SECURITY NOW!

Transcript of Episode #760

Folding Proteins

Description: This week we examine some consequences of increased telecommuting with the use of RDP and VPNs skyrocketing, along with a new bug in iOS's handling of VPN connections. We look at Google's unrelenting quest to get the "www" out, and note some changes to Firefox and further revisions of browser release schedules. We take a deep dive into a very welcome forthcoming code security feature for Windows 10. We share an action item for users of OpenWRT routers, and the result of an audit of Cloudflare's privacy-enforcing DNS service. We divulge a few interesting bits of feedback and some SQRL and SpinRite miscellany, then finish by examining a new opportunity to donate our unused CPU cycles for help with COVID-19 research.

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-760.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-760-lq.mp3

SHOW TEASE: It's time for Security Now!. Steve Gibson is here. We've got RDP. We've got VPNs. We'll talk a little bit about something Intel's calling "Shadow Stack" and why you will want this on every x86 architecture. And, finally, Folding@home, how you can help conquer COVID with your spare cycles. It's all coming up next on Security Now!.

Leo Laporte: This is Security Now! with Steve Gibson, Episode 760, recorded Tuesday, March 31st, 2020: Folding Proteins.

It's time for Security Now!, the show where we cover security, privacy, and how things work.

Steve Gibson: Eventually.

Leo: Eventually, in obsessive detail. We're only half an hour late, Steve. This is a new world record.

Steve: Well, yes.

Leo: Welcome, Steve Gibson, host of our show.

Steve: Great to be here, Leo, with you for Episode 760 for the last day, this is a Tuesday, the last day of March, which means that April's Patch Tuesday, as I've noted before, will be as late in the month as it's possible to have it, as happens from time to time. We've got a bunch to talk about, bunch of fun stuff, sort of a potpourri.

We're going to look at some consequences of increased telecommuting with the expanded use of RDP and VPNs skyrocketing. We've got a new bug, I heard you mention it in MacBreak, in iOS's handling of VPN connections, which is not a huge problem, as you said then. I'm sure Apple will fix it. But it's kind of interesting. We also look at Google's unrelenting quest to get the www out of URLs, against also unrelenting pressure against doing that, apparently. We have some changes to Firefox, and further revisions of browser release schedules as our major browser producers work to figure out this new world that we're living in.

We're going to take a deep dive into a very welcome forthcoming code security feature for Windows 10. We're going to share an action item for users who have put the OpenWRT firmware onto their routers. Also the result of an independent audit of Cloudflare's privacy-enforcing DNS service. Then I want to update our listeners on a few interesting bits of feedback regarding SQRL and SpinRite. And then we're going to finish by examining a new opportunity, since SETI@home has shut down, to donate our unused CPU cycles for help with COVID-19 research, thus the title of this podcast, "Folding Proteins."

Leo: Ooh. I'm curious about this because I'm a little skeptical. You know, we did SETI@home for 20 years. And I'm a little skeptical.

Steve: Well, maybe it's a little different to be searching for space aliens from the background cosmic radiation, different from proven technology to understand the molecular modeling of amino acids.

Leo: And Folding@home has some success stories to tell.

Steve: Yes.

Leo: That's the key; right? They've actually used these, what is it, multi teraflop, multi petaflop...

Steve: We're now at exaflop, believe it or not.

Leo: Exaflops, wow.

Steve: The current Folding@home base of computers has a power equal to the sum of the world's seven largest supercomputers combined. It is a stunning resource. And in fact that's part of the problem is it's got too much power. They're like apologizing for the fact that people's computers aren't busy all the time. It's like, we're working as hard as we can to keep them busy. So anyway, we'll talk about that at the end of the podcast.

Leo: Nice.

Steve: So we have our typical geeky Picture of the Week, which due to the situation the world is in right now takes another look at coronavirus. We've got someone talking to a doctor, or somebody who has a mask on and a Red Cross hat. So this first person says, "Well, coronavirus is just simple DDoS," which first of all I thought was kind of clever, a distributed denial of service, as indeed we're all living through right now. The doctor replies, "Yes, so close your ports and stay 127.0.0.1." So another fun take on the techie side of this.

And speaking of VPNs, that was the first bit of news that I wanted to share today. Not surprisingly, although in the case of RDP somewhat worryingly, both RDP and VPN use are skyrocketing since the beginning of the year, you know, to no one's surprise. The use of the cloud and all remote access technologies has jumped up significantly since this whole stay-at-home went into effect in many parts of the U.S. and also increasingly globally. We have some numbers for that.

Microsoft's perpetually security-challenged Remote Desktop Protocol usage, based on the presence of available ports - that is, 3389, publicly present on the Internet - has jumped 41%. And popular VPN port appearances have gone up by a third, 33%. And this is according to data compiled by Shodan, the online public scanning service that we're now talking about all the time these days. The number of publicly visible RDP endpoints - which is to say servers answering TCP connections at port, in this case, 3389 - has gone from roughly three million at the start of the year to nearly 4.4 million the day before yesterday, just this most recent Sunday, March 29th. And this only reflects RDP servers listening on the default RDP port 3389.

John Matherly, who's Shodan's CEO and founder, noted in an interview with ZDNet that a similar surge has also been seen - I got a kick out of this - on port 3388, which of course is 3389 minus one, which he says is regularly used by system admins to, quote, "hide," unquote, the RDP service from attackers. Okay. Now, of course all the attackers know that now. So for that not-very-well-hidden port, the number has also jumped, in this case by 36.8% from roughly 60,000 3388 instances of RDP listening on that port to 80,000, from 60,000 to 80,000 currently.

And I know I'm a broken record about this, but seeing a jump in exposed RDP ports is horrifying since all of our experience informs us that Microsoft has never managed to make RDP safe to expose publicly. It should always be safely tucked behind a strong VPN, yet another reason to use a VPN. And then the VPN should itself use some form of multifactor authentication, as all good strong enterprise VPNs do. You don't want it to just be username and password unless you've got some very strong protection against someone just doing, as it's now called, a credential stuffing attack where usernames and passwords are guessed until they get in.

