The Mystery of CVE-2023-38606

Description: After everyone is updated with the state of my still-continuing work on SpinRite 6.1, and after I've shared a bit of feedback from our listeners, the entire balance of this first podcast of 2024 will be invested in the close and careful examination of the technical details surrounding something that has never before been found in Apple's custom proprietary silicon. As we will all see and understand by the time we're finished here today, it is something that can only be characterized as a deliberately designed, implemented, and protected backdoor that was intended to be, and was, let loose and present in the wild. After we all understand what Apple has done through five successive generations of their silicon, today's podcast ends, as it must, by posing a single one-word question: Why?

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-955.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-955-lq.mp3

SHOW TEASE: It's time for Security Now!. Steve Gibson is here, and we have perhaps the most interesting story. What a way to kick off 2024, our first episode of the new year. And Steve breaks down what looks to be a massive conspiracy around the security of the last five generations of iPhone. The details coming up on Security Now!.

Leo Laporte: This is Security Now! with Steve Gibson, Episode 955 for Tuesday, January 2nd, 2024: The Mystery of CVE-2023-38606.

It's time for Security Now!, the show where we cover your privacy, your security, your safety online, how computers work, and a whole lot more with this guy right here, Steve Gibson, the autodidact who knows all, tells all, and is back again for 2024. Happy New Year, Steve.

Steve Gibson: Happy New Year to you, Leo. And I was pleased to see that we've got continuing sponsors in 2024 that'll be familiar to our audience.

Leo: Yes, yes. Your show will never be hard to sell. People want to be on Security Now!, that's for sure.

Steve: Okay.

Leo: But by the way, we're always looking for new advertisers. I don't want to discourage anybody.

Steve: No.

Leo: We always want - if you work for a company, and you say, boy, they really - I hear this a lot - they really ought to advertise on Security Now!, by all means contact us, advertising@twit.tv.

Steve: I can definitively state, Leo, that Apple will not choose to be an advertiser on this podcast.

Leo: Well, I can think of many reasons for that. But what do you mean?

Steve: Well, we know they have a very thin skin; right? Like, you know, you haven't been allowed back after you said one comment once, or looked at your phone during a presentation or some nonsense. I don't remember.

Leo: Yeah, no, I'm on a blacklist for sure, yeah.

Steve: Okay. This podcast has a very dry title, "The Mystery of CVE-2023-38606.

Leo: Hmm, whatever could that mean?

Steve: Uh-huh. After everyone is updated with the state of my still-continuing work on SpinRite 6.1, and after I've shared a bit of feedback from our listeners, the entire balance of this first podcast of 2024 will be invested in the close and careful examination of the technical details surrounding something that has never before been found in Apple's custom proprietary silicon. As we will all see and understand by the time we're finished here today, it is something that can only be characterized as a deliberately designed, implemented, and protected backdoor that was intended to be, and was, let loose and present in the wild. After we all understand what Apple has done through five successive generations of their silicon, today's podcast ends, as it must, by posing a single one-word question: Why?

Leo: Well, well. I think you're probably right, but I can't wait to - we talked a little bit about it on MacBreak Weekly. And I even said, "I bet you Steve will have something to say about this." So I'm very, very interested in the deep details, the breakdown.

Steve: We've got the details. Everybody is going to understand it. We're left with some questions. Not just one, like how did this happen? Because there are things unfortunately we're never going to know. But there are definitely things we do know. And it really takes a little bit of the polish off the Apple.

Leo: Oh, my.

Steve: Yeah.

Leo: Oh, my. Well, I'll be very interested to hear more about this. All right.

Steve: And we do have, oh, my goodness, a great Picture of the Week. I gave it a caption that really...

Leo: I actually like the caption, "Can you say the word 'ad hoc' in a sentence." I don't know what it means, but we will get to the Picture of the Week. All right. Picture of the Week.

Steve: So this picture was of an insane person, something that a crazy person did. Apparently they were traveling somewhere in Europe, where the plugs are two round pins, offset from the center.

Leo: Yeah.

Steve: But they had a U.S. standard two-prong, straight-prong power plug. And they couldn't, I guess, go downstairs to the hotel gift shop and buy themselves an adapter. So they managed to put one together.

Leo: Oh, lord. This is insane. Oh, my god.

Steve: So I looked at this, and I...

Leo: Would this work?

Steve: Oh, yeah. I gave this - although, you know, you want to keep your distance. I gave this the caption "Please use the term 'ad hoc' in a sentence." And the sentence is "This is a photo of an ad hoc EU to US power adapter."

Leo: It looks like he's got forceps and a key and a padlock.

Steve: There's forceps stuck in one of the round European outlet holes. And then on the U.S. plug the two prongs have holes through them, as many of ours do, so the plug was - one of the plug's holes was threaded through the handle of the forceps, giving it an electrical attachment to one side of the European power.

Leo: It's kind of clever because it's taking advantage of the hole in the U.S. plug.

Steve: Yes. That was clever. Well, then, so does the padlock, which has been stuck, the hasp of the padlock has been stuck through the hole in the other prong of the U.S. plug, the padlock then closed. And of course it's got a key, linking it with a key ring to another key, and that second key is plugged into the opposing hole in the adjacent EU power outlet, thus completing the circuit.

Leo: I mean...

Steve: If you use the term "circuit" very loosely.

Leo: I'd be a little concerned about the impedance of the key ring.

Steve: Yeah. Yes. You're right, that little key ring loop, if you're drawing too much power from the circuit, it's going to get...

Leo: It's going to start to glow.

Steve: It's going to get a little warm. Oh, god. Anyway...

Leo: This is hysterical.

Steve: A great photo.

Leo: Don't ever try this at home, kids. This is not recommended by any means.

Steve: Yes. We're not suggesting this. We're just saying, if you are really desperate...

Leo: No, no, no, no. Nobody's that desperate.

Steve: Then even don't do it.

Leo: No, even then, no.

Steve: Okay. So as I mentioned two weeks ago, I was hoping to be able to announce that SpinRite 6.1 was finished. And really it is all but finished. If I were not still making some incremental progress that matters, I would have declared it finished. I've been wrestling with the behavior of some really troubled drives, like finally someone Fedexed me - I paid the FedEx shipping because there was no way I was going to have someone who's been testing SpinRite pay shipping. He Fedexed me his drive Friday that arrived three days ago, on Saturday, because SpinRite was getting stuck at 99.9953% of the drive. Well, stuck is no good.

Turns out it was - that drive, it was a Western Digital, was returning a bogus status which was tripping up SpinRite, which wasn't prepared for like a ridiculous status byte coming back from the drive. So SpinRite's better now, and three other people's drives that were also getting stuck for the same reason no longer do. So it's like the edgiest of edge cases. But I want this thing to be perfect. And the things I'm doing will carry forward into the future. So I would have to do it sooner or later. I might as well do it now.

Anyway, I need to give it a little more time to settle. But what I know is that, as a consequence of this intense scrutiny at the finish line, this SpinRite really does much more than any version of SpinRite ever has before. So anyway, I'm just days away.

Okay. So I have a couple bits of Closing the Loop feedback. This is interesting. This is from Rotterdam in the Netherlands. The subject was "Which root certificates should you trust?" Remy wrote: "I made the tool requested in 951." He wrote: "Hi, Steve. Happy SpinRite customer here. Can't wait for 6.1. In Episode 951 you discussed root certificates, and Leo asked for a 'Prune your CA' app. You said: 'The good news is that this is generic enough that somebody will do it by the next podcast.'" He said: "Well, fun challenge, and I happen to know a bit of C++, so here is my attempt." And we have a link in the show notes. It is also this week's Shortcut of the Week.

Anyway, he wrote: "It parses the Chrome or Firefox browser history and fetches all certificates. It's a reporting-only tool that does not prune your root store, since I suspect," he says, "that could break stuff. So if people want that, they can figure out how to do that themselves and make a backup." He says: "I've compiled it as a 32-bit Qt 5.15 C++ application, specifically for you, so it runs on Windows 7."

