SECURITY NOW!

Transcript of Episode #669

Cellular Location Privacy

Description: This week we examine some new side-channel worries and vulnerabilities. Did Mandiant "hack back" on China? More trouble with browsers, the big Google Firebase mess, sharing a bit of my dead system resurrection, and a look at the recent Supreme Court decision addressing cellular location privacy.

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-669.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-669-lq.mp3

SHOW TEASE: It's time for Security Now!. Steve Gibson is here. There's lots, well, there's actually not that much to talk about. Steve said it's a light day. I think we'll find a few things, though, on the agenda. A Picture of the Week that involves magnets. A lot more about side-channel attacks. Even WebAssembly is vulnerable, believe it or not. And the scoop about the scandal at FireEye's Mandiant Group. It's all coming up next on Security Now!.

Leo Laporte: This is Security Now! with Steve Gibson, Episode 669, recorded Tuesday, June 26th, 2018: Cellular Location Privacy.

It's time for Security Now!, the show where we cover your privacy and security online with this guy right here, Mr. Steve Gibson of the GRC Corporation. Good afternoon, Steve.

Steve Gibson: Good afternoon, Leo. Good to see you again.

Leo: Nice to see you.

Steve: I decided to name this podcast in honor of a recent Supreme Court decision about cellular location privacy. I read the decision. The good news is we'll summarize it. But I'm going to spend some time with it because what I love about Supreme Court decisions is the clarity of the writing, I mean, and how exact it is, and how careful they are about what they say. I mean, that's sort of what I try to do in a whole different realm on this podcast, as we know.

And this is the result of a challenge from, I think it was 2011 is when, you know, because these things all take a long time to roll through the courts and then to get appealed and then the appellate court says, no, we think that was probably the right decision, and then they go, oh, we're taking this to the Supreme Court. Well, this is about the bar required by law enforcement to obtain the so-called business records of cellular service providers, which in this case can be used to locate people. And anyway, so the case is interesting. But the exact way the result was phrased I think is really interesting. So that's sort of our main topic.

And for whatever reason there wasn't a huge amount of stuff this week. But that's good because a number of the topics that we want to discuss are bigger. So there's a surprising amount, actually, of breadth and depth to new side-channel attack worries and vulnerabilities. There's an interesting question about whether the security firm Mandiant, which was, after they did what they're alleged to have done, purchased by FireEye, whether they actively hacked back some Chinese hackers and showed that to a very respected journalist, David Sanger, who explained all this in his recently released book, which has caused a bunch of furor because it's like, whoops, that's illegal.

Anyway, we also have more trouble with browsers, a big mess with a company named Firebase that Google purchased, I think it was in 2014, about four years ago. They offer a bunch of cloud connectivity backend stuff which makes it really easy to add cloudness to apps, largely Android, but also some iOS. Unfortunately, somehow, nobody was worried even recently about security. And so a startling number of apps that affects well over half of industries due to their prevalence have completely insecure connections to the cloud. And this company whose name I can't - anyway, it's in the notes - did a complete analysis and report that we need to talk about just because it would be good to know if anyone is using any of these.

Also I had an interesting adventure. You know that here I am today, Tuesday, a week and a day after having come to my work area and found my trusty beloved Windows XP machine dead, unrevivable, unresuscitatable, just gone. Anyway, I decided I had a need to get to reboot it, essentially, because I wanted to move some licenses off of it that you can only do from the running instance. You can't do it from the file system. The hard drive was fine. It was a big four-drive RAID and all that.

So I solved the problem, and I thought our listeners would find it interesting, just as an interesting forensics experiment. And, boy, I also came away very impressed with how far the virtualization industry has come since I last had an occasion to look at it. So, and we have a couple little bits of miscellany stuff. And we'll talk about what the Supreme Court thinks. So I think another great podcast. And a fun Picture of the Week.

Now, it's funny, when you mentioned it, I was choosing among them, and I had it in my head that it was different than this one. And I agree with you, this one looks a little bit made up. But still kind of - it's definitely...

Leo: Not a few of yours look like they might have been staged. But, you know, people have done worse; right?

Steve: Yeah.

Leo: Should we save it?

Steve: Well, we can just tell people now. It's just great. It shows a 3.5" floppy that's labeled with a nice Sharpie marker: "System restore disk. Do not erase." So this was somebody who put all of their system restore information there, whatever that is, and wanted to make sure that nobody would erase it. It's shown stuck to a refrigerator with a big bar magnet.

Leo: Save this. Keep it here.

Steve: That's right. Do not erase.

Leo: Do not erase.

Steve: It's called "bulk erasing," for any of you who do not know, yes.

Leo: Or degaussing. We used to call it degaussing.

Steve: Degaussing, yeah, right. Remember when, exactly, our old-school televisions would have kind of like a green tinge on sort of one area of the screen. And later they got automatic degaussing. But for a while there was actually a coil you could buy, and you would move this hoop around the screen and then very slowly pull it away in order to leave it with no net magnetic field.

Leo: I wouldn't be surprised if young people today look at the way we lived 20 years ago and think, "They were savages. Just savages."

Steve: Yes. And it required three people to lift the CRT tube.

Leo: Yeah.

Steve: Remember those huge - the Sony XBRs? I mean, they were doing big screen the old-fashioned way.

Leo: But it wasn't that big. A big screen TV in those days was 21 or 27 inches. But it was at least that deep.

Steve: Oh, you needed to cut a hole in the wall and protrude into the next room, yeah. Ah, those were the days.

Leo: Ah, the days. Those were the days, my friend.

Steve: So I stumbled into this because the idea or the issue caught my eye, and that was that the OpenBSD project announced they were suspending support for hyperthreading due to a forthcoming emerging concern that they were rather disgruntled about not being told of.

Okay. So backing up a little bit, OpenBSD is the branch of the Unix project which prioritizes security above all else. There's NetBSD, FreeBSD, OpenBSD, YourBSD, MyBSD, OurBSD. Anyway, OpenBSD security. And the reputation is well deserved. I think, I don't remember now what the stat is, but something like since 1996 they've only had two remote code execution vulnerabilities, whereas everybody is drowning in them. So focusing on security I think pays off.

Okay. So Theo de Raadt, and that's not R-A-T, it's R-A-A-D-T, he's German, Theo de Raadt...

Leo: Dutch.

Steve: Dutch, okay. Of course, Dutch, right.

Leo: And he's just kind of a legend, frankly.

Steve: Oh, and it's funny, too, because, yes, I saw his headshot, and I thought, wow, I recognize him. I mean, it's like he only ever took one picture, and it's still...

Leo: It's the same one. It's like Linus. Kind of like Linus, yeah, yeah.

Steve: Yeah. It was exactly the one I remembered of him. It's like, oh, wow. He's not aging, either.

Leo: Yeah.

Steve: Although you wouldn't know that from how grumbly he is about this. So he heads the OpenBSD project. And in their coverage of it last Thursday, he told IT Wire...

Leo: Oh, I'm sorry. He's South African.

Steve: Really.

Leo: It's a Boer name, and he lives in Canada now. I'm just looking at his bio.

Steve: Oh, okay, good. Well, Theo de Raadt. Do you say de Raadt?

Leo: I say de Raadt, but I don't know. Who knows.

Steve: Yeah. We don't want to call him "The Rat" because...

Leo: Theo the Rat.

Steve: Anyway, so he was interviewed by IT Wire, who said that he'd spent about a month trying to discern the problems that he said, quote, "Intel would not disclose to us." So here he is - and actually, I mean, he's the perfect guy to run the project because I did a little bit of, I mean, I've known of him, and he really is, I mean, he's just a security hardliner. So what he explained is that a paper is coming out which will be presented at the forthcoming Black Hat USA 2018 Conference, this coming August, which he got access to. So I thought, oh, that's interesting. What's that?

