

Google explains: "HTTP/2 uses 'streams,' bidirectional abstractions used to transmit various messages, or 'frames,' between the endpoints. 'Stream multiplexing' is the core HTTP/2 feature which allows higher utilization of each TCP connection. Streams are multiplexed in a way that can be tracked by both sides of the connection while only using one Layer 4" - meaning TCP - "connection. Stream multiplexing enables clients to have multiple in-flight requests without managing multiple individual connections."

Okay, now, I'm going to switch back and note that we talked about this a million years ago on the podcast. Well, okay, 2015. With HTTP/1.1, a single TCP connection can be created to a server and a client. It's able to send multiple queries to the server sequentially in a sort of pipeline. But the server must receive them in order and reply to each query in order so that the sequence of replies matches the sequence of queries. The big trouble with this is that many smaller and faster-to-answer queries could get held up behind an earlier query that takes a server more time to reply to. Perhaps it needs to query something else to obtain an answer. Meanwhile, some queries like for simple images, for example, you know, little icons and embellishments, would be held up waiting behind the big slow query to be returned.

When Google says that HTTP/2 uses "streams," they mean that ID tags are added to individual queries, and ID tags are returned with their matching replies. This creates a sort of parallel abstraction where, even though there's still only one connection, a highly capable HTTP server could accept every incoming query the moment it arrives, assign an independent task or thread to begin assembling each query's reply individually, and then send back that query's reply with its associated ID tag as soon as its reply is ready.

Leo: That makes sense because modern machines are multithreaded.

Steve: Yes.

Leo: So they're not doing a single process anymore. That's ancient history.

Steve: Right.

Leo: So they could take advantage of that.

Steve: Yes.

Leo: A lot of programming languages have streams built in for this very reason.

Steve: Right. So Google continues. They said: "One of the main constraints when mounting a Layer 7" - meaning an HTTP-level, application-level - DoS attack is the number of concurrent transport connections. Each connection carries a cost, including operating system memory for socket records and buffers, CPU time for the TLS handshake, as well as each connection needing a unique four-tuple, that is, the IP address and port pair for each side of the connection, which constrains the number of concurrent connections between two IP addresses.

"In HTTP/1.1, each request is processed serially. The server will read a request, process it, write a response, and only then read and process the next request. In practice, this means that the rate of requests that can be sent over a single connection is one request per round trip, where a round trip includes the network latency, the proxy processing time, if there's a proxy, you know, in front, and backend request processing times. While HTTP/1.1 pipelining is available in some clients and servers to increase a connection's throughput, it is not prevalent among legitimate clients."

Now, I thought that was interesting. What Google just said was that, while HTTP/1.1 does technically support allowing clients to "send ahead" queries, as I originally mentioned, in the process known as pipelining, in practice that has not ever been widely adopted, and now it won't be because HTTP/2 has come along now, and all the browsers do that. And the reason was it's a little scary to like send requests blind and just hope that the answers come back in the right sequence, and you not get confused in counting them. Much cleaner if you tag them with individual IDs so you're sure about which query matches which request.

So anyway, it wasn't widely adopted, so the clients wait until they receive the reply to their single outstanding query before they're comfortable to submit the next query, even though technically they could.

Google says: "With HTTP/2, the client can open multiple concurrent streams" - so again, get this, that a stream is just an abstraction. In the same way that packets are an abstraction of a connection on the Internet between two points, we say we're connected, but it's actually a stream of packets. Similarly, streams are an abstraction of multiple concurrent connections over a single TCP connection.

So they said: "Each stream corresponding to one HTTP request. The maximum number of concurrent open streams is, in theory, controllable by the server. But in practice, clients" - here it is - "may open 100 streams per connection, and the servers process these requests in parallel." Meaning that a client could connect using TCP, bring up a TLS connection to go HTTPS, then generate - then when the server has responded, it'll say that it is able to handle HTTP/2, so the client says great. It then basically just pours as many queries for items on the server into this connection as possible, giving each one a unique ID tag, knowing that as the server is able to handle them, and in any order that the server may choose, it will start sending the replies back tagged with the same ID as the query that it generated. And a hundred of them can be outstanding at once. You can just pour a hundred in.

