CNAME Collusion

Description: This week we discuss a welcome change coming soon to the Chrome browser, and a welcome evolution in last week's just released Firefox 86. We're going to look at questions surrounding the source of the original intrusion into SolarWinds servers, and at a new severity-10 vulnerability affecting Rockwell Automation PLC controllers. We'll touch on VMware's current trouble with exploitation of their vCenter management system, and I want to share a recent code debugging experience I think our listeners will enjoy and find interesting. Then we're going to conclude with some information about something that's been going on quietly out of sight and under the covers which must be made as widely public among web technologists as possible.

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-808.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-808-lq.mp3

SHOW TEASE: It's time for Security Now!. Steve Gibson is here. Lots to talk about, including breaking news. Seven, count them, seven zero-days in Microsoft Exchange Server. And we'll talk about a sneaky new thing that some websites are using to track you, even if you have tracking protection turned on. We'll tell you how to avoid it, CNAME collusion, coming up next.

Leo Laporte: This is Security Now! with Steve Gibson, Episode 808, recorded Tuesday, March 2nd, 2021: CNAME Collusion.

It's time for Security Now!, the show where we cover your security and privacy and well-being online with this cat right here, Mr. James - James? Started calling you James Tiberius Gibson. But no.

Steve Gibson: I knew where you were going with that, yeah.

Leo: Steve Gibson of the GRC Corp. He has danced...

Steve: It was the Vulcan wave.

Leo: That's what did it.

Steve: The Vulcan hand sign. That's what got you tripped up.

Leo: That's what did it, yup. I don't know what Mr. Spock's middle name is. I don't even know what his first name is. But I do know James Tiberius Kirk.

Steve: Actually we're having fun. We are rewatching "Fringe," which Lorrie never saw.

Leo: Oh, yeah, yeah.

Steve: And of course he is Bell. And at the end of the first season he comes out of the shadows. And of course I recognized his voice.

Leo: Shatner. You knew immediately, yeah.

Steve: Yeah. And then she's like [gasps]. So anyway, we're having fun with that.

Leo: Nice.

Steve: This is going to be one of those episodes, Leo. This is important. No fooling around this time. There's something which has come to light which is a form of escalation of the browser tracking fight which is very disturbing because of what the tracking companies are asking websites to do, thus the "collusion" part of this. There is collusion involved. And the consequences of what's been done in order to avoid those who wish not to be tracked, to circumvent anti-tracking, is deeply disturbing.

So there was more news, but I wanted to be able to spend enough time on this because, I mean, we do have news. But as I looked into this, I just thought, okay. And I say this in the show notes later because I just wrote this about 10 minutes ago. We can't fault people when cookies began being abused by third parties, even though that was never their design, because people didn't know. I mean, random users, they didn't see the tracking going on. The technologists knew, and we did nothing. And we're here again. We're at something that is going on that no one is seeing, that by the end of this podcast every one of the podcast listeners is going to understand. And something - we just can't do nothing again.

Anyway, the topic, the name of the podcast is CNAME Collusion. I was going to go with the CNAME Conspiracy, but that word's been a little overused lately. And so I thought collusion sounds better, and it's actually more accurate. But first we're going to discuss a welcome change coming soon to a Chrome browser, probably near you. And a welcome evolution in last week's just-released Firefox 86, which actually 86's something. We're going to look at questions surrounding the source of the original intrusion into SolarWinds servers, which there's some controversy about. The old CFO and the new CFO have differing opinions about how this thing crawled in in the first place.

Also we're going to look at a new severity-10 vulnerability affecting Rockwell Automation PLC controllers. CISA is warning the world about this. We'll touch on VMware's current dilemma with the exploitation of their vCenter management system as a consequence of a patch they just released which was instantly reversed and for which there are, last time I checked, six public exploits which immediately started getting used. I also want to share, because I think our listeners will find it really interesting, I certainly did, a recent code debugging experience I had with that system that I mentioned last week, that it

was the only system that I had that was still causing me trouble. And I now know why. And, oh, is it weird.

And then, as I said, we're going to conclude with some information about something that's been going on quietly out of site and under the covers, which must be made as widely public among web technologists as possible. And so we're going to do our part on this podcast.

Leo: As always. That sounds interesting. I will be staying tuned. Not like I can go anywhere, but I'll be listening with interest.

Steve: Just don't fall off your ball, Leo.

Leo: I might. Depends on how shocking the CNAME Collusion is. All right, Steve.

Steve: So our Picture of the Week is just sort of a fun one. I titled it "Not exactly confidence inspiring." And this is showing an ATM, a contemporary-looking ATM. And on the screen it shows - apparently the processor has crashed and rebooted, and it shows Microsoft Windows NT Workstation. And I went down, I squinted at the screen, and I thought, oh, well, at least it's Workstation 4.0. So that's good. It's not 3.51 or whatever it used to be.

But anyway, yes, if your ATM is running on Windows NT, I mean, that doesn't mean that it's insecure because of course new viruses won't infect Windows NT. But it does sort of suggest that it isn't receiving any regular maintenance and love and care, that maybe the 300 baud modem which it's using to connect to the rest of the world is the reason why your transactions are running a little slowly.

Leo: Used to be you'd hear that. You'd hear the [mimicking modem noise].

Steve: Oh, yeah.

Leo: It's also a little concerning that we're seeing the boot screen on the ATM. I don't think that's a good sign, either.

Steve: That's true. Yeah. That probably means that the 10MB hard drive...

Leo: It's full.

Steve: ...is retrying or was unable to get off the ground or lord knows. Maybe the floppy that it's running inside is having read errors. So I am delighted to announced that the forthcoming Chrome v90, which is slated for release around mid-April, will finally assume that any non-specific, or as Google terms it, "schemeless" URL which is entered into its omnibox, which is what they call the URL field, will be assumed HTTPS before falling back to HTTP. As our listeners know, I've mentioned often that it seems well past time for our browsers to assume HTTPS rather than HTTP. They don't do that yet. But it appears that's finally going to happen. This will likely influence certainly all other Chromium-

based browsers, and we can expect that probably Firefox and Safari will follow suit. I think it's 80 - I saw the figure recently, actually when I was researching this. Eighty-some, oh, maybe it's in here.

So last Wednesday Google's Emily Stark tweeted, if you're running Chrome Canary, dev or beta, so that's not the one we get yet, or that the rest of us have yet, and you want some more HTTPS in your life, which, okay, you can't avoid having it in your life. Anyway, she said: "Go to chrome://flags and search for 'omnibox-default-typed-'" - actually all you have to do is "omnibox-," and you could type a "d" for default. I did it. There aren't any other collisions in the search. So anyway, it's omnibox-default-typed-navigations-to-https, which allows you to enable Chrome first trying HTTPS rather than HTTP by default.

Anyway, and then the next day she followed up, tweeting: "Currently, the plan is to run as an experiment for a small percentage of users in Chrome 89" - okay, so that will be the next one we get, which probably is sometime soon, later this month, and then will launch fully. Assuming presumably that nothing blows up from their small percentage of user tests in 89, it'll be on in Chrome 90. So, yeah, the current public release is 88. And I checked. 88 doesn't yet have anything like that option. 89 will. So a test percentage of 89 users will get that. The rest of us, if we're curious, and I will when 89 comes out, will turn it on because it's clear that when you just type GRC.com, you should go to https://GRC.com.

I have long had code that redirects any HTTP person coming in immediately to HTTPS. Well-behaved sites do that. But again, HTTP should die eventually. Certainly the browsers are doing everything they can to kill it off. So, yeah, this is just, you know, makes sense that this would eventually happen. Chrome is leading the way. Edge and Brave and everybody else, Vivaldi, all the other Chromium-based browsers will probably do the same thing. And I imagine then Firefox and Safari will follow.

Leo: So Brave does HTTPS Everywhere. I think it has it built in. Is that the same thing?

Steve: Okay, yeah, kind of. So that says it's typically an add-on on current browsers.

Leo: Yeah, it's a Chrome extension, I think, yeah.