And so I tracked it down in the Briefings Schedule. It's a paper titled "TLBleed: When Protecting Your CPU Caches Is Not Enough." Well, and I smiled when I saw that it was by an old friend of the podcast, Ben Gras. Ben and I corresponded, and I shared this with the podcast, a couple years ago. These are the guys who, with their Professor Herbert Bos at VU Amsterdam did the Flip Feng Shui, which was the weaponization of RowHammer. And back then we covered this on podcast 577 in 2016, and he wrote to us.

He said: "As one of the authors, thank you for your knowledgeable and detailed exposition. We're honored by it and your kind words." Oh, because I called it a sublime righteous hack, I mean, it was really - they really did some clever stuff. And so I said, oh, thanks, and wow. And then he said: "Thank you again for your Flip Feng Shui coverage. Because of more recent coverage, I'm a bit of an expert on how well our work is understood. I don't hesitate to rate your coverage at a 95th percentile rating for expertise and quality of exposition."

So we've established a relationship. And I imagine that, before long, August is month after next, we'll probably be hearing from Ben when he has something to talk about because he's kept in touch since, and we've covered a bunch of their work since then. So it's apparent that he and his team, presumably still led by this Professor Herbert Bos, who we wanted to make sure in our earlier correspondence I always gave credit to, have been up to more mischief.

The abstract for the Black Hat paper reads: "We present TLBleed." And it's no coincidence that the first three letters are TLB. He says: "...a novel side-channel attack that leaks information out of Translation Lookaside Buffers. TLBleed shows a reliable side channel without relying on the CPU data or instruction caches," which puts it in a different category similar to Spectre and Meltdown, but importantly different because this isn't microcode fixable.

He says: "This therefore bypasses several proposed CPU cache side-channel protections, such as page coloring, CAT, TSX, and so forth. Our TLBleed exploit successfully leaks a 256-bit EdDSA key from cryptographic signing code, which would be safe from cache attacks with cache isolation turned on, but would no longer be safe with TLBleed."

So basically they've got like a next-generation side-channel attack that, once again, leverages cross-thread state leakage in contemporary CPUs. He says: "We achieve a 98% success rate after just a single observation of signing operation on a co-resident hyperthread and just 17 seconds of analysis time. Further, we show how another exploit based on TLBleed can leak bits from a side-channel-resistant RSA implementation." So, I mean, this is cutting through a lot of exploit mitigation.

He says: "We use novel machine-learning techniques to achieve this level of performance. These techniques will likely improve the quality of future side-channel attacks," suggesting that this is also an advance in technology. "This talk contains details about the architecture and complex behavior of modern, multilevel TLBs [Translation Lookaside Buffers] on several modern Intel microarchitectures that is undocumented and will be publicly presented for the first time."

Okay. So that's what had Theo's nose out of joint is he found out this was happening. He tried to get information from Intel, and Intel was not being forthcoming. So Theo said, in his interview with iTWire, said he could not be more specific about the nature of the vulnerability that had led to the disabling of hyperthreading because the paper has not yet been published. So he's under NDA to have access to the paper. And Ben has done the same thing with me several times when he's wanted me to see what he was doing, but I couldn't talk about it until it was released publicly, of course.

So as a consequence, though, well, and fact is, I mean, knowing the technology, knowing what the abstract says that's public, knowing what OpenBSD did, there's really not much left to know except some of the details. Theo said of the paper: "It was shared with us, allowing us to fix it. The solution is trivial and temporary." He says: "Maybe Intel tells us a different way. I'm waiting... I'm waiting... I better not keep waiting," he said. And then he gets grumbly, saying: "Only Tier 1 companies received advance information, and that is not responsible disclosure. It is selective disclosure. Everyone below Tier 1 has just gotten screwed," he told iTWire.

Okay. So what we know from the many conversations we've had about these various speculative attacks is that when different interests share the same hardware, the optimizations which have been employed for years to speed up the performance of our processors, when those optimizations leave a residual state that has changed the future of what the processor will do - and after all, that's their whole point is to change the future. They want to speed up the future. When that happens, it's possible for an adversary running on the same system to obtain information. We understand that.

Okay. So hyperthreading, as I mentioned, I was talking just I guess it was last week or the week before about thread context switching and how when you're executing code you have - oh, yeah, it was in the context of the multimedia buffers, the MMX and so forth registers. I remember, in fact it was last week's topic, it was the Lazy Floating Point Restore, where I was explaining about threads and context switching and where, in order to switch to executing a different, like at a different location in the system, you have to store away all the registers that have been in use and then reload them all from the thread of execution that had been suspended, and then let it go.

And the idea is by - what Intel did that we talked about was probing to see whether a thread actually attempted to do any floating point operations because restoring those registers - there's a lot of them, and they're big - that's an expensive thing to do. So they said, hey, we can save some time, which is the whole point.

Okay, well, another thing that Intel did years ago that I referred to just briefly before I knew about this this week, is hyperthreading, which was - it was a low performance gain, but it was inexpensive for them. I think I recall it was like 8% maybe, which is not nothing, but still it wasn't - the idea was it would allow you to - this is before multicore, so you just had one core, and you were happy to have one. They were looking for more performance. And so they realized that they could, with minimal additional hardware, have like a shadow set of registers, that is, have two full sets of registers and, with just flipping a single bit, switch register contexts. And what that would so is it would give you what they called "hyperthreading." You didn't actually - you were sharing everything else. But it saved the time-consuming register save and restore.

Now, the problem with that is that a modern operating system doesn't just have two threads. It'd be wonderful if you only had two threads ever. Then you could just ping-pong back and forth between them really fast. Typical operating systems are now running thousands of threads. So the idea that you can bounce between a pair, well, okay, it's become less useful as we've become much more aggressive with threading.

And so what's happened is, rather than for example just extending that to more hyperthreads per core, they actually did more cores. But because it was there, because it's been there forever, because operating systems know how to turn it on, that is, the hyperthreading - and in some old BIOSes you see that option, Enable Hyperthreading. It's like, oh, okay. Well, yeah. There has been, until now, no reason not to have it because - so Intel just sort of left it in there, even though its value has been diminishing as the opportunity to switch between one additional context when you actually have thousands of them in the system, that's not that useful.

So what this means is that what Ben and his group discovered was a high reliability way for the other thread of the pair of a hyperthread pair to obtain information about its mate, essentially. So the immediate reaction of the security-conscious OpenBSD folks was just to say, okay, fine, we're turning off hyperthreading. Again, it'll be a little bit of a performance hit. Probably not a lot. And they're just doing it preemptively.

Intel has apparently responded to all of this, saying, you know, this is really not that big a deal, folks. Intel's position is that the existing mitigations surrounding cache hit and cache miss and cache eviction will be enough to solve this problem. I don't see how that's the case. I don't see how the microcode updates that we've received for Spectre apply to this because what the Translation Lookaside Buffers are, and I've also talked about my amazement with that portion of Intel's architecture over the years, they're the things that enable virtual memory. Essentially, the TLBs are the hardware implementation for virtual memory. Virtual memory creates an abstraction between the memory any of our applications, any of our processes running on an operating system see, with the actual memory, the actual physical memory in the system.

