Adiantum

Description: This week we look at Apple's most recent v12.1.4 iOS update and the two zero-day vulnerabilities it closed, as well as examine the very worrisome new Android image display vulnerability. We dive into an interesting "reverse RDP" attack, look at the new LibreOffice and OpenOffice vulnerability, and consider Microsoft's research into the primary source of software vulnerabilities. Mary Jo gets an early peek at enterprise pricing for extending Windows 7 support. China and Russia continue their work to take control of their countries' Internets. Firefox resumes rollout of its AV-warning Release 65. We offer up a few more SQRL anecdotes, share a bit of listener feedback, then see how Google does the ChaCha with their new "Adiantum" ultra-high-performance cryptographic cipher.

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-701.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-701-lq.mp3

SHOW TEASE: It's time for Security Now!. Steve Gibson is here. Coming up we'll look at the iOS zero-days and the patch to fix them; an issue with RDP that Microsoft says it's not a high priority to fix that; and issues with OpenOffice, LibreOffice, and Russia's plans to disconnect from the Internet. It's all coming up next on Security Now!.

Leo Laporte: This is Security Now! with Steve Gibson, Episode 701, recorded Tuesday, February 12th, 2019: Adiantum.

It's time for Security Now!, the show where we cover your security and privacy online with the commander in chief of the privacy brigade, Mr. Steven Gibson.

Steve Gibson: The privacy - every week it changes, Leo. I never know what you're going to come up with, but it's always fun.

Leo: You should see the uniform. It's all lead foil. It's fabulous. Fabulous.

Steve: Yeah, yeah. A little heavy to wear if you get the thick version of that.

Leo: I was reading that tinfoil actually conducts and probably is not ideal for a hat.

Steve: Well, as long as you didn't have any open seams it'd be essentially a Faraday cage, so nothing could get in. But if it's just a little cap on top, that's just not going to do the job. That's not going to happen.

Leo: That's why I said the uniforms, the entire uniform.

Steve: So this episode is titled "Adiantum," which is a new term that Google has coined. And it allows me to use the phrase "Google does the ChaCha," for reasons we will be getting to. Some very, very cool technology. And as we know, oftentimes among the news of the week nothing really stands out. This time, last Thursday Google announced something which is significant for our industry. So Adiantum is its name.

But first we're going to look at Apple's most recent 12.1.4 iOS update and the two zero-day vulnerabilities it closed, in addition to fixing the FaceTime problem. Also a very worrisome new Android image display vulnerability involving PNG files, which make Stagefright probably look like nothing by comparison. This is not good. We're also going to do a bit of a dive into the research into a reverse RDP attack. We've talked about the Remote Desktop Protocol and how bad it is for people to leave their RDP servers exposed to the public Internet. Well, it turns out you need to trust the server you're connecting to every bit as much. So that was some really interesting research that Check Point did.

Also there's a problem with LibreOffice and OpenOffice, which one of the two has been fixed. But until it is, it's raising all kinds of alarm bells. I wanted to share quickly Mary Jo's early peek at enterprise pricing for extended Windows 7 support which you guys talked about last week on This Week in Windows. Also China and Russia are both continuing their work to take control of their own countries' Internet. We'll touch on what that's about. We've got Firefox's resumption of its AV warning, well, remember that they had a problem we talked about last week because it was causing such problems for AV. They've resumed releasing Firefox 65 as a consequence of some changes. I have a few more SQRL anecdotes to share, a bit of listener feedback, and then we're going to see how Google does the ChaCha with their new Adiantum ultra-high-performance cryptographic cipher mode.

Leo: Oh, sounds cool.

Steve: So lots of fun stuff to deal with today.

Leo: Awesome. Steve?

Steve: Actually, come to think of it, I could have done a different picture. I should have done the picture of the algorithm for this new cipher, Adiantum.

Leo: Is it pretty?

Steve: Yeah, it is. And it is in the show notes later on, so we will get to it.

Leo: We'll get to it, all right.

Steve: But anyway, I just liked this. This was this beautiful piece of work, this ridiculously inexpensive PDP-11/70 kit. Someone tweeted...

Leo: This is Oscar's kit?

Steve: Yeah, this is Oscar's kit. And look at the, I mean, he just does just a stunning job. I mean, that's just the...

Leo: That's the insides.

Steve: Yes, that's the PC board that no one will ever see, which is just, you know, gold-plated and a beautiful legend mask and spaces for all the lights and switches and things. And he's got light. You can see up at the top of the picture our light masks that help to align the LEDs so that they line up properly, you don't have to fudge them manually. And they also provide inter-LED isolation so the light from one doesn't bleed into the other.

Leo: He's Swiss, isn't he.

Steve: Yes.

Leo: You can kind of tell.

Steve: Yeah, yeah. Anyway, just a beautiful piece of work. So I wanted to just sort of share that since someone had sent that. And he did the injection molding to make those funky switch toggles. The DEC switch handles are this weird triangle with skirts, and they don't exist in the world. And so he made them so that they fit on top of these beautiful Alcoswitch toggles. And, I mean, it's just a work of art.

Leo: What's the address? Or maybe we shouldn't send people there anymore; right?

Steve: We probably should. I think if you put in, what was it, guaranteed...

Leo: Obsolescence guaranteed.

Steve: There it is, "obsolescence guaranteed" and then "PDP-11," maybe. Or just "obsolescence guaranteed." But look at it, I mean, it's just - and so the whole thing is - he did an injection-molded bezel so that you have the - and silk-screened front panel. I mean, it's only about two or three inches deep because it's based on the Raspberry Pi that is emulating the PDP-11. And so really all this is, is it's a physical I/O device. It's switches and lights to emulate the PDP-11. And he sat down with a friend, videotaping as they, like, toggled switches slowly on an actual PDP-11 to get every light correct. So it's just, again, yes, his perfectionism is something I completely relate to. And that's just a beautiful piece of work. And it's $250 for this amazing kit that does not take a rocket scientist to build. So I just wanted to show it again. It's just a beautiful piece of work.

Leo: I did get on the list, and I got an email from him a little while ago saying fill out the form. And I've done that, and I'm just waiting for more information.

Steve: Yeah, I mean, he's just a hobbyist who's nuts.

Leo: Yeah, yeah, that's awesome.

Steve: You say, well, this turned into a little bit of a career, more than he expected. It's like, okay, as long as you want to do it.

So I didn't have a single one of my iOS devices prompt me to go get 12.1.4, yet on every one of them, when I went looking, it's like, oh, we have a firmware update for you. So you may want to get it because, aside from fixing the unattended FaceTime - did I say "Facebook" before? Probably - FaceTime eavesdropping bug that we talked about either a week or two ago, the one where somebody could add themselves to a group while calling you that would cause your phone to think that the person you're calling had answered so that you'd get this zero notification activation of your microphone and camera. So Apple responded by quickly shutting down the group servers that took that feature away from FaceTime.

Anyway, 12.1.4 is the fix that we expected to be immediately forthcoming. But in addition to that, there are three additional flaws which were fixed, two of which are being actively exploited in the wild as zero-day exploits. Google's Project Zero researchers found those two zero-days, which they privately disclosed to Apple. And the third one Apple independently discovered and fixed. The two from Google, there's a memory corruption issue that could allow a malicious application to gain elevated privileges on the vulnerable device, and a different memory corruption issue that could allow malicious application to execute arbitrary code with kernel privileges.

So as always, we never get any further information. Apple's not saying anything. Google's not saying anything. It's just, okay, let's get this fixed. But that's sort of demonstrates that, as much focus as Apple puts on securing these things, we're still finding problems.

And then the third problem, which is also fixed, was discovered by Apple themselves. It's a different flaw in FaceTime involving live photos, which they have now foreclosed. So probably worth updating. Again, as I said, none of my devices said, hey, we've got an update. I don't know, like, some of them downloaded them. Some of them needed to download this update. Maybe they just wait for you to go looking or to see the little red badge on the settings icon and then go, oh, there's something I should do. I don't know. But eventually...

Leo: They roll it out bit by bit. That's the problem.

Steve: Yeah.

Leo: And so there are people vulnerable who - like Lisa this morning said, oh, there's an update, did you know? And I said, well, yeah, kind of, but...

Steve: Exactly, yeah. And you're right, Leo, the problem is there are just so many of these devices out there that it's going to take a while. Oh, and let's not forget macOS. As a consequence of this increasingly shared codebase, it is vulnerable to all of these problems. So macOS, you want to go to Mojave 10.14.3, which fixes all three of those iOS vulnerabilities which macOS also had. So update all your Apple stuff. And a little bit later I'm going to talk about this continuing problem that we have with security and why we just never seem to get this right.