Steve: Yes. Which does transmute the non-specific URL into HTTPS. So yes. Basically that HTTPS Everywhere, our listeners know, has been around for a couple years now. It's finally getting built in. But so if you don't specify, Chrome will go HTTPS first. If that doesn't answer, there's nobody home there, then they'll go, huh, and try HTTP. And of course then all kinds of other alarms and bells are going to go off because Chrome doesn't want to talk to anybody over HTTP. It's like you get all kinds of scary, like oh, wait a minute. And like, yeah, yeah, yeah. But I think it was 83% of all websites are now HTTPS, which is really great.

Leo: That's great. And that's - you can thank Chrome for that because Google really pushed this; right?

Steve: Yes.

Leo: They said you'll rank higher in the search.

Steve: Chrome and Let's Encrypt. Let's Encrypt was arguably...

Leo: They made it easy, yeah.

Steve: Well, they made it free. That was the big deal.

Leo: Right.

Steve: All the old grumbly Linux farts [cross grumbling]. That's not the way the Internet was meant to be. It's supposed to be free.

Leo: Well, it is pricey, I have to - or was pricey.

Steve: As in beer or something. What is it, beer free? I don't know if beer is free.

Leo: No, it's not. It's liberated.

Steve: Liberated, yes. Anyway.

Leo: So, yeah, because Brave does have that. So Brave has done that for a while, for as long as I can remember, this HTTPS.

Steve: They may have been, you know, maybe Chrome was looking at that going, huh, that works better than we thought. So, yeah. So, okay. So that's the first bit of good news on the browser front. The second is that apparently my wish list is getting some long overdue attention this week. Mozilla just announced that with their recently released Firefox 86, the long-running abuse of third-party cookies would finally be 86'd.

Of course as I've long lamented, and we'll be talking about this toward the end, the use of third-party offsite cookies for tracking was never part of the plan. Netscape invented first-party cookies in order to implement a simple session maintenance mechanism which for the first time back then, and still today, enables the concept of a user logging into a website and then being known as they moved about. Essentially it amounted to them being tracked as they moved about that one site. But being a first-party cookie, it only worked for that one site.

What was never intended was that third-party advertisers, or dark and unseen analytics providers, or Google Analytics, would insinuate themselves pervasively throughout the web and employ their own cookies for tagging and tracking the activities of individual users. But as we know, what can be done will be done. And tracking is what resulted.

So here's what they've done. With the release of Firefox 86, that just ended. I mean, like really. At the top of their posting titled "Total Cookie Protection," Mozilla wrote: "Today we are pleased to announce Total Cookie Protection, a major privacy advance in Firefox

built into ETP Strict Mode. Total Cookie Protection confines cookies to the site where they were created, which prevents tracking companies from using these cookies to track your browsing from site to site."

Okay. So in other words, third-party cookies are not blocked, but they are stovepiped. So assuming, for example, that third-party cookies were enabled at all in your browser, which is the case typically, any third-party entity may still give the user's browser its cookie, like an advertiser or like Google Analytics or like anyone providing content to your page when you visit a specific site. They're welcome to give your browser its cookie. But now, I mean, like Firefox right now, I have 86, and you get it. If you go to About Firefox you'll probably see that you have 85, and then it'll offer - although I did get an update yesterday without asking.

But now Firefox will associate the third-party cookie it received with the website where the user was when that cookie was received. The two will be paired. So if the user returns to the same site, that third-party cookie will be returned to the third-party site. So it's not breaking third-party cookies. But if the user visits a different site with content, an advertisement or tracking code from that same third-party site, because the user is visiting a different website, there will be no third-party cookie associated with the visited site. So cross-site tracking is defeated.

Now, if we wanted to use some modern high-falutin' language to describe this, we would say that Mozilla has segmented their browser cookie name space, creating individual cookie name spaces for each website. So I think this is it. I mean, this is a wonderful compromise between allowing and refusing all third-party cookies just as a binary choice. With this solution, third-party cookies are still allowed. Nobody can say, oh, you turned off third-party cookies. You're bad. No. They're on. But you don't get that same third-party cookie when you go somewhere else that has that same third-party site. You're at a different location. So it's treated as a new third-party cookie, not cross-associated among websites.

So this is just great. We know that the pressure to track is significant. Even though, when it really does work, it's frequently reported as being a bit unnerving, like when some recent activity that you have had somewhere, like you're in Amazon searching for something, that turns up in an ad somewhere else a few minutes later. That freaks people out. It's like, wait a minute. I mean, notice that when people actually experience being tracked, they're not happy. They're like, wait. But most of the time you don't experience it. And I'm not convinced it even actually provides any substantial, like, worthwhile revenue. But it sure is a big business. So anyway, big props to Mozilla for adding this welcome and long-overdue feature to Firefox.

Okay. SolarWinds. It's going to be a while before we really get through with this topic.

Leo: The gift that keeps on giving, yeah.

Steve: Yeah. As we know, post-intrusion forensics is always difficult. Like the famous Hacking 101 is, well, once you break in, you delete the log files of your break-in, and thus you eliminate some of the tracks. So that's sort of a classic example. But, yeah, you can kind of do that. And we've talked about how much the designers of the intrusion, how much effort they went to to deliberately thwart post-intrusion forensics from figuring out where they came from. Like to decouple the intrusion from SolarWinds into other systems, to prevent it being tracked back to SolarWinds.

So the previous SolarWinds CEO, Kevin Thompson, says that it may - this was in front of the Senate or the House, I guess it was, a representative - oh, yeah, the U.S. House of

Representatives Oversight & Homeland Security Committee. He said: "It may have all started when an intern" - right, blame it on an intern, they're gone - "when an intern set an important password to solarwinds123." Fortunately, not monkey. But still it was SolarWinds password set to solarwinds123. And then, adding insult to injury, the intern posted it on GitHub.

Leo: Love it.

Steve: Oh. Kevin Thompson, he told a, as I said, a joint U.S. House of Representatives Oversight & Homeland Security Committee hearing that the password was "a mistake that an intern made. They violated our password policies, and they posted that password on their own private GitHub account. As soon as it was identified and brought to the attention of my security team, they took that down."

Okay, now, as soon as, you know, when you learn to speak that way, you get the C of the CEO in front of your designation. Kevin's responsible-as-possible-sounding testimony was contradicted by SolarWinds' current CEO, Sudhakar Ramakrishna, who confessed that the password solarwinds123 was in use by 2017. The researcher Vinoth Kumar, who discovered the leaked password to one of SolarWinds' file servers had earlier stated that SolarWinds did not change the password until November of 2019, after he had discovered it on the Internet and reported it. So about two years that password solarwinds123 was in use. Now, the insecure password - yeah.

Leo: You have to wonder at all the employees who use it who didn't think twice.

Steve: Yes, exactly. Exactly. In two years that's how they logged onto that server. Oh, thank goodness this is easy to remember.

Leo: So easy to remember.

Steve: Don't have to look it up. It's not a bunch of gibberish. I can just say it over the phone. How nice. The insecure password is one of three possible avenues of attack, which SolarWinds has been investigating as it tries to determine how it was first compromised by the hackers. The two other theories are brute-force guessing of company passwords, so-called "credential stuffing," as we call it now, as well as the possibility the hackers could have entered via compromised third-party software. That's right. Let's blame it on somebody else, if not this intern that we had a couple years ago whose password was allowed to sit there for several years.

Then some other third-party software, and of course they're the third-party software that really did the damage. In other words, they don't know for sure. And since post-intrusion forensics is difficult, we might never be certain. But publicly posting a private password on GitHub is certainly not the way to keep it a secret.

Okay. Speaking of things that you really wish you could keep secret, but you can't, Rockwell Automation has a CVE-2021-22681. It is a 10 out of 10 Critical, assigned by CISA. The security of nearly all of Rockwell Automation's PLCs, which are Programmable Logic Controllers, are affected by the use of a single globally shared static encryption key. Oh, yes. Every one of the, now, I wrote "hundreds of thousands." It's probably millions. I mean, these things are everywhere. I'll explain what PLCs are. But they're all

"protected," in air quotes, by the same single key. So of course there just might as well not be one.