And speaking of VPNs, Shodan has seen the number of servers running VPN protocols, the traditional enterprise protocols like IKE and PPTP, jump up by a third from about 7.5 million systems to nearly 10 million now. IKE and PPTP are typically what enterprises use as gateways into their Intranet and their internal corporate networks.

And apropos of the sponsor of this podcast, the use of consumer-grade VPNs has also seen a sudden surge in usage. And of course, exactly as Leo was suggesting, in this case it's most likely for use in bypassing so-called "geofencing" for online content while people are stuck at home. There's been a 165% rise in users since just before the middle of March in several cases. There's a site called Top10VPN that sort of is a third-party VPN service rater.

And they wrote of Netflix-compatible VPNs, they said: "To bypass Netflix's location restrictions and unblock all the 'hidden' TV series and movies," they said, "you need to know what's the best VPN for Netflix." And they said: "Unfortunately, not all VPNs work with Netflix. But don't worry, we're here to help. Our team of experts regularly tests 72 VPNs to see if they unblock Netflix libraries in the U.S., the U.K., and many other countries."

And, you know, they go on to talk about this. And in their monitoring of VPN usage, they've seen a recent 65% overall increase in consumer use of VPNs. So I think in general, because VPNs in general do a better job of security, I would always recommend to our users that they use multifactor authentication if possible. On the other hand, a consumer use of a VPN is going to be on the client side. So that's a different concern than this jump in listening RDP servers. There we're talking about a big jump in exposed RDP usage.

And, you know, I shudder to think where all of these RDP servers came from. You know, it doesn't feel like, I mean, maybe enterprise has responded that quickly. I worry, though, that people have put their home machines or maybe their work machines now have some sort of an RDP presence on the Internet because people want to be able to get to them remotely. And so credential stuffing attacks have been shown to be surprisingly effective. So it would be so much better to hide everything behind a VPN and then need to log into the VPN in order to get access to an RDP server behind it. I have a feeling we'll be doing some coverage of updated RDP attacks here before long.

And speaking of VPN handling, we got an interesting feature update to iOS. And Leo, I meant to mention to you, Paul Thurrott's rave about iOS with a mouse, iOS devices with a mouse last week, I caught the tail end of it on Windows Weekly, and I just was amazed by that. And in fact his raving about it induced me to hook up a Bluetooth mouse to an iPad, and I have to say he's not wrong. It's an interesting usability solution. So I thought that was just very cool.

But anyway, one of the things that happened, actually it was first occurring in v13.3.1, and it's extended into 13.4, is it has come to the community's attention, actually it was ProtonVPN who were just sort of doing some packet-sniffing work and noticed something weird going on. A ProtonVPN researcher was using Wireshark to monitor the packet traffic from an iOS device when he noticed that, even after the VPN was brought up and its tunnel was active, non-tunneled traffic from before the VPN was enabled continued to be exchanged with the iOS device. iOS was not closing its existing connections and then reconnecting them once the VPN was in place as one would have expected it to. So as a consequence, any connections that were initiated after the VPN was active would be securely routed through the VPN. But any preexisting connections would not be protected unless the individual services close or reset the connections themselves.

From a practical standpoint, we know that nearly everything these days is HTTPS, so it's not like any important data would be in the clear. That would be unlikely. But of course that's often not the point since users have an expectation that their IP address would be protected while using a VPN. And in this case it would not be. On the other hand, their IP wasn't being protected before they turned the VPN on. So, you know, it's not a huge big deal.

The ProtonVPN folks found a simple sort of brute force workaround, although it's a bit annoying because you have to do it by hand. After you bring the VPN up so that it's online, then briefly switch the device to airplane mode or flight mode, which of course turns off the WiFi completely. Then restore the device so that WiFi is enabled. Turn off airplane or flight mode. WiFi will come up. All of the connections will reestablish themselves and be tunneled through the VPN. Apple's suggestion about this is to configure "always on" VPN via their Mobile Device Management, MDM. But it takes some work with putting the device into supervise mode, then using the Apple configurator.

I've talked about this before, and I think I must have taken our users step by step through it because one of my iPads is that way. It's got persistent VPN. And I forgot, I mean, it's been years now. I took a little screenshot actually because I thought, oh, look at that, there it is. That's what I thought. It's got, under the General Settings menu, airplane mode, then WiFi, then Bluetooth, then cellular. Then it's got an additional line that my other devices don't have, VPN. And it's on, and I can't turn it off. It just insists on being on. I'm not even sure now what server it's VPNing to. But that's the configuration on one of my iPads. So it's definitely possible to do that.

On the other hand, Apple's going to fix this. I'm sure it's already been fixed in-house. They're probably moments away from pushing out a fix to this because it's gotten a lot of attention in the industry, and Apple doesn't force themselves to stick to any schedule. So I imagine within a few days this'll be fixed. But for what it's worth, in the meantime, just bringing up the VPN, switching off WiFi, switching it back on, pushes everything back through the VPN tunnel, and you'll be good.

Wow, talk about sort of a ridiculous and fraught issue. Google has continued to battle the industry and its own users over this display of both "www" and "https" in their Chrome browser URLs. It insists on characterizing www and also m-dot, remember "m" as in the mobile version of a domain, as trivial. And they just don't want to show it to people. It was two years ago, back in 2018, that Google first announced its intention to have then Chrome 69 "elide," as they term it, the www from Chrome's displayed URL. The uproar of opposition caused them to rethink that decision and end up backing away from it.

But someone somewhere inside Google is very stubborn about this, and they have not let it go. Last summer, with the release of Chrome 76, we talked about it at the time, the www was finally disappeared from Chrome's URL display. And many vocal people have been unhappy about it ever since.

At the time, Google wrote, they said: "The Chrome team values the simplicity, usability, and security of UI surfaces. To make URLs easier to read and understand, and to remove distractions from the registrable domain, we will hide URL components that are irrelevant to most Chrome users." So of course, conversely, typical complaints are things like: "It causes confusion in that what the user sees as the URL in the omnibox is not reflected in the actual value when copied, it does not match the SSL certificate, and there are many sites that do not automatically map the naked domain to www." In other words, the www is often not superfluous.