Google announced some things they fixed in their February 2019 updates for Android. And there were a trio of surprisingly worrisome Android PNG display bugs, image display, which means this is in the PNG interpreter which is invoked when anything on your Android device needs to render a PNG image, which says, what, the web. Any web page, any ad on a web page, a picture in an email that you receive and open, a PNG received through instant messaging. So, I mean, it is a big vulnerability because it can execute arbitrary code.

Google, in explaining what they had fixed, now that they have fixed it, said the most severe of these issues is a critical vulnerability in framework that could allow a remote attacker using a specially crafted PNG file to execute arbitrary code within the context of a privileged process, meaning that this thing that's doing the interpretation is privileged. It's down in the kernel, not up where the user is. So this is a service being provided by the OS which means, if you have a problem there, the code that gets loose is getting loose with the privileges of the process that was running it, which is the kernel.

So they said: "The severity assessment" - which was highest. "The severity assessment is based on the effect that exploiting the vulnerability would possibly have on an affected device, assuming the platform and service mitigations are turned off for development purposes or if successfully bypassed." They said: "We have had no reports" - this is important - "no reports of active customer exploitation or abuse of these newly reported issues." They said: "Refer to the Android and Google Play Protect mitigations section for details on the Android security platform protections and Google Play Protect, which improve the security of the Android platform."

Under Framework they said: "The most severe vulnerability in this section could enable a remote attacker using a specially crafted PNG file to execute arbitrary code within the context of a privileged process."

The Hacker News site framed it a bit less clinically, and probably more realistically. To paraphrase what they wrote, they said: "Using an Android device? Beware. You need to be more cautious when opening an image on your smartphone downloaded from anywhere on the Internet or received through messaging or email apps. Yes, just viewing an innocuous-looking image" - and of course they showed one of a kitten - "could hack your Android smartphone thanks to three newly discovered critical vulnerabilities that affect millions of devices running recent versions of Google's mobile operating system, ranging from Android 7.0 Nougat to its current Android 9.0 Pie. Although Google engineers have not revealed any technical details, the updates mention fixing a heap buffer overflow flaw, and also errors in SkPngCodec, and bugs in some components that render PNG images."

So we know that Google, as we talked about last year, Google is working diligently to improve the Android update ecosystem that is working to require manufacturers to be responding to Android patches and to push them within some length of time. But what has already shipped has already shipped. And most of Google's efforts are necessarily forward looking. You know, they can't retroactively amend the contracts that they've had people sign about what they've already done. So that means that the bad guys now know that there are three vulnerabilities, one of which at least is very bad, allowing arbitrary

code execution, and PNG rendering image flaws in Android since v7, which was released on August 22nd of 2016.

So 2.5 years ago this flaw was introduced. Every version since and every phone sold since has it, until it's updated with these February patches which have just been released and announced by Google. And of course they made them available to partners, but we also know that doesn't mean a lot at the moment. So every phone sold in the interim for the last 2.5 years, until and unless patched, will be vulnerable to these flaws.

Those who have been listening to this podcast since the summer of 2015, that is, the year before that, will recall the Android Stagefright exploit which caused quite a bit of activity on both sides of the law, given that this problem only needs to display a PNG of a specially crafted image, and that these fixes are probably reverse-engineerable by the bad guys since Android has the mixed blessing of being way more open than Apple is with their iOS platform. I have a feeling we haven't seen the end of this. So it may well be that a month from now we are talking about people receiving just an image on their phone and having their phone commandeered. So we'll see.

What it means for our listeners is, if you are using Android, hopefully you have a phone from a manufacturer who will make those updates available, maybe directly from Google, maybe through a responsible manufacturer. Maybe you can go get them. Whatever it is, it would be good to get them because I will be surprised if the bad guys don't jump on this. The problem is too many Android devices are either never going to get patched or are going to create a large window until they are patched. And the idea of just showing somebody a picture is, well, I mean, is really bad.

We recently covered the news of exposed remote desktop servers. Microsoft's RDP, Remote Desktop Protocol, is what typically Windows servers expose, which allows people with an RDP client to connect in order to get a remote desktop. And I guess apparently, based on behavior, either people believe it's safe to leave an exposed RDP server unmonitored to the Internet for anyone who wants to, I mean, it's what, port 3389? Shodan, just do a search, you find all these port 3389s, happily accepting connections. Hey, I'm RDP. Come guess my username and password. You can have a desktop. Oh, my lord. Anyway, that's the world as it is now.

And, you know, the new term is "credential stuffing," which is the term that's now in vogue for brute force automated guessing of logon credentials. We used the term "credential stuffing" last week when we talked about the Japanese government, who was going to be scanning hundreds of thousands of their own citizens' IPs performing credential stuffing attacks on anything they can find, typically IoT things, and see whether they can get in, in which case they're going to work with those people to tighten their security. So, I mean, I use RDP because I'm a Windows guy. I've got Windows servers. You won't find a port 3389 open anywhere because people would be loony, just it's crazy to have a server like that exposed. But Check Point Research became curious, not about the security of RDP servers, which we already know is frightening, but about the security of the client connected to an RDP server.

The RDP protocol is extremely complex and was developed in the dark as opposed to in the light of day, being as it is a proprietary protocol of Microsoft's. So of course this hasn't prevented its reverse engineering by others, and several third-party and open source clones exist, client clones exist of RDP clients. But as we know, a "complex protocol," if you put that in quotes, should give any security-aware person pause since we know that complexity is the enemy of security. No one should be surprised to learn that the RDP clients are rife with exploitable flaws.

Check Point wrote in their disclosure, they said as an overview: "Used by thousands of IT professionals and security researchers worldwide, the Remote Desktop Protocol (RDP) is

usually considered a safe and trustworthy application to connect to remote computers. Whether it is used to help those working remotely or to work in a safe VM environment, RDP clients are an invaluable tool. However, Check Point Research recently discovered multiple critical vulnerabilities in the commonly used Remote Desktop Protocol that would allow a malicious actor to reverse the usual direction of communication and infect the IT professional or security researcher's computer."

In other words, if the RDP server you connect to with your RDP client has been maliciously modified, it can attack your client, your machine from which you are connecting with a vulnerable RDP client - and it turns out they all are - in order to hurt you. They found 16 major vulnerabilities, and a total of 25 security vulnerabilities overall.

They looked at three clients. The de facto default Microsoft built-in RDP client is mstsc, which is Microsoft Terminal Services Client is their name. But there are two others that are popular and mature open source clients. There's one called FreeRDP, which is the most popular RDP client, on GitHub. And there's rdesktop, which is an older open source RDP client, which happens, as an interesting twist, to come by default in Kali Linux distros, which is interesting because Kali Linux is what many red teams use as their security platform. And should they use rdesktop to work to penetrate a remote machine, it could penetrate them instead.

Okay. So Check Point wrote: "As is usually the case, we decided to start looking for vulnerabilities in the open source clients. It seems it will only make sense to start reverse engineering Microsoft's client after we will have" - their English is a little broken, but I'm going to read what they wrote - "after we will have a firm understanding of the protocol. In addition, if we find common vulnerabilities in the two open source clients, we could check if they also apply to Microsoft's client. In a recon check, it looked like rdesktop is smaller than FreeRDP, has fewer lines of code, and so we selected it as our first target."

I should explain. I'm going to go into some detail here because I think our listeners will find the strategy that they adopted and the way they talk about what they found to be interesting, sort of as an inside look into how an organization like Check Point, that is very mature security research, how they tackle something like this. So first they went for the open protocol, or rather the open source, because that was examinable. And then they went for the smaller of the two just because there was less to examine, the argument being, if they find problems there, then that would give them a starting point for looking at the closed source Microsoft implementation.

So they said: "We decided to perform an old-fashioned manual code audit instead of using any fuzzing technique. The main reasons for this decision were the overhead of writing a dedicated fuzzer for the complex RDP protocol, together with the fact that using AFL" - that's the American Fuzzy Lop is one of the often-used fuzzers - "using AFL for a protocol with several compression and encryption layers didn't look like a good idea." In other words, fuzzing, as we've talked about before, is the process of just throwing stuff at an API, just giving it a bunch of stuff and see if the system crashes.

Leo: It's like system events, though, always is like clicks and mouse moves and stuff like that? Or is it...

Steve: Yeah, well...

Leo: It's mostly system events.

Steve: Yeah. So when you're actually connecting to the server, there is, in the case of RDP, a deep API.

Leo: Oh, so you have all sorts of garbage over the API.

Steve: Yes. And the problem is so much of it is going to be that - so the idea is the API is so highly structured that meaningful exploits are probably not going to be random. There are so many ways that the API would reject fuzz that your likelihood of finding something is really low. That would require that they write a custom fuzzer that is protocol-aware in order not to just be rejected out of hand. And so they thought, okay, we're not going to write custom fuzzing code. Let's just take a look at the code. So they just looked, they just started reading the source.