Leo: You've got a reputation, Steve.

Steve: "And up." He says: "Qt 6, the current version of the C++ framework, does not support Windows 7. Cheers, Remy." So anyway, since I found this as I was pulling together today's podcast, I've not yet had the chance to examine it closely. But I quickly went over to Remy's page, and it looks like he has done a very nice piece of work.

Leo: Yeah. It's very cool, yeah.

Steve: Yeah. And I don't know if you noticed, but there are counts on the number of instances of each of those certificates that he found. And he's also supporting by, I mean, he's sorting by the most hits on a given cert. And not surprisingly, Let's Encrypt is up there at the top.

Leo: Right.

Steve: Anyway, so he provides an installer for Windows users with the full source code. So for Linux users he suggests installing QT and compiling his what he calls "Certutil" from the provided source code. He's deliberately removed all Google ads, Google and other tracking from his site. So he suggests that, if you find his work useful, you might perhaps buy him a cup of coffee. The link to Remy's page, as I said, is this week's GRC shortcut, so grc.sc/955. And that'll just bounce you directly to Remy's page. And so Remy, thanks for this.

Leo: This is read-only, and I understand his reluctance to put a surgical tool in there. But how hard, if one's seeing these, how hard would it be to remove them? Is there a mechanism for that manually?

Steve: Yes. Well, under Windows, you down in the search box put MMC and hit ENTER. That brings up their desktop. MMC ENTER. And there it is. And so that brings up - and yes, you want to run it. Yup. And so then you go to File up in the File Menu and add snap-in. Add/remove snap-in. Now find Cert and add that. You have to move it into the box on the right.

Leo: Oh, you can't just click it, you have to drag it.

Steve: Yeah.

Leo: Okay.

Steve: Add/remove snap-in.

Leo: So there's the certificates snap-in.

Steve: And then click on that little thing that...

Leo: Oh, I see, add it here, I get it. Oh, yeah. I haven't used this in a while. All right.

Steve: And my account.

Leo: There it is. Now certs are in there. And now I can see my certificates.

Steve: Now you've created a certificate viewer which allows you to see all the certs you've got on your machine.

Leo: And the one you care about is Trusted Root; right?

Steve: Yes. And if you click there...

Leo: Oh, there you go.

Steve: ...there they all are.

Leo: Now, let's say I don't want Comodo. Can I right-click on it and delete? Yeah.

Steve: Yup.

Leo: So you totally can do that manually, one by one.

Steve: You absolutely can.

Leo: Okay.

Steve: Yup. And we'll see how this goes because this also could be automated. All the certs are in the registry. This is basically a...

Leo: It's a regedit, yeah.

Steve: ...convenient viewer into the registry that does a lot of interpretation for you, too.

Leo: This is good. I mean, honestly, this is the tool that's kind of sufficient. It's just that you have to do it onesie-twosie.

Steve: Well, and you've got to know which ones. So his certutil shows you...

Leo: That's useful to report. Yeah, yeah, yeah.

Steve: ...the entire, given your entire current browser history, what certificates has your browser ever touched?

Leo: And what hasn't it touched because there's a whole listing of unused CAs.

Steve: Exactly.

Leo: Yeah. And those are the ones you might consider removing.

Steve: Yeah. Actalis. Who's ever heard of IT Milan Actalis?

Leo: It's Italian. It's an Italian...

Steve: I don't think we need them.

Leo: Yeah, we don't need them.

Steve: You know, we don't - and of course what this means is anything they ever sign our browser absolutely trusts. You know, that's the great danger of this whole system.

Leo: Right.

Steve: Is that it's carte blanche trust for every single certificate in our root.

Leo: And as you pointed out when we first started talking about this, there's only a handful of certs that do 90% of the traffic into your computer.

Steve: Seven certs gives you 99.96% of all non-expired browser certificates.

Leo: Wow. Okay. I could probably remove the China Financial Certification Authority from my EV roots.

Steve: I think so.

Leo: I have a feeling I won't be needing that.

Steve: Yeah. Now your light switch might not work anymore. But still.

Leo: So that's the question. What does the - you would get - it would tell you this is an untrusted site.

Steve: Correct.

Leo: But you still could go to it. You could bypass it.

Steve: Absolutely. You could push past it. And, you know, maybe that's what you want is something that says whoa, wait a minute, why am I going to the Chinese laundry? I don't really need that.

Leo: Yeah, I don't need that one, yeah. Yeah, okay. Very handy. Thank you. Good job, Remy. Yeah.

Steve: Thank you. Martin Bergek was referring to last week's Schrodinger's Bowls Picture of the Week, or rather two weeks ago. This was great. He said: "Showed the Picture of the Week to my son, age 13. He immediately gave the obvious answer: 'Break the top glass. Obviously that is cheaper to replace than the plates.'"

Leo: Ah. Very clever.

Steve: And he said - yes. He said: "Slightly annoyed I didn't come up with it myself." Well, none of us did, Martin. "But happy that the future is looking bright."

Leo: That's great.

Steve: So anyway, yeah. The upper glass was in front of an empty shelf. So you could break the glass, then lift the shelf and have full access to all the bowls that are precariously trying to fall but are being held in place by the pane of glass below that. So very, very clever.

@sammysammy222 tweeted @SGgrc: "Steve, I planned to use DNS Benchmark today. When I ran the download by VirusTotal, three security vendors flagged this file as malicious. While I trust you, I wanted to check in, in case there is a problem with the file. Thanks."

Okay. So I just dropped the file on VirusTotal, and at the moment I did, four of the AV tools at VirusTotal dislike it. Now, I'll note, however, that they're all rather very obscure AV, so it's not Google or Microsoft or anybody. And since GRC's DNS Benchmark is approaching 8.8 million downloads at a rate of 4,000 per day, I suppose it's a good thing that not everyone is as cautious as sammysammy222 or there would be a lot more concern than is being expressed. But seriously, I poked around a bit in VirusTotal because it, like, allows you to see what things are setting off alarm bells. And I looked at the behavior that is concerning to VirusTotal, and I can't really say that I blame it. The program's behavior, if you didn't know any better, would peg it as some sort of bizarre DNS DOS zombie. You know, it scans network adapters. It's chock full of every DNS server you can imagine. There's 505 of them or something.

Leo: It does look kind of suspicious, come to think of it, yes.

Steve: Yeah.

Leo: Good point.

Steve: And, you know, it's got all sorts of mysterious networking code. Unfortunately, this is the world we're living in today. It doesn't matter that that program which has been around since, what, 2007 or something I think I wrote it? You know, it's never hurt a single living soul, and it never would. You know, it's even triple SHA-256 signed, with various of GRC's old and new certificates, all of them valid. Do the signatures matter? Apparently not. It still looks terrifying.

So anyway, to answer your question, SammySammy, the file's code has not changed in nearly five years, since I remember I tweaked it, I think to add 1.1.1.1 or something, you know, or 9.9.9.9, some updated server I wanted to put in there, and that was five years ago. It's never actually harmed anyone. Yet I can understand why VirusTotal would go ugh, you know. So yeah, anyway, I'm pretty sure you can rely on it being safe.

Christian Nilsson said: "Hello, Steve. I'm one of all those who is so glad you decided to go past 3E7 hex." So we know that's 999 in decimal. He said: "Thank you. Speaking of Tailscale, et cetera, I strongly recommend Nebula Mesh. Open source all the way, and very easy to manage. I know it's been on your radar before. What's the reason for not advocating it in SN-953?"

Christian, no reason at all. I just forgot to mention it among all the others. And I agree 100% that Nebula Mesh, which was created by the guys at Slack, is a very nice-looking overlay network. We did talk about it before. Slack wrote it for their own use because nothing else that existed at the time, this was back now five years ago in 2019, did what they needed out of the box. So they rolled their own.

Nebula Mesh's home on GitHub is under SlackHQ's directory tree, and I have the link in the show notes. But you can just search "nebula mesh" to locate it. It's a fully mutually authenticating peer-to-peer overlay network that can scale from just a few to tens of thousands of endpoints. It's portable, written in GO, and has clients for Linux, MacOS, Windows, iOS, Android, and even my favorite Unix, FreeBSD. So absolutely thank you for the reminder.