So, for example, every process can think that it loads at location 10,000. Normally there is like a base processor load point. Every process thinks it's running starting at location 10,000, when in fact they're scattered arbitrarily through physical memory. And it's a series, it's a hierarchy of translation tables which maps or converts the addresses that the process sees to the actual physical memory that the processor reads and writes to and from. And in Intel systems there's like four layers of these TLBs, Translation Lookaside Buffers, in order to support context switching and process switching and all of the virtual memory operations that our modern systems perform for us without us being aware of it.

So this is a form of caching, and you can't turn it off. I mean, there's just - our systems wouldn't even, I mean, I've always been amazed that they run at all through this level of hardware abstraction. But somehow they manage to. Pages are allocated typically in 4K blocks, but they can also be, I think it's 2MB and one - couldn't be 1GB. Anyway, you're able to change the page size. Normally 4K is what all of our modern processors use, and operating systems.

So what's clearly happening is that, because a thread executing and asking for data forces these Translation Lookaside Buffers which implement the virtualization of memory, that forces them to be loaded, which forces the eviction, because they're limited in size, of what was loaded before. Certainly it would be immediately clear to a hardware architect, as it was to Ben and his group, that here is another way in which two threads sharing the same core are able to leak information.

What will probably happen - so I guess my first reaction to Intel's denial of this being yet another problem, I mean, they must be getting tired of this over there, is I don't see how you work around this. You need to do something - it'll be really interesting to see, when we have the information from the Black Hat presentation, what these guys are able to suggest as a means for solving this problem. The only thing that comes to mind is that you change what's known as the "thread scheduler," the process scheduler and the thread scheduler, in operating systems so that you are never having different processes sharing hyperthreads. And that requires restructuring and rearchitecting our systems to some degree, but I don't see it as a huge problem.

Most processes are already multithreaded. So they're already asking for multiple threads of execution for their own work. So all you have to do is make sure that both threads of the hyperthread are being used by the same process, rather than by different processes. And if you do that, then you prevent cross-process information leakage via Translation Lookaside Buffers. And I can't think of any other way to solve the problem than that. Again, it's not a patch, it's some restructuring. And it would be a shame, too, because this SMT, Simultaneous Multi Threading, is the feature that Intel offers with hyperthreading. And, I mean, it's a benefit to have it in today's architectures. So not having it would be a loss. But it's not at all clear how we get around that problem.

So we'll stay tuned. I'm sure that we will be hearing from Ben and his group and probably come back and take a look at this after Black Hat and after the information has been made fully public. In the meantime, OpenBSD doing what they do, they just said, okay, we're turning that off. We're just going to - I mean, and again, not a huge performance hit. And I like it that these guys are prioritizing security first. I mean, that's their deal. So yet one more way. I mean, it's really looking like everything which has been done historically to give us more performance from clever engineering is now, as they say, these chickens are coming home to roost. It's like, well, we've been sort of happily, naively skipping along, enjoying the performance that we have. We're going to have to now, in today's climate, scale this back and get some more control.

And I think ultimately it's going to mean that the microcode provides more hooks for next-generation operating systems that can arrange to prevent these sorts of problems robustly. And probably it means that, in the case of hyperthreading, that the multiple threads offered by a single core will need to be kept within a single process. But it'll be interesting to see if these guys come up with a different solution.

Okay. So second question. Did FireEye's Mandiant Group hack back at China? What happened is, David Sanger, who is a well-respected journalist, he's with The New York Times, he's their national security journalist, and he's a Professor of National Security Policy at Harvard's Kennedy School of Government. I've seen endless interviews of him through the years. I mean, he's the real deal. So when he writes a book, as he has, and describes what he saw in it, you have to take it with more than a grain of salt.

First of all, the book is itself interesting, and I think it would be of great interest to our listeners. It's just out. He's been making the rounds, talking about it. It's titled "The Perfect Weapon: War, Sabotage, and Fear in the Cyber Age." I've not read the book. However, I have seen David interviewed about this book, and he makes an interesting case, and that is that launching missiles is an overtly real-world physical act, and attribution is absolute. You know where the missile came from. Cyberattacks sort of slip under the radar. I mean, literally. And they are much less expensive. They require much less physical real-world development and so forth. So this whole cyberwar thing fits into a different category. And David's position is that we need to be paying more attention to this.

Okay. So the book is $15 for Kindle, $20 hardback, and free for an Audible trial. So I really think, again, it's titled "The Perfect Weapon: War, Sabotage, and Fear." Because it's just out, there aren't many reviews on Amazon. But one that caught my eye I wanted to share from someone who is himself an author. He says: "I came to this book the long way around. Knowing that I had just published a military thriller in which North Korea crashes the electrical grid for the greater D.C. area, my brother-in-law sent me a link to David Sanger's recent interview on NPR," regarding this book.

He said: "Listening to Mr. Sanger confirmed some of the scariest parts of my own research. I discovered that my fictional scheme for robbing the U.S. government of electrical power is uncomfortably similar to an actual cyberattack that flatlined large segments of the Ukrainian grid in 2015. Far from being worst-case imaginary scenarios, some of the concepts I've written about," he writes, "have already played out in the real world, usually in countries distant from the United States and under circumstances that either don't make the news or don't create an impression on the public consciousness."

He says: "I burned through this book in less than a day. 'The Perfect Weapon' has the page-flipping intensity of the best techno-thriller novels, with the gravitas of meticulously sourced nonfiction. If I had to sum up this book in one word, it would be 'terrifying.'" He says: "With true stories from the cyber sabotage of the Democratic National Committee to the penetration of the White House computer networks, this book is a wake-up call for our technology-dependent civilization. I just hope we don't hit the snooze button and go back to sleep."

Okay. So that's the book. David Sanger is the real deal, a serious old-school journalist. So that makes this controversy which has arisen interesting. Bleeping Computer, our friends Lawrence Abrams and his team, have some terrific coverage of this that helps to sort of summarize this. They said that U.S. cybersecurity firm FireEye has denied claims that have been ramping up on social media all of last week about illegally hacking back a Chinese nation-stage cyberespionage group. The claims and social media discussions started after the publication of "The Perfect Weapon." And they go on to explain what that is, and we just talked about that.

And then Bleeping Computer writes: "In the book, Sanger recounted a series of events from 2013, in the lead-up to FireEye publishing a report called 'APT1, Exposing One of China's Cyber Espionage Units.'" And actually we talked about this at the time. It was one of the things that we covered on the podcast five years ago. And then Bleeping writes: "At the time, the report was a landmark moment in the cyberintelligence community as it exposed the activities of Chinese hackers in a depth of detail like never before, even going as far as pinning the hacking on Unit 61398 of China's People Liberation Army (PLA), an attribution level unheard of at the time."

Okay. So what Sanger wrote, and I have in the show notes for anyone who's interested, is a link to a tweet by Thomas Rid, who's a political scientist known for his work on the history and risks of information technology in conflict. He's also a professor of strategic studies at Johns Hopkins University School of Advanced International Studies. So he was the one who sort of brought this to light, snapping some pieces of Sanger's book.

Because what David Sanger wrote was: "Ever resourceful, Mandiant CEO Kevin Mandia's staff of former intelligence officers and cyber experts tried a different method of proving their case. They might not be able to track the IP addresses to the Datong Road high-rise itself, but they could actually look inside the room where the hacks originated. As soon as they detected Chinese hackers breaking into the private networks of some of their clients" - now, remember, this is Mandiant previous to the acquisition by FireEye - "mostly Fortune 500 companies, Mandia's investigators reached back through the network to activate the cameras on the hackers' own laptops. They could see their keystrokes while actually watching them at their desks."