They first started, as I mentioned, with rdesktop v1.8.3. They said: "After a short period, it looked like the decision to manually search for vulnerabilities paid off. We found several vulnerable patterns" - and I'll explain that in a second - "in the code, making it easier to 'feel' the code and pinpoint the location of possible vulnerabilities." And this is sort of, it's like when you read code you can get a feel for the personality of the person who wrote it.

I mean, if you looked at my assembler, you would absolutely get a sense for me. I mean, code is that expressive, where you can see, like, oh, okay, Steve, he's weird. He'll push some arguments onto the stack because he's going to later need to use them when he calls something. And rather than popping them off the stack just to then push them on the stack as arguments to the call, he'll prepush them a ways ahead to have them there and then not go through popping and then repushing because he understands what's actually happening in the processor. And so you'll look at that and go, you know, but I'm careful to comment that because it's like, okay, someone looking at my code will go, wait, why is he pushing these over here? And so I always comment, as much for myself as for someone else, it's like, these are being pushed because these are arguments that'll be used down lower a ways. And so I might as well just have them there waiting for that function.

Anyway, the point is code really can reflect the personality, for lack of a better term, of the person writing it. And what they saw was they saw some habits that this coder had that were bad. They were bad coding habits. So not just obscure, which is what you might call mine, but actually dangerous. So what that did was that allowed them to then look for instances of the use of this bad habit and see if they were located in vulnerable places. And they found 11 vulnerabilities with a major security impact and 19 vulnerabilities overall in the library, just after getting a sense for the habit of the coder where they saw that, oh, he's doing some things that aren't really that safe, and that caused some problems.

They also noted as an aside that the other end of this rdesktop connection was an xrdp server, which is also open sourced and based on that same rdesktop code, which led them to believe that, well, in fact they did some additional recon. And based on their findings, it appears that similar vulnerabilities can be found in RDP, as well. And that's a concern because that is a server as opposed to a client, which might be exposed and now is believed to have some problems.

So anyway, I'm not going to go into detail because I just kind of wanted to give our listeners a sense for the way they operated with this. In their disclosure, and I've got a link in the show notes - yup, I do, just checked. In their disclosure, they've got screenshots and details for anyone who's really curious about the nature of the programmer's personality that they found and what it meant. But to sum it up, they

characterized what they found as leading up to, as they put it, massive heap-based buffer overflows. They wrote: "By chaining together these two vulnerabilities" - they were sort of related - "each pair found in three different logical communications channels," they wrote, "we now have three remote code execution vulnerabilities" they found in the client.

And then dealing with a different instance of a vulnerability, they wrote, and this one had been given a CVE of 2018-8795, remote code execution, they said: "Another classic vulnerability is an integer overflow when processing the received bitmap for screen content updates." Remember that this is remote desktop, so the server is sending the client chunks of its screen, like regions of screen that have changed and need to be updated. So the server is saying, here's a rectangular bitmap region of the screen with this width, this height and this pixel depth.

So what they noted was, although the bitmap width and height parameters are only 16 bits each, by multiplying them together with the bits per pixel, the pixel depth, it's possible to trigger an integer overflow, meaning that 16-bit value times a 16-bit value times the bits per pixel depth can overflow 32 bits, which is the integer value that was being used here, and lead to badness. They said: "Later on, the bitmap decompression will process our input and break on any decompression error, giving us a controllable heap-based buffer overflow." And they said: "This calculation can be found in several places throughout the code of rdesktop, so we marked it as a potential vulnerability to check for in FreeRDP," that is, the other open source RDP client.

So then of FreeRDP they wrote - and they tested 2.0.0 RC3, so Release Candidate 3. They said: "After finding multiple vulnerabilities in rdesktop, we approached FreeRDP with some trepidation. Perhaps only rdesktop had vulnerabilities when implementing RDP." They said: "We still can't be sure that every implementation of the protocol will be vulnerable. And indeed, at first glance," they wrote, "the code seemed much better. There were minimum size checks" - oh, and technically that's what was missing from that code pattern that they kept seeing in the way rdesktop was coded. They weren't checking for minimal - they weren't doing a sanity check, as I call it in my code, for minimum size. So "...before parsing the data from the received packet, which was the primary feature missing from rdesktop," they said, "and the code feels more mature."

They wrote: "It's going to be a challenge. However, after a deeper examination, we started to find cracks in the code, and eventually we found critical vulnerabilities in this client, as well." And I should note that here's another perfect example of the fact that, just because the code is open and can be audited doesn't automatically mean it's more secure. Nobody ever apparently looked before. So there it is, and it's open, and it's free, and it's full of bugs. And we've talked about this back and forth, is closed better than open and so forth. And it's like, yes, open source has the potential to be verifiably secure, but somebody has to go verify it. Just the fact that it's there doesn't mean that it's any better than something that somebody wrote and didn't publish open.

So they said they found critical vulnerabilities in this client, as well. They found five vulnerabilities with major security impact and six vulnerabilities overall in the library. Then some additional sleuthing discovered that the RDP client NeutrinoRDP, which is a fork of an older version of FreeRDP, therefore probably also suffers from the same vulnerabilities. It exists and probably bears scrutiny, if anyone is using it.

They said: "At the end of our research we developed a proof-of-concept exploit for this client," which they were able to demonstrate. And then also there's something they called this "same integer overflow." They said: "As we saw earlier in rdesktop, calculating the dimensions of a received bitmap update is susceptible to integer overflows. And indeed, FreeRDP shares the same vulnerability." So there was some, you know, even if neither project inherited code from the other, both implementations failed to check for,

basically believed the server they were connecting to and were not being defensive about the data coming from the server. And basically these clients, as we often see, are interpreters. And so these interpreters, these RDP client interpreters were just taking on faith the safety of what the RDP server was providing and were very vulnerable to that.

So what about Microsoft? The original RDP client from which these were reverse engineered - since as far as I know Microsoft never published, formally published the protocol. And of course Microsoft's original RDP client is the one that's built into all of our Windows machines, and it's the one which Windows users will be using. If you are on Mac or on Linux, then you're probably using one of these other clients if you're connecting with remote desktop to a Windows server.

So these guys wrote: "We started by testing our proof of concepts" - the ones they had developed, the proof of concept exploits they developed for the other two - "for the vulnerabilities in the open source clients. Unfortunately" - well, unless you're a Windows user. They said unfortunately for the success of those proof of concepts. But they wrote: "Unfortunately, all of them caused the client to close itself cleanly, without a crash." So the client just said, okay, I don't know what I've connected to, but this doesn't seem like a valid Windows server. I'm going to go away now. So just it shut down without crashing.

Then they said: "Having no more excuses, we opened IDA" - which of course we've discussed recently also, that's IDA, the Interactive Disassembler. And they said: "We started to track the flow of messages. Soon enough, we realized that Microsoft's implementation is much better than the implementations we tested previously." They said: "Actually, it seems like Microsoft's code is better by several orders of magnitude, as it contains several optimization layers for efficient network streaming of the received video; robust input checks; robust decompression checks to guarantee that no byte will be written past the destination buffer; and additional supported clipboard features. Needless to say, there were checks for integer overflows when processing bitmap updates."

So Microsoft's client is written to protect the system using it to connect to a remote foreign server. However, Check Point researchers did find some troubling flaws in clipboard sharing. They go into great detail in their posting, and I didn't want to drag us through that because there isn't much, I mean, there is a lot of technical detail, but nothing of much interest. But Remote Desktop Protocol supports clipboard sharing. And it turns out that it's possible for a malicious RDP server with Microsoft's current RDP client to use classic path traversal attacks to place files anywhere on the user's system, limited only by the user's current privileges.

So the RDP client is running in the context of the current user, so that's good. It's not running with any elevated privileges. But Microsoft's client is susceptible to a path traversal attack so that, if the server you're connecting to were modified, or the protocol intercepted or somehow manipulated, files could be placed anywhere that you have permission to do so. And you have permission to put files in your startup folder, which would mean that they would be executed next time you log in or start the system. So that's not good.

They also discovered that the RDP server, if it were not well behaved, that is, if it were malicious, could dynamically monitor the client's clipboard because there's clipboard sharing which is enabled by default; and that anything that the user even transiently places on their clipboard while you have RDP open, a connection open, the server gets a notification and the contents of the clipboard on the fly. So, for example, if you just pasted a complex admin password, that goes to the server, even if you don't do anything with it explicitly over the connection, and even if it's only briefly present.

So then in their posting they outlined their disclosure timeline because they did full responsible disclosure for this. And the short version is the FreeRDP and the rdesktop supporters immediately responded, quickly fixed the problems, asked that Check Point verify the fixes and give them the green light. They did fix the problems. Check Point gave them the green light. And there are now new versions of both FreeRDP and rdesktop.