So Google said: "For example, the client can open 100 streams and send a request on each of them in a single round trip. The proxy" - which is the frontend to a cloud service - "will read and process each stream serially, but the requests to the backend servers can again be parallelized. The client can then open new streams as it receives responses to the previous ones." Meaning it can always have a hundred of them outstanding, a hundred pieces of work to be done. So as streams complete by their reply coming back, that says, oh, there's room for another stream, and so it just immediately emits another query. So Google says: "This gives an effective throughput for a single connection of 100 requests per round trip, with similar round trip timing constants to HTTP/1.1 requests. This will typically lead to almost 100 times higher utilization of each connection."

Okay. So this is where, why, and how HTTP/2 offers potentially far greater performance over HTTP/1.1. HTTP/2 solicits clients, our web browsers, to dump all their queries into a single connection at once. Since it's just a single TCP connection, they cannot actually move across the wire simultaneously, but they can be queued up and sent so they're packed very tightly, and so that fewer small and less efficient packets are sent. Remember we talked about this back then. If you send small queries, then you're only using a part of a packet, yet you're still having to switch and route an entire packet. If instead you're able to pack queries literally where the last byte of one starts with the first byte of the next, then all your packets are full, and so all of your switching and your packet processing rate also benefits. I mean, it's really the way that this should have been done. But, you know, it was simple back then. As I said a couple weeks ago, nothing is simple anymore, unfortunately. It's all getting complicated.

So anyway, assuming that the backend has sufficient parallel serving capability, all of the then-outstanding replies can be assembled in any order. The moment a reply is ready it's queued up for return to the client with its ID tag identifying which query it's the reply for. It is beautiful and elegant, and it is a future that is here. Unfortunately, it can be abused like nothing else ever has been.

So note that the most modern preexisting high request rate attacks are already leveraging this parallel streaming capability. As Google said, since 2021 the largest attacks we've been seeing are HTTP/2 because they're using these streams in order to make this happen. So HTTP/2 already allowed much higher request rates than were possible under HTTP/1 or 1.1. But now here's what you've been waiting for, the problem that's created by the new abuse of this fancy parallel architecture.

Google writes: "The HTTP/2 protocol allows clients to indicate to the server that a previous stream should be canceled by sending a RST_STREAM frame. It's all you need to do at the client end."

Leo: What could go wrong with that?

Steve: Basically, never mind. You send a never mind.

Leo: Never mind. Forget I asked.

Steve: The protocol does not require the client and server to coordinate the cancellation in any way, meaning there's no need for it to be confirmed. TCP automatically gives us reliable transport. So TCP's got that covered. If the client sends a RST_STREAM frame, it can assume the server has, and it is then able - that frees up one of its 100 for another query. Thus the client can do this unilaterally. The client may also assume that the cancellation will take effect immediately when the server receives the RST_STREAM frame before any other data from that TCP connection is processed.

Google says: "This attack is called Rapid Reset because it relies upon the ability for an endpoint to send a RST_STREAM frame immediately after sending a request frame, which makes the other endpoint start working, and then rapidly resets the request. The request is canceled, but leaves the HTTP/2 connection open for additional requests and cancellations. Therefore, the HTTP/2 Rapid Reset attack which is enabled by this capability is simple," says Google. "The client opens a large number of streams at once as in the standard HTTP/2 DDoS attack. But rather than waiting for a response to each request stream from the server or the proxy, the client cancels each request immediately. And since HTTP/2 specifies that the client may assume that any canceled stream is immediately available for another request, the HTTP abusing attacker may immediately follow a stream reset with another new request using the same stream.

"The ability to reset streams immediately allows each connection to have an indefinite number of requests in flight. By explicitly canceling the requests, the attacker never exceeds the limit on the number of concurrent open streams, which is typically 100. The number of in-flight requests is no longer dependent on the round-trip time, but only on the available network bandwidth.

"In a typical HTTP/2 server implementation, the server will still have to do significant amounts of work for canceled resets, such as allocating new stream data structures, parsing the query and doing header decompression, and mapping the URL to a resource. Moreover, in reverse proxy implementations, the request may be proxied to the backend server before the RST_STREAM frame is processed. This requires the proxy to then forward the stream reset to the appropriate back-end server. By comparison, the attacking client has almost no cost for sending the requests. This creates an exploitable cost asymmetry between the server and the client."

Leo: Oh, I was just saying there's the problem. Clearly.

Steve: Yup. Exactly.

Leo: Right? It's from a spam. It costs them nothing to send it.

Steve: Exactly.

Leo: Yeah. Yeah. Wow.