The Programmable Logic Controller fills an important gap within any process control system, or an important need. For example, you might be on an oil rig, where the pressure of a feeder line, and I don't even know what that is, but it sounds good, a feeder line must be maintained within a range. But the pressure might need to be taken at several different places and averaged. And there might be multiple upstream sources of pressure controlled by valves with actuators. So in a sense it's a small closed system having a handful of inputs and outputs. And once its function is defined, it can and should just be left alone to work. You just want that feeder line pressure maintained. And if you've got to open some valves using actuators to maintain the pressure, then that's good. And you want to average the pressure in a few locations. So not a big complex job, but it's a job that needs to get done.

So how do you build the control system for that? Once upon a time, before computers, a custom circuit would have been created, and some tech would have built it from scratch. Today, you go over to Rockwell Automation's website and pick the PLC, the Programmable Logic Controller, that's just big enough to - because there's a whole bunch of them of different sizes. You want the one that's just big enough to handle the number and types of inputs and outputs that you need. Then an engineer who's been trained up uses Rockwell software, which is called Studio 5000 Logix Designer, to program the little computer that resides inside that industrial oil rig-tough little box. And you hook it up. And that little world will now take care of itself.

So a PLC does a limited set of very specific things. Once upon a time, as I said, it might have been done with discrete circuitry, like a bunch of clacking relays, all wired together to implement the sequencing logic required to route vials of newly synthesized vaccine around their carousel, counting them as they pass, and then to flip routing gates open and closed at the exact time needed to fill the waiting containers. Another typical job for a PLC. But today all of that is handled, not with a bunch of clacking relays, but by an unseen Rockwell Automation PLC. It's programmed once, and it effectively becomes part of the overall machine.

In any industrial setting, where things are moving, spinning, whirring, valves are opening and closing and stuff's happening, there are tasks that don't require a general purpose computer. And god knows you sure don't want Windows anywhere near the control loop of systems like that. It'll decide it's time to update, and your vaccine vials will go flying all over the place. So yes, Windows hosts Rockwell's Studio 5000 Logix Designer app, which is used in a cool way to interactively design the logic that will be programmed into this. But once that PLC device is programmed, it's blessedly off on its own. Leave it alone, it'll just get the job done.

So everything would be great with these workaday PLCs. But apparently some bozo somewhere decided that needing to go down to the shop floor to tweak the controller of the machine that's squeezing bottle caps onto Coke bottles was too much to ask. So let's put it on the network. Believe it or not, these perfect little happy worker controllers have received IP addresses. Stop me if you've heard this one before. What could possibly go wrong? And yes, as I mentioned above, not only do they have IP addresses, often with public presences on the Internet, again, things that are just supposed to get their job done, just leave them alone. They're working. But they are all being protected by the same, now well known, cryptographic key.

Last Thursday, the U.S. CISA warned of a critical vulnerability, giving it a 10 out of 10, which is a hard score to earn, that 10. You can get to 9.8 without trying that hard. But getting a 10, it's like the Olympics. So that allows hackers to remotely connect to Logix controllers and from there alter their configuration or applicable code.

CISA stated that the vulnerability requires a low skill level for exploitation. Quoting them, they said: "The vulnerability tracked as CVE-2021-22681 is the result of the Studio 5000 Logix Designer software making it possible for hackers to extract a secret encryption key." And they didn't put "secret" in quotes, but, you know.

Leo: Semi-secret.

Steve: Yeah. We wanted it to be secret. So could you please, let's keep it a secret, everybody. Let's agree. "This key is hard-coded into both Logix PLC controllers and engineering stations, and verifies communication between the two devices. A hacker," they're writing, "who obtained the key could then mimic an engineering workstation and manipulate PLC code or configurations that directly impact a manufacturing process." It's just why must we put everything on the Internet, Leo. We just, you know...

Leo: Because it's there, Steve. Because it's there.

Steve: But people's garage doors are on the Internet. And, I mean, if it moves, hook it up. Give it an IP address.

Leo: That's it.

Steve: Oh, my lord.

Leo: My garage door is on the Internet. Is that a bad thing?

Steve: I really didn't mean to pick on you, Leo.

Leo: It makes it easy to open it from anywhere.

Steve: On the other hand, I should have said, of course your garage door is on the Internet.

Leo: Of course it is.

Steve: We know your front door is on the Internet.

Leo: No, no, no. Well, it is. The camera. Yeah, yeah, yeah.

Steve: I know. I know.

Leo: You're right. You're right.

Steve: Okay. So VMware's got some problems.

Leo: Uh-oh.

Steve: I mean, it's not their fault. They did things responsibly. As we know, there are companies that get notified of a problem, and they don't fix it for a long time, and they force the people who notify them to release the information for the benefit of the world since the company refuses to do anything until apparently made to do so. That didn't happen here. What happened was a very bad flaw was found in the entire population of VMware's vCenter servers. VMware could do nothing but patch it; right? Like here, everybody update your VMware vCenter systems. Unfortunately, they're a big target.

And the bad guys got right on this. As a consequence, hackers are currently mass scanning the Internet in search of VMware servers that have not yet patched this newly disclosed code execution, remote code execution vulnerability which carries a severity rating of 9.8 out of 10. As I said, 9.8 you can get. To hit 10, you've got to really be doing something.

Leo: You've got to be good. You've got to be really good.

Steve: Yeah. So the VMware problem is CVE-2021-21972. It's a, as I said, remote code execution vulnerability. vCenter is an application for Windows or Linux used by admins to enable and manage virtualization of large networks. Within a day of VMware issuing a patch for this very bad problem, proof-of-concept exploits appeared from at least six different sources. The severity of the vulnerability, combined with the availability of working exploits for both Windows and Linux machines, immediately motivated hackers to scramble to find vulnerable servers.

Troy Mursch, a researcher with Bad Packets, wrote: "We've detected mass scanning activity targeting vulnerable VMware vCenter servers." He said that the BinaryEdge search engine - which we know is sort of another version of Shodan. The BinaryEdge search engine found almost 15,000 vCenter servers exposed to the Internet, while Shodan searches revealed about 6,700. The mass scanning is aiming at identifying servers that have not yet installed the patch.

So the flaw is just about as bad as it gets. It allows a hacker with no authorization to upload files to vulnerable vCenter servers that are publicly accessible over port 443, which as we know is TLS, HTTPS. Successful exploits will result in hackers gaining unfettered remote code execution privileges in the underlying operating system. The vulnerability stems from a lack of authentication in the vRealize Operations plugin, which unfortunately is installed by default. So they're all going to have it.

In their blog posting, Positive Technologies, who discovered and responsibly privately reported the flaw to VMware, wrote: "In our opinion, the RCE (Remote Code Execution) vulnerability in the vCenter Server can pose no less a threat than the infamous vulnerability in Citrix." And here they're referring to CVE-2019-19781, which was widely implicated in the mass of ransomware attacks on hospitals, those early ones, back in 2019.

They said: "The error allows an unauthorized user to send a specially crafted request, which will later give them the opportunity to execute arbitrary commands on the server." So they're being a little bit cagey because they don't want to give away everything. They're being responsible still. They said: "After receiving such an opportunity" - you

know, on the other hand, right, six public exploits with full source are available. So the cat's out of the bag.

"After receiving such an opportunity," they wrote, "the attacker can develop this attack, successfully move through the corporate network, and gain access to the data stored in the attacked system, such as information about virtual machines and system users. If the vulnerable software can be accessed from the Internet" - which of course is the case in vCenter systems, those 15,000-plus of them - "this will allow an external attacker to penetrate the company's external perimeter and also gain access to sensitive data. Once again," they write, "I would like to note that this vulnerability is dangerous, as it can be used by any unauthorized user."