Which brings us to today. There must have been a large enough outcry over the loss of the "www" and the "https" to again cause Google, after what, 10 months or 11 months, to rethink it. No, maybe eight months. So Chromium developers are now testing a new omnibox context menu that will give users the option to "Always Show Full URLs." So I guess it seems like it's a bit begrudging. Google continues to believe that showing what it calls a "trivial subdomain" will distract users when making security assessments. I mean, by this time everyone's used to www. Sometimes it's there; sometimes it's not.

So this feature is currently in the Chrome 83 Canary build. We're currently running 80 publicly; 81 is due soon, within a few weeks. So it's in the 83 Canary build, appearing in a new context menu that drops down from the omnibox, you know, the thing that contains the URL. So after this setting has been set, it will be retained until changed to always display the full web address, including "https" and "www" or "m-dot." They outlined their plan for users to opt out of the URL snippage in a post on the bug tracker titled "Implement omnibox context menu option to always show full URLs."

The post's author, Livvie Lin, wrote: "The omnibox context menu should provide an option that will prevent URL elisions for the entire Chrome profile. However, showing the full URL may detract from the parts of the URL that are most important to making security decisions on a web page." So again, they're not letting go of it, but they're acquiescing apparently to the constant annoyance that some old-school people have about actually seeing the URL that you are visiting.

So anyway, we will have, probably when Chrome 83 is out in the main release, we'll be able to right-click on the URL bar, select I want to see the truth, the whole URL. We'll get it in Chrome, and we will keep it. Oh, and I did see a note that it is for desktop versions of Chrome only. I think it may not apply to our mobile smartphone versions. So that's to be determined.

And then another interesting piece of good news. Firefox 76 will finally stop assuming HTTP when nothing is specified. Our listeners will know that this is something I've been commenting about being wrong about our browsers for some time. With the majority of web sites now HTTPS, and there being a definite bias in favor of moving everything there, over to HTTPS, as I've noted several times, it has seemed wrong to me that if someone were only to enter Amazon.com into their web browser, their browser would assume the protocol prefix of http://. So then first jump to http://amazon.com, whereupon Amazon's server would almost always, as mine does, GRC does, redirect you over to https://amazon.com.

Anyway, Firefox is going to make the change. On their Bugzilla page, which is both where bugs go to die and new features are born, sort of being bugs because they don't yet exist, there's an entry for Bug 1613063 which is flagged as "experimental," and it's titled "HTTPS Only Mode." The description reads: "Currently, if a Firefox user types foo.com in the address bar, then our internal machinery establishes an HTTP connection to foo.com. Within this project we will expose a preference which allows end users to opt into an 'HTTPS Only' mode which tries to establish an HTTPS connection rather than an HTTP connection to foo.com. Further," they said, "we will upgrade all subresources on the page to load using HTTPS instead of HTTP."

So then finally they concluded, under "Implementation Considerations," they said: "For top-level loads which encounter a time-out, we could provide some kind of error page with a button which would allow the end user to load the requested page using HTTP." Meaning that, if their default promotion from HTTPS to HTTP fails, then they would result in a page saying, okay, fine, try using HTTP. So the user would press that.

Then they said: " For subresource loads we would fail silently and just log some info to the console." So this setting will initially be off, but then power users could flip it on if they wished to in the UI. And I'm sure that their instrumentation will monitor the experiences of those who choose to turn it on. And then at some point in the future, assuming that all goes well, they would flip it to "on" and then provide a useful UI fallback. So anyway, nice to see that we're moving forward. At this point I think that really makes sense, if you just put in the root domain name. Why assume HTTP; you know? Actually it would be nice if the browser did both and waited to see who responded. But they're not doing that.

I mentioned last week that Google had put future Chrome releases on hold; that they were going to, like they were stopping work, they said, on any feature releases for the next Chrome. We're currently on 80, so that would have been for 81. They were only going to be doing bug fixes. But I guess they got comfortable pretty quickly with the way things were working with their engineers telecommuting. So that's no longer the case.

In a Chrome blog last Thursday, Google updated, saying that it plans to resume work on Chrome releases. They said that Chrome 81, which was originally scheduled for release on March 17th, has now been rescheduled for release on April 7th, so a couple, what, I guess next week, around the middle of the week. And at that time web developers and sysadmins would have had time, or their web developers and system administrators would have had time to adapt to their new working conditions.

So they're also saying that this does, I guess because it kind of squeezes it, it will result in dropping release 82. They're just going to skip the number altogether with its new features, which will be merged into release 83 and subsequent future releases. And 83 is now expected to be released sometime around mid-May. So I think they basically just postponed 81 a couple weeks because they've got an existing release pipeline set up. They're going to just skip over 82. There won't be one. And they'll jump over to 83 rather than forcing a renumbering of everything else downstream.

Last week we also noted that Edge was pausing their own releases to stay in sync with Chrome. And then last week, citing the impact on its customer base from COVID-19, Microsoft announced that, starting in May, optional Windows 10 cumulative updates would also be paused. So also a pause on the Windows 10 side. And as we know, Mozilla did not pause Firefox updates, but it did roll back the termination of TLS 1.0 and 1.1 so that access to government-based websites that weren't yet running 1.1 and 1.2 would be unimpeded. So that's cool.

Okay. So a bit of a deep dive on a forthcoming feature for Microsoft Windows 10. This is to support something known as shadow stacks. It's something we talked about quite a while ago, so we'll sort of refresh what's going on and what problem this solves. But this is very welcome for future versions of Windows 10. Last Wednesday, Microsoft's Hari Pulapaka, who's their group program manager for the Windows kernel, updated the world on the state of Microsoft's plans to add hardware-enforced stack protection to Windows. In this case, hardware-enforced stack protection takes the form of shadow stacks.

Through the last several decades, as we know, we've talked about the problem with the stack, buffer overruns, this concept of stack canaries, adding a little unpredictable bit of data that an attacker wouldn't be able to have to the stack in order to verify that there hasn't been an overwrite of a return address before the processor takes the return. You know, there have been all kinds of ways to try to shore up the problem that the stack inherently presents. In this case it requires hardware support from Intel in the form of a new feature being added to the most recent chips known as CET, Control-flow Enforcement Technology.