And for those who are looking for a good, 100% open, unlike Tailscale, where they are keeping a couple of the clients closed source and the rendezvous server, which those nodes need to find each other, closed, Nebula's 100% open. So again, Christian, thanks for the reminder.

Ethan Stone said: "Hi, Steve. I've noticed something recently that you might be interested in. First on my Windows 10 machine, then on my Windows 11 machine, the Edge browser has kept itself running in the background after I close it. The only way to actually shut it down is to go into Task Manager and manually stop the many processes that keep running after it's closed. I noticed this because, as one of your most paranoid listeners" - he says, parens, "(I know that's a high bar) - "I have my browsers set to delete all history and cookies when they shut down." Boy, he must like logging into sites. "And I have CCleaner doing the same thing if they don't. CCleaner seems to have caught on to Microsoft's little scheme and now notes that Edge is refusing to shut down and asking me if I want to force it. Anyway, it seems like it's just another little scheme to keep all of my data and activity accessible to their grubby little data-selling schemes."

And actually I think it's probably just, you know, who knows what. They figure just get more RAM if you don't want Edge continually running. Actually what it probably is, is it allows it to launch onto your screen much more quickly when you start it again; right? Because it never really stopped. So it's already, it's like still there. It just turned off the UI. So you click on the icon and, like, bang, it's like whoa, okay. So anyway, I understand you want things that you don't need and that you stop to actually stop. That's a good thing.

He finally says: "I love the show, and I'm looking forward to new ones in 2024 and the advent of SpinRite 6.1, although I really, really, really need native USB support. So I hope SpinRite 7 isn't far behind. Also, I'm looking forward to not having to log into Twitter to send you messages, although please give priority to SpinRite 7."

Okay. So I have been gratified with the feedback from Twitter users that they're looking forward to abandoning Twitter to send these short feedback tidbits. My plan is to first get SpinRite 6.1 finished. As I said, that's just like moments away. Okay. But then I need an email system, which I don't currently have, in order to notify all v6.0 owners through the last 20, what, 21 years - 20 years because it was in 2004 I began offering 6.0. So I need, you know, an email system to let them know that it's available and basically of what has turned out to be a massive free upgrade to 6.0. So that need will create GRC's email system, and we'll experiment then with my idea for preventing all incoming spam without using any sort of heuristic false-positive tendency filter.

And then I plan to immediately begin working on SpinRite 7. I don't want to wait for several reasons. For one thing, I currently AM SpinRite. I have the entire thing in my head right now. My dreaming at night is about SpinRite. And I know from prior experience that I will lose it if I don't use it. You know, I barely remember how SQRL

works now. So I want to immediately dump what's in my head into the next iteration of SpinRite code. Another reason that Ethan referred to is about USB. Unfortunately, USB is still SpinRite 6.1's Achilles heel. All of SpinRite has historically run through the BIOS, but I've become so spoiled now by 6.1's direct access to drive hardware that the BIOS now feels like a real compromise.

Yet even under 6.1, all USB-connected devices are still being accessed through the BIOS. Since SpinRite's next platform, which I've talked about, RTOS-32, already has USB support, although not the level of support required for data recovery - that's what I'll be adding - still, it made no sense for me to spend more time developing USB drivers from scratch for DOS when at least something I can start with is waiting for me right now under RTOS-32. So yes, I'm a hurry to get going on SpinRite 7. It'll be happening immediately.

Oh, and then I had already replied to Ethan. And when I checked back into Twitter, he had replied to me. And he wrote again, "Okay, but please spend less time on the email thing than you did on SQRL." And so yes, I am not rolling my own email system from scratch this time. I've already purchased a very nice PHP-based system that I've just been waiting to deploy. So I'll be making some customizations to part of it, but that's it, and then we'll have email.

Okay, Leo. I'm going to catch my breath, you tell us why we're here, and then, oh boy, we're going to...

Leo: So the whole show's going to be this Apple thing?

Steve: Yes.

Leo: Holy cow. There is a lot to talk about.

Steve: Some other things happened, but they pale in comparison.

Leo: Yeah.

Steve: To what we now absolutely know that Apple has done.

Leo: Well, and they said in the information that it was ARM devices, other ARM devices might also be susceptible.

Steve: All iDevices.

Leo: All iDevices. But ARM makes stuff, like my Pixel phone, no, not my Pixel, my Samsung phone is ARM based. So that, well, we'll find out.

Steve: Yup.

Leo: We'll find out. And as far as I know, those CVEs have been patched on Apple devices. But we'll find out about that, as well.

Steve: Well, we're going to find out why this was here for five years.

Leo: Yeah. Whoof. Wow. I used your name in vain on MacBreak Weekly. I said if there's one thing I've learned from listening to Steve for almost 20 years is interpreters are a problem.

Steve: I heard you, yes.

Leo: Yeah, yeah.

Steve: And you're absolutely correct.

Leo: We know this. Okay. Let's get the CVEs here.

Steve: I have in the past suggested that you buckle up. And this is no exception.

Leo: I don't have a seatbelt on this bicycle seat that's attached to a stick.

Steve: Oh, that's right. You're not on the ball anymore. That's good. That's good.

Leo: I will plant my feet firmly on terra firma.

Steve: Plant your feet, yeah, firmly apart. Okay. Our long-time listeners may recall that during the last year or so we mentioned an infamous and long-running campaign known as "Operation Triangulation," which is an attack against the security of Apple's iOS-based products. This breed of malware gets its name from the fact that it employs canvas fingerprinting to obtain clues about the user's operating environment. It uses WebGL, the Web Graphics Library, to draw a specific yellow triangle against a pink background. It then inspects the exact high-depth colors of specific pixels that were drawn because it turns out that different rounding errors used in differing environments will cause the precise colors to vary ever so slightly. We can't see the difference, but code can. And since it uses a triangle, this long-running campaign has been named "Operation Triangulation."

Okay. In any event, the last time we talked about Operation Triangulation was in connection with Kaspersky - Kaspersky. I heard you putting the accent on the second syllable, and I went and checked, Leo, and you're exactly right. I have been always saying Kaspersky, but it's Kaspersky.

Leo: I just put myself in kind of the mind of Boris Badenov.

Steve: The Russian, the Russian...

Leo: And I say, how would Boris Badenov pronounce it? Kaspersky. And then that's how I say it, yeah.

Steve: That's right. That's right. Well, you got it right. Okay. And we talked about it with Kaspersky Labs because someone had used it, that is, this Operation Triangulation attack, to attack a number of the iPhones used by Kaspersky's security researchers. In hindsight, that was probably a dumb thing to do since nothing is going to motivate security researchers more than for them to find something unknown crawling around on their own devices. Recall that it was when I found that my own PC was phoning home to some unknown server that I dug down, discovered the Aureate Spyware, as far as I know coining that term in the process since these were the very early days, and then I wrote "OptOut," the world's first spyware removal tool. My point is, if you want your malware to remain secret and thus useful in the long-term, you need to be careful about whose devices you infect.

Although we haven't talked about Kaspersky and the infection of their devices for some time, it turns out they never stopped working to get to the bottom of what was going on, and they finally have. What they found is somewhat astonishing. And even though it leaves us with some very troubling and annoying unanswered questions, which conspiracy theorists are already having a field day with, what has been learned needs to be shared and understood because, thanks to Kaspersky's dogged research, we now know everything about the "What," even if "How" and "Why" will forever probably remain unknown. And there's even the chance that parts of this will forever remain unknown to Apple themselves. We just don't know.

Kaspersky's researchers affirmatively and without question found a deliberately concealed, never documented, deliberately locked but unlockable with a secret hash, hardware backdoor which was designed into all Apple devices starting with the A12 chip, the A13, the A14, the A15, and the A16.