Microsoft responded: "Thank you for your submission. We determined your finding is valid but does not meet our bar for servicing." And I put in here parenthetically, "(In other words, we have MUCH bigger problems over here.)" Anyway, they said, so, yeah, we agree, that could happen. We're not going to fix it.

Leo: You'd have to log into a malicious RDP server, though.

Steve: Correct, correct.

Leo: So that's probably what they're thinking is, well, it's on you, then, buddy.

Steve: Well, and how likely is it, really. I mean, a Windows user is probably always going to be logging into a Windows RDP server, not something else. So seems very unlikely that that would be the case. So out of an abundance of caution, they do note that the clipboard sharing can be disabled. It's a simple configuration checkbox over on your client, whether you want to share your clipboard with the server. And so if you have occasion or any reason to mistrust the server you are connecting to, you could disable clipboard sharing.

That's the only problem that they found in the Microsoft RDP client. And for what it's worth, if you are a Linux or Mac user, it's worth updating. This all just happened. So do get the later versions of rdesktop or FreeRDP, if you use those. Again, in any event, this is a reverse attack against you by a malicious server you connect to. Interesting from a technical standpoint. Interesting from a let's take a look at how defects are born. But not nearly as big a concern, for example, as all the Android devices for the last 2.5 years that can be compromised when displaying a PNG image that they receive from anywhere. That's one to worry about.

So any users of LibreOffice and Apache's OpenOffice should update. There was a classic path traversal vulnerability in those. It's funny how that's always bit us. The idea of ../../.., it's sort of a clever hack; but, boy, has it caused problems historically. A security researcher, Alex Infuhr, has discovered a severe remote code execution vulnerability in both of these open source office suites that can be triggered just by opening a maliciously crafted ODT. That's the OpenDocument Text file format. The attack relies on exploiting, as I mentioned, a directory traversal flaw, which has a CVE-2018-16858. It automatically executes a specific Python library bundled within the software.

Both of these office suites bundle their own Python interpreters so you don't have to have one separately installed. It's there. And by mixing a white-colored link with an onmouseover event and turning the whole page into a link, he's able to get this thing to trigger. So all it requires is that a user open one of these maliciously crafted documents, and bad guys' code runs on your system.

He wrote in his disclosure that the Python file named "pydoc.py" that comes included with LibreOffice's own Python interpreter accepts arbitrary commands in one of its parameters and executes them through the system's command line or console, thus

essentially creating a pipeline that allows the attacker supplied command to execute, even though it's in a document. That pydoc.py has a security problem with it, I mean, like fundamentally. He provided a proof-of-concept video demonstration which shows how he's able to trick that event, the onmouseover event, into calling a specific function within the Python file, which then executed the payload that he had provided through a Windows command line, and showed no indication, no warning dialog of any kind to the user. And he didn't really have great success.

We've talked before about the problems that researchers have. Some researchers have even been grumbly about how difficult it has been to report problems. There was a little bit of that. He said: "At first I reported it via the LibreOffice Bugzilla system. Apparently for security issues it's better," he said, "to send an email to officesecurity@lists.freedesktop.org," he said, "but I didn't know that." In fact, we've talked about the initiative for there to be some sort of consistent reporting path independent of organization, some way for people who discover vulnerabilities to know who to talk to, especially if you're not doing this all the time. It takes some digging around.

He said, anyway: "So my Bugzilla report got closed, but I convinced them to have another look. The bug was picked up and moved to a thread via officesecurity@lists.freedesktop.org, which is where it should have gone." He said: "The issue was verified and fixed quite quickly." So that was on the LibreOffice side. OpenOffice confirmed via email that they recognize there's a problem, but OpenOffice does not allow the passing of parameters. So his proof of concept doesn't work as it was developed for LibreOffice. But the path traversal is still there and can be abused to execute Python script from another location on a local file system.

So at this point I would opt for LibreOffice, if I had a choice, because those guys jumped on it and actually fixed the problem. So no real big message there beside the fact that, if you are using LibreOffice, you want to just check for an update which is waiting, depending upon how long it's been.

Also I picked up an interesting little tidbit from a BlueHat security conference which occurred in Israel last week. Microsoft security engineer Matt Miller said that, over the past 12 years, not changing very much, around 70% of all Microsoft patches were fixes for memory safety bugs. I guess it shouldn't really surprise us. Leo, you've got the graph that Mike provided in his slides from his presentation on the screen, basically showing kind of a, you know, looks like one of those tanks with oil and water in it, where the oil doesn't mix with the water, and it sort of has a wave going back and forth. It's just sort of a wavy blue line separating the light blue from the dark blue, with a dotted red line right at 70%. And it's maybe a little more than average over 70, but it's kind of around 70, dipped down below a little bit for a while in 2012.

But basically the point is, over a decade and two years, 12 years, we've not seen much change. It's always been around 70. And despite everyone's efforts at improving things, address space layout randomization and doing everything we can, I guess I would argue that it would have gotten a lot worse if we hadn't fought back with technology to work against these memory-related problems. But it's pretty much been a standoff at 70%. Which is sort of interesting. So that would be all memory-related things. Now, arguably it's a big bucket, a catchall. There's buffer overflow. There's memory-related race conditions, page faults, null pointers, stack exhaustions, heap exhaustions and heap corruptions, use after free vulnerabilities, or double free vulnerabilities, where you free something and then free it again, and that causes another upset. So that is arguably a bunch of stuff. But again, it's been hovering at 70%.

Anyway, the reason Matt explained for this high percentage is because Windows, he says, and I agree, has been written mostly in C and C++, which he describes as two

memory unsafe programming languages that allow developers fine-grained control over the memory addresses where their code can be executed and where data can be stored. Just a simple mistake in the developer's management of memory can lead to exploitable memory errors that attackers can exploit to obtain remote code execution or the elevation of their privilege within the system.

And, I mean, we know that's the case. Anyone who's looked at C realizes, I mean, you can put anything you want to into a pointer. And if you dereference that pointer, you're accessing your memory space. So, I mean, there's no abstraction. There's nothing between you and the bare metal. Which is, I mean, that's why C was written. We've talked about this a number of times through the years is Kernighan and Ritchie developed C to be like, I mean, they wrote the first Unix in assembler. And then they said, okay, let's do a minimal abstraction on top of assembly language in order to give us some convenience.

So C is a very small language. It adds just a small layer on top. And then it uses libraries in order to further expand the language. So again, programmers like it because they want the power that comes with it. Unfortunately, with that power comes responsibility. And we seem to be having a problem with that.

So as a consequence, all of this makes memory management and safety errors today's biggest attack surface for hackers. And these errors are what attackers are capitalizing upon, thus the reason that what Microsoft patches are memory-related problems. His presentation, Mike Miller's presentation asserted that use after free and heap corruption vulnerabilities remain the preferred go-to bugs for attackers searching for exploitable vulnerable code behavior. And so there's no big surprise there.

And I've been mentioning recently that it's somewhat surprising to see how deeply entrenched existing computing technology actually is, meaning how reluctant we appear to be to really change anything fundamental, despite the fact that fundamental change is what we clearly need. My own current focus, as everybody knows, has been on usernames and passwords to log into computers. But we're largely using the same technology today that, I mean, we've added password managers in order to deal with the problems of the fact that websites are unable to keep our secrets. We're using the same technology today that mainframe terminals used in the early 1970s, which is now approaching 50 years ago. That hasn't changed. Which is mindboggling.

But similarly, coding has always not appreciably changed throughout all that time. Yeah, there have been, and we hear about various dynamic programming languages from time to time. There have been academic experimental memory safe languages, where the allocation is done for you. You're protected from yourself. Your variables are garbage-collected after you so you don't have to worry about the whole allocation problem. But for many different reasons, they never gain foothold as a mainstream core implementation language.

And if the language itself were to be safe, the problem is that they're still running on old-school hardware architectures and being compiled, or more likely interpreted, because these fancy languages tend to be interpreted, they're being interpreted by buggy interpreters running on old-school hardware. So it never really gets us anywhere. And then of course there's programmer hubris, which is never to be underestimated in scope or depth. No programmer ever thinks they're going to make a mistake. And when they inevitably do, it's considered a mistake. Whoops. And it's treated as though it won't ever happen again. But of course it will and it does, over and over and over.

If this is going to change at some point, we need to rethink the whole thing. And I don't know where that rethink comes from. Maybe from some ivory tower somewhere. But I also wonder, will it really ever change at this point, since it's been so long, Leo, that

we've had what we've had. I mean, there was even a Forth processor, a Forth chip that was, you know, ran Forth. And that sort of just never went anywhere.

Leo: I think it's probably easier to mess things up with Forth than almost anything, to be honest.