And back when we talked about the way processors operated in general, I talked then about how the stack itself, the concept of an execution stack, is one of the key innovations in the design of CPU architectures. It's been around for a long time, but not forever. For example, the earliest DEC PDP minicomputers and other early minicomputers back in the '70s did not have a stack. When I was writing those demo programs for the PDP-8s behind me, those PDP-8 emulators, the lack of a stack was an annoyance because it's so handy.

And one of the primary things that a stack does is manage subroutine calls, that is, you're executing code. You want to call a subroutine, a commonly used procedure to do something, for example, maybe it's to get a keystroke from the keyboard. And because in many different places in your code you may want to get a keystroke from a keyboard, rather than duplicating that code, the keyboard reading code all over, you just do it once. You create a routine that does that. And instead you call that routine, that is, you jump to it from all over in your code. In low-level computer programming, a jump instruction just changes the location where the computer is running. A subroutine call somehow saves where you are so that, after running the subroutine, you can return to it.

Well, the PDP-8 doesn't have a stack. So they came up with a kludge which was to store the return address in the first storage word of the routine, and then to start executing with the second storage word in the subroutine so that the subroutine would return to its caller by looking at the address in the first word of the routine and jumping there, an indirect jump through the first word of the routine. Well, that was clever, and it was simple to implement in logic. But one of the problems is that you cannot have recursion if you do that. That is, you could never have that routine call another routine that might then come back and call that first routine because it would overwrite the return instruction that was first stored the first time the routine was called.

The beauty of a stack is it solves this problem with recursion in a very elegant fashion. The stack is just a region of memory set aside in the system, and a register points to what is kind of arbitrarily decided as the top of the stack. Normally, stacks grow downward, that is, as you push data on the stack, the pointer is decremented to earlier or lower addresses in memory to successively store that data.

Okay. So later minicomputers like the PDP-10, which was a 36-bit machine, and the famous PDP-11 where Unix was first written, thank goodness it was a stack-based machine because otherwise Unix would have been a lot harder to pull off. I don't think they would have bothered, actually. I mean, a stack is so incredibly useful. So the main characteristic of a stack is that it's visible to the programmer. As I said, the processor has a pointer to the top of the stack, called the "stack pointer." And there are instructions for pushing various amounts of data onto the stack for temporary storage and popping it off of the stack when it's no longer needed, in order to retrieve it, or in some cases just to throw it away.

When parameters are passed to a procedure in a procedural programming language, the parameters are typically passed by having the caller of that procedure push those parameters onto the stack in a predefined order. Then the calling procedure then calls the subroutine, the procedure that it's looking for, which pushes the caller's return address onto the stack as the system then jumps to the beginning of the procedure. And if that procedure needs to allocate some space for local variables, that space will be allocated by moving the stack pointer downward to sort of create a buffer region of available space, again on the stack.

So the point is that the stack serves as a sort of multifunction scratch pad which does a very efficient job of giving transient data a place to live. And the stack is visible to the programmer, whose caller's parameters and its own local variables are all present and accessible by offsets into that stack. And as we know all too well, programmers often allocate, unfortunately, temporary communications buffer space on the stack, which if they're not careful allows more to be read onto the stack, thus overflowing data and causing buffer overruns on the stack. But when all goes well, a stack is an incredibly elegant innovation for CPUs.

Okay. So what's a shadow stack? I deliberately noted the many different sorts of things that cohabitate and share that single stack. And as we've seen historically, the biggest danger is that the control-flow data, that is, the subroutine return addresses, share the stack with the many other sources and types of non-control-flow data. And remember, all of it is visible to the programmer, that is, it's their stack. It's the currently executing thread's stack. So, for example, if a programmer wished to cause his subroutine to return somewhere other than back to its caller - hard to see a good use case for that. But if the programmer wanted to, it would be trivial for him to arrange. He'd simply overwrite the correct subroutine return address on the stack, waiting, with any other address in the system. And upon executing from the subroutine, the CPU would dutifully read that modified address from the stack and jump there.

And of course the point is, if malicious code somehow managed to arrange to do the same thing, that opens up a huge vulnerability. So by comparison, a shadow stack, this hardware-enforced shadow stack lives like its name suggests, in the shadows. Unlike the primary system stack, it is not visible to the programmer. It's not something the programmer can see or has any access to. This means that it's also not visible to any malicious code that might get loose. So Intel has added this shadow stack feature to their future CET-equipped CPUs.

Unlike the main system stack, which contains this wonderfully dynamic hodgepodge of data and control-flow, the shadow stack contains only a - it's only a stack of return addresses. And it's managed by the CPU behind the scenes invisibly. When a programmer makes a call to a subroutine procedure, this CET-equipped Intel CPU pushes the return address, the caller's return address, onto both the main system stack and the invisible shadow stack. The main stack receives and holds all manner of other information, as I was talking about, before and after and during the call.

But the shadow stack only has return addresses. When the called procedure eventually returns, executes a return instruction, this CET-equipped Intel CPU pops the return address both from the main stack and also separately behind the scenes from the shadow stack, and it compares them. They are guaranteed to match, so long as nothing nefarious or maybe inadvertent has modified the return address on the system stack. So it performs a nice sort of local stack verification in hardware, stack status verification in hardware. But those two return addresses are guaranteed not to match if anything might have modified or overwritten the original value stored on the big visible system stack.

So this makes for a very slick way to enable Intel's CPU hardware to catch any of the very common stack overwrite mistakes, as well as buffer overruns and other common stack-based security flaws. It's going to have, you know, essentially zero software or timing overhead, operates entirely on hardware based on this additional microcode and hardware support from Intel. It has been lagging. Last Wednesday's announcement was that support of this technology was now under development by Microsoft, and a preview is available in the Windows 10 Insider Preview builds, you know, the so-called "fast ring."

The specification for this has been public for several years. They first started talking about it about four years ago, in 2016. And support for it has preceded the wide availability of the hardware that actually supports it. The very popular GCC compiler suite and Glibc both added support several years ago. But once all of the pieces come together, what we'll effectively have is a significant step forward in our functional CPU architecture.

As I've said, the innovation of the stack as a general purpose, very efficient catchall for the storage of temporary dynamic data and control-flow has been a huge innovation in PC architecture, or CPU architecture. But it's always suffered from being also a little bit brittle and prone to either inadvertent mistakes or malicious abuse. So adding this invisible control-flow shadow stack will solve this problem of this multipurpose stack very elegantly for the industry.