This now publicly known backdoor has been given the CVE which is today's podcast title, thus CVE-2023-38606, though it's really not clear to me that it should be a CVE since it's not a bug. It's a deliberately designed-in and protected feature. Regardless, if we call it a zero-day, then it's one of four zero-days which, when used together in a sophisticated attack chain, along with three other zero-days, is being described as the most sophisticated attack ever discovered against Apple devices, and that's a characterization I would concur with.

Okay. So let's back up a bit and look at what we know thanks to Kaspersky's work. I can't wait to tell everyone about 38606, but to understand its place in the overall attack we need to put it into context. The world at large first learned of all this just last Wednesday, on December 27th, when a team from Kaspersky presented and detailed their findings during the 37th four-day Chaos Communication Congress held at the Congress Center in Hamburg, Germany. The title they gave their presentation was "Operation Triangulation: What You Get When Attack iPhones of Researchers," which, I think, is a perfect title because yeah. On the same day last Wednesday they also posted a non-presentation description of their research on their own blog, titled "Operation Triangulation: The Last Hardware Mystery."

I've edited their lengthy posting for the podcast, and I'm not going to go through all of it. But I wanted to retain the spirit of their disclosure since I think they got it all just right, and they did not venture into conspiracies. So after some editing for clarity and length, here's what they explained. They said: "Today, on December 27th, 2023, we delivered a

presentation titled "Operation Triangulation: What You Get When Attack iPhones of Researchers" at the 37th Chaos Communication Congress held at the Congress Center, Hamburg. The presentation summarized the results of our long-term" - it is multiyear - "research into Operation Triangulation, conducted with our colleagues.

"This presentation was also the first time we had publicly disclosed the details of all exploits and vulnerabilities that were used in the attack. We discover and analyze new exploits and attacks such as these on a daily basis. We've discovered and reported more than 30 in-the-wild zero-days in Adobe, Apple, Google, and Microsoft products. But this is definitely the most sophisticated attack chain we have ever seen. Here's a quick rundown of this zero-click iMessage attack, which used four zero-days and was designed to work on iOS versions up to iOS 16.2."

Okay. So "Attackers send a malicious iMessage attachment, which the application processes without showing any signs to the user. This attachment exploits the remote code execution vulnerability" - and these are all CVE-2023, so I'm going to skip that. So it's "the remote code execution vulnerability 41990 in the undocumented, Apple-only ADJUST TrueType font instruction. This instruction had existed since the early '90s until a patch removed it.

"It uses return/jump-oriented programming and multiple stages written in the NSExpression/NSPredicate query language, patching the JavaScriptCore library environment to execute a privilege escalation exploit written in JavaScript. This JavaScript exploit is obfuscated to make it completely unreadable and to minimize its size. Still, it has around 11,000 lines of code, which are mainly dedicated to JavaScriptCore and kernel memory parsing and manipulation. It exploits the JavaScriptCore debugging feature DollarVM to gain the ability to manipulate JavaScriptCore's memory from the script and execute native API functions.

"It was designed to support both old and new iPhones and include a Pointer Authentication Code bypass for exploitation of recent models. It uses the integer overflow vulnerability 32434 in the OS's memory mapping syscalls to obtain read/write access to the entire physical memory of the device. It uses hardware memory-mapped I/O (MMIO) registers to bypass the Page Protection Layer known as PPL by Apple." This was mitigated as that CVE that is this podcast's title, 38606. A lot more on that in a minute.

"After exploiting all the vulnerabilities, the JavaScript exploit can do whatever it wants to the device running spyware. Which is to say it is at this point the iDevice, because this is in all the A12 through A16 chips, in all of Apple's devices. The device is cracked wide open. At this point they've achieved absolute dominance. So they say after exploiting all the vulnerabilities, the JavaScript exploit can do whatever it wants to the device, including running spyware. But the attackers chose to, A, launch the IMAgent process and inject a payload that clears the exploitation artifacts from the device; and B, run a Safari process in invisible mode and forward it to a web page with the next stage.

"The web page has a script that verifies the victim and, if the checks pass, receives the next stage, the Safari exploit. The Safari exploit uses 32435 to execute shellcode. The shellcode executes another kernel exploit in the form of a Mach object file. The shellcode reuses the previously used vulnerabilities 32434 and 38606. It is also massive in terms of size and functionality, but completely different from the kernel exploit written in JavaScript. Certain parts related to exploitation of the above-mentioned vulnerabilities are all that the two share.

"Still, most of its code is also dedicated to parsing and manipulation of the kernel memory. It contains various post-exploitation utilities, which are mostly unused. And finally, the exploit obtains root privileges and proceeds to execute other stages, which

load spyware. And those subsequent spyware stages have been the subject of previous postings of Kaspersky.

Okay. So now I'm going to summarize this, and then we're going to zero in. The view we have from 10,000 feet is of an extremely potent and powerful attack chain which, unbeknownst to any targeted iPhone user, arranges to load, in sequence, a pair of extremely powerful and flexible attack kits. The first of the kits works to immediately remove all artifacts of its own presence to erase any trace of what it is and how it got there. It also triggers the execution of the second extensive attack kit, which obtains root privileges on the device and then loads whatever subsequent spyware the attackers have selected. And all of that spyware has full rein of the device because that subsequently loaded spyware is under, you know, obtains the permission of these kits.

So it uses CVE 41990, a remote code execution vulnerability in the undocumented, Apple-only ADJUST TrueType font instruction. That's where it begins. With that still somewhat limited but useful capability, it uses "living off the land" return/jump oriented programming, meaning that since it cannot really bring much of its own code along with it at this early stage, instead it jumps to the ends of existing Apple-supplied code subroutines, threading that together to patch the JavaScriptCore library just enough to obtain privilege escalation.

Leo: So they call that "living off the land."

Steve: Right.

Leo: Because it's not using - you can't put much of your own code in there, so you have to use existing code.

Steve: Right. Right. And it just jumps to just before other subroutines occur.

Leo: So there might be a little memory there somewhere that you can...

Steve: Yeah. There are a few little instructions just before a subroutine finishes and returns. So all they can do is like jump to a sequence of the ends of existing code in subroutines as like little bits of worker code, threading that together to achieve what they want.

Leo: Now, to do that you have to know exactly where that stuff lives; right? You have to have the actual memory event. That's why address randomization works; right?

Steve: Yes.

Leo: Yeah.

Steve: Address space layout...

Leo: Apple doesn't do ASLR, though, I guess, with this code?

Steve: Well, remember that they said the bulk of that code is analyzing the kernel memory. So it's looking for all of these things in order to figure out where everything is.

Leo: Oh. So it may be moving around, but they look for a chunk, and they recognize it and say, oh, it'd be right after this, wherever that is.

Steve: Yup. Yup.

Leo: Holy cow. This is so sophisticated.

Steve: Or they find like an API jump table and then know that the fourth API in that table will be taking them to the subroutine, the end of which they need.

Leo: So that's why ASLR doesn't always work, even if you use it, because...

Steve: Right. Right. It definitely makes the task far more difficult. But here they're using it. And they've managed. So, I mean, it really raises the bar. But it can still be worked around.

Leo: Right.

Steve: So after that, they have obtained privilege escalation. By using the library's, the JavaScriptCore library's debugging features, then they're able to execute native Apple API calls with privilege.

Leo: Oh, my god.

Steve: It then uses another vulnerability, 32434, which is an overflow vulnerability in the OS's memory mapping system API, to obtain read/write access to the entire physical memory of the device.

Leo: Oh, my goodness.

Steve: Yeah, that's different than the normal access because normally you are constrained, like a program is constrained, to what you have access to. So they use an integer overflow, another zero-day to get read/write access to the entire physical memory, and read/write access to physical memory is required for the exploitation of the title of this podcast, CVE-38606, which is used to disable the entire system's write protection. Normally you can't change anything. You can't write anywhere. And so they figured out how to turn off write protection.

Leo: That's amazing. I mean, this is what the most sophisticated hacks often do is chaining vulnerabilities.

Steve: Yes.

Leo: And you slowly escalate your capabilities till you get what you want.