Sanger writes: "The hackers, just about all of them male and most in their mid-20s, carried on like a lot of young guys around the world." He writes: "They showed up at work around 8:30 a.m. Shanghai time, checked a few sports scores, emailed their girlfriends. Then, when the clock struck 9:00, they started methodically breaking into computer systems around the world, banking" - I think he meant banging - "on the keyboards until a lunch break gave them a moment to go back to the sport scores and girlfriends."

He says: "One day I sat next to some of Mandia's team, watching the Unit 61938 hacking corps at work. It was a remarkable sight. My previous mental image of PLA officers was a bunch of stiff old generals sitting around in uniforms with epaulets, reminiscing about the glory days with Mao. But these guys were wearing leather jackets or just undershirts, and probably saw Mao only if they visited his mausoleum in Tiananmen Square."

Leo: I don't know where else they would have seen him. They're dreaming. They're dreaming about him.

Steve: Yeah, exactly.

Leo: He hasn't been around a lot lately.

Steve: No. He says: "'They were such bros,' Andrew Schwartz, one of Mandia's communications specialists, recalled later." So now FireEye claims it was a misrepresentation or misunderstanding. And of course people are like, wait a minute. How does seeing what the clothes of these people wearing, how was that a misunderstanding? Okay, so I'm getting ahead of myself.

Bleeping Computer says that, in a statement that [FireEye] just released: "FireEye refutes these claims. The company says that Sanger mischaracterized what really happened and might have simply misunderstood what he was shown that day when he was allowed to sit with Mandiant employees. FireEye says Sanger never observed real-time hacking, but only prerecorded videos of APT1 operators interacting with computers on the network of compromised companies." And I should mention that they went as far as to show a video of a remote terminal where the cursor is moving and menus are clicking themselves and so forth. But it's like, okay, that's way different than seeing the output of a video camera looking at people banging on their keyboards.

"Furthermore, FireEye says it obtained permission from these companies to leave the compromised PCs intact and observe what the hackers were doing, and at no point its employees used offensive hacking techniques." And then they say specifically, because they're really worried about what Sanger has said in his book: "Mr. Sanger suggests our '...investigators reached back through the network to activate the cameras on the hackers' own laptops.' We did not do this, nor have we ever done this. To state this unequivocally, Mandiant did not employ 'hack back' techniques as part of our investigation of APT1, does not 'hack back' in our incident response practice, and does not endorse the practice of 'hacking back.'"

They say: "The conclusion that we hacked back, while incorrect, is understandable. To someone observing this video over the shoulder of one of our investigators, it could appear as live system monitoring. Nevertheless, Mandiant did not create these videos through hacking back or any hacking activity. All of these videos were made through information obtained via consensual security monitoring on behalf of victims' companies that were compromised. The videos Mr. Sanger viewed were from Windows Remote Desktop Protocol network packet captures of Internet traffic at these victim organizations." Okay. "Mandiant has never turned on the webcam of an attacker or victim system. In short, we do not fight hackers by hacking, but by diligently and legally pursuing attribution with a rigor and discipline that the cause requires."

Okay. So naturally this has caused quite a furor. And in an emailed response, Dave Sanger responded to Mandiant's statements, Mandiant/FireEye's statement, saying: "Mandiant gave us" - meaning him - "extraordinary access to their investigation as we were preparing to write about Unit 61398 in late 2012, and the result was our story in the Times, and the company's report, in February of 2013."

David wrote: "I spent considerable time with their investigators and saw the images of the hackers as described in [his book] 'The Perfect Weapon.' Mandiant now says that all those images came from 'consensual monitoring,' in other words, that everything they received, from code to message traffic to imagery, was visible because the hackers themselves were transmitting them in the course of breaking into the systems owned by Mandiant clients. While that wasn't my understanding at the time, passive monitoring is a reasonable explanation of how the company came to link the hacks to specific individuals, several of whom have since been indicted by the United States."

Okay. So that's the end of David's response. And then in their conclusion Bleeping Computer said that some industry insiders - oh, actually I think this was some coverage in CyberScoop, yes. Some industry insiders told CyberScoop they were less than shocked by the claims. Broadly speaking, hacking back to gain novel insight into an attacker is fairly well known, despite it being illegal under U.S. law. The practice is often the subject of commonplace rumors amongst those in the industry. So just sort of an interesting look into the world and how it works.

People who've been listening to this podcast for a long time, and I think this actually occurred before the podcast, back in the early days when GRC was suffering its first DDoS attacks, may remember that I did my own forensic sleuthing in order to figure out who was behind the DDoS attacks that we were suffering. And this is a long - this is, like, 15 years ago. And I tracked down somebody whose handle was Wicked, learned that his name was Michael, and talked to him on the phone and said, "Hey, knock it off."

So it is possible to do this stuff without hacking back. But it sure sounds like, given David Sanger's reputation, I'm inclined to believe that he was shown some cool things. And who knows. You would expect that savvy hackers would have black tape over their own cameras. Maybe they don't because they were using them for video conferencing or videoing with their girlfriends. Who knows.

Leo: Well, that's kind of why I believe Mandiant's story, because that would make more sense, frankly.

Steve: Yeah.

Leo: Because you're right. I mean, now, even maybe in those days they didn't put black tape over cameras. Maybe that's a new thing. Why even have cameras; right?

Steve: I agree. They're just, well, I need one to see you, Leo.

Leo: Right. That's all you need.

Steve: And that's all I use it for.

Leo: That's all you need.

Steve: I turn it off after we're done and get back to work.

Leo: Do monitors now routinely have cameras on them?

Steve: Yeah. I think every laptop.

Leo: Laptops I know. But monitors, I mean. If you have a tower computer, you're not going to have a camera.

Steve: I'm not aware of it. I have - I'm looking at one, two, three, four, five, six, seven, eight. I have got monitors around me, and not one of them has a camera.

Leo: Yeah. I mean, I guess some might because it's a convenience. But, yeah.

Steve: Yeah. Okay. So, boy. As I said at the beginning of the year, when we first ran across the Spectre and Meltdown problems, this felt like something that would be going on and on and on because it was such a fundamental mistake. It was something that wound the industry up because it wasn't something you could easily patch. Well, another technology already exists and is evolving and has hit the same sort of problem, well, yeah, essentially the same problem with Meltdown, Spectre, and now potentially TLBleed. And that's something that we've spoken about, an evolving technology that was adopted very quickly across the web browser space, which is WebAssembly. WebAssembly solves the JavaScript performance problem by creating a - by shifting the compilation away from the browser so that the browser receives a precompiled, almost ready to execute blob of code.

Traditionally, as we know, JavaScript is ASCII. It's text. And so the browser downloads this text. And then in the browser the compilation process/interpretation, the interpreters have gotten so fast that they now do sort of incremental, or it's called JIT, Just In Time compilation, only they're not bothering to compile or deal with code that they haven't encountered yet. But as they do, they compile it on the fly in order to then be able to execute it repetitively, quickly. You can think of it as sort of a caching approach for compilation.

Well, the problem is that takes time, and people want their pages to appear in front of them and run immediately. So WebAssembly allows, first of all, different high-level languages to be used to compile to a common .wasm is expression, .wasm for web asm file, which is a binary file. It has a text representation so that you can actually read the WebAssembly. But what the browser downloads is a binary blob which allows I to just run immediately.

So it's very popular. I think it started, the work began actually well before it was on our radar, back in 2012. But just last year it got adopted across the board - Chrome, Firefox, Edge, Safari, all the major browsers now support WebAssembly, which is a faster adoption than we've typically seen with new technologies, probably because it's so performance oriented. No browser - as we know, they all compete on performance. No browser wanted to be the one didn't support WebAssembly and therefore suffered from benchmark comparisons against all the others. So anything that could make the browser faster, they all jumped on.