I'm super excited that Windows 10 will be getting this. And Intel's chips, I tried to dig around and figure out at which point this would actually appear. I was unable to find anything definitive about which version of Intel hardware did actively have this. But it's something where the Intel, I don't know if it's Spectre and Meltdown or if they realized, oh, my goodness, we've got some other problems in the microcode with this that's kept pushing it back. But people have been waiting for this to get added to the Intel architecture for quite a while, and it looks like we're seeing it finally beginning to happen.

And it must be in the latest hardware. Otherwise there would be nothing to test it with under Windows 10. But that'll be nice when this problem is solved because, as I've been saying, it feels like we have some fundamental problems with the way our software works, and it needs to be rethought. One possibility is to make our languages far more automatic, that is, take the power out of the hands of the programmer, although programmers lose power kicking and screaming. The alternative is to come up with good ways to make our hardware help these problems much more than they have been. And this is just a perfect example of that happening.

So it's going to be great when we have Windows support. And it looks like, as soon as we get the various Linuxes and probably Mac recompiled with these features turned on, they'll have them, too. Oh, and if these features are turned on, and you're running on a chip that lacks the hardware support, it's just ignored. It just doesn't raise an exception because there is no comparison of the two stacks performed during a return jump.

So we have a instance of the press running around again with its hair on fire, screaming in one case "Patch now! Critical flaw found in OpenWRT router software." Okay, kind of. The situation is not good, but it's also not the end of the world. But anyone who would knowingly be running OpenWRT, and since that's an open source alternative that's loaded on top of other router firmware, I imagine anybody who is running OpenWRT knows they are. So there is a potential supply chain exploit, and it's very good that it was found and has now been fixed since it might conceivably have been exploited in either targeted or widespread attacks.

The problem was discovered and responsibly disclosed earlier this year by a guy named Guido Vranken, who was working with a company called ForAllSecure. And I think our listeners will find his description of his discovery interesting. I have the link for the full blog post. I'm just reading the top of it because he then goes into a much more detailed description. But just his introduction is neat. He said: "For ForAllSecure, I've been focusing on finding bugs in OpenWRT using their Mayhem software." And Mayhem is a fuzzer. We were talking about fuzzers recently.

He said: "My research on OpenWRT has been a combination of writing custom harnesses" - meaning interfaces for the fuzzer - "running binaries of the box without recompilation, and manual inspection of code." He said: "I found this vulnerability initially by chance when I was preparing a Mayhem task for Opkg." Opkg is the OpenWRT package manager. Anyway, he said: "Mayhem can serve data either from a file or from a network socket. Opkg downloads packages from downloads.openwrt.org, so my plan was to let this domain name point to 127.0.0.1 from which Mayhem is serving."

He said: "To test if Opkg would indeed download packages from a custom network connection, I set up a local web server and created a file consisting of random bytes. When I ran Opkg to install a package, it retrieved the file as I had intended, but then threw a segmentation fault." You know, it blew up based on this noise that it had downloaded. So of course we know, whoops, that's a red flag.

He said: "I didn't understand why an invalid package would cause this error. After all, the package should not be processed if the SHA-256 hash was incorrect. My initial hunch was that Opkg would download the package, unpack it to a temporary directory, and only then verify the SHA-256 hash" - excuse me.

Leo: You want to take a little break?

Steve: Just a little sip of coffee.

Leo: I didn't give you your usual coffee break. Sorry.

Steve: Thank you, buddy.

Leo: Yeah, yeah, yeah.

Steve: So anyway, so he said: "...and only then verify the SHA-256 hash before definitively installing it to the system." He said: "I suspected that the unpacker couldn't deal with malformed data." Okay, there's another red flag, but we're going to run across many of them before we're done here. He said: "Like the file with random bytes served from my web server." And I have to mention that it should raise an alarm for everyone that he was able to set up a web server which Opkg would download from because, like, wait, what?

Leo: Not good.

Steve: Like, what? Uh-huh. So he said: "Further inspection" - I hope everybody with OpenWRT is sitting down at this point. "Further inspection showed that the SHA-256 hash wasn't being checked at all, which is the basis of the vulnerability at hand." Meaning this first of many. He said: "I was right about the unpacker being buggy, though; malformed data would lead to a variety of memory violations."

He said: "Once I confirmed that Opkg would attempt to unpack and install any package it downloads, I was able to recreate the findings with Mayhem with just a slight modification to Opkg." He said: "I set up a Mayhem task for 'opkg install attr.'" He said: "Attr is a small OpenWRT package," obviously to show attributes. He said: "And implicitly, Mayhem was able to find the remote code execution bug by detecting the memory bugs in the package unpacker. If OpenWRT's SHA-256 verification had worked as intended, Opkg would simply discard the package and not process it, and no segmentation faults would transpire."

He says: "Mayhem is capable of fuzzing binaries without recompilation or instrumentation. Coming from a workflow that involves writing many custom harnesses for software libraries," he says, "which Mayhem also supports, this has been a delightful experience, and it has allowed me to set up targets for dozens of OpenWRT applications in just weeks, and more vulnerability disclosures are forthcoming."

Okay. So in other words, first of all, there are a bunch of problems with the OpenWRT codebase that should put everyone on, well, notice, and frankly on edge. But first and foremost is that its package manager has not been bothering to check the hash of anything it downloads. And I should mention since 2017. And believe it or not, this problem is significantly compounded by the fact that these updates are over HTTP. Yes, I'll say it again, HTTP and not HTTPS. That's the only way this guy was able to set up a local web server on 127.0.0.1 and probably used the hosts file to redirect whatever that was, downloads.openwrt.org, to his local IP. If you are downloading stuff over HTTP, as we know, that means there's no certificate to verify that the package is being obtained from the correct server, which makes DNS spoofing or any other type of man-in-the-middle-style traffic interception easy to pull off.

I guess I'm not quite back to a hundred percent. But nearly so.

Leo: Well, it could just be, you know, allergies or something.