Steve: Yup.

Leo: Wow.

Steve: Okay. So, so far we've used three vulnerabilities: 41990, 32434, and 38606. The big deal is that by arranging to get 38606 to work, which requires read/write access to physical memory because that's how it's controlled, this exploit arranges to completely disable Apple's Page Protection Layer, which Apple calls PPL. It is what provides the great deal of the modern lockdown of iOS devices. Here's how Apple themselves describes PPL in their security documents.

They said: "Page Protection Layer (PPL) in iOS, iPadOS, and watchOS is designed to prevent user space code from being modified after code signature verification is complete. Building on Kernel Integrity Protection and Fast Permission Restrictions, PPL manages the page table permission overrides to make sure only the PPL can alter protected pages containing user code and page tables. The system provides a massive - this is Apple - a massive reduction in attack surface by supporting system-wide code integrity enforcement, even in the face of a compromised kernel. This protection is not offered in macOS because PPL is only applicable on systems where all executed code must be signed.

Leo: So what's interesting is this comes after system integrity is complete.

Steve: Right.

Leo: So you've verified. You go, okay, we've got to make sure everything's good. Everything's good.

Steve: Lock it down.

Leo: Lock it down. And you've gotten in after that.

Steve: Yeah. The key sentence here was: The system, PPL, the system provides a massive reduction in attack surface by supporting system-wide code integrity enforcement, even in the face of a compromised kernel. So that means that defeating the PPL protections results in a massive increase in attack surface.

Leo: Sure, because it's assumed you're good.

Steve: Right. Once those first three chained and cleverly deployed vulnerabilities have been leveraged, as Kaspersky puts it, "the JavaScript exploit can do whatever it wants on the device, including running spyware." In other words, the targeted iDevice has been torn wide open.

Okay, now, the next thing Kaspersky does is to take a closer look at the subject of their posting and of this podcast. But before we look at that I need to explain a bit about the concept of memory-mapped I/O. From the beginning, computers used separate input and output instructions to read and write to and from their peripheral devices, and read and write instructions, you know, read and write, to read from and write to and from their main memory. This idea of having separate instructions for communicating with peripheral devices versus loading and storing to and from memory seemed so intuitive that it probably outlasted its usefulness.

What this really meant was that I/O and memory occupied separate address spaces. Processors would place the address their code was interested in on the system's address bus. Then, if the IO_READ signal was strobed, that address would be taken as the address of some peripheral device's register. But if the MEMORY_READ signal was strobed, the same address bus would be used to address and read from main memory. To this day, largely for the sake of backward compatibility, Intel processors have separate input and output instructions that operate like this, but they are rarely used any longer.

Leo: As with a lot of stuff, it was for efficiency; right? So peripheral devices could just sit there, and then every once in a while instead of you - it was like a spooler to the printer. Instead of having to wait for the printer to ACK it and all that, you just stick it there, and the printer gets it when it wants it, basically.

Steve: Well, yes. And, for example, behind me we have my PDP-8 working boxes.

Leo: I'm sure they used it, yeah.

Steve: So when you only had 12 bits of address space, which is 4K, you don't want to give any of that up to peripheral I/O. You need every single byte to be actual memory. So the idea was that the peripheral devices occupied their own memory space, or I'm sorry, their own, not memory space, their own I/O space. I/O space was separate. It was completely disjoint from memory space. And you access the I/O space with I/O instructions, and memory space with standard read and write instructions.

Okay. So to be practical, one of the requirements is that the system, if you want to - okay. I jumped ahead. What happened was that someone along the way realized that, if a peripheral's device hardware registers were directly addressable just like regular memory, meaning in the system's main memory address space, right alongside the rest of actual memory, then the processor's design could be simplified by eliminating separate input and output instructions, and all of the already existing instructions for reading and writing to and from memory could perform double duty as I/O instructions.

Leo: Because it's all in the same spot now. It's all in the [crosstalk].

Steve: Right.

Leo: Yeah.

Steve: But to be practical, one of the requirements is that the system needs to have plenty of spare memory address bits. But even a 32-bit system, which can uniquely address 4.3 billion bytes of RAM, can spare a little bit of space for its I/O.

Leo: Probably not a lot of space; right? I mean...

Steve: Right. And that's exactly what transpired. So, for example, when today's SpinRite wishes to check on the status of one of the system's hard drives, it reads that drive's status from a memory address near the top of the system's 32-bit address space. Even though it's reading a memory address, the data that's read is actually coming from the drive's hardware. This is known as memory-mapped I/O because the system's I/O is mapped into the processor's memory address space, and discrete input and output instructions are no longer used. And for example, in the case of ARM, they don't even exist. You know, ARM has always been just direct memory-mapped I/O.

Leo: Interesting. So just by polling that address, you trigger a transfer from the I/O device.

Steve: Exactly.

Leo: You're not really reading RAM. You're reading the I/O device.

Steve: Exactly. And, you know, if you've got 64 bits of address [crosstalk] have anything in most of that space.

Leo: Yeah, right, right.

Steve: So today's massive systems and by "massive" I mean an iPhone because it has grown into a massive and complex system they use 64-bit chips with ridiculously large memory spaces. So all of the great many various functions of the system, you know, fingerprint reader, camera, screen, crypto stuff, enclave, all of that stuff are reflected in a huge and sparsely populated memory map. Okay, now, in the past we've talked about the limited size of the 32-bit IPv4 address space and how it's no longer possible to hide in IPv4 because it's just not big enough. The entire IPv4 address space is routinely being scanned.

This is not true for IPv6 with its massive 128-bit address space. Unlike IPv4, it's not possible to exhaustively scan IPv6 space. There's just too much of it. So here's my point. When there is a ton of unused space, it's possible to leave undocumented some of the functions of memory-mapped peripherals, and there's really no way for code to know whether anything might be listening at a given address or not. And not only that, there may be too many addresses to reasonably check.

Leo: There could be millions.

Steve: Billions. Trillions. Gazillions.

Leo: Yeah.

Steve: Yeah. I think...

Leo: Oh, this is interesting.

Steve: Yeah.

Leo: So this reminds me of the old peek and poke commands. Right?

Steve: Yeah.

Leo: On older machines like Commodores and Ataris, you'd peek an address, or you'd poke an address. And sometimes that would be an I/O, like if you wanted to write to the hard drive, you'd poke it to an address, and it wouldn't be a memory address, it would poke it to the hard drive. So that's still built in to modern processors. In fact, it sounds like more so because...

Steve: Yes.

Leo: There are many, many more devices attached.

Steve: It is now the way - yes. Or things, yes. And it is so convenient to just be able to use the existing architecture of reading and writing to and from actual memory to do the same with your peripherals.

Leo: It's a clever hack. And you know what that means.

Steve: What Kaspersky discovered was that Apple's hardware chips, from A12 through A16, all incorporated exactly this sort of hardware backdoor. And get this. This deliberately designed-in backdoor even incorporates a secret hash. In order to use it, the software must run a custom, not very secure but still secret, I mean, it's not crypto quality, but it doesn't need to be, hash function to essentially sign the request that it's submitting to this hardware backdoor.

Here's some of what Kaspersky's guys wrote about their discovery. And note that even they did not discover it, because it's explicitly not discoverable. They discovered its use by malware which they were reverse-engineering. The question that has the conspiracy folks all wound up is how did non-Apple bad guys discover it?

Leo: Yeah, where did they get the information of where to peek and poke?

Steve: And we're going to be spending some more time on that.

Leo: Because presumably they couldn't brute force it. They couldn't go through every address and see what happened.

Steve: It is not brute forcible.

Leo: Yeah.

Steve: As we'll see. In the section of their paper titled "The mystery of the CVE-2023-38606 vulnerability," Kaspersky wrote: "What we want to discuss is related to the vulnerability that has been mitigated as 2023-38606. Recent iPhone models have additional hardware-based security protection for sensitive regions of the kernel memory." This is the PPL they're talking about. "This protection prevents attackers from obtaining full control over the device if they can read and write, that is to say, even if they can read and write kernel memory, as achieved in this attack by exploiting CVE-32434."