The problem is that it breaks, WebAssembly inherently breaks the mitigations which the browser vendors immediately put in place at the beginning of the year to protect us from Spectre and Meltdown. As we know, the problem with Spectre and Meltdown is cross-process leakage, where a hostile entity would have some way of getting into or running code on a core which is shared by others. And as I've been saying, this isn't a problem really for end users because, if you've got hostile code in your computer, it's already in your computer. The big threat was in cloud or shared hosting environments, where one of the hosted systems could be malicious and could be working to obtain secret information from other entities that are hosted in the same environment.

Well, one place where end users do have exposure is when they're surfing the web and downloading code into their browser, which is running in order to do the things that we want our browsers to do now. So there you are constantly running code from third parties. It's not technically on your computer. It's contained in your browser. And as we know, lots of work has gone into maintaining the sandboxing of the browser, maintaining that containment. But it is a place where we're constantly exposed to the potential of threats.

So the first thing that the browser manufacturers did was they fuzzed the time that was available to threads in JavaScript. There are APIs which used to be high resolution, where the JavaScript could say, "What time is it," and it would get back an instantaneous timestamp which was as accurate as the system was able to provide. Typically, extremely accurate. And then the JavaScript would say, okay, thanks very much.

Well, by taking a snapshot of now, and then taking a snapshot again after setting up a bunch of circumstances, the JavaScript could detect how much time elapsed between now and then, and use that to infer, believe it or not, the content of caches globally that is outside of the browser, thus creating a trans-browser leakage of information. So because you could argue the browsers don't need super high resolution, like script running in a browser, JavaScript doesn't have a clear need for super high-resolution timing or super-accurate timing, deliberate jitter was added, and the resolution of the API's returned results was reduced so that JavaScript could not know exactly what time it was inside of its own code execution. So it's like, okay, whew, problem solved; right?

Well, it turns out that there was, again, people are clever. There was another technology which allowed high-resolution access to time indirectly. And that is, in JavaScript you can have workers, which are essentially threads, individual threads of execution, which are going about doing their own thing in the code. So what developers worked out and security people worked out was a backdoor, essentially, to high-resolution timing, not using timers. There's something known as a Shared Array Buffer, SAB for short, Shared Array Buffers in JavaScript, which is a means of allowing communication between these worker threads in JavaScript. They share an array of memory.

Well, if one of them has the job of doing nothing but incrementing a value, incrementing a counter in one cell of this shared array, then another thread that wants an accurate timing can read repetitively from that rapidly incrementing value in order to recover high-resolution timing information which has otherwise been deliberately hidden from code running in browsers.

So what happened is that the WebAssembly project has a roadmap of the features that they are planning to implement in the language, and they had not yet gotten to implementing shared array buffers as the language spec is maturing over time. And in an interesting back-and-forth discussion that I saw online and saw some coverage of, they ended up deciding to punt. Their feeling is there is no clear way of allowing shared array buffer support in WebAssembly where the whole point of WebAssembly is running full out, I mean, just absolutely as fast as you can.

Oh, and I forgot to mention that in digging around in this, I ran across a comment that just made me smile, which is there is no browser-based crypto miner which does not use WebAssembly. Which how many times have I just scratched my head, saying I just can't believe - you just can't do crypto mining in JavaScript. Well, it turns out, right, nobody does. All of this cryptocurrency mining is courtesy of WebAssembly because it is as close to native code execution as you could get. So that explains to me and for the record, for all of our listeners, how it is possible to do cryptocurrency mining in a web browser. Well, it's actually essentially native code, courtesy of WebAssembly.

And so the point is that the only way the WebAssembly developers can see to prevent essentially cache-busting timing would be to somehow stall the WebAssembly threads so that they just can't know whether they were explicitly stalled or it took longer as a consequence of a cache miss. The problem is stalling WebAssembly threads is exactly what you don't want to do. You don't want to cripple the performance that you've created WebAssembly for in the first place.

Leo: I'm surprised, though, that something running in a virtual machine could execute these kinds of timing attacks.

Steve: I know. I am, too. And in fact there have been proof of concepts done, and I've seen code snippets where one workers just sits there and does nothing but just as fast as it can increments a counter. And another worker thread reading that counter is given enough information about the passage of time in order to do cache-busting, so much so that the Shared Array Buffers have been removed from JavaScript and will not be implemented by the WebAssembly people. And everybody is just, like, basically they're saying we want this, but we can't offer it safely at the moment. We're hoping that Intel and/or the OS vendors working together, basically we're hoping for some other resolution to this problem.

And the big concern is, with this advent now of the TLBleed, which is hyperthreading, I think the only solution there is going to be to keep the threads in the same process so that there isn't any - so that adjacent hyperthreads can't be running in different processes. That's the only solution I see there. And I just think we're going to have to tolerate a performance hit in the short term until we end up with next-generation processors.

And as we know, Leo, Intel's got a deep pipeline of processor architecture. There's no question, if they've had to scramble around and change the microcode in as many of their existing chips as they have, that there was some immediate design impact on the things that were further down the production line and probably had to get pulled back in order to have some features added in order to solve this problem in the right way. So serious impact on our industry.

And I think this is the last big piece of information before we get on to our miscellany stuff and then talk about the Supreme Court. And this is Firebase. Firebase, Inc. is a mobile and web application development platform which was created by, well, Firebase itself, created by Firebase, Inc. seven years ago in 2011, which was acquired by Google four years ago in 2014. So they are now at Firebase.Google.com and have a very nice professional-looking website. And they claim that Firebase helps mobile apps to succeed.

They say: "Build fast apps without managing infrastructure. Firebase gives you functionality like analytics, databases, messaging, and crash reporting so you can move quickly and focus on your users. Backed by Google, trusted by top apps." They say: "Firebase is built on Google infrastructure and scales automatically for even the largest apps."

So essentially what this is, is a cloud-based database service where, rather than asking applications which are already connected apps, that are already cloud-based, mobile, connected, Firebase is saying, we'll store all your data for you. You can offload all of that stuff, and we'll handle it. So they have something called Cloud Firestore, which is store and sync app data at a global scale; Cloud Functions, which allows to run mobile backend code without managing servers. They have authentication, and therein lies the rub: "Authenticate users simply and securely." Which turns out to, unfortunately, not be as true as we and the users of these apps would wish. Also Cloud Storage: Store and serve files at Google scale.

So the firm whose name I could not remember at the top of the show is Appthority, which is kind of a clever name, authority, but Appthority. And they've got, Leo, probably one of the best-named vulnerabilities around. They're looking at data leakage through backend stores.

Leo: Oh, dear. Oh, dear. I'm worried now.

Steve: Uh-huh. They called it...

Leo: Backend data leakage would be really a problem. What did they call it?

Steve: They called it "HospitalGown" vulnerability.

Leo: Oh, dear. Oh. Well, that's better than it could have been.

Steve: Yeah.

Leo: Data leakage? You're sitting in it. No, no, no.

Steve: So they said: "In 2017 the Appthority Mobile Threat Team discovered" - actually and they named - "the HospitalGown vulnerability. The vulnerability, named for data leaking through backend data stores that are unsecured, results from app developers' failure to properly secure backend servers with firewalls and authentication, leading to data exposure." They wrote: "Our initial report in May 2017" - so just about a year ago - "revealed that weakly secured backend databases were being accessed via apps used by employees, partners, and customers and resulted in numerous security risks including extensive leaks of sensitive data, easier data access and exfiltration, and increased risks for spear phishing, social engineering, data ransom, and other attacks."