Steve: Yes. Yes.

Leo: You know, it's a stack-based language. You can put anything on the stack, pop it off the stack and, you know.

Steve: Well, and as we've often said, it's a write-only language because you are basically the compiler, and it is really difficult to read Forth after the fact. But I wonder really if there's any motivation in this direction, any true motivation. People say they want security, but they really don't insist upon it. They complain when something is insecure, but users cannot see security. They don't understand what it really is. It's not a feature. It's not an obvious benefit. It's a complete intangible.

And of course we've used the analogy before how we smugly lock the front doors of our homes that have easily broken glass windows. So it's like, okay. A home would be much more secure if it were a bunker with no windows. But nobody wants to make that tradeoff, even though, yes, it would be much more secure. So the presumption is, well, it's secure enough. Right, until someone breaks in. And it's well understood that Apple's iOS-based devices are significantly more secure than Android. They're not perfect.

Leo: Swift does a lot of memory management and garbage collection protection.

Steve: Yes. Yes. On the other hand, most Android devices are much less expensive, so that's what most people buy. I mean, I know people with Android. And it's like, it's good enough, and it was cheaper. So even though Apple has problems, too, people buy Android. So I guess not only don't and won't people actually pay more for security, but this industry still doesn't actually have true security to sell them. So maybe we are going to need more than three digits for this podcast, Leo.

Leo: There's got to be a - good, I hope so. There's got to be a way, though. I mean, NASA designed software that can't be patched.

Steve: Oh, but at such cost. At such cost.

Leo: Maybe that's the answer is it's too expensive, yeah.

Steve: And that's the problem. Yes. Using today's technology, today's computing technology, we can produce provably secure software. But no one has time. It's just too expensive. So we don't bother. And speaking of updates, Mary Jo on This Week in Windows last week shared with you and Paul some news from Microsoft. We know that Windows 7 is now on extended...

Leo: Life support.

Steve: ...security, yes, yes, life support. Until this time, until February of 2020, so one year from now. We've talked about the idea that - because it only just happened, despite all the crazy pressure Microsoft has put on the world to go to Windows 10, it only just happened that Windows 10 passed Windows 7 in usage. And nobody who has 7, mostly enterprises, wants to leave. They're like, everything's working. And besides, Windows 10 has a bad rap, or rep, or both, when it comes to privacy. I mean, Leo, have you installed Windows 10 on a new machine?

Leo: Dozens of times, yeah.

Steve: It looks like an arcade game. I mean, stuff is flipping up and down and jumping around, and you're being asked if you want candy corn dropped on you or I don't even know what this is. It's just insane. I just look at this, and I think, you've got to be kidding me. Anyway, so it's no surprise that enterprise people want nothing to do with this nightmare. So anyway, Mary Jo shared last week, and I have a picture of what she posted over on ZDNet in the show notes. Windows Enterprise versions starting next year will be able to pay $25 per seat for continued updates for the year. Well, actually, wait a minute, January 20 through January 21. I don't understand that.

Leo: This doesn't take effect till January of 2020, till next year. The extended updates expire 2020.

Steve: Oh, 2020, right, 2020. Okay. So next year $25. The year after, $50 per seat.

Leo: Doubles every year.

Steve: Yes. And the year after, $100 per seat. That's the Enterprise, which is the cheaper of the two. If you are on Windows 7 Pro, it's $50 the first year, $100 the second year, and $200 the third year. So the idea is, yes, and this is not for end users. End users are going to get left in the cold. You know me. Actually, I mean, I have Windows 10 all over the place now. And I have arranged to strip the crap out of it. There's a bunch of things you can do in PowerShell that just removes all of this, whatever the heck that crap is that they have in there now, the new apps that nobody wants. I heard Paul a couple weeks ago just saying to you, "It's all junk. Have you looked at those Windows Store apps? They're just junk." It's like, yeah, but I guess they're secure junk, so okay. Anyway, I'm making peace with Windows 10.

Leo: This is making peace? I can't wait to hear when you make war. Okay, yeah.

Steve: Anyway, the point is, for what it's worth to our people who are in enterprises, Microsoft is going to start pushing hard to, if you want to keep security updates, and if you want to stay on Windows 7, they're going to start pushing hard, starting next year, to move people off. I don't know what enterprises will do. Maybe they'll just say, fine, we'll pay. Maybe they'll give up and move to 10. Or maybe, you know, there is a version of 10, I think I heard Mary Jo talking about it, or I read about it - no, she did refer to it. There's a version of 10 that is available to enterprises that doesn't have any of this

nonsense in it. End users can't get it. We can't get it. But there is one that Microsoft makes available that is tamed.

And so maybe that'll get made more widely. I mean, nobody wants Xbox junk that you can't uninstall in your Windows 10, if you want nothing to do with Xbox. It's just loony tunes. And then ads for games and, ugh. Anyway, so now we know what it's going to cost. We can't get it, enterprises can, and it's going to cost a bit.

Okay. Whew. Maybe I've had too much caffeine. China's cybersecurity law actually went into effect last November. And we didn't talk about it much. But in light of Japan's forthcoming test, their own essentially pen test of their own network's IoT devices coming up here soon, I just wanted to note that - and we'll be talking about Russia here next, in a second. But China has, not surprisingly, given themselves some expansive powers. So I wanted to follow up on our discussion of Japan.

This newly enacted law allows China's Ministry of Public Security, the MPS, which is the same agency which maintains China's Great Firewall and runs its national facial recognition system and the surveillance camera network, has been since November empowered to conduct in-person or remote inspections of the network security defenses taken by companies operating in China, check for prohibited content banned inside China's border, log security response plans during onsite inspections, copy any user information found on inspected systems during onsite or remote inspections, perform penetration tests to check for vulnerabilities, perform remote inspections without informing companies, share any collected data with other state agencies, and has the right to have two members of the People's Armed Police, the PAP, present during on-site inspection to enforce procedures.

So again, as we know, it's not a democracy. But I'm feeling like, as a consequence of this, as a consequence of what Japan is going to do, they're going to end up with much more secure network infrastructures, that is, public infrastructures, than we, the U.S., who so far refuse to allow white hat or government agencies to do the same. And I don't think this is going to stand. I bet you that we're going to see Australia follow suit, given the penchant that Australia has seen. And then probably the U.K., and then I wouldn't be surprised if the U.S. does. I just think, nice as it is to say, okay, nobody gets to do this, to only allow the bad guys to do it, look at the preponderance of evidence this last year of podcasts that we've been talking about. It was nice once, but it's just no longer practical.

And I said we were going to talk about Russia. This is going to be really interesting to see. Russian authorities and major Russian Internet providers are planning to disconnect the country from the Internet as part of a planned experiment. The ultimate goal is for Russian authorities to implement a web traffic filtering system like China's Great Firewall and to also arrange to have a fully working, self-contained, country-wide Intranet - essentially that's what you would call it if it's no longer global - in case the country needs or chooses to disconnect. And you could imagine hearing in a Russian accent: "Why do we need the rest of world?"

So the initial experiment, which is intended to occur before, but hopefully not on, April 1st, will serve to provide insight and feedback and possible modifications to this law which is proposed to allow this to happen, which has been introduced in the Russian Parliament last December. The first draft of the law mandated that Russian Internet providers should be able to ensure the independence of the Russian Internet space, which they refer to as "Runet," in case of foreign aggression, to disconnect the country from the rest of the Internet. Additionally, Russian networking firms would have to install "technical means" to reroute all Russian Internet traffic to exchange points approved or managed by, Leo, our favorite organization, Roskomnadzor.

Leo: The FCC.

Steve: Yes.

Leo: Should we do this in the U.S.? What do you think? I don't think this is such a bad idea.

Steve: It's not such a bad idea.

Leo: I mean, obviously in the U.S. you'd only do it in case of war. I mean, you wouldn't do it just because you didn't like what you were hearing from the outside world.

Steve: Right. Well, yes. And in fact I wouldn't be surprised if we don't already have...

Leo: A kill switch.

Steve: Cutoff switches on all cables coming, all transatlantic cables coming into the U.S. and satellite stuff.

Leo: That's really interesting.

Steve: We probably do. The problem is it would cause chaos. Our system would collapse if it didn't have access to - because there's just the assumption of all these resources located globally. To their credit, what Russia is doing is they're being upfront about it. They're saying, you know, we're going to start running experiments, a series of drills.

Leo: Let's just see what would happen.

Steve: Yes, exactly.

Leo: What would happen?

Steve: Let's just pull the switch.

Leo: Might be okay. We'll see.

Steve: Let's pull the plug and, yeah. So anyway, Roskomnadzor will inspect the traffic to block prohibited content. And that's really interesting, too, because, okay, they're saying they're wanting to experiment with routing everything through Roskomnadzor, but everything is TLS now. So that suggests everybody...