So the point being even if you have read and write to kernel memory, you still can't do anything with it. And that's what PPL, Apple said, protects from; right? This thing is so locked down that you still can't get into user space. They said: "We discovered that to bypass this hardware-based security protection, the attackers used another hardware feature of Apple-designed Systems on a Chip.

"If we try to describe this feature and how the attackers took advantage of it, it all comes down to this: They are able to write data to a certain physical address while bypassing the hardware-based memory protection by writing the data, destination address, and data hash to unknown hardware registers of the chip unused by any firmware. Our guess is that this unknown hardware feature was most likely intended to be used for debugging or testing purposes by Apple engineers or the factory, or that it was included by mistake. Because this feature is not used by the firmware, we have no idea how attackers would know how to use it. We're publishing the technical details so that other iOS security researchers can confirm our findings and come up with possible explanations of how the attackers learned about this hardware feature.

"Various peripheral devices available in the System on Chip may provide special hardware registers that can be used by the CPU to operate these devices. For this to work, these hardware registers are mapped to the memory accessible by the CPU and are known as "memory-mapped I/O." Address ranges for MMIOs of peripheral devices in Apple products (iPhones, Macs, and others) are stored in a special file format called the DeviceTree. Device tree files can be extracted from the firmware, and their contents can be viewed with the help of the DT (stands for device tree) utility.

"While analyzing the exploit used in the Operation Triangulation attack I," wrote the Kaspersky researcher, "discovered that most of the MMIOs used by the attackers to bypass the hardware-based kernel memory protection do not belong to any MMIO ranges defined in the device tree." In other words, they're deliberately secret. "The exploit targets Apple A12 through A16 Bionic Systems on a Chip, targeting unknown MMIO blocks of registers which we have documented."

He said: "This prompted me to try something. I checked different device tree files for different devices and different firmware files. No luck. I checked publicly available source

code. No luck. I checked the kernel images, kernel extensions, iBoot, and coprocessor firmware in search of a direct reference to these addresses. Nothing. How could it be that the exploit used MMIOs that were not used by the firmware? How did the attackers find out about them? What peripheral devices do these MMIO addresses belong to?

"It occurred to me that I should check what other known MMIOs were located in the area close to these unknown MMIO blocks. That approach was successful, and I discovered that the memory ranges used by the exploit surrounded the system's GPU coprocessor. This suggested that all these MMIO registers most likely belonged to the GPU coprocessor. After that, I looked closer at the exploit and found one more thing that confirmed my theory. The first thing the exploit does during initialization is write to some other MMIO register, which is located at a different address for each version of Apple's System on Chip."

In the show notes I have a picture of the pseudocode that Kaspersky provided. It shows an "if cpuid =" and then a big 32-bit blob. And the comment is CPUFAMILY_ARM_EVEREST_SAWTOOTH (A16). And if it's equal, it loads two variables, base and command, with two specific values. Then "else if cpuid =" and another 32-bit blob. That refers to CPUFAMILY_ARM_AVALANCHE_BLIZZARD (A15). And if that cpuid equal is met, it loads two different base and commands into those values. And so forth for A14, A13, A12. In other words, all five CPU families from Bionic A12, 13, 14, 15, 16, require their own something. And what we learn is that it's their own special unlock in order to enable the rest of this. It is different for each processor family.

Leo: Wow. And it seems like Apple did a good job with this. That's what's interesting; right?

Steve: Oh, Leo. Wait, wait. They did a great job with this. There's more. Kaspersky's posting shows some pseudocode for what they found was going on. Each of the five different Apple Bionic A12 through A16 is identified by a unique CPU ID which is readily available to code. So the malware code looks up the CPU ID, then chooses a custom per-processor address and command, which is different for each chip generation, which it then uses to unlock this undocumented feature of Apple's recent chips.

Here's what they said: "With the help of the device tree and Siguza's utility, PMGR [power manager]," he said, "I was able to discover that all these addresses corresponded to the GFX register in the power manager MMIO range. Finally, I obtained a third confirmation when I decided to try to access the registers located in these unknown regions. Almost instantly, the GPU coprocessor panicked with a message of "GFX ERROR Exception class" and then some error details. This way, I was able to confirm that all these unknown MMIO registers used for the exploitation belonged to the GPU coprocessor. This motivated me to take a deeper look at its firmware, which is written in ARM code and unencrypted, but I could find not anything related to these registers in there."

Leo: Hmm.

Steve: "I decided to take a closer look at how the exploit operated these unknown MMIO registers. One register, located at hex" - and I put this in the show notes just to give everyone an idea of how just obscure this is. "One register located at hex 206040000 stands out" - I know.

Leo: I think it's more than 999. I think. That's a pretty large number, yes.

Steve: It is. It is. He says: "It stands out from all the others because it is located in a separate MMIO block from all the other registers. It is touched only during the initialization and finalization stages of the exploit. It is the first register to be set during initialization and the last one during finalization. From my experience, it was clear," he said, "that the register either enabled or disabled the hardware feature used by the exploit or controlled interrupts. I started to follow the interrupt route, and fairly soon I was able to recognize this unknown register, and also discovered what exactly was mapped to the register range containing it."

Okay. So through additional reverse-engineering of the exploit and watching it function on their devices, the Kaspersky guys were able to work out exactly what was going on. The exploit provides the address for a Direct Memory Access, a DMA write, the data to be written, and a custom hash which signs the block of data to be written. The hash signature serves to prove that whomever is doing this has access to super-secret knowledge that only Apple would possess. This authenticates the validity of the request for the data to be written.

And speaking of the hash signature, this is what Kaspersky wrote. They said: "Now that all the work with all the MMIO registers has been covered, let us take a look at one last thing: how hashes are calculated."

Leo: Breaking down - but we should mention, though, Apple has patched all four CVEs. Right?

Steve: Hardware's still there.

Leo: Oh. So is some of this not patchable? Is that what you're saying?

Steve: No, no, no. It has been closed, and we're absolutely going to...

Leo: We'll get to that, okay. I just want to say that upfront so people don't go "Oh, no," and put their iPhone in a steel blank box or something.

Steve: Yeah, just definitely update it.

Leo: Yeah. Keep it updated.

Steve: Okay. So Kaspersky says: "Now that all the work with all the MMIO registers has been covered, let us take a look at one last thing: how hashes are calculated. The algorithm is shown, and as you can see, it is a custom algorithm." And Leo, it's in the show notes on the next page, on page 11 on my show notes.

Leo: It's a lookup table; right?

Steve: They said, well, okay, I'll get there. "With the hash is calculated by using a predefined SBox table." He says: "I tried to search for it in a large collection of binaries, but found nothing. You may notice that this hash does not look very secure, as it occupies just 20 bits, but it does its job as long as no one knows how to calculate and use it. It is best summarized with the term 'security by obscurity.'"

Leo: You'd have to know the values in this SBox.

Steve: Every single value.

Leo: Because they're random. They're not consecutive or anything like that, yeah.

Steve: Correct. He said: "How could attackers discover and exploit this hardware feature if it is never used, and there are no instructions anywhere in the firmware on how to use it?" Okay. So to break from the dialogue for a second, in crypto parlance, an "SBox" is simply a lookup table.

Leo: In this case, what, 256 values, it looks like? Yeah.

Steve: Exactly. It's typically an 8-bit lookup table of 256 pseudorandom but unchanging values. The "S" of SBox stands for substitution. So SBoxes are widely used in crypto and in hashes because they provide a very fast means for mapping one byte into another in a completely arbitrary way. The show notes shows, as I said, this SBox and the very simple lookup and XOR algorithm below that's used by the exploit. It doesn't have to be a lot in order to take an input block and scramble it into, basically, it is a 20-bit secret hash.