Steve: And so another advantage the attacker gains is that the explicit cancellation requests immediately after creation means that a reverse proxy server, which is what Cloudflare has on their perimeter and all the big cloud providers do, won't send a response to any of the requests. Canceling the requests before a response is returned thus reduces the returning bandwidth to the attacker. Meaning the upstream back to the attacker won't get clogged up with much bigger replies since they've been canceled. So all of the work is loaded onto the servers with very little traffic returning. Although it's diabolical, it's not a flaw in HTTP/2. It's just an abuse of a deliberate design feature.

So what does Google recommend? They begin by explaining. They said: "We don't expect that simply blocking individual stream requests is a viable mitigation against this class of attacks. Instead, the entire TCP connection needs to be closed when abuse is detected." Get this. "HTTP/2," they wrote, "provides built-in support for closing connections, using the GOAWAY frame." You've got to love that. The designers of the HTTP/2 protocol defined a "GOAWAY" frame that could be sent back up the link to the client telling it to, well, go away.

Google says: "The RFC defines a process for gracefully closing a connection that involves first sending an informational GOAWAY that does not set a limit on opening new streams, and one round trip later sending another that forbids opening additional streams." So there is a process basically for allowing the client to gracefully shut down rather than just terminate all of its pending queries. "However," they said, "this graceful GOAWAY process is usually not implemented in a way which is robust against malicious clients."

Leo: Well, we've got to work on that.

Steve: So that, maybe we're going to see about fixing that.

Leo: We need a graceful GOAWAY, for sure.

Steve: "This form of mitigation leaves the connection vulnerable to rapid reset attacks for too long, and should not be used for building mitigations as it does not stop the inbound requests. Instead, the GOAWAY should be set up to limit stream creation immediately." So maybe we will see an HTTP 2.1 that will tweak the definition of this.

They wrote: "This leaves the question of deciding which connections are abusive. A client which cancels requests is not inherently abusive. The feature exists in the HTTP/2 protocol to help better manage request processing. Typical situations," says Google, "are when a browser no longer needs resources it had requested due to the user, for example, navigating away from the page, or applications using a long polling approach with a client-side timeout." Mitigations or, you know, maybe when you've got multiple mouse pointers, and they're in disagreement about what should be done next.

Leo: You see? There's a good reason for that browser.

Steve: "Mitigations for this attack," they said, "can take multiple forms, but mostly center around tracking connection statistics" - basically detecting abuse - "and using various signals and business logic to determine how useful each connection is. For example, if a connection has more than 100 requests with more than 50% of the given requests canceled, it could be a candidate for a mitigation response. The magnitude and type of response depends on the risk to each platform, but responses can range from forceful GOAWAY frames as discussed before to closing the TCP connection immediately." Okay, just hang it up. Basically go away, then hang up. "To mitigate against the non-canceling variant of this attack, we recommend that HTTP/2 servers should close connections that exceed the concurrent stream limit. This can be either immediately or after some small number of repeat offenses."

Leo: In other words, there's a fingerprint for this kind of attack.

Steve: Yeah.

Leo: You can immediately detect.

Steve: Yeah, it's very obvious if like every request is immediately being canceled. You know, I'd hang up on that jerk immediately.

Leo: Yeah, yeah. Well, that's good news. I mean...

Steve: Oh, yeah. So that's the story. Exactly as Google initially characterized the problem, the deliberate design decision of HTTP/2, which can be used to significantly increase its efficiency, and does increase its efficiency, can also be used to significantly increase the potency, the potential and the potency of DDoS attacks. Attackers figured out that, rather than using HTTP/2 simply to simultaneously ask for tons of resources and then be flooded by their return, they could instead immediately cancel their request and reissue another. Against this threat, today's servers, especially those in the cloud which have distributed their request handling among multiple backend components, which might make canceling those issued requests much more tricky because those also have to be distributed, make it more time consuming, would be seriously taxed by this new attack strategy.

It will be interesting to see whether anything can be done to change HTTP/2 to prevent or limit this abuse. At the moment, the various servers are testing themselves and modestly tweaking their request handling to do a "less bad" job of dealing with this abuse of this HTTP/2 feature. Since, as I mentioned above, it's not possible to spoof the IP addresses of anything that's riding on top of TCP, the best solution might be to dynamically blacklist, or at least significantly throttle, any IP that is found to be abusing HTTP/2 Rapid Reset. In that way, the bots would be recognized and quickly ignored at the perimeter of a large hosting provider like Cloudflare.

Leo: We do something like that all the time with password mitigation; right? You don't let somebody log in - my light just went out. It's a sign, isn't it. You don't let somebody log in an infinite number of times, one after the other. In many cases you just say, oh, six times you're out, or 10 times you're out.