Okay. And they go on to update us basically with their new report, which was just released, focusing on this Mobile Threat Team's latest discovery, which is "a new variant of the HospitalGown vulnerability, which occurs when app developers fail to require" - it's hard even to believe I'm saying this - "app developers fail to require authentication to a Google Firebase cloud base." And then they explain: "Firebase is one of the most popular backend database technologies for mobile apps but does not secure user data by default." It's like, what?

Okay. I'll say that again. "Firebase is one of the most important backend database technologies for mobile apps, but it does not secure user data by default, does not provide third-party encryption tools, or alert developers to insecure data and potential vulnerabilities. To secure data properly, developers need to specifically implement user authentication on all database tables and rows, which rarely happens in practice.

Moreover, it takes little effort for attackers to find open Firebase app databases and gain access to millions of private mobile data app records." And then they go on to explain how easy it is to find this. I won't bog us down in those details.

But in the show notes, Leo, I have a table from this report, from Appthority's report. It's a grid showing total apps with Firebase IODBs, Android predominantly. For example, there's a total of a little over 28,000. 27,277 are Android; 1,274 are iOS. Then the number of the total 28,502 which were vulnerable, a little over 3,000 - I'm sorry. Of the 28,502 total apps using Firebase, a little over 3,000 of those were vulnerable. I'm just...

Leo: The big one is that half of all iOS apps that use Firebase are vulnerable.

Steve: Yes.

Leo: That's kind of stunning.

Steve: Yes, yes.

Leo: Now, I gather that Firebase is very popular.

Steve: So, yes, it's very popular.

Leo: Geez.

Steve: I know. Okay. So what did they find when they looked? Under Scope of Threat they explain. Out of a total of, now, they analyzed 2,705,987 apps. That's from which they found 27,227 Android apps and the 1,275 iOS apps that use the Firebase database. So a lot of them do. Of those connected apps they found that one in 11 Android apps, that's 9%, and almost half of iOS apps, 47%, that connect to a Firebase database were vulnerable. As you said, Leo, almost half of iOS. And still, one in 11 of Android. And actually, because it's so much a stronger usage in the Android platform, the number, the gross number of Android apps is higher. More than 3,000 apps were leaking data from 2,300 unsecured servers. Of these, 975 apps were in active customer environments. One in 10 of the Firebase databases are vulnerable.

So Firebase, this company that's making this service available, 10% of their databases are vulnerable. Vulnerable Android apps alone were downloaded over 620 million times. So these are not obscure apps. These are popular apps. Over 100 million records containing 113GB of data has been exposed. They said most organizations were affected. They found 62% of enterprises have at least one vulnerable app within their mobile environment. No type of organization was immune. Organizations with data exposed to this HospitalGown variant, as they call it, the Firebase variant, included banks, telecommunications companies, postal services, ridesharing companies, hotels, and educational institutions. And there's no geographical preference or boundaries. The U.S., Europe, U.K., Argentina...

Leo: Argentina.

Steve: I always say that wrong. I get an extra word. I paused because I knew I was going to get it wrong. Argentina, Brazil, Singapore, Taiwan, New Zealand, India, and China all exposed.

Okay. So what about data? They looked. 2.6 million plaintext passwords and userIDs. More than 4 million protected health information records, including chat messages and prescription details. 25 million GPS location records. 50,000 financial records, including banking, payment, and bitcoin transactions. More than 4.5 million Facebook, LinkedIn, and Firebase and corporate datastore user tokens. And a lot of it is regulated. Some of the data leaked includes highly sensitive private information subject to regulatory requirements, including HIPAA and the GDPR and PCI.

Medical information: Chat messages between patients and pharmaceutical sales representatives, together with their prescriptions and orders, leaked from a Mexico-based pharmaceutical app. Information about medical consultations, medical history, and diagnostic information was also leaked. Information was protected, of course, by HIPAA regulations and subject to breach notification requirements. They found sensitive personal data, email addresses, phone numbers, full names, geolocations, and Facebook OAuth tokens were all leaked in violation of data privacy protection laws such as the GDPR.

Vehicle license plate numbers: Some apps exposed vehicle license and registration numbers as well as geolocation data. California, as we know, has numerous data privacy laws, some of which require companies that expose names accompanied by license plate numbers and other automobile details to disclose these breaches. And credit card numbers, which were exposed, subject to PCI DSS regulations. And on and on and on. So personal data. Sensitive enterprise data.

Anyway, I'm glad that this has come to light. I'm sure that Google's owned Firebase property will get on this, if they haven't already, and get this cleaned up. It's clear that, if 90% of applications did employ security, you'd have to consider it negligent that one in 11 Android apps and certainly half of iOS apps did not do so. It's cool that there's this service that allows apps to offload all of these functions to the cloud. But it's just sort of stunning that in this day and age this information would be stored in the clear.

I didn't go into the details, but you're able to easily find this on the Internet and add like a /.json file extension and immediately begin retrieving data, essentially in plaintext, containing all of this. So yet another example of I guess the triumph of convenience over security. It's very convenient to do this, but stunning that this company would not enforce security to be used for anyone who is using their service.

Okay. So I had a little adventure. As our listeners know, my trustee Windows XP machine croaked late Sunday night, early Monday morning a week ago. And I've been picking up the pieces ever since. I'm now in front of that system that our listeners will remember I built about two and a half years ago, when Microsoft had announced that they were no longer going to support Windows 7 on newer hardware. The uproar from that, after I immediately purchased the hardware that would still be supported by Windows 7, Microsoft said, whoa, whoa, whoa, that's not quite what we meant. There was a lot of pushback from enterprises, and they have since said, okay, we'll go further. I think it was Sandy Bridge was the last architecture they said they were going to support.

So since that was 15 years newer than the system I was using, I built up a beautiful machine. But because I knew it was going to be a pain in the butt in order to have downtime to switch over, and XP was still working for me, and I've got lots of my own layers of security surrounding me, I've just continued using it till it died.

Okay. So on Thursday morning I brought up the RAID that had been down since Monday and got access to the very latest snapshot of all of my files. So I lost no data whatsoever. I had images and things, but it's good to have the last instance of everything. And then I immediately rebuilt my toolchain, and I can now reassemble SQRL and the Net Engine, and I'm still in the process of deciding what I want to do about an editor because I've decided, okay, it's time to give up completely on using Brief in a DOS box, even though there are a number of, you know, Brief was so popular that there are a number of updated modern clones of it. But it's like no, no, no, it's time to - since I'm doing this, it's time to move on.

But there were some licenses that I had on software on that machine that could not be extracted except using the running live program. And they were expensive. So I thought, oh, crap. I need to boot this. So I thought, okay, well, I have the C drive. It's sitting here. I'm looking at it. And some people in the GRC newsgroup where I've been hanging out, in the SQRL newsgroup, suggested that I just move this to a virtual machine, bring it into a VM. And I thought, really. Okay, except wait a minute, this is an old Windows XP OS that has all of its drivers, I mean, it's booting a RAID, so it's got a RAID driver. And I've got different graphics cards in it than I have now, and on and on and on. And anyone who's used Windows long enough knows that you really, if you change the motherboard out from under an installed Windows system, you're in for some pain. I mean, it maybe won't even boot at all. Or, if so, you boot in safe mode, and nothing really works and so forth and so on.

Anyway, so I documented in the show notes a page that I found. First of all, lots of information about this sort of stuff. But normally what people are doing is they're cloning a running system into a VM. That is, and there are plenty of tools you can use for capturing the current system and creating a virtual machine image from that. Unfortunately, that opportunity was foreclosed. That is, I had the C drive, but I had it as just a mounted file system. I didn't have it running. So most of what I found when I was digging around the 'Net for how to pull this off didn't apply.