So, yeah, we got the message. Again, it's one of those races against time, essentially. One day after this patch was released, six proofs of concept existed, and mass scanning began. So my mantra has been make sure you are maintaining an open line of communication with the vendors of all the front line equipment that you're maintaining and make sure this doesn't go into a, yeah, we do these on Friday sort of mailbox, but really comes to the attention of somebody who it may be necessary to get out of bed or, yes, reboot a server that you'd rather not reboot because you're going to have to kick everybody off that's currently using it. But boo-hoo. This thing needs to get fixed now. So this is one of those.

And there's just, you know, there isn't any way around this. They had to release the patch. The only thing they could do was to hope that the news got out, that the people who are in responsible positions would respond to it instantly, inside of the timeframe required to reverse-engineer it and for the bad guys to start scanning. Who knows how, I mean, and this is going to typically be a larger company running a VMware vCenter. They have a huge percentage of install base. I saw that it was like 80% market share at the high end of this kind of virtualization up in the high fortune companies.

So those are the targets that the ransomware guys want. So no surprise that this was, you know, not only was it simple to execute once you had the proof of concept code, obviously it's simple to reverse engineer because it only took a day. But the target was juicy. The target set would also be juicy, targets of opportunity. So this thing just had everything.

Last week - I want to share with our listeners, and I think everyone will get a kick out of it - I mentioned that one troublesome machine owned by a tester in Germany was again causing my new code some trouble. Back in the earlier ReadSpeed Benchmark development days, I was worried about wearing out my welcome with him because, like, he had a problem. And the only thing I could do was to produce a series of test releases in what was ultimately a futile attempt to zero in on the trouble and fix it. I mean, I added auditing code that was spitting out like leaving breadcrumbs as it went. And then they just disappeared. And it's like, okay. And then I would try something else, and I would zero in on where the breadcrumbs disappeared, you know, did everything I could remotely by just giving him test after test after test to try and run and then report on what happened.

Then, miraculously, his system started working. And of course that always makes me nervous because if you don't really do something to fix a problem, then that miracle might choose to reverse itself at any point. And sure enough, when I moved the new code over into SpinRite and released the first test releases of it there, it no longer worked on his system. So I had purchased one of the very same old Gigabyte motherboards from eBay. And it came last week. So, and I may have had it, in fact, but hadn't - what?

Leo: You got it.

Steve: Oh, yeah, yeah. In fact, I have...

Leo: Such dedication. I can't believe it.

Steve: This is, oh, that's standard operating procedure for me. I've got controllers and drives and motherboards. And a lot, almost everything I can do just by sort of looking at the symptoms, looking at an audit, and then making a good guess. And then the guy will say, yay, it works now. It's like, congratulations. Like, okay, good. Whew. But sometimes it's just nothing I can do. So I put the motherboard in an ATX case that I had.

Anyway, I thought I'd give our listeners a sneak peek into this microdrama because it demonstrates a bit about the process of debugging code and serves as a beautiful example of the weird sorts of things that we face in the real world where code which is born in the lab actually needs to function in the real world eventually. And as it turns out I now know I could never have figured this out remotely. I mean, I would have thought he was nuts. I mean, I just wouldn't have known what to think because, even with the machine sitting in front of me, what I saw made absolutely no sense.

The problem was occurring in a simple routine which copied the contents of a disk sector buffer from high XMS memory above 1MB down into traditional x86 segmented memory below 1MB. Should be a piece of cake. But the machine went into that subroutine, and it never came back, never came out. So, okay. At least now I had apparently located the location of the problem that Chris and his machine in Germany was having. So I fired up my debugger. And I followed the processor into that simple subroutine.

As we've talked about a lot, one of the reasons I find the Intel chips enjoyable to program, and also why I never want to program a RISC chip, an ARM-based architecture, for example, is that the Intel chips have a CISC architecture, Complex Instruction Set Computer, CISC, C-I-S-C. The ultimate example of a CISC ISA, an ISA, Instruction Set Architecture, was probably the DEC PDP-11 and the VAX machines. They were designed back at a time when a lot of code was still being written in their assembly language. And when compiler design was still sort of a nascent art, I mean, those were the early days. So the chips themselves presented a sort of high-ish level language at the assembly language, at the machine language, to the people who were going to program them. They wanted to make it pleasant.

Okay. So for example, the Intel x86 architecture includes an instruction that I used in that subroutine. It's a byte-range copy instruction that no self-respecting RISC chip would ever abide. They'd be like, what? We're not copying a range of bytes. We do one thing really well. And if you want more of that, you've got to ask for it. But not Intel. So the starting address of the source range is placed into the chip's SI register, SI standing for Source Index. I'm not kidding. And the starting address of the destination range to copy to is placed into the chip's DI register, DI standing for Destination Index. And the number of bytes to be copied is placed into the CX register, C as in Count.

Then a single instruction, like one byte opcode instruction is executed, which causes the heavily microcoded Intel chip, or actually in this specific case an AMD processor, which is a Phenom II that was on this Gigabyte motherboard, to fetch a byte from where the SI register points, store it to where the DI register points, increment both the SI and DI registers so that they will now each be pointing to the next byte in their ranges, then decrement the CX register. If the CX register has not just been decremented to zero, then repeat. Copy the next byte and so on.

So essentially, by executing a one-byte instruction, after setting things up by putting specific pointers into specific registers, the Intel will copy a block of data from one place to another. So I'm explaining all this because as I single-stepped the processor, instruction by instruction, watching it like put the data for S into the SI register, no problem. Put the data into the DI register. That worked. Store the 512-byte count into CX. Yup. I then stepped into that byte range copy instruction, and nothing happened. It was as if the instruction was taking forever to execute. It just went into the debugger. The debugger didn't come back to me. So one of the tricks we all learned, and Leo, I know you know, back in the early days of the PC, and a system appeared to lock up, was to hit the Num Lock on our keyboards a few times.

Leo: Forgot about that.

Steve: And, oh, yeah, I still do it. Of course I'm still living back in this world.

Leo: It doesn't do anything; does it?

Steve: Well, it turns the light on and off. And so if the Num Lock key on your keyboard works, that means that keyboard interrupts are being serviced.

Leo: Oh, okay, good.

Steve: That the BIOS has seen that you hit Num Lock, and it sends a message to the keyboard to turn that light on or off. So it sort of says, oh, look.

Leo: I'm not dead yet.

Steve: Yeah. It's just a flesh wound.

Leo: Yes.

Steve: So essentially back then it meant that there was still some hope. But if you hit Num Lock a few times, and the light didn't change, then probably not even the famous three-finger salute of CTRL+ALT+DELETE would bring the system back. It was time to reach over for the Reset button in order to get things restarted. But in this case Num Lock was still toggling. So the debugger showed that it executed this instruction, and nothing happened. Yet Num Lock was toggling.

And Chris had originally also noted that the little ASCII character spinners, if you may remember, when the ReadSpeed Benchmark is running, I cycle the characters at the ends of the title banner, which are vertical bars. I switch them to left-leaning slashes, then minus signs, then right-leaning slashes, then back to vertical bars. So it makes little spinners. He commented that they were still spinning. His system stopped working right in the middle, but they were still spinning.

Leo: Interesting.

Steve: Which told me again that meant that the timer interrupt was being serviced because the timer interrupts at 18.2 Hz, and so I count several of those, and then I advance the ASCII character to the next position. So things were still working. The processor was still running. But it was also apparently...

Leo: [Crosstalk] loop, apparently; right? Just stuck in a loop.

Steve: Well, yeah. It was apparently just sitting at that single instruction, doing nothing. Now, Intel chips have some built-in debugging support. And this debugger works using that. It sets a hardware breakpoint on the instruction after the one that's about to be stepped through. That way, when the processor comes out on the other side of the instruction, the debugger retains control. It updates the screen to show the current processor state, and you can see where you are. But that breakpoint was never being tripped because the AMD Phenom II processor was apparently never stepping out of that instruction to the next one.