Leo: You can only turn it off or on. You can't really...

Steve: Yeah. You can't look in it unless you get the Russian root cert that they're going to force everybody in Russia to have.

Leo: Oh, boy. Well, they might do that.

Steve: I think China is going to do it. And anybody, any state that insists on being able to inspect the traffic, the contemporary traffic of their citizenry has to put a root cert on every machine and then perform fake cert creation and proxying at the border. That's the only way to do it is to perform a state-sanctioned man-in-the-middle inspection of traffic.

Leo: Wow.

Steve: Anyway, the Russian government has been working on this project for years. And in 2017 Russian officials said they plan to route 95% of all Internet traffic locally by 2020. That is, meaning through Roskomnadzor.

Leo: I like how you say that.

Steve: Buckle your seatbelts. To pull this off, they'll still need their own DNS. So authorities have built in-country backup of the entire domain name system. It was first tested in 2014, and then again last year. It will be a major component of Russkinet - wait, no, Runet.

Leo: Oh, I like Russkinet. That's good.

Steve: Russkinet.

Leo: Russkinet.

Steve: Russkinet - when ISPs plan to pull the plug and disconnect the country from the rest of the world. So we're going to, ooh, sometime before long, next month or maybe, I guess, oh, no, before the first of April. So we're going to be talking about what happened when Russia pulled the plug on the Internet. Woohoo.

Leo: Wow. Wow. Very interesting.

Steve: Yeah.

Leo: Yeah, I mean, if you use 1.1.1.1 in the U.S., and we cut off our Internet connection, your DNS would go down.

Steve: Yeah.

Leo: Right?

Steve: Yeah.

Leo: So we're so interconnected at this point, you can't...

Steve: Yeah, but I really, you know...

Leo: And you need a kill switch because...

Steve: I'm sure there's no chance right now, given what we know about the Internet, that there isn't one that can isolate the country. And the problem is...

Leo: It would isolate the country.

Steve: Everything will break.

Leo: That's the problem.

Steve: Yes. Everything will break.

Leo: Very interesting.

Steve: So I just wanted to mention also that we talked about Firefox's suspension of Release 65, which was warning its users who had Avast and AVG AV and also some others. All the vendors have now issued an update, the AV vendors, to suspend their scanning of traffic for users of Firefox. So Mozilla has resumed rolling out 65. So for what it's worth, if Firefox users want AV scanning, you'll need to switch to a different browser, probably use Chrome. Or just rely, as you and I do, Leo, on Defender, that does very good, it's rated now, it's ranked right up there with the other guys, very good antivirus. And it does so because it's native to the OS. It doesn't cause Firefox a problem. So yay.

Leo: Steven?

Steve: So I'm going to share three anecdotes and then talk a little bit about why I'm not doing more. Domsch is his handle in the SQRL forums. And he said: "Just wanted to chime in. I created my first SQRL ID months back without having a use for it. Of course I lost the restore codes. So after listening to SN-700" - meaning last week's episode - "on my commute yesterday and today, I set up a new identity created on my Android phone with the awesome Android app, signed onto the forums in seconds, and five minutes later signed onto the forums on my work PC without any problems."

He wrote, "This is awesome. The biggest thing for me, it's independent of a password manager." Which is not to say we don't love password managers, I'm glued to mine, but we all need them right now. His point is, and he writes: "The password manager extension is one of the big things keeping me with Firefox. I'd really like to switch to Falkon, but without my passwords that's a no-go."

He says: "So here is me hoping SQRL will find wide adoption on all platforms so I can finally leave my password manager behind." He wrote: "Thanks, Steve, for creating this awesome authentication method, and especially for not going with the 'early bird, alpha, preorder, green light' approach that's so common nowadays, but instead doing it right and delivering an actually 100% working product."

Then CormaP said, and I just picked these comments up in the forums, he says: "I installed the Android SQRL client a week or two ago and have just used it to register here for the first time. I foolishly forgot my password for my identity." And I think, okay. He says: "So I reset the password using the rescue code. Painless. Logged into the forums using SQRL. Painless. Set up two-factor using the LastPass authenticator. Painless."

He said: "All good so far. My question is, can I avoid having to reenter my password when logging into a site on my phone? Or can I just use the phone's fingerprint sensor? Or is it simply because I have only logged in twice? Am I missing the point/doing something wrong? Thanks." He said Cosma, so maybe it is Cosma. Or maybe I just, no, maybe - anyway, he spelled it differently.

And to answer his question, first of all, he used the features that I built into the whole system to allow people to do their own password recovery because exactly what he had happen is going to happen. And with SQRL, because it is rigorously two-party, there's no one to go crying to if you forget the one password you only ever need. The good news is you only need one ever for all your sites that use SQRL, so you're much less likely to forget it. But it could happen. Or you could reenter it twice, and then forget - you could change it and forget what you changed it to.

So anyway, this solves all those problems, and it worked for him. I'm asking Jeff Arthur, who's the author of the iOS version, to be less rigorous with unlocking SQRL on the iPhone under the assumption that, if you've unlocked your phone with your fingerprint or your face, then you've already just - in fact, you have to do it per authentication. So you are right then reauthenticating yourself to the SQRL app.

Right now he's been following the protocol I use on my Windows client where it's a little finickier about making you reenter your whole password because in its default mode Windows doesn't have biometrics built in, whereas the newer iOS phones, as we know, do, and even tablets. And Android, the most recent Android Pie has a usefully secure fingerprint authentication also. So to answer his question, yes, when this settles down, you will be able to only use a contemporaneous reauthentication in order to log on.

And lastly, KenRS said, and he was writing to Jeff in the forums because all of the authors of the clients have their own development spaces there. He said: "Jeff, just trying your client on this forum, and it's great to see it working. The identity import using the printed QR code worked perfectly. Once set up, I scanned the SQRL QR code appearing on my laptop screen with my iPad and was instantly logged in on the laptop." And he says: "That part does seem like magic." He says: "Logging in 'on-device' on the iPad also worked. The only thing that still is rough is signing in using the fingerprint authentication on the iPad. Instead of logging in instantly, I'm left with Safari searching for localhost. Canceling that leaves me logged in. Regards, Ken." And, yes, that's something that Jeff will be fixing.

So anyway, we're getting close. I'm sharing these anecdotes while a great deal of work is underway behind the scenes, as we sort of are in countdown. It's become clear to the authors of the other SQRL clients that this bread is just about baked and that SQRL really does work and is actually going to happen; at least, I mean, like we'll have multiple clients and multiple servers and a working website where people can go and experiment with this.

So right now work is proceeding at an increased pace as we approach the finish line. I'm working to flesh out the SQRL forums content so there's material there for newcomers. At the same time, I need to update SQRL's documentation, the online documentation, so that those who want to evaluate its underlying technology and operation will be able to do so because those docs are five years old now, and they are only barely reflective of the way things actually turned out.

So I guess my point is that right now we're sort of getting a trickle of people, and those who really know it already, who have been in the newsgroup, have moved over and are experimenting. It's sort of just the right level of activity that we can see problems and work to fix them. While you've been doing the sponsor breaks, Leo, I've been researching why I'm getting a very occasional server error 503 from the FastCGI module of IIS because it's running PHP. And now I think I know why, thanks to you talking to our listeners about Sophos.

Leo: He uses those breaks intelligently, ladies and gentlemen.

Steve: Yeah, that's right. When you see me doing this when you come back to the camera, there's a reason for it.

Leo: I love it. You waste no time. I love it.

Steve: So anyway, I mean, this is - I'm getting very excited and also very nervous. You can imagine, you know, this has been a huge effort. But, I mean, it really all is working. And it is very exciting. Oh, there's also a web extension running both under Firefox and Chrome. So you don't even need to have - and that'll give us Mac and Linux. And there are people running my Windows app under Wine on a Mac. And so anyway, lots happening. I know I'm teasing people, but it's getting close. And right now I have enough happening that, if there was a flood of people, it would kind of overwhelm us. So we're just working out the details because you know me, I'm a "measure it twice or maybe 10 times and then cut once accurately" person. So we're getting there.

Leo: Good. How exciting.

Steve: I'm very excited. So three things about pfSense real quick. Dave, who tweeted from @darkmatter_0x00, he said: "Hi, Steve. Great podcast. Quick question. A few episodes ago" - oh, and we just heard a yabba-dabba-doo, so thank you. Someone just purchased a copy of SpinRite, which I didn't talk about in this podcast. But, I mean, it absolutely is the reason SQRL exists, because our listeners and the public and those needing to keep their hard drives in shape and to recover from problems have supported this effort as a consequence of purchasing something which is hopefully useful to them immediately and which I have promised will be useful moving forward in the future, and which I will get back to as soon as this SQRL project is behind me.