I did find a page at UltraGeek.com which was exactly what I needed. So I have a link in the show notes for anyone who's interested. And doing this, to give you a sense for what was necessary, was a multistep process. First of all, the only tool that I could find which this page pointed me to was by our old buddy Mark Russinovich from Sysinternals. Mark created something called Disk2VHD. VHD is Microsoft's version of - VHD, as the acronym sounds, is a virtual hard drive. So Mark has, and there's a free, you can download it from the Sysinternals site, sysinternals.microsoft, Disk2VHD. You give it a mounted drive letter like I had, in this case it was Q, and it will virtualize that drive letter, creating a Windows VHD. Okay, so that's the first step.

Leo: That's handy. That's neat.

Steve: It is very cool.

Leo: Cool tool, yeah.

Steve: Very cool. Okay. But I wanted to use VMware. And in fact VMware has the third tool that I need. So I needed to convert a VHD into VMware's version, which is a VMDK. So there's a company called StarWind that has what they call a V2V converter. That is, it is an inter-virtual machine converter, and it's free. They get an email address from you for the privilege, but fine. I gave them my Gmail account. So it's a free tool called

StarWind V2V Converter. And I used it - by the way, this was a 93GB file. So none of this was fast. But I wanted it. So it converted the VHD to a VMDK.

So that got me over into VMware land from Microsoft land, which is where Mark's tool allowed me to get. Then VMware has something called the vCenter Converter Standalone, VMware vCenter Converter Standalone, which again I downloaded from VMware at no cost. And that's the miracle. And I am so impressed with this thing. You give it a virtual machine image, which is what StarWind V2V converter produced. But of course that wouldn't boot because, I mean, it's got all of the Windows drivers and configuration and everything for a motherboard which is long since dead. Well, actually not long since. By that time it was four days.

Leo: Long in seconds.

Steve: But it's 15 years old. So I'd be hard pressed to find another one. And it was never really clear, didn't seem to be the power supply. I think it was the motherboard. And it had been kind of dying slowly, like the PS/2 port on the back died, so my clanky keyboard had to have a USB interface in order to still get used. I mean, it was sort of showing signs of getting tired.

Anyway, this thing, this VMware vCenter Converter Standalone, it's able to reach into an offline virtual machine image and change the drivers, to go into the registry which is offline, it just exists there in this virtual machine image, and go in and change the drivers to the drivers that VMware environment offers - IDE, SCSI, keyboard, video, I mean, everything, USB - and install them into the image and then insert them into Windows. So this thing that would have never in a million years booted in VMware, can. So once it was through, I just kind of scratched my head, thinking, okay, can this possibly work? And I fired it up in VMware, and it was like an old friend coming back to life. It was like, oh, there's my world.

Leo: You know, it will probably still be faster on that new machine than it was on the original hardware.

Steve: Oh, it was wonderful.

Leo: And I bet it's faster.

Steve: It booted up much faster. And then I was curious, I opened up Device Manager. Not a single wrong device. All of the old things that were not there had disappeared. There were no yellow triangles warning you of anything unknown. It was a miracle. Anyway, I just wanted to share it with our listeners. Just if at any time in the future you find yourself, like you have a file system, a bootable file system from where, somewhere, doesn't matter when because this thing goes back to Windows 95, you can bring it back to life. It's amazing. It's wonderful. Oh, and you can use VM Player. So again, I think VM Player is free. I have a full VM Workstation because that's one of the reasons I got this big monster machine was I wanted to be able to run multiple VMs with different environments and so forth, and have Linux and so forth.

So anyway, I just wanted to share my adventure and my amazement that this thing works. Oh, and I did, I fired it up, and I was able to recover the licenses. And also I was able to - I have a couple very mature registry trees. And so by bringing the machine back up and being logged in, and then I'm able to export the registry and then was able to import that registry tree into my new machine, which is not something you can do from just the file system laying there because it's all in reg files. I've wanted to do it in the past, and it's just you can't.

So anyway, it's very cool. It's very possible now, and not spend any money, to bring up an old drive under any of these new virtual machines. And I'm sure that - I didn't look at all the other V2V things. I only wanted to go from the Microsoft virtual hard drive to the VMware. But it looked like it supported a number of different options. So anyway, I'm impressed with where we are with virtualization these days. This may be old news for some, but I just wanted to make sure that our listeners knew about it because it was - I was just amazed. It was really cool.

And lastly, I saw a bit of feedback that I wanted to share. It's sort of feedback/errata. Ciprian in Germany, his subject was "Huge Cisco Security patches number." And his English has a German lilt to it. Anyway, he said: "Hello. Cisco released again tons of patches for big iron routers." He says: "I got 24 individual emails. What I wanted to remind is that Cisco had, is, and will acquire companies for their products; and in those purchasing, not all the backdoors are known. My 2C. Regards." Oh, and he also said: "P.S.: Currently SpinRiting a huge 24-drive Dell enclosure to see what I can reuse."

Anyway, I thought it was a very good point. I was a little rough on Cisco a couple weeks ago, just scratching my head, wondering how in the world you could possibly explain, like what was the culture there that they would have all these unknown, apparently even to them, or certainly to officialdom, backdoors in their products and then be launching, to their credit, audits of their own stuff and finding things they didn't know about, and in some case having other people finding things.

And so I thought this was very on point. And so I thanked Ciprian for this, that Cisco is acquiring companies. It's very much like FireEye, who bought Mandiant. If that hacking back of China occurred, it was before FireEye owned Mandiant. So you really can't hold FireEye responsible, even though Mandiant is now their employees. Similarly, things you acquire from other people, well, you get what you get.

And I did want to mention that through the years a lot of people have used SpinRite as Ciprian is. You get drives of unknown background and heritage, you run SpinRite on them to just make sure they're okay before you decide that you want to put them into use. So anyway, thanks for the feedback, Ciprian. And I'm really glad for the observation that these may not have been homegrown backdoors as a consequence of some bizarre culture at Cisco. They could have just been acquired.

Leo: We didn't write the backdoor, we just bought it.

Steve: Yeah, that's right. Handy little thing to have.

Leo: All right. Back to you, Steve.

Steve: Okay. So as we know, and this has been in the news a lot recently, increasingly local police and other law enforcement agencies have been obtaining court orders to compel local cell carriers to disclose the location data of people they suspect of crimes. But significantly, these have not been probable cause search warrants as permitted under the Fourth Amendment to the U.S. Constitution. They've simply been requests from courts for the business record documents of cellular carriers. And in previous decisions it's been held that business records are subject to court order and can be obtained by law enforcement for whatever purposes they wish.

Okay. So there was a burglar ring operating, I think it was in 2011 they were caught. The ringleader was a guy named Timothy Ivory Carpenter, who was accused of planning these crimes. I guess some of his other cohorts fingered him as the boss, and so he got in trouble. He petitioned, I mean, so he was first convicted from - I should explain that the FBI, after they got his name, got a court order to cause the cellular carrier in the region to turn over the records of his cell phone location, which did identify him as being present in the location and time of four of the burglaries. He was found guilty in court, sentenced for like a huge number of years, more than several lifetimes, as I recall.

He appealed, and the appeal was heard. The appellate court said, no, we think this is proper, so sorry about that. And it went to the Supreme Court. The decision of the Supreme Court a day or two ago came down and overturned what the appeals court and the lower courts had said. And this was interesting enough, and I thought would be interesting enough to our listeners, that I've pulled together sort of a summary of this because I wanted to go over exactly what this is that was decided.