So I stared at that for a while, thinking, what? It made no sense. It had to work. And I think I mentioned last week that Chris had observed that everything worked just dandy if he booted from a diskette. But not when he booted from a USB thumb drive. And in my subsequent experimentation before rolling up my sleeves, I learned that all was okay when I booted from any mass storage device. And in subsequent testing, I determined that it wasn't actually what booted the machine, but from where the program was run.

In other words, this instruction, this one instruction would hang if I booted from hard drive, but then ran the code from a USB thumb drive. Yet everyone else who has been testing this code all along is also typically booting and running from the code from their USB thumb drives. Yet no one else was seeing this problem. Which was really not surprising since this problem could not possibly be happening in the first place. Yet it was.

Okay. So because what I was seeing was impossible, I decided to decompose that fancy single-instruction Intel block copy into a series of individual instructions that would accomplish the same thing. You know, do my own loop, rather than use this built-in looping behavior of the Intel chip. And again, I single-stepped. And again, the system hung at one indivisibly simple instruction, when the processor attempted to load the accumulator register with the contents of the location in upper memory.

So now we're just at a move instruction. There's nothing more basic, more simple than a move instruction; right? You've got to move data from one place to another. It never completed. But also it never completed only when the code was run from USB. And DOS doesn't load anything on the fly. It's not doing anything fancy. It's old school. It loads everything first into RAM before it starts to run the program. So how could there be any memory of where the byte came from, the evil byte, the move instruction that it just wasn't going to run.

So anyway, I have no idea why running from USB could possibly matter. The only thing I can conclude is that there's some bizarre subtle bug in that old AMD Phenom II processor. The Intel x86 architecture provides us with six segmentation registers. I was using the default, which is DS, which stands for Data Segment. So what I was seeing was impossible. In a Hail Mary, I changed the code to use the FS segment register, and everything worked perfectly every time. So henceforth, none of SpinRite's code will ever set the data segment to zero and attempt to use it to access 32-bit flat memory storage. Doing that should work. And notice that it works for everybody else. And as far as we

know, everywhere except on a Gigabyte motherboard with an AMD Phenom II processor, when the code is loaded from USB. Welcome to my world.

When I published a test release for Chris to try, and also to check my own sanity, it did indeed fix his trouble, too. And someone else who had never reported in, but who had been watching, wrote to say that his similar AMD Phenom II-based Gigabyte system had also never worked before, but now it does. So anyway, I thought our listeners might get a kick out of a peek inside a bit of last week's work. Most problems that I track down and resolve teach me something that I don't know. That's what makes this journey so interesting. I can't say that I learned anything from this problem except what not to do for magical mysterious reasons, which I will never do in the name of achieving total compatibility. Once upon a time, back in '04...

Leo: Way back.

Steve: Yeah, when SpinRite 6.0 was released, one of the reasons it developed such a strong following was that it just always worked. I am now in the process of wrestling this new and soon-to-emerge SpinRite 6.1 back into that state, where it just always works. And when I'm finished, it will.

Leo: I'm surprised, given that bug, that you still prefer segmented memory architectures and x86 to the flat memory model of ARM processors. I mean, why do you prefer x86?

Steve: I wouldn't say that I prefer it.

Leo: Okay.

Steve: And in fact I'm having so much...

Leo: I mean, this bug comes from a segmented memory flaw of some kind.

Steve: Yes, yes, yes. What I appreciate is that it is often the case that you don't always need access to 32 bits of stuff. That is, you know, that's 4.3GB. Most things fit within 64K, which means you could access them with 16 bits. But if you were going to access them with 16 bits, then you need to choose which 16 bits in the system. And that's what segmentation does is it gives you an offset to the beginning of a 64K range of bytes. And it's efficient, you know, in terms of back once upon a time, when efficiency actually mattered, you know, people say, Gibson, I can't believe your Benchmark is 13K. It's like, yeah, 13,000 bytes. And a lot of that is text because I wanted to say something on the screen.

The point is we've completely lost touch with how efficient code can be. And I understand I'm the weirdo here. But these are the problems that I like to solve. But at the same time, Leo, I am so anxious to move to this new 32-bit platform for SpinRite 7. I'm just - I cannot wait to get there because it will be nice. I've been in programming Windows, as we know, since I stopped programming SpinRite. And I've really gotten spoiled by just being able to point to something, go, yeah, I want that. Even though it's a long way away, I still want it.

Leo: And we should point out Steve is one of the handful of people that addresses his own registers and pays attention to this stuff. Almost everybody else is using a higher level language that you can completely ignore what's going on at the processor level. It's just you, Steve, let's face it.

Steve: Yes. I know. I know.

Leo: But as a connoisseur of assembly language, I respect your opinion on this because you're the last man standing. Congratulations. No, I'm sure there are plenty of people, especially in the gaming industry, who are writing the most time kind of code.

Steve: Oh, and Leo, the guys who do those demos, the demos that fit within a certain size and do this amazing stuff with graphics, holy crap are - well, and hackers. Hackers are all living here because this is where they are.

Leo: Right.

Steve: Is down in buffer overruns and register contents and things. So, yeah, at the application programmer level, nobody needs to worry about this. But anything you do where it actually is necessary to know exactly what's going on, well, exactly what's going on is in the registers. That's where the action is.

Leo: This man lives in the registers. That's the truth of it. Yeah.

Steve: Okay.

Leo: And actually it's really valuable to understand that, by the way, because there are subtle bugs that happen in higher level languages that it's helpful to understand why, especially like numeric overruns and things, where you understand that's a 32-bit integer or a float. And you can't get that precision or that kind of thing.

Steve: Well, and what would you do? If you had a higher level language, and you executed an instruction that was supposed to copy something, and it just went in and never came back.

Leo: Happens all the time. I sent you a wonderful article, I don't know if you read it, about debugging the loading code in Grand Theft Auto IV, which is notoriously horrific, like...

Steve: Oh, yeah. You sent that to me for reading while I was recovering from my second vaccine shot, yeah.

Leo: Yeah. So this guy, it's famous, GTA IV takes sometimes 20 minutes before you can play the game. I mean, it's insane. And it turns out it's because they're parsing a big JSON file using sscanf(), and it's got a subtle bug in it. And it's actually not a bug, but it's something that causes - it's an inefficiency.

Steve: Right.

Leo: You probably, at the level you're coding, pay no attention to efficiency, the Big O notation or anything like that. That doesn't come up; does it?

Steve: Oh, yeah. Remember when I was doing the LRS, the Longest Repeated Strings?

Leo: Oh, that's right.

Steve: I was working on that. That was a serious...

Leo: Linear versus geometric expansion of the time, yeah.

Steve: P vs. NP sort of problem.

Leo: Right, right, right. Okay. So you do pay attention to that stuff. Usually in assembly it's probably not - doesn't come up that much. All the loops are unrolled and everything, you know. You're not recursing or anything.

Steve: Yeah. I dropped, in fact, I used our own podcast, something I learned from the podcast. I had in the original ReadSpeed code, I was hashing the boot sectors of drives with SHA - I don't remember now. Maybe SHA-1 because I really didn't need that much. But it was still a rather large algorithm.

Leo: Right.

Steve: And when I moved into SpinRite, I was upset that just a hash function was taking up so much space.

Leo: I love it.

Steve: So I switched to the FNV function, FNV-1, which we talked about on the podcast, which simply multiplies a byte by a specific prime, and it's what the hackers were using in the ransomware to create a high-speed, very small, lightweight hash function. SpinRite now has that.

Leo: Interesting. Yeah, I remember that, yeah.

Steve: And I saved myself several K worth of hash lookup tables by switching to a very economical, basically FNV. It doesn't have to be cryptographically strong. I just needed a good hash. And now I've got one that takes up no space.

Leo: Well, that was kind of the bottom line in this fix. This guy, by the way, didn't have access to the source code. He disassembled - that was the other thing. I thought about you.

Steve: Good for him.

Leo: He disassembled it and did it all by hand. He didn't have a symbol table or anything, was able to figure it out, modify the DLL. And it really came down to the fact that these folks at Rockstar had written their own JSON parser, which is horrifically inefficient. And bottom line is use libraries. Don't try to - you're nuts to write your own JSON parser. That's just nuts. Why do that? He fixed it. He got about 70% improvement in loading time.