So the $64,000 question here is how could anyone outside of Apple possibly obtain exactly this 256-entry lookup table, which is required to create the hash that signs the request which is being made to this secret, undocumented, never seen hardware which has the privileged access that allows anything written to bypass Apple's own PPL protection? Some have suggested deep hardware reverse-engineering, you know, popping the lid off of the chip and tracing its circuitry.

Leo: Wow.

Steve: But because this is a known avenue of general vulnerability, chips are well protected from this form of reverse engineering. And that sure seems like a stretch in any event. Every one of the five processor generations had this same backdoor, differing only in the enabling command and address.

Leo: Do they have the same SBox?

Steve: Yes, same SBox.

Leo: Yeah, they didn't want to change that every time.

Steve: Some have suggested that this was implanted into Apple's devices without their knowledge, and that the disclosure of this would have come as a horrible surprise to them. But that seems far-fetched to me, as well. Again, largely identical implementations with a single difference across five year separation generations of processor, and then to have the added protection against its use or coincidental discovery of requiring a hash signature for every request.

To me, that's the mark of Apple's hand in this. If this was implanted without Apple's knowledge, the implant would have been as small and innocuous as possible, and the implanter would have been less concerned about its misuse, satisfied with the protection that it was unknown, that there was this undocumented set of registers floating out in this massive address space that only they would know about. They would want to minimize it. Add to that the fact that the address space this uses is wrapped into and around the region used by the GPU coprocessor. To me this suggests that they had to be - the GPU coprocessor and this had to be designed in tandem.

Okay. After Apple was informed that knowledge of this little hardware backdoor had somehow escaped into the wild and was being abused as part of the most sophisticated attack on iPhones ever seen, they quickly mitigated this vulnerability with the release of iOS 16.6. During its boot up, the updated OS unmaps access to the required memory ranges through their processor's memory manager. This removes the logical to physical address mapping and access that any code subsequently running inside the chip would need.

Leo: Doesn't that break stuff, though? I mean...

Steve: No. Just that, I mean, it wasn't supposed to be there anyway.

Leo: Oh.

Steve: So it's, yeah, it's literally just those critical ranges. The Kaspersky guys conclude by writing: "This is no ordinary vulnerability, and we have many unanswered questions. We do not know how the attackers learned to use this unknown hardware feature or what its purpose was. Neither do we know if it was developed by Apple, or it's a third-party component like ARM CoreSight.

"What we do know, and what this vulnerability demonstrates, is that advanced hardware-based protections are useless in the face of a sophisticated attacker as long as there are hardware features that can bypass those protections. Hardware security very often relies on 'security through obscurity,' and it is much more difficult to reverse-engineer than software. But this is a flawed approach because sooner or later all secrets are revealed. Systems that rely on 'security through obscurity' can never be truly secure." That's how they end. And Kaspersky's conclusions here are wrong.

Leo: Oh.

Steve: The Kaspersky guys are clearly reverse engineering geniuses. And Apple, who certainly never wanted knowledge of this backdoor to fall into the hands of the underworld for use by hostile governments and nation states, owes them big-time for figuring out that this closely held secret had somehow escaped. But I take issue with Kaspersky's use of the overused phrase "security through obscurity." That does not apply

here. The term "obscure" suggests discoverability. That's why something that's obscure and thus discoverable is not truly secure, because it could be discovered.

But what Kaspersky found could not be discovered by anyone, not conceivably, because it was protected by a complex secret hash. Kaspersky themselves did not discover it. They watched the malware, which had this knowledge, use this secret feature; and only then did they figure out what must be going on. And that 256-entry hash table which Kaspersky documented came from inside the malware which had obtained it from somewhere else, from someone who knew the secret.

So let's get very clear because this is an important point. There is nothing whatsoever obscure about this. The use of this backdoor required a priori knowledge, explicit knowledge in advance of its use. And that knowledge had to come from whatever entity implemented this, period. Also, we also need to note that, as powerful as this backdoor was, access to it first required access to its physical memory-mapped I/O range which is, and was, explicitly unavailable.

Software running on these iDevices had no access to this protected hardware address space. So it wasn't as if anyone who simply knew this secret could just waltz in and have their way with any Apple device. There were still several layers of preceding protection which needed to be bypassed. And that was the job of those first two zero-day exploits which preceded the use of 38606.

So we're left with the question, why is this very powerful and clearly meant to be kept secret faculty present at all in the past five generations of Apple's processors? I don't buy the argument that this is some sort of debugging facility that was left in by mistake. For one thing, it was there with subtle changes through five generations of chips. That doesn't feel like a mistake, and there are far more straightforward means for debugging. No one hash-signs a packet to be written by DMA into memory while they're debugging. That makes no sense at all.

Also, the fact that its use is guarded by a secret hash reveals and proves that it was intended to be enabled and present in the wild. The hash forms a secret key that explicitly allows this to exist safely and without fear of malicious exploitation without the explicit knowledge of the hash function. This is a crucial point that I want to be certain everyone appreciates. This was clearly meant to be active in the wild with its abuse being quite well protected by a well-kept secret. So again, why was it there? Until Apple fesses up, we'll likely never know.

Leo: So let me, before you go on, does this functionality have any purpose that we know of? Is it used in any way by any Apple...

Steve: No.

Leo: No.

Steve: And that's the point.

Leo: It's weird it's in there.

Steve: There is no reference to it. And that's what the Kaspersky guy looked for. He looked everywhere. Nothing uses it. There is no reference to it anywhere.

Leo: So if that's the case, it implies it might have been put in there for this purpose. I mean, what purpose...

Steve: That's what I believe.

Leo: What other purpose could - if it's not being used currently...

Steve: Exactly.

Leo: You make some extreme efforts to not only put it in there, but to protect it.

Steve: Yes.

Leo: You have a 256-byte secret that presumably like one Apple engineer knows, you know, it's kept under lock and key.

Steve: Right, right.

Leo: In a basement office somewhere in Cupertino.

Steve: Right.

Leo: Would the chip manufacturers know this hash?

Steve: All they see is a mask which they are imbuing in the silicon.

Leo: They don't, they can't see it.

Steve: No.

Leo: So there is a secret here which Apple holds.

Steve: Yes.

Leo: Presumably holds tightly.

Steve: Yes.

Leo: To a functionality which nobody apparently uses except maliciously.

Steve: Yes.

Leo: Well, that's interesting.

Steve: That's exactly it. Okay. So could this backdoor system have always been part of some large ARM silicon library that Apple licensed without ever taking a close look at it? Okay, well, that doesn't really sound like the hands-on Apple we all know. But it's been suggested that this deliberately engineered backdoor technology might have somehow been present through five generations of Apple's custom silicon without Apple ever being aware of it.

One reason that idea falls flat for me is that Apple's own firmware Device Tree, which provides the mapping of all hardware peripheral into memory, the documentation of the mapping accommodates these undocumented memory ranges without any description. Unlike any other known component, nowhere are these memory ranges described. No purpose is given. And Apple's patch for this, that's what this CVE fixes, Apple's patch for this changes the descriptors for those memory ranges to "DENY" to prevent their future access.

Leo: Well, that further confirms that there's no reasonable use for this, if you could just turn it off.

Steve: Right.

Leo: It doesn't break anything.

Steve: Right.

Leo: It's not being used by anybody.

Steve: That's right.

Leo: But it's in there, and it's carefully put in there.

Steve: Yup. The other question, since the hash function is not obscure, it is truly secret, is how did this secret come to be in the hands of those who could exploit it for their own ends? No matter why Apple had this present in their systems, that could never have been their intent. And here I agree with the Kaspersky guys when they say "This is a flawed approach because sooner or later all secrets are revealed." Somewhere, Leo, exactly as you said, people within Apple knew of this backdoor. They knew that this backdoor was

present, and they knew how to access it. And somehow that secret escaped from Apple's control.

Leo: Do you think it's possible Apple put that in there at the bequest of a government, a China or Russia?

Steve: Well...

Leo: As a backdoor? And yeah, we have a backdoor you can use?

Steve: That's two paragraphs from here.

Leo: Okay. Keep going, sorry.

Steve: What we do know...

Leo: You've got my mind going. This is fascinating.