Steve: Yeah.

Leo: So this would be a simple mitigation. It doesn't change the protocol, though; right? It changes the way the server operates.

Steve: Exactly.

Leo: Yeah. You would - see, I feel like this is a really good example of - and I'm sure these days when you're in an IETF meeting or a W3C meeting, and they propose these protocols, they hammer, you know, they try to think of ways people would take advantage of it. And somebody just forgot to say, oh, you know, somebody could just keep sending rapid resets, and it would overwhelm us because we can - we'll start a million threads per user. We should build into the protocol there's a limit. Or, you know...

Steve: Right.

Leo: And they just didn't. They missed it.

Steve: Right.

Leo: It's not that the feature's a bad feature; right? I mean, the idea of these with a well-behaved client is fine.

Steve: Well, and consider that this is eight years old, that is, the protocol.

Leo: Just took them a while.

Steve: It took them this long, I mean, it took the bad guys this long to go, hey, you know?

Leo: So I'm not surprised that a bunch of engineers didn't figure it out and say...

Steve: Our attack, we're only able to have 100 concurrent outstanding requests. So we're having to wait for the replies to come back to free up a stream. So why don't we just cancel that request? Then we don't have to wait for the answer.

Leo: Yeah. And it's pretty - I think it would be pretty easy to spot somebody behaving badly, that there would be - I guess the key is that there's enough distinction between a well-behaved user - this is the problem. If it's not distinct, you don't want to hang up on, well, this is why SYN/ACKs work because you can't tell the difference between somebody who's sending you a reasonable SYN and somebody who doesn't care. But if you could tell the difference between a bad actor and a normal user pretty readily, which it sounds like you could, I think mitigation might be doable.

Steve: And here's where the term "heuristic" comes to our aid.

Leo: Oh, yeah. That's a heuristic, isn't it.

Steve: We would be using a heuristic. We would say, you know, we've done stats on, you know, for like the last month, on all of the HTTP/2 connections. We've never seen more than 5%.

Leo: Exactly.

Steve: Rapid resets.

Leo: Yeah.

Steve: So if anybody does 25, you know, they don't deserve our attention today.

Leo: Right. You always want to have it so that legitimate - it's like spam [indiscernible]. Legitimate users, you don't want false positives. So you want legitimate users who are doing things right, but maybe are demanding.

Steve: Yeah, I mean, how many times do we have - is someone required to say, if you didn't get our email, check your spam folder. Because presumably you wanted that because you were talking to them, but it got routed into spam because unfortunately we've, you know, again, a heuristic that is not sharp enough.

Leo: Yeah, it was too aggressive. This one seems like it would be doable, but maybe, I don't know, maybe it's not.

Steve: Well, and that's I'm sure what the server vendors are scurrying around now doing in anticipation of this. And I should mention that all of these servers have the ability to turn off this protocol. So if you were your own HTTP/2 site, and you were being attacked, just tell your server to turn off support for HTTP/2, be a 1.1 server. And yes, your performance will drop, but it's better than being held off the Internet by one pissed-off bot.

Leo: It really drops when you get 20,000 people doing hundreds of requests.

Steve: I mean, I wouldn't be at all surprised if one attacker could not hold, I mean, not Cloudflare, but a standard website with no upfront protection.

Leo: Well, and a static site like mine, for instance, I'm not worried about mine being DDoSed, but a static site like mine, for instance, I don't need HTTP/2.

Steve: No.

Leo: HTTP/1.1 would be fine.

Steve: Yup.

Leo: I'm using NGINX. Is NGINX one of the server...

Steve: Yes, NGINX is great, and they are working on a mitigation.

Leo: We use DOS mitigation on our site, but I won't tell you by whom. Not Cloudflare. But there's a lot of companies that offer DOS mitigation, DDoS mitigation.

Steve: Anybody who's curious could follow your track [crosstalk].

Leo: I guess it wouldn't be hard to figure out.

Steve: It would go through your DOS mitigator.

Leo: All right, Steve. What a great, great topic. Maybe I was more engaged because they weren't feeding me and asking me questions and saying, Leo, can you come down the hall and check on this. So maybe I was more engaged. But what a good, I think a really good show, really good.

Steve: Yeah, well, and I think it's important for us all to note that there is now a new DDoS technology, a way of doing a DDoS that is effective against the latest HTTP/2 protocol, which nearly 40% of the web, 39% of the web is using.