Steve: Just a cough. Okay. So the OpenWRT project has known about this since the beginning of the year. They recommend carefully upgrading - I think I put the word "carefully" in because, again, if it's over HTTP, you've got to be sure you're downloading your update from the proper place - to the latest version. That means making sure you're connecting to the correct server. Make sure that your DNS has not been changed.

And interestingly, although I haven't reported it just because I just didn't have time with everything else that's been going on, there have been a bunch of router DNS attacks recently. So definitely make sure that your router is using the valid DNS servers that you deliberately configured it for and that it might not have been changed because of course that is the trivial way that you pull off a man-in-the-middle to a non-HTTPS site, just by redirecting downloads.openwrt.org.

Okay. So the bug was introduced. It's been given CVE-2020-7982. It was introduced in early 2017. It affects OpenWRT versions 18.06.0 through 18.06.6, so that range of 18.06 versions, and 19.07.0, and also separately the OpenWRT LEDE fork version 17.01.0 through 17.01.7. The fix was applied to an updated version of that 18.06 series, taking you to 18.06.7, which is what you want if you were using 18.06, and to 19.07.1, if you currently have 19.07.0. It was released at the beginning of February. It's been around. Maybe your device has updated itself. In any event, you want to get that fixed if you're an OpenWRT user because now it's been made publicly widespread, widely known, and people who don't fix that could find themselves victims of the problem.

Cloudflare and 1.1.1.1 had themselves audited by KPMG. As we know, Mozilla's decision to route all of its browsers' DNS queries by default via DoH, DNS over HTTPS, to Cloudflare raised a bunch of noise when it was first announced. Many creaky old-school Unix diehards chafed over the loss of DNS's inherent distributed design. You know, people complained that this was, you know, becoming sort of a monoculture of DNS, if all Firefox users' DNS went through one provider. And perhaps some or much of this was due to their lack of long-term knowledge of who Cloudflare is. You know, I wasn't worried.

They may have believed, if they didn't know otherwise, that it was just some random provider. And in truth, if Cloudflare was not as well known to me and us, you know, Leo and our listeners, I'd have been concerned by the idea of trusting some random single provider. But for us, Cloudflare is not some random provider. We do know Cloudflare and many of the people behind the name. So it always seemed like an excellent choice and a good idea. And we have said, as our own ISPs decide that they want to bring up DNS over HTTPS, they're welcome to do that with their own local DNS servers, and then people could choose not to focus on 1.1.1.1.

But Cloudflare has made some very strong both claims and commitments and promises about the way they're going to manage and their eyes-off and hands-off policy for all of the DNS traffic which then will be transiting their network. So the good feeling we have always had about them has been solidified because they opened for a long period of time last year their entire network to KPMG for the sake of allowing a fully independent audit. The auditor's report has a ton of detail, bullet points about specifically the things they did and saw and found and were fine about. And I've got a link in the show notes to the entire KPMG report.

But it concludes: "In our opinion, management's assertion that the 1.1.1.1 Public DNS Resolver was effectively configured to support the achievement of Cloudflare's Public Resolver commitments for the period from February 1, 2019 through October 31, 2019, based on the criteria above, is fairly stated in all material respects." In other words, they got a green light from an independent KPMG auditing group who they let see exactly what they were doing.

So if anyone was harboring any concerns, I'm not surprised by this outcome. I'm delighted by it. And props to Cloudflare for saying, okay, we're going to handle this, and we'll let somebody come in and take a look at what we're doing. And actually, based on the detailed report, they really dug in. This was not some token transient BS bureaucratic audit. It was down at the edge resolver, syslog is logging or not logging, and what is logging, and the retention of packets and, I mean, it was in real detail. So I came away completely pleased and convinced.

Oh, and I wanted to acknowledge everybody - and yes, even you, Evan Katz - for informing me that the new Edge Chromium browser is going to be receiving vertical tabs. Yay.

Leo: That's your feature that you keep using Firefox, the sole feature that keeps you using Firefox.

Steve: Yes, it is. Nothing replaces that, yeah. And of course all of our listeners know it. And so I was the subject of a Twitter storm. They're going to be adding vertical tabs. And all I can hope is that Google will, for whatever reason, who has stubbornly refused to do so, will say, hmm, looks kind of popular over there. Looks kind of good. Maybe we should consider doing that because...

Leo: Tell me why you like vertical tabs?

Steve: Well, because a tab is vertical. I mean, I'm sorry, a tab is horizontal. A tab is horizontal.

Leo: Yeah, it is, yeah.

Steve: And so you can stack them much more efficiently vertically than you can stack them horizontally. And all of our screens are no longer 4x3. They're all 16x9 or more. So typically, if you bring your browser full screen, now it's just sitting in, you know, your browser is sort of - the content is in the middle with a lot of dead space on both sides. So put the tabs there. You can see them easily. It just makes sense to sort of square the browser window by filling the left side with tabs that work vertically. It just seems, like, crazy that it isn't a default. So anyway, apparently someone at Microsoft agrees. And, yay, Edge will be getting those with a Chromium engine on the back end. So everybody gets to win.

Leo: How do you - you're not using an extension to do it in Firefox. You're just using the little browser, what do they call that, the little browser gutter on the left; right? You're turning that on?

Steve: Well, no. No, no. I actually do have...

Leo: You use an extension for it, okay.

Steve: I have something called Tree, yeah, I have Tree Style Tabs, which creates a hierarchy of tabs, which I like. I don't think Firefox does it natively.

Leo: Well, it does, but it's kind of a weird - here, I'll put it on the left so you'll feel more at home. This is that gutter I was talking about. And you can have it be a variety of things, including synced tabs, which is all the tabs that all your machines have.

Steve: Right.

Leo: So you turn off the - I don't know. I still have tabs, though, so I don't really know how that would go.

Steve: Yeah.

Leo: So you'd want to have the extension.

Steve: Yeah. I do use one. And there are a number of vertical tab extensions for Firefox.

Leo: Weird that, I mean, if it's just an extension, that Chrome doesn't have an extension that does it. That seems odd.

Steve: There is one. I think Chrome's UI for some reason fights it. There is an extension that hangs a separate window, sort of as a sidecar, off the side. But it just doesn't feel right. And besides, I've solved the problem with Firefox. I just want to have it.