He said: "Quick question. A few episodes ago you mentioned a small firewall VPN device/router that can be bought. What was the name of it? Only has a few ports and ran pfSense. Maybe I'm wrong about the 'pf' part. Thanks." And of course that was the SG-1100. And think of Steve Gibson, although there's no affiliation, as in SG - what does it stand for? Probably secure gateway? Anyway, SG-1100 [Security Gateway]. If you google that, it ought to take you to Netsense, I think it is, or Netgate. They're the people.

So that's what it is. And yes, it is three ports, so a WAN port and two on the LAN side, which allows you to do network isolation. It does run pfSense, which is absolutely the drop dead manager that you want. pfSense has just really, well, first of all, it runs everything and has a web browser-based interface that allows you to do stuff. For example, I'm pulling off all kinds of networking magic, thanks to having the flexibility that it offers.

Bldg, which looks like Building 35, tweets @Bldg35. He said: "Hi. Just wanted to say thanks for planting the idea in my head regarding pfSense. I've got my SG-1100 online now. Reconfigured my ASUS RT-AC5300 to AP [Access Point] /AiMesh mode." And actually that's exactly what I did with mine. He said: "I was surprised that I didn't lose any WiFi features, such as guest node isolation, et cetera."

He said: "It didn't take too long to get an OpenVPN server running in pfSense for remote access. Also, for some reason, Internet access feels noticeably faster. For instance, my 10 Ring security cameras [wow] now come up on my phone much faster in live view and recorded videos. Not sure why that is, but I'll take it." And for one thing, it is five times faster, the SG-1100, five times faster than the predecessor, SG-1000. So it's one thing to have gig ports. It's another thing for them to be saturatable. And this little SG-1100 is great for 150 bucks. It's able to do that.

And, finally, Andy Blak, B-L-A-K, who tweeted under the same name: "Hey, Steve. Recently you mentioned the Netgate pfSense products on Security Now!. I purchased one after the show for my home network, though I'm having trouble configuring it. Do you have any recommendations for who could assist with this? Netgate charges $900+ minimum for their support." He says: "I called them." He says: "Just looking for a more reasonable option. Many thanks. Andy."

And the answer is yes. The Google is your friend. The upside of pfSense is that, I mean, it can do anything. The downside is that it can do anything. It's way more comprehensive and therefore configurable and complete than any router. But with that comes some configuration overhead. The good news is pfSense is the software you need to focus on, and there is a ton of how-to stuff on the Internet. So just google any question you have: pfSense; OpenVPN, how to set up OpenVPN; pfSense, you know, how to brew my espresso stronger. Any question you could possibly ask, pfSense probably does it, and you'll find somebody who can help you.

So don't be shy. And, yikes, that is expensive support. In fact, I got a different email I didn't mention, someone who said that they couldn't get the firmware from Netgate. They wanted a support contract. And it's like, well, that's ridiculous. The pfSense firmware is freely available for the SG-1100 on the Internet. Once again, Google is your friend.

Okay. What is Adiantum? I guess the question is, how can crucially important full-disk encryption be offered on inexpensive commodity hardware which lacks any encryption acceleration? That's the question Google set themselves. Because they have Android, they want to offer full-disk encryption, but we know what that does if you run it on a platform without some assistance for AES. And, you know, think about this for a moment. Many of us now take for granted the fact that our devices at rest are fully

encrypted, meaning that the mass storage that our devices carry internally cannot simply be read out as a file system, either by good guys or bad guys who want to know what's in there.

I mean, again, I feel like we now just take that for granted on our mobile devices. But anybody running Android who doesn't have full disk encryption enabled has their file system storage as an unencrypted file system. And so anybody who gets a hold of it can just get it. I mean, it's just all there. So it's crucial that it be encrypted and decrypted on the fly. But as we've mentioned before, the reality for users of low-end Android smartphones who, initially wanting full security, activate their drive's whole disk encryption, only to suffer such a reduction in performance that they're effectively forced to back out and just use their systems without that kind of encryption.

The trouble is caused by our chosen, the industry's chosen best-of-class cipher, Rijndael, which was chosen, as we know, to be the AES, the Advanced Encryption Standard, which while it's wonderfully secure, comes at the significant cost of per-byte encryption performance. And that overhead, that performance hit is bad enough that, back in 2010, when Intel introduced their new core, their Intel core processor family with Westmere, that new silicon and all silicon since has incorporated six new instructions known as the AES-NI, which is for AES New Instructions. Because I guess all the cooler names were already taken. They specifically speed up the use of the AES cipher on Intel architectures.

And it's not that AES cannot be done without those instructions. It can, of course. But each of those instructions is specific to an aspect of what the AES cipher algorithm needs so that each is able to replace a great many more standard instructions to get the same job done much more quickly. The point being, yes, AES is super secure, but not very quickly without some help. And also, as we've discussed before, the reason encrypting the entire drive is so sensitive to the performance of its encipherment is that bulk enciphering must happen continuously on the fly for every byte of data read and written to the drive. That cipher is stuck in-line and cannot be bypassed.

So happily, the bright lads at Google decided to take aim at this problem to see how they might improve upon the situation. Since it's not simply to speed up the Rijndael - it's not possible to simply speed up the Rijndael cipher without hardware, they knew that they would need to replace Rijndael with something else. This is feasible in any closed system where the cipher is only used for internal purposes. For example, when transacting over TLS, before the ChaCha20 cipher got added to TLS, which has happened recently, there was no choice other than to use AES, since its use there is for an external standard where you want to connect to a server remotely over a cipher that you both have. Maybe the other server doesn't have anything other than AES, so you've got to use it.

But internally, deep inside an Android device, Google is 100% free to choose anything that works. And what works is this cipher I mentioned, ChaCha. There's a chart, big chart in the show notes showing the performance of the bulk encryption mode known as XTS, which is what typically AES-256 cipher uses. It's the mode used for bulk-encrypting mass storage, versus Adiantum, which uses the ChaCha cipher, although in its reduced 12-round strength, rather than its normal 20-round strength because 12 is strong enough, and they're going for speed here. The more rounds you have, the slower it is.

Anyway, this bar chart shows about a five-factor performance improvement, which is in fact what Adiantum delivers with ChaCha versus AES-256 with equivalent security margin. And I smiled when I saw ChaCha because it is the brainchild of the same person who designed the 25519 elliptic curve cipher that is entirely responsible for enabling SQRL's solutions. That person is Daniel J. Bernstein, Dan Bernstein.

So Google's online security blog last Thursday, February 7th, was entitled "Introducing Adiantum: Encryption for the Next Billion Users." I guess they have high hopes for the

use of Android. But really, I mean, they've nailed it. And I can't say any better than they have. I'm going to quickly share what they wrote.

They said: "Today, Android offers storage encryption using the Advanced Encryption Standard. Most new Android devices have hardware support for AES via the ARMv8 Cryptography Extensions. However, Android runs on a wide range of devices. This includes not only the latest flagship and mid-range phones, but also entry-level Android Go phones sold primarily in developing countries, along with smart watches and TVs. In order to offer low-cost options, device manufacturers sometimes use low-end processors such as the ARM Cortex-A7, which does not have hardware support for AES. On these devices, AES is so slow that it would result in a poor user experience. Apps would take much longer to launch, and the device would generally feel much slower." And remember that speed and power consumption also go hand in hand. You'd really like to run the chip slower and cooler and thus use less battery, if you could.

"So while storage encryption," they write, "has been required for most devices since Android 6.0 in 2015, devices with poor AES performance," and they say 50 Mbps and below. And maybe that's megabytes. There's that MiB again. That's got to be megabytes [mebibyte] - "50 million bytes and below are exempt." So that is to say, to make sure everyone heard that, even with Android 6.0, which requires hardware encryption, the AES hit is so onerous that, if devices have lower performance, they are exempt from Google's own requirement.

They said: "We've been working to change this because we believe that encryption is for everyone. In HTTPS encryption," they say, "this is a solved problem. The ChaCha20 stream cipher is much faster than AES when hardware acceleration is unavailable, while also being extremely secure. It is fast because it exclusively relies on operations that all CPUs natively support: additions, rotations, and XORs. For this reason, in 2014 Google selected ChaCha20 along with the Poly1305 authenticator, which is also fast in software, for a new TLS cipher suite to secure HTTPS Internet connections. ChaCha20-Poly1305 has been standardized as RFC 7539, and it greatly improves HTTPS performance on devices that lack AES instructions."

So just to pause for a second, that means that light bulbs and low-end IoT devices that are performance challenged, as long as the server that they're connecting to offers this ChaCha20-Poly1305 mode of encipherment, which is now part of the TLS cipher suite spec, will be able to negotiate that with the server and end up with a much higher performance, yet still equally secure protocol for moving bulk data across an encrypted connection. So that's all for the good.