Steve: Nice.

Leo: Yeah. We'll see if Rockstar pays any attention. Anyway, that's why I sent it to you. I thought while you're curled up on the couch you'd enjoy this tale of disassembly.

Steve: Perfect story.

Leo: All right, Steve. On we go.

Steve: One little bit of news. Paul, @whatsupdoc114, shot me a note that Exchange Servers all over the world were currently under attack. I did a little quick checking, and sure enough, today on March 2nd, Microsoft just released seven remote code execution vulnerability patches for Exchange Server, the most three recent, I think 2019, 2016, and 2013 or 12. Anyway, I did a little digging, and I found a note. The risk is still extremely high.

The exploit allows an attacker to perform a Pre-Auth RCE and essentially end up with the ability to run commands with system privileges since most customers don't use split permissions or have not performed the steps required to remove excessive permissions from Exchange Servers and Active Directory, it's likely that the attacker may be able to gain highly privileged rights in your on-premises domain. So it must be, I don't know where the - oh, yeah, you found it. Emergency patches for four - and actually there are seven that I saw.

Leo: Yeah, so this is out of date. This is Dan Gooden writing at Ars Technica. He's always very good. He pushed this about an hour ago. It was four, and it's now up to seven zero-days in Exchange Servers.

Steve: Wow.

Leo: That's not good. And by the way, Microsoft says hackers have been using it on behalf of the Chinese government. So that's not good. That's not good. Well, thanks for the update. That's why people listen. We don't normally do breaking zero-days.

Steve: No.

Leo: But when they happen, we've got to; right?

Steve: Yup. And so here is some news that is just - it's beyond distressing. Okay. So Criteo, I guess that's how you pronounce it, C-R-I-T-E-O, is a leading tracking company. They send website administrators with whom they already have a tracking and analytics relationship an email. It asks them to make a quick change which will "only take two minutes," and it will "adapt their website to the evolution of browsers." Which is to say that it will work around their own website's visitors' attempts to block tracking and to reenable tracking to their site's visitors in a "more optimal way."

Then in this email, after presenting instructions for the site's webmaster about how to make the required change, which will indeed only require a couple of minutes, in the particular instance of email that I saw, they conclude with, if this is not done, you may lose 11.64% of your sales, 11.53% of your gross turnover, and 20.82% of your audience. And this brings us...

Leo: Geez.

Steve: ...to some recently published research, which explores just how prevalent and pervasive this new technique has grown over the past few years. The group of five researchers will be presenting their work at the 21st Privacy Enhancing Technologies Symposium (PETS) 2021 this July. But we have the research now. I'm going to quickly read the abstract of their paper and then explain in detail because this is really important and horrifying what this means.

Their abstract says: "Online tracking is a whack-a-mole game between trackers who build and monetize behavioral user profiles through intrusive data collection, and anti-tracking mechanisms, deployed as a browser extension, built into the browser, or as a DNS resolver. As a response to pervasive and opaque online tracking, more and more users adopt anti-tracking tools to preserve their privacy. Consequently, as the information that trackers can gather on users is being curbed, some trackers are looking for ways to evade these tracking countermeasures.

"In this paper we report on a large-scale longitudinal evaluation of an anti-tracking evasion scheme that leverages CNAME records to include tracker resources in a same-site context, effectively bypassing anti-tracking measures that use fixed hostname-based block lists. Using historical HTTP Archive data we find that this tracking scheme is rapidly gaining traction, especially among high-traffic websites.

"Furthermore, we report on several privacy and security issues inherent to the technical setup of CNAME-based tracking that we detected through a combination of automated and manual analysis. We find that some trackers are using the technique against the Safari browser, which is known to include strict anti-tracking configurations. Our findings show that websites using CNAME trackers must take extra precautions to avoid leaking sensitive information to third parties."

Okay. So here's the story. So first of all, what are CNAME records? They're not something we've had much occasion to talk about in the past. I have a feeling that's going to change. But on the other hand, DNS is something we're pretty much always talking about. A CNAME record is simply another type of DNS record. As we know, a DNS "A" (standing for Address) record resolves a specific domain name to a dotted quad IPv4 address. Similarly, a DNS "AAAA" record resolves a specific domain name into an IPv6 address.

An SMTP email server might query a domain, like for example GRC, for any MX records which will point to one or more IP addresses, which are used for email servers for that domain. And for various reasons, a domain's text records might be queried for information, like to provide the public key used to check a domain's anti-spam signatures. So although DNS's primary purpose is to look up and return IP addresses, it's also a nifty general purpose distributed Internet directory, capable of containing and returning all sorts of other information. And another of those types of queries is the CNAME.

CNAME (C-N-A-M-E) stands for Canonical Name. Whereas an "A" query returns an IPv4 address, a CNAME query returns another domain name. The domain name being queried is considered to be an alias, and what's returned is the canonical name for that alias. And CNAME records are handled specially by DNS. As Wikipedia explains, CNAME records are handled specially in the domain name system and have several restrictions on their use. When a DNS resolver encounters a CNAME record while looking for a regular resource record, it will restart the query using the canonical name instead of the original name. The canonical name that a CNAME record points to can be anywhere in the DNS, whether local or on a remote server in a different DNS zone. So you can think of it as a pointer. It is a pointer to a different DNS name.

So here's what's evil, and what that email above was asking website admins to do, and which many - these guys counted more than 10,000, I'll get to that in a minute - websites have done. They were asked to say, okay, say at example.com, to place a CNAME record into their site's DNS such that some arbitrary but specified subdomain of example.com, like say dyzxrdb.example.com, would be an alias for the canonical name web-trackers-R-us.com. So what that does exactly is anytime someone wants to look up the IP address for dyzxrdb.example.com, their assigned DNS resolver, which is performing the DNS resolution for them, will query the example.com domain's name servers for that subdomain. But because that subdomain record is a CNAME record, that's what will be returned, not an IPv4 IP address, but what will be returned to the querying DNS resolver is a CNAME result with that web-trackers-R-us.com as its answer.

The resolving DNS server that understands that is the canonical name for which dyzxrdb.example.com was an alias will then ask web-trackers-R-us.com silently, like on its own, because this is all built into DNS, for its IP. Whereupon the user's DNS resolver will return that IP as the IP of dyzxrdb.example.com to the person, the browser, the web browser that asked for it. From the user's perspective, they asked for the IP of a subdomain of example.com, and they received an IP. But due to prior collusion between the website they're visiting and web-trackers-R-us.com, the IP they received was for web-trackers-R-us.com. From the standpoint of the user's browser, this is an in-domain, same-domain query. So third-party cookie restrictions do not apply. And the user's web browser will treat this query as a subdomain of the website being visited.

Okay. So what we have so far is a horrifically sneaky means of deliberately overriding a user's wishes for anti-tracking by websites that feel that they have a superior right to track and obtain leverage from their visitors. But believe it or not, it's much worse. Cookies set on specific domains are accessible to, and sent to, anyone who queries their subdomains. This means that by colluding in this way to allow an untrustworthy third-party tracking entity to pretend to be within a website's domain...

Leo: Ooh.

Steve: Yes, Leo.

Leo: That's sneaky as hell.

Steve: Well, the cookies being held...

Leo: So it's not a third party, doesn't appear to be a third-party cookie, even though it is.

Steve: Correct. It doesn't appear to be a third-party query, a third-party request. It actually is. But this means the cookies being held by the browser of visitors to that site will be sent to that third-party entity because the browser won't know any better. The website's visitors' logon session authentication cookies will be sent outside of that domain...

Leo: Oh, that's not good.

Steve: No, to untrusted and, I would argue, untrustworthy third-party tracking and analytics companies.

Leo: Wow, that's really not good. You're sending your Facebook login cookie to them, in effect.

Steve: Exactly.

Leo: They can post as you. Geez.