Leo: You don't need it. You don't care.

Steve: Yeah. Although I would love to have it native. And some day the world is going to capitulate, apparently, and we're going to get vertical tabs.

Leo: That's good.

Steve: SQRL.

Leo: SQRL.

Steve: Yeah, SQRL now has a beautifully written, open source, cross-platform Windows, Linux, and macOS native SQRL client and library.

Leo: Nice.

Steve: So SQRL for Linux, SQRL for macOS, no more need to use WINE. I meant to put screenshots in the show notes, and I just forgot. This is the work of Jose Gomez and Alex

Hauser. Jose previously wrote the OAuth2 provider for SQRL, and Alex did a lot of work with Daniel's Android client.

Leo: Yeah. Jose helped me get it on the TWiT Community, yeah.

Steve: Right, right. So this client now has a new forum over on GRC's SQRL forums at sqrl.grc.com, not a trivial subdomain. And the project is being hosted on GitHub. I've got the link in the show notes for anyone who is interested. It's github.com/sqrldev/SQRLDotNetClient. So those guys have just produced a beautiful cross-platform client. So all of our desktops are covered now.

And as for SpinRite, I was, as I noted, a bit slowed down by what I strongly suspect, without evidence yet, was COVID-19. As we know, there's increasing talk about an antibody test, and I will be first in line for it, as soon as it's available. I'm going to be very disappointed...

Leo: We should take bets on whether you had it or not.

Steve: Yeah, I just - Lorrie actually thinks not. She thinks...

Leo: Because you didn't have a fever, which is, as we were talking about it before the show, a common symptom.

Steve: It is. It is.

Leo: Like very common, yeah.

Steve: It is. Oh, I also didn't have - I had respiratory compromise, I mean, I had this weird dry cough.

Leo: You're still coughing, Dude. You definitely had that.

Steve: Yeah.

Leo: Whatever it was you had.

Steve: For weeks. So anyway, the moment there will be an antibody test. And, you know, it's going to be a few more weeks, I think. But there's a big push for it because people who are positive for the antibody are presumably immune for some length of time. The most recent medical feeling is one to three years. We just don't really know. But based on other known coronaviruses, the expectation is at least a couple years of immunity. And then you may get it again, but be asymptomatic. Your body might just deal with it.

Leo: Right, or it'd be more mild.

Steve: Exactly. Exactly. Anyway, so while I was unable to code, I did manage to read the second of Hamilton's "Salvation" trilogy, and it did not disappoint. I'm now one book behind on Ryk Brown's "Frontiers Saga." I've read and very much enjoyed the first 27 books, so book number 28 awaits.

Leo: Geez.

Steve: But those are much lighter reading.

Leo: Geez, that's a lot.

Steve: All that said, I was back to work on SpinRite's AHCI driver over the weekend, and I will be returning to it this evening, and I'm hoping to have some new code for our anxious SpinRite testers to test soon. So moving forward on all fronts once again.

Leo: Before we get to Folding, I did want to mention there's been another Marriott breach.

Steve: Oh, 5.2 million, yup.

Leo: You saw it, okay. I wasn't sure if you missed it or not. This one's not as significant because they got credentials for some Marriott employees through the app. It was actually a pretty hare-brained scheme. And I guess there was no protection. So having employee credentials gave them access to guest information for 5.2 million customers since last January, since January. So, yeah, once again, guest names, addresses, birthdays, emails, phone numbers, and loyalty reward programs for both hotel and partner airlines. But no passport information, no driver's license, no passwords.

Steve: Yeah. There is a service you can use to find out if you are subject to the breach. And I don't know. Maybe they'll say they're sorry.

Leo: They didn't say they were sorry this time. We're not sorry. Sorry, not sorry. It was a breach. Sorry, not sorry.

Steve: Yeah, wow.

Leo: Anyway, you left it...

Steve: Someday, Leo, we're going to figure out how to make a secure website. But that doesn't seem to be happening anytime soon.

Leo: You left it out because you thought it was insignificant, or just you ran out of space?

Steve: Well, yeah.

Leo: Just another one.

Steve: I looked at the, what is it, 358 million from the Starwood Hotel's breach. And I thought, okay, you know.

Leo: If you're a Marriott person, you're used to it.

Steve: Breaches are happening now just so commonly.

Leo: It really is true, isn't it, yeah.

Steve: In fact, there was even some discussion about whether we now should really call it a "breach." If someone puts their data on the Internet, is it a breach or an unauthorized access?

Leo: That's a good point.

Steve: You know, it's like, well, you published your database on the Internet. You shouldn't have, but you did. So it didn't really require any great hacker skills in order to download the database, which was there on an API. So is that a breach? You're unhappy, but maybe not.

Leo: You gave them the information, yeah.

Steve: Yeah. So we now can donate our unused CPU cycles to help provide answers to COVID-19. It's a happy coincidence that just as the SETI project decided that it just had no longer any need for the raw signal processing number crunching that all of its users were doing, that a crucial new need for distributed computing at massive scale would arise, because our unused CPU cycles can now help to increase our understanding of the structure and function of the COVID-19 virus.

It turns out that molecular modeling can be used to identify therapeutic drugs that might be of use in preventing the COVID-19 spike protein from binding to the ACE2 receptors of the cells in our lungs. In other words, our medical science and molecular modeling technology has progressed to the point that the field of computational biology is now a thing. And, oh, baby, is it compute-intensive. But it's possible to run simulations entirely using very sophisticated math to predict the 3D molecular shape and thus the interactions between biological compounds.

Wikipedia has a terrific and up-to-the-date intro that even mentions about Folding@home, which is even mentioning COVID-19. They said: "Folding@home" - and that's FoldingAtHome.org, for those of you who don't want to wait - "is a distributed computing project for performing molecular dynamics simulations of protein dynamics. Its initial focus was on protein folding, but has shifted more to biomedical problems such as Alzheimer's disease, cancer, COVID-19, and Ebola.

"The project uses the idle processing resources of personal computers owned by volunteers who have installed the software on their systems. Folding@home is currently based at the Washington University in St. Louis School of Medicine, under the directorship of Dr. Greg Bowman. The project was started by the Pande Laboratory at Stanford University, under the direction of Professor Vijay Pande, who led the project until 2019. Since 2019, Folding@home has been led by Dr. Greg Bowman, who is a former student of Dr. Pande."