Google said: "However, disk and file encryption present a special challenge." Oh, note that we're talking about a stream cipher in ChaCha20. ChaCha20 generates, given a key, generates a stream of pseudorandom bits which then is XORed with the plaintext to create the ciphertext. As opposed to AES, which is a block cipher, which produces blocks of eight bytes, thus 128 bits at a time - wait, blocks of 16 bytes, thus 128-bit blocks, which are then used in various modes to encrypt the plaintext into ciphertext, typically CBC or some other block chaining mode, to create an inter-block dependency. We've talked about all this in podcasts past. But ChaCha20 is a stream cipher. Anyway, I wanted to make sure people understood the distinction.

They said: "However, disk and file encryption present a special challenge. Data on storage devices is organized into sectors which today are typically 4096 bytes. When the filesystem makes a request" - and actually they're noting, they're kind of mixing sectors and clusters. You know, clusters are typically clusters of sectors. Sectors still tend to be logically 512 bytes. But we now have clusters of eight bytes. So, yes, clusters of eight sectors, which end up with allocation units of 4096 bytes. So Google says: "When the file system makes a request to the device to read or write a" - they said "sector," but they

really meant a cluster - "the encryption layer intercepts that request and converts between plaintext and ciphertext. This means that we must convert between a 4096-byte plaintext and a 4096-byte ciphertext. But to use RFC 7539" - that's the streaming cipher used for TLS - "the ciphertext must be slightly larger than the plaintext. A little space is needed for the cryptographic nonce and for the message integrity information."

In other words, and we've talked about this all before, in these normal block chaining modes you have an initialization vector, a nonce that is at the front end, doesn't need to be secret, but does need to be unique. And then you also have an authentication tag added to the end which is how you verify that nothing has changed in the stream. Both of those things slightly increase the length of what you're transmitting. But of course, since it's a communication stream, that doesn't matter. It's a tiny little bit added in front and at the end. No one cares. But you can't think about it. If it's a hard drive, they're like fixed chunks of space. There's nowhere to stick extra stuff. They're blocks.

And so you need to take, and this is the point Google is making, need to take a 4096 block and go [sound], that's the technical term, and turn it into a 4096 ciphertext and vice versa. You can't ask for a little bit extra space in the beginning and the end. There's none because you've got the block ahead and the block behind. So Google says: "There are software techniques for finding places to store this extra information" - that would be a mess. They're talking about adding metadata to the filesystem. But they said: "But they reduce efficiency and can impose significant complexity on a filesystem design.

"Where AES is used, the conventional solution for disk encryption is to use the XTS or CBC-ESSIV modes of operation, which are length-preserving. Currently, Android supports AES-128-CBC-ESSIV for full-disk encryption and AES-256-XTS for file-based encryption. However, when AES performance is insufficient, there is no widely accepted alternative that has sufficient performance on lower end ARM processors.

"To solve this problem, we have designed a new encryption mode called Adiantum, A-D-I-A-N-T-U-M. Adiantum allows us to use the ChaCha stream cipher" - which is very fast on commodity hardware without acceleration - "in a length-preserving mode, by adapting ideas from AES-based proposals for length-preserving encryption such as HCTR and HCH. On ARM Cortex-A7, Adiantum encryption and decryption on 4096-byte sectors is about 10.6 cycles per byte, making it around 5X faster than AES-256-XTS."

They said: "Unlike modes such as XTS or CBC-ESSIV, Adiantum is a true wide-block mode. Changing any bit anywhere in the plaintext will unrecognizably change all of the ciphertext, and vice versa. It works by first hashing almost the entire plaintext using a keyed hash based on Poly1305 and another very fast keyed hashing function called NH. We also hash a value called the 'tweak' which is used to ensure that different sectors are encrypted differently."

Well, in fact the tweak in practice is typically the cluster number. It's called a "tweakable block cipher," and each block, the number of the block, the offset of the block in the filesystem is the tweak factor. Which means that, if you were to encrypt the same data in two different places on the drive, it would look completely different and no one could tell it was actually the same. You want to prevent even that slight leakage of information. So this is known as a tweakable block cipher, where the tweak adds - you can think of it as like another key to the keyed cipher.

They say: "This hash is then used to generate a nonce for the ChaCha encryption. After encryption we hash again so that we have the same strength in the decryption direction as the encryption direction. This is arranged in a configuration known as a Feistel network, so that we can decrypt what we've encrypted. A single AES-256 invocation on a 16-byte block is also required, but for 4096 inputs this is not performance critical."

So in the show notes I have a picture of the algorithm showing how it all works. I won't go into it because that was a description of it just now. And they note that: "Cryptographic primitives like ChaCha are organized in 'rounds,' with each round increasing our confidence in security at a cost in speed. To make disk encryption fast enough on the widest range of devices, we've opted to use the 12-round variant of ChaCha rather than the more widely used 20-round variant." For example, the communications protocol, NTLS uses ChaCha20.

"Each round vastly increases the difficulty of attack. The seven-round variant of ChaCha was broken in 2008 and, though many papers have improved upon this attack, no attack on eight rounds is known today." And we've got eight, nine, 10, 11, 12. So a five-round pad with each round, as they said, vastly increasing the difficulty. This ratio of rounds used to rounds broken today is actually better for ChaCha12 than it is currently for AES-256. So higher security margins than AES right now.

They said: "Even though Adiantum is very new, we are in a position to have high confidence in its security. In our paper, we prove that it has good security properties, under the assumption that ChaCha12 and AES-256 are secure. This is standard practice in cryptography. From 'primitives' like ChaCha and AES, we build 'constructions' like XTS, GCM, or Adiantum. Very often we can offer strong arguments, but not a proof that the primitives are secure; while we can prove that, if the primitives are secure, the constructions we build from them must be, too. We don't have to make that assumption about NH or Poly1305 hashes. These are proven to have the cryptographic property that they rely on.

And, finally, Adiantum, as I was mentioning I think before we began recording, Leo, is named after the genus of the maidenhair fern, which in the Victorian language of flowers, floriography, represents sincerity and discretion.

Leo: They're quite beautiful, actually.

Steve: And Adiantum is a fern which water runs off of, interestingly.

Leo: I think they wanted to have it be adamantium, but they didn't know how to spell it. Which is the metal in Wolverine's fingers.

Steve: Right, right.

Leo: That would have been better.

Steve: It would have been, yeah. Maybe it was taken. I mean, like maybe it's...

Leo: Well, it's also trademarked, probably, so maybe they couldn't get adamantium.

Steve: Yeah. And maybe they were trying to be close, but not, you know - no, we were talking about the fern.

Leo: It's barely pronounceable, to be honest with you. It's not...

Steve: I know. You have to really struggle to, you know, I was rehearsing before the podcast, Adiantum, Adiantum.

Leo: Adiantum. Adiantum. Well, there you go.

Steve: Yes. Yup. So we have a new, a very cool new high performance, low processor requirement cipher mode, appropriate for block ciphers. And I take my hat off to Google. That's going to be super useful.

Leo: They solved a problem they had, and quite elegantly.

Steve: Yup, yup.

Leo: Friends, we've come to the end of this fine show, but not the end of the conversation.

Steve: No.

Leo: You can go right to GRC.com, that's Steve's website. Not only get copies of the show and transcripts, you can get SpinRite, the world's best hard drive maintenance and recovery utility. You can join the SQRL groups and participate there. There's lots of other great free stuff there. GRC.com. We have audio and video of the show at our site, too, of course: TWiT.tv/sn. We do Security Now! on Tuesdays, 1:30 Pacific, 4:30 Eastern, 21:30 UTC. If you want to watch it, just go to TWiT.tv/live. You can watch or listen live. And if you do that, join us in the chatroom at irc.twit.tv. That's where the other people watching live hang out.

But as usual, on-demand versions are the best way to do it. And again, TWiT.tv/sn. You can listen. You don't have to miss your jazz radio show or anything like that. You can listen at your leisure, at your convenience, and even at high speed, as some people do. I don't blame them.

Steve: Okay.

Leo: All of our shows sound - it's funny. I had somebody come in yesterday or Sunday who was listening to TWiT. He said, "You sound so weird because I listen at double speed almost all the time."

Steve: Ah.

Leo: "You sound a little drunk." This is what we normally sound like. Don't forget our TWiT survey, TWiT.to/survey19. We do this every year to get a better idea of what you're interested in, who you are. It helps us sell advertising. We are not collecting personal information of any kind. Don't worry. But it just gives us a general idea of the audience. So TWiT.to/survey19. Thanks, Steve. Have a great week.

Steve: Thank you, my friend. Talk to you next week for Episode 702.

Leo: [Whistling] Bye.

Steve: Bye.