Steve: Yes. The fact that it's done over HTTPS provides no security. Anyone at any of those tracking, advertising, analytics firms of which 13 have now been identified by the researchers could trivially impersonate any user of any website who didn't explicitly log off, and who therefore still have a valid authentication cookie. It is an unbelievable breach of trust and abuse of web technology.

I ran across a wonderful website that allows us to play with and explore our own browsers' cookie and subdomain handling to understand exactly this issue. And I made it this week's Security Now! podcast shortcut of the week, so it's https://grc.sc/808 because this is podcast number 808. That will bounce you over to scripts.cmbuckley.co.uk/cookies.php.

And I played with it this morning because I was wondering whether my Firefox 86 with its new full "Total Cookie Protection" that I opened with fully enabled would help. No. It is not blocking any of this leakage because these are not third-party cookies. These are subdomain cookies. What this cool site, again, grc.sc/808, it allows you to set cookies in

different domains and subdomains of that site and see which ones are returned. It's a very cool little page. So I'm sure that our listeners will get a kick out of it.

Okay. So how widespread is this behavior? Thankfully, these researchers have gone to some effort to unearth the extent of this currently spreading industry-wide website collusion with the tracking industry. It's not difficult. You just, for example, resolve any subdomains of the primary domain that you receive from a website. You do that DNS lookup for yourself as if you were a recursive DNS resolver, and you see whether you receive a CNAME record that points to any one of the 13 current providers of this form of CNAME tracking. And you've got something up on the screen, I see, Leo, from that page.

Leo: So this site, basically what I've done is I've told cmbuckley.co.uk to set my cookie. And it returns my cookie, even though it says it looks like a first-party cookie, but it's really my third-party cookie. Is that what's happening here?

Steve: Well, so and then, if you click down below in that text in the lines below, see that they have an a.something.

Leo: Yeah, a script. Yeah, oops.

Steve: So that's a subdomain of the root domain, and you can see it receive your cookie.

Leo: Right, same thing, there's a cookie, yeah. And then let's zoom in on this.

Steve: Yes. And so that's the point is that because it's a subdomain, you set the cookie on the root domain, subdomains all get the same cookies.

Leo: I see.

Steve: So with the collusion enabled by the CNAME, some random gibberish dot, like .amazon.com, it would get Amazon.com's cookies. All the amazon.com cookies you have would go to the third party. It's that bad, Leo. It's unbelievable that this has been happening. Who would know?

Leo: And of course they're saying, oh, well, we'll never do that. We just want to know who's visiting your site. This is not - this is for you. We would never steal your authentication cookies.

Steve: Right. This is for your benefit. We're making more relevant ads. And it keeps the Internet free, and it wouldn't be free otherwise.

Leo: This is one way people are getting around ad blockers and trackers is looking - if everything looks first party, they don't block it. None of these block if it's first party; right?

Steve: Right.

Leo: So if you can make a third-party ad or a third-party cookie look...

Steve: Be a subdomain, yup, yup. Okay. So the researchers found this technique currently in use on a total of 10,474 websites. And of the top 10,000 websites overall, 9.98%, one in 10 of the top 10,000 are currently employing this form of CNAME tracking, cloaking, subdomain collusion.

Leo: Are they sending the cookies to themselves or to this third party?

Steve: That's the problem. There's no distinction, Leo.

Leo: We don't know. We don't know.

Steve: No, no, no. I mean, we do know. There's no distinction. Our browsers, what that page you were playing with demonstrates is our browsers send cookies set on the root domain to their subdomains.

Leo: Right.

Steve: That's the way browsers are designed. They're supposed to do that. Once upon a time you could start the root domain with a dot. It'd be like .amazon.com. And that would say don't share these root cookies with my children, with our subdomains. That behavior went away years ago. It's now ignored by browsers, even if they still see it.

Okay. So their research furthermore observed what they termed "targeted treatment of Apple's Safari web browser" where the advertising technology company I mentioned before, Criteo, who mailed the letter I opened with, switched specifically to CNAME cloaking to bypass Safari's otherwise strong privacy protections.

And it's worse. Data leaks. Significant cookie data leaks were found on 95% of the sites that use that CNAME tracking, so 95% of the 10,474 sites, all of which sent cookies containing private information such as full names, locations, email addresses, and even session authentication cookies to trackers of other domains without the user having any knowledge or control. Again, remember that the entire presumption of cookies is that bad and abuse-prone as they may be, at least they stay within the domain that set them. At least their content, whatever it might be, even if it's a user's actual name and real world identity, bad practice as that would be, at least it remains between those two parties.

So while cookies can be used for tracking, the only data that's ever returned to a domain is something that that domain had previously sent. Thus by definition it's not secret to that domain. But now, thanks to the horrendous abuse of CNAMEs being used to deliberately confuse cookie domains, data is being sent with queries by the user's browser to entities who never set that data in the first place. That tracker never set the authentication cookie, which is how the user is staying logged in. They never put the user's name and location and email address in the cookie. The domain did, figuring,

okay, yeah, it's bad practice, but what the hell. It's going to stay here. It's only between the users' browser and us. And, oh, look, we're HTTPS now, so no one can spy on it.

CNAMEs break that so that all the information which was private between the users' browser and the domain is now being sent to any subdomains, and this subdomain is now a tracker, an analytics company. As the researchers noted, that data which should never be exposed to any third party often contains information that tracking firms would die to have and leverage. And now they don't have to. They just need to get websites to collude with them by adding a CNAME record to that domain's DNS.

There is a bit of good news. The only good news here is that good old Gorhill's uBlock Origin add-on is at least partially effective at spotting and blocking accesses to these despicable subdomains. I have in the show notes a table from the research, Table 1, titled "Overview of the analyzed CNAME-based trackers," based on the HTTP Archive dataset from October 2020.

The number one tracker is a company called Pardot, P-A-R-D-O-T. Unfortunately, it's owned by Salesforce. That had the highest number, just shy of 6,000 detected websites. 5,993 detected websites were using Pardot, a Salesforce company which, I mean, we all know Salesforce, they say "powerful business-to-business marketing automation," stating that "Pardot offers powerful marketing automation to help marketing and sales teams find and nurture the best leads, close more deals, and maximize ROI." And they are the number one user and abuser of this technology.

Number two position is Adobe Experience Cloud that is also doing this. And then there's a list of all 13 of these companies. But in a note on this table, the researchers did observe that Pardot is being blocked because the third-party script being sourced from Pardot.com is being blocked, that is, by uBlock Origin; and that, if that script was not blocked, then CNAME abuse would succeed. The researchers had the following to say about countermeasures against CNAME.

They said: "In response to a report that a tracker was using CNAMEs to circumvent privacy block lists, uBlock Origin released an update for its Firefox version that thwarts CNAME cloaking. The extension blocks requests to CNAME trackers by resolving the domain names using the browser's own browser.dns.resolve API to obtain the last CNAME record, if any, before each request is sent." And by that they mean CNAME records can actually chain; right? Because it's a pointer that causes the DNS to go to another domain, that one could have a CNAME that could point it somewhere else. So it wants to follow the chain to the last CNAME record. Then it checks whether the domain name matches any of the rules that it is blocking. So it's got a robust CNAME chain following a technology in uBlock Origin.

He says: "Subsequently, the extension checks whether the domain name matches any of its rules in its block lists, and blocks requests with matching domains while adding the outcome to a local cache." And then they said: "Although uBlock Origin also has a version for Chromium-based browsers, the same defense cannot be applied because Chromium-based browser extensions do not have access to an API to perform DNS queries. As such, at the time of this writing, it is technically impossible for these extensions to block requests to trackers that leverage CNAME records to avoid detection.

"uBlock Origin for Chrome, which does not have an explicit defense for CNAME-based tracking, still manages to block several trackers. This is because the requests to the trackers matched an entry in the block list with a URL pattern that did not consider the hostname. Unfortunately, it is fairly straightforward for the tracker to circumvent such a fixed rule-based measure by randomizing the path of the tracking script and analytics endpoint, as is evidenced by the various trackers that could only be blocked by the uBlock Origin version on Firefox."