Steve: This is very good, Leo. Now you know why we're talking about nothing else this week.

Leo: Yeah.

Steve: What we do know is that, for the time being at least, iDevices are more - that is, after this patch - more secure than they have ever been because backdoor access through this means has been removed, after it became public. But because we don't know why this was present in the first place, we should have no confidence that next-generation Apple silicon won't have another similar feature present. Its memory-mapped I/O hardware location would be changed, its hash function would be different, and Apple will likely have learned something from this debacle about keeping this closely held secret even more closely held.

Do they have a secret deal with the NSA to allow their devices to be backdoored? Those prone to secret conspiracies can only speculate. And even so, it does take access to additional secrets or vulnerabilities to access that locked backdoor. So using it is not easy.

Leo: Could Apple, though, have gone to somebody and said here's the step-by-step? They would know. I mean, it does involve this font resizing in TrueType to get in in the first place.

Steve: Well, yeah, those are particularly - those are two particular zero-days. As we know, Apple is constantly patching against zero-days.

Leo: They don't want these, yeah, yeah.

Steve: Right? So...

Leo: They may have a more direct access to this that doesn't require a zero-day, perhaps.

Steve: Well, remember this thing has been in place now since the A12.

Leo: Five years.

Steve: Yes, yes. So it's been blocking things, yet people have still been getting in. This may have been used for a long time, and Kaspersky, I mean Kaspersky, finally just found it.

Leo: And we don't know - the reason they found it was it was embedded on iPhones of many Kaspersky employees.

Steve: Yes.

Leo: But what we don't know is...

Steve: The bad guys...

Leo: ...who did that. It's almost certainly a nation-state. Would you agree?

Steve: You know...

Leo: It's got to be [crosstalk] or somebody; right?

Steve: It was a mistake to attack Kaspersky.

Leo: Yeah, maybe that, yeah.

Steve: Now the world knows, and Apple turned it off. Was Apple forced to turn it off? Are they glad they turned it off? Were they surprised? You know, I would prefer to believe that Apple was an unwitting accomplice in this, you know, that this was an ultra-sophisticated supply chain attack, that some agency influenced the upstream design of modules that Apple acquired and simply dropped into place without examination, and that it was this agency who therefore had and held the secret hash function and knew how to use it.

As they say, anything's possible. But that stretches credulity farther than I think is probably warranted. Occam's Razor suggests that the simplest explanation is the most likely to be correct. And the simplest explanation here is that Apple deliberately built a secret backdoor into the past five most recent generations of their silicon.

One thing that should be certain is that Apple has lost a bit of its security shine with this one. A bit of erosion of trust is clearly in order because five generations of Apple silicon contained a backdoor that was protected by a secret hash whose only reasonable purpose could have been to allow its use to be securely made available, after sale, in the wild. And that leaves us with the final question: Why?

Leo: And we can only speculate. I mean, we don't know who put it on the Kaspersky iPhones. Could have been the Russian government. Much more likely the U.S. government. Or the NSA.

Steve: Somebody had knowledge, yes.

Leo: It's speculation, but it's not a long stretch to think that the NSA required a backdoor of Apple. Apple complied, gave the NSA...

Steve: Made it super inaccessible.

Leo: Made it hard.

Steve: It was safe. It was safe to use.

Leo: If you were going to design a backdoor, Steve, this is how you would do it.

Steve: Yes. Yes. It is absolutely safe. And so yes, you're right, Leo, the NSA could have said, you know, we require a means in. You know? Make it secure. And they made it secure.

Leo: And we know because of the Patriot Act they could have done it with a National Security Letter that would require Apple not reveal that this had been done.

Steve: Yup, ever.

Leo: Ever.

Steve: And certainly allowed Apple to foreclose it the instant it became public. And they did. And as I said, there is nothing now that would lead us to believe it will not reappear under a different crypto hash in the next iteration of silicon.

Leo: Well, and not to be conspiracy theory minded, but there's nothing to make us think that every other phone manufacturer hasn't done the same thing for the NSA.

Steve: Now, one interesting thing, and maybe this will come out after people have had a chance to look at other ARM silicon...

Leo: That's a good question, is this an ARM problem.

Steve: If this was inherited by an upstream supply, we would expect other devices to have the same thing.

Leo: Yeah. Yeah. So that's what's unknown. Is this Apple only?

Steve: Is this exclusive to Apple, or do other devices have this, too? The problem is...

Leo: It looks like it is, though. Yes?

Steve: You have to catch it in use. I mean, Kaspersky only caught it because they had malware that was doing it.

Leo: It makes for, by the way, an excellent spy novel or movie because the secret is a 256-byte chunk that could be sitting on any variety of things. It could be put in a JPG with steganography. It makes for a very interesting spy novel. Possibly the NSA got to an Apple engineer. But it'd have to be the Apple engineer. I can't imagine this secret was held by many people. Maybe not by any one person at all. If I were Apple, I'd split it up. Very interesting.

Steve: Yeah.

Leo: Very interesting. You know, I mean, the most obvious explanation is, as you said, the NSA and a secret backdoor.

Steve: Yup, said you have to put this in because we need access.

Leo: Yeah. In which case maybe you can't find fault with Apple because that's the problem with being a business in the U.S. You have to follow the laws of whatever country you're doing business in, whether it's China or the U.S. They're not in Russia anymore.

Steve: And Apple could argue it is safe. I mean, what they did, it might as well not exist except for the fact that the secret got loose. Well, or maybe it didn't. Maybe the NSA still has it closely held. On the other hand, reverse engineering the malware, and now we see.

Leo: That's right, we only know because we've got the malware.

Steve: Right.

Leo: You know, somebody in the Discord is pointing out, Adrod points out that both the Chinese government and the Russian government have within the last year forbidden the use of iPhones in government business.

Steve: Yeah, that's true.

Leo: Wow. This is quite a story.

Steve: This has been present for the last five years. This went into - this first appeared in the A12, five generations ago.

Leo: And if they can do it once, they can do it again. There's no guarantee that...

Steve: That's exactly my point. If they turn this one off, it'll be in the next generation of silicon. And again, we'll have no idea.

Leo: Well, for all we know there's three backdoors in the existing generation.

Steve: That's a very good point. That's a very good point. There could be two different, two other hashes and the same access so that if one is discovered, the other two remain active.

Leo: I am going to vote in favor of Apple and say they were compelled to do this by the NSA. They had no choice. They couldn't reveal it. They did it in as secure a way as possible.

Steve: All the evidence suggests that so far.

Leo: And I think in their favor they had no choice. I mean, if the NSA says it, you get an actual security letter, you don't have a choice. And there's a FISA court somewhere that ordered this.

Steve: Wow.

Leo: Wow. The problem with this is you just listened to one of the most technical podcasts for the last two hours. There are very, very few people who would listen to this, understand it, and understand the implications. And that's another thing that - that's another form of obscurity because I wouldn't expect even Ron Wyden or his

very, well, Chris Soghoian would understand. Maybe Chris Soghoian is whispering in Ron Wyden's ear right now and saying, hey, you know, this is a problem.

Steve: Yeah, we've got to ask Apple what the hell.

Leo: What is going on here. What a must-listen-to episode. As always, Steve, thank you so much for your deep work. When I said on MacBreak Weekly I'm sure Steve will cover this, I had no idea how deep this rabbit hole was going to go. Fascinating.

Steve Gibson is at GRC.com. SpinRite, what's going on? How soon? Don't say a date. That's all right. You can't. You can win with it.

Steve: I can't. I was actually a little frantic over the holidays trying to make my own deadline of before the end of the year.

Leo: No, no, no. Do it right. Do it right.

Steve: And I have no choice. Everyone will get the payback of my getting it done right.

Leo: This is why we love Steve. He does it right, first. He doesn't wait, say I'll fix it in post. GRC.com. You can get a 6.0 version of SpinRite, which is the current version. You'll get a free upgrade to 6.1 as soon as it's out. You'll be the first to know. You can even participate - as you can see, many have - in its development.