They wrote: "The project has pioneered the utilization of CPUs, GPUs, PlayStation 3's, Message Passing Interface used for computing on multicore processors, and some Sony Xperia smartphones" - okay, don't do that - "for distributed computing and scientific research. The project uses statistical simulation methodology that is a paradigm shift from traditional computing methods. As part of the client-server model network architecture, the volunteered machines each receive pieces of a simulation (work units), complete them, and return them to the project's database servers, where the units are compiled into an overall simulation. Volunteers can track their contributions on the Folding@home website, which makes volunteers' participation competitive and encourages long-term involvement." It's like, hey, how many work units have you achieved, you know, and so forth.

"Folding@home is one of the world's fastest computing systems." Get a load of this. "With heightened interest in the project as a result of the 2019-2020 coronavirus pandemic, the system achieved a speed of approximately 768 petaflops, or 1.5 x86 exaflops at March 25, 2020" - so last Wednesday - "making it the world's first exaflop computing system. This level of performance from its large-scale computing network has allowed researchers to run computationally costly atomic-level simulations of protein folding thousands of times longer than formerly achieved. Since its launch on the first of October 2000, the Pande Lab has produced 223 scientific research papers as a direct result of Folding@home. Results from the project's simulations agree well with experiments."

So basically this is doing, out on this massive network - as I mentioned to you before, Leo, the sum of processing power by the world's top seven fastest supercomputers is less than the current power of the Folding@home network. Since February, the Folding@home community has been working on the computationally heavy work of finding out how the COVID-19 virus protein binds to cells.

When the outbreak was picking up steam, the Folding@home project asked for volunteers to donate their computers' unused computational power to help accelerate the open science effort to develop new lifesaving therapies as part of a collaboration of multiple laboratories around the world. Folding@home says, get this, there's been a roughly 1,200% increase in contributors, with 400,000 new members signing up in the past two weeks. So I would imagine the listeners of this podcast will probably be responsible for another serious jump in that number.

Due to the project basically being swamped recently, the Folding@home researchers are working to generate enough work for this massive influx of new processing power to tackle. They indicated that there might be a bit of downtime, that is, like people would be looking, and their Folding@home cycles were not being sucked up, while new simulations are being set up. They said: "Usually your computer will never be idle, but we've had such an enthusiastic response to our COVID-19 work that you will see some intermittent downtime" - meaning lack of use of the resources that you are making available - "as we sprint to set up more simulations. Please be patient with us. There is a lot of valuable science to be done, and we're getting it running as quickly as we can."

So I would say a very worthwhile effort for those of us who leave machines on and unattended while we're not using them. Let it download little work packets of computational biology to perform, and then get that work done and send the results back to them. We'll all be doing our share. Basically, a mathematical biological laboratory, which is just so cool.

Leo: And they have some evidence that this is a worthy thing to do? I mean, I know in theory it is. It uses a lot of electricity, so we would want to make sure that it is in fact going to do some good instead of just kind of make you feel good.

Steve: Yeah. Given that or bitcoin pooling...

Leo: Well, that's dumb, yeah.

Steve: Yeah.

Leo: But what's, I'm just curious, I mean, again, it makes everybody feel good. But is it really doing anything?

Steve: I'm not enough an expert to comment either way. Maybe we'll do some more digging or have some people who know more.

Leo: It's not insignificant, if they're driving that many exabytes, insignificant use of power. I mean, somebody once observed that the bitcoin mining is using more power than the entire solar power installations of the globe could provide, which means it's not a good thing. As long as it's doing something of value. But if it's just making people feel "I'm solving the problem," you know, I'd like to know, I'd like to see their success stories, I guess.

Steve: Yeah, that would be good. I cannot speak to it authoritatively one way or the other.

Leo: I think on the website, I looked once, they have some success stories. But I'm not probably qualified to judge it. So forgive my skepticism, but I'd like to know more.

Steve: Skepticism's a good thing, my friend.

Leo: I'm always skeptical of things that make everybody feel good, but don't have any apparent real value. But who knows? And you're certainly welcome to do it. I did it for a while, and it used so much power and got my machine so hot, I thought, maybe I won't do this. The fans were on all the time. It was crazy. Crazy. But

Washington University at St. Louis School of Medicine is a nice organization, so I'm sure it's fine.

Steve, we're done. Go rest your throat, you poor fellow.

Steve: We are. I'll do that.

Leo: Yeah. Feel - I know you're feeling better, but drink some honey. Honey and whiskey.

Steve: Actually, that's what Lorrie's been making me is Earl Grey with caffeine and some honey.

Leo: Nice. Perfect.

Steve: In order to soothe my throat.

Leo: Perfect. Thank you, Steve. Steve Gibson does this show every Tuesday, and you're welcome to watch us do it live. That's about 1:30. It's right after MacBreak Weekly, so it's maybe 2:00 o'clock Pacific time. That would be 5:00 p.m. Eastern time. That'd be 21:00 UTC if you want to tune in. TWiT.tv/live has video and audio streams. Of course most people want to listen on demand because you want to have a whole set. You want to collect all 700 and whatever episodes.

So if you're a collector, if you'd like to have the complete set of 760 episodes, go to Steve's site, GRC.com. You'll find little teeny-weeny 16Kb versions of the show. You'll find the full 64Kb audio of the show. You'll also find something he has uniquely, which is the transcripts of the show. He pays to get those done every week. And that's really great if you like to follow along while you're listening. That's all at GRC.com.

While you're there, pick up SpinRite, the world's best hard drive recovery and maintenance utility. And there's lots of other free stuff there. Browse around because it's a treasure trove for people who like to learn things. You can also get copies of the show at our website, TWiT.tv/sn. We have audio, and we also have video, oddly enough. You can get it there. And best thing to do probably would be subscribe in your favorite podcast application. Then you don't even need to have to think about it. It'll just be there of a Tuesday evening or a Wednesday morning, just in time for your commute from your bedroom to your living room.

We hope you enjoy the show, and we'll see you next time. Bye, Steve.

Steve: Bye-bye, buddy.