And you see that in this table. They have a column for uBlock Origin where Firefox was successful, and uBlock origin where Chrome was successful. uBlock Origin on Firefox, because it actually is doing CNAME block lists, is able to be more effective.

Leo: So it's interesting because I use Firefox and NextDNS. So it looks like I have pretty full coverage if I do both of those.

Steve: Yup. NextDNS is also a good CNAME blocker, yes.

Leo: They're kind of like a piehole in the sky.

Steve: Right.

Leo: Yeah, I love that.

Steve: So the researchers wrap up their research with the following conclusion. They said: "Our research sheds light on the emerging ecosystem of CNAME-based tracking, a tracking scheme that takes advantage of a DNS-based cloaking technique to evade tracking countermeasures. Using HTTP Archive data and a novel method, we performed a longitudinal analysis of the CNAME-based tracking ecosystem using crawl data of 5.6 million web pages. Our findings show that unlike other trackers with similar scale, CNAME-based trackers are becoming increasingly popular, and are mostly used to supplement 'typical' third-party tracking services.

"We evaluated the privacy and security threats that are caused by including CNAME trackers in a same-site context. Through manual analysis we found that sensitive information such as email addresses and authentication cookies leak to CNAME trackers on sites where users can create accounts." In fact, I didn't mention this, but in their report they took a handful of these sites where you can create an account. They created accounts. They tracked and verified the leakage. They used the leaked authentication tokens to impersonate themselves, and it all worked. So this is not a theoretical problem. And this is, remember, 10% of the top 10,000 websites are doing this now. It's just...

Leo: It's so amazing.

Steve: It's just horrible.

Leo: Yeah, horrible.

Steve: So what we have is a real mess. No form of explicit tracking was ever designed into our use of the web. It happened as an unintended consequence of single advertising services having appearances on multiple hosting websites. And those providers were allowed to set cookies in our browsers just like their originally intended first-party cousins. I would argue we should have stopped it then. We should have just said no. But the trouble was this tracking was effectively invisible. Users didn't see it. It went completely unseen by the public. And it wasn't the public's responsibility to stop it. I would argue it was technologists' job to say no because it was the technologists who

were abusing this technology. But of course those wearing white hats didn't say anything. No one said no.

Then, when an awareness began to emerge, and third-party cookies were being threatened and sometimes disabled or deleted, browser fingerprinting emerged as a means for allowing what had grown into a tracking industry to retain its grip on our browsers and on us. Since fingerprinting was more difficult to defend than cookies, it received a stronger pushback from browser vendors who didn't like the idea that cookies were being bypassed as a means of tracking their users and that this kind of slimy fingerprinting was going on behind the scenes.

And now we have what is perhaps the ultimate abuse in tracking technology. Thanks to explicit collusion among a growing number of websites, third parties, those same tracking third parties and analytics firms, are being allowed to receive a website's cookies, apparently without the website knowing or caring. Our logged-in session authentication cookies are being received by third-party tracking entities with whom users have no relationship; and with whom they would surely refuse to share their logon session and various other possibly personal details if they were made aware of what the technology they are using was doing behind their backs.

But once again, end users have no idea. They just use this, and they assume that people who do know are going to do something, that they're being protected by people who are going to be responsible. This has been going on for years and has been growing slowly, and it needs to be fixed. As with the original abuse of third party cookies, where all this began, it cannot be the responsibility of those who do not understand this to say no. It's got to be those of us who do understand this to push back in every way possible. So I am so glad that this research has shined a bright light on this next generation of tracking. It needs to be shut down immediately. But as I said, it takes those who are technologically savvy in the web industry to make it happen. The sooner the better.

Leo: Yeah. It's amazing. But I think that this is the cat-and-mouse battle between tracking companies and advertising companies and ad tech companies and users who just reasonably say "Don't track me, bro." And they're going to pull out all the stops, including posing as a first-party site.

Steve: Yeah. It's going to take legislation. That's the only thing that can happen is that we just have to say, sorry, you just can't track people. I mean, imagine right now...

Leo: Well, awareness is good because the market can respond. The market can, I mean, I want to know what these top 10 sites are so we can say something.

Steve: Yeah, 10%.

Leo: Yeah.

Steve: Yeah. So 1,000 of the top 10,000 sites are doing this.

Leo: So if those are well-known sites, and I think they probably are, those names need to be revealed so that we can say something. Because that's how you, you

know, that's not okay. They think they're getting away with it because nobody will ever notice.

Steve: Right. Exactly.

Leo: Well, now we know, yeah. Wow. I'm glad I use NextDNS. That's all I can say. And uBlock Origin.

Steve: That would have been a good name for this podcast, Leo. "Well, now we know."

Leo: So now you know the rest of the story.

Steve: Now we know.

Leo: That's pretty much every podcast with Steve Gibson. It's a great informative thing, and I'm glad you're here to hear it. And tell your friends; you know? If they're smart enough to understand what we just said, anyway. Some people maybe this would go [sound effect], but - most people. That's the problem is this stuff is [crosstalk].

Steve: Yeah, we should mention NextDNS, I mean, because if your DNS, if the people you are sending those queries to are on the ball, then they're the ones that will be asking example.com for the value of this CNAME. They will see that it is a domain that is a tracking domain, and they'll just say, oops, sorry.

Leo: And it's my guess that if Quad9 and Cloudflare and the others don't already protect against this, they could, and I imagine they'd be implementing that pretty quickly. I'm just pleased that this NextDNS.io, it's free for the first 300,000 queries. But what happens is you get to that number pretty quickly because you put it on everything. I have it - it's running on my network.

Steve: And god, in this day and age, 300,000 queries is like lunch.

Leo: Goes like that. So I buy it, but it's not, it's like two bucks a month or something. It's fairly inexpensive. And boy, is it well worth it. It really - it does a whole lot more than that. Lot of security stuff. NextDNS.io.

Steve Gibson is at GRC.com, and that's where you'll find so many great things like this show: 16Kb audio versions for the bandwidth impaired; full human-written transcripts that are great to read along while you listen. You can also use them for searching. And of course 64Kb audio, all at GRC.com. While you're there, pick up SpinRite. Release 6 is current. But as you can see, Steve's working pretty darn hard to get 6.1 out the door. So buy v6 now, you'll get 6.1 free when it's available. You'll also get to participate in the early testing and so forth. So it's a good thing. Everybody needs SpinRite. ShieldsUP! there, lots of other stuff for free. He's a very generous soul. I have copies of the show, as well, or we do. There is no "I" in TWiT. Actually, there is.

Steve: But it's a lowercase "i."

Leo: It's a lowercase "i." It's just a humble "i," a little "i." It's at TWiT.tv/sn. That's where you'll find audio and video, too, actually. There's also a YouTube channel you can watch dedicated to Security Now!. What else? You can subscribe in your favorite podcast application. Simple enough. And that way you'll get it the minute it's available of a Tuesday afternoon.

We do the show around 1:30 Pacific, 4:30 Eastern, 21:30 UTC. You can watch us stream it live, if you like to watch the behind-the-scenes chitchat at TWiT.tv/live. Watching live, chat live at irc.twit.tv. After the fact, our forums are at www.twit.community, no third-party tracking cookies involved, as far as I know. I guess it's possible that - I'll check the CNAME real quick, just to make sure. No, of course not.

And same thing with our Mastodon instance. Everybody's saying, "Where's Steve?" I said, "It took me years to get Steve to use Twitter. Just relax, okay? We'll move him over to Mastodon in time." It's the federated open source version of Twitter. Our Mastodon server that's a federated server is at TWiT.social and gives you access to all the other Mastodon servers, as well. TWiT.social. I'll see you in there.

Thank you, Steve. Have a wonderful week. I'm glad you got your second Fauci Ouchy. We wouldn't want to lose you.

Steve: No, ready to go. Bring it on.

Leo: Good. We'll see you next week on Security Now!.

Steve: Bye.