Chief Justice Roberts delivered the opinion of the court. And from the document which I've excerpted from it reads: "This case presents the question whether the Government conducts a search under the Fourth Amendment when it accesses historical cell phone records that provide a comprehensive chronicle of the user's past movements."

Okay. So I'll skip over what we already know from a technology standpoint, but I'll just say that it opens with a clear and perfect succinct explanation of cell tower location, how the increasing popularity of cellular communications has caused providers to increase the density of towers and thus increases the accuracy of tower-based location, approaching that of GPS, especially in urban areas. And I do have in the show notes at the beginning of this a link to the PDF of the full, it's 158 pages, I think it was, or that might have been one of the other PDFs, it was like at least 119, of the opinion.

Anyway, so here's the summary of the decision. It reads: "Cell phones perform their wide and growing variety of functions by continuously connecting to a set of radio antennas called 'cell sites.' Each time a phone connects to a cell site, it generates a time-stamped record as Cell Site Location Information (CSLI). Wireless carriers collect and store this information for their own business purposes.

"Here, after the FBI identified the cell phone numbers of several robbery suspects, prosecutors were granted court orders to obtain the suspects' cell phone records under the Stored Communications Act. Wireless carriers produced CSLI" - that is, the Cell Site Location Information - "for petitioner Timothy Carpenter's phone, and the Government was able to obtain 12,898 location points cataloging Carpenter's movements over 127 days, an average of 101 data points per day. Carpenter moved to suppress the data, arguing that the Government's seizure of the records without obtaining a warrant" - that is to say a search warrant - "supported by probable cause violated the Fourth Amendment.

"The District Court denied the motion, and prosecutors used the records at trial to show that Carpenter's phone was near four of the robbery locations at the time those robberies occurred. Carpenter was convicted. The Sixth Circuit affirmed, holding that Carpenter lacked a reasonable expectation of privacy in the location information collected by the FBI because he had shared that information with his wireless carriers."

So what the Supreme Court held is, one, the Government's acquisition of Carpenter's cell site records was a Fourth Amendment search. Which is to say, qualifies under Fourth Amendment protection, and a search warrant with probable cause needed to be obtained in order for the cell site records to be had. They wrote: "The Fourth Amendment protects not only property interests, but certain expectations of privacy, as well."

And then they cite a previous case, Katz v. the U.S.: "Thus, when an individual 'seeks to preserve something as private,' and his expectation of privacy is 'one that society is prepared to recognize as reasonable,' official intrusion into that sphere generally qualifies as a search and requires a warrant supported by probable cause." Then they cite previous case Smith v. Maryland.

"The analysis regarding which expectations of privacy are entitled to protection is informed by historical understandings 'of what was deemed an unreasonable search and seizure when [the Fourth Amendment] was adopted.'" And again a previous case, Carroll v. U.S. "These founding-era understandings continue to inform this Court when applying the Fourth Amendment to innovations in surveillance tools."

Then they said, next paragraph B: "The digital data at issue, which is personal location information maintained by a third party, does not fit neatly under existing precedents, but lies at the intersection of two lines of cases. One set addresses a person's expectation of privacy in his physical location and movements," and then they refer to U.S. v. Jones, "where five Justices concluding that privacy concerns would be raised by GPS tracking."

Then they say: "The other addresses a person's expectation of privacy in information voluntarily turned over to third parties." Then they cite U.S. v. Miller and also U.S. v. Smith. And this comes back a couple times, these Smith and Miller cases, "where expectation of privacy in records of dialed telephone numbers conveyed to a phone company." And those were found not to be protected. That is, there was no expectation in using a telephone dialing numbers.

Then they say, the next paragraph, C: "Tracking a person's past movements through CSLI partakes of many of the qualities of GPS monitoring considered in" that Jones case, where it was decided that GPS tracking had privacy concerned. "It is detailed." So they're saying: "CSLI partakes of many of the qualities of GPS monitoring. It is detailed, encyclopedic, and effortlessly compiled. At the same time, however, the fact that the individual continuously reveals his location to his wireless carrier implicates the third-party principle of Smith and Miller. Given the unique nature of cell site records, this Court declines to extend Smith and Miller to cover them.

"A majority of the Court has already recognized that individuals have a reasonable expectation of privacy in the whole of their physical movements. Allowing government access to cell site records, which 'hold for many Americans the privacies of life,'" and they cite Riley v. California, "contravenes that expectation. In fact, historical cell site records present even greater privacy concerns than the GPS monitoring considered in Jones. They give the Government near perfect surveillance and allow it to travel back in time to retrace a person's whereabouts, subject only to the five-year retention policies of most wireless carriers. The Government contends that CSLI data is less precise than GPS information, but it thought the data accurate enough here to highlight it during closing argument in Carpenter's trial. At any rate, the rule the Court adopts 'must take account of more sophisticated systems that are already in use or in development,' and the accuracy of CSLI is rapidly approaching GPS-level precision.

"The Government contends that the third-party doctrine governs this case because cell site records, like the records of Smith and Miller, are business records, created and maintained by wireless carriers. But there is a world of difference between the limited types of personal information addressed in Smith and Miller" - that's the phone dialing - "and the exhaustive chronicle of location information casually collected by wireless carriers.

"The third-party doctrine partly stems from the notion that an individual has a reduced expectation in information knowingly shared with another. Smith and Miller, however, did not rely solely on the act of sharing. They also considered the nature of the particular documents sought and limitations on any legitimate 'expectation of privacy' concerning their contents. In mechanically applying the third-party doctrine to this case, the Government fails to appreciate the lack of comparable limitations on the revealing nature of CSLI.

"Nor does the second rationale for the third-party doctrine, which is voluntary exposure, hold up when it comes to CSLI. Cell phone location information is not truly 'shared' as the term is normally understood. First, cell phones and the services they provide are 'such a pervasive and insistent part of daily life' that carrying one is indispensable to participation in modern society," citing Riley v. U.S. "Second, a cell phone logs a cell site record by dint of its operation, without any affirmative act on the user's part beyond powering up."

And so they trim this a little bit, saying: "This decision is narrow. It does not express a view on matters not before the Court; does not disturb the application of Smith and Miller or call into question conventional surveillance techniques and tools such as security cameras; does not address other business records that might incidentally reveal location information; and does not consider other collection techniques involving foreign affairs or national security.

"The Government did not obtain a warrant supported by probable cause before acquiring Carpenter's cell site records. It acquired those records pursuant to a court order under the Stored Communications Act, which required the Government to show 'reasonable grounds' for believing that the records were 'relevant and material to an ongoing investigation.' That showing falls well short of the probable cause required for a warrant. Consequently, an order issued under 2703," which is what was cited before, "is not a permissible mechanism for accessing historical cell site records.

"Not all orders compelling the production of documents will require a showing of probable cause. A warrant is required only in the rare case where the suspect has a legitimate privacy interest in records held by a third party. And even though the Government will generally need a warrant to access CSLI, case-specific exceptions, for example exigent circumstances, may support a warrantless search," and they provide some citings, "reversed and remanded."

So, bottom line, and good news for individuals, in my opinion, is that the Supreme Court ruled that the acquisition of CSLI data, that is, the entire history of an individual cell phone's movement back through time, as far back as five years, can only be acquired by law enforcement if they have probable cause and are able to get a search warrant from a court in order to request the acquisition of those records from the cell carrier.

So I think the Supreme Court did us right in this case, and I'm glad that this will happen. There have been recently an increasing number of instances, I don't think I've talked about it on the podcast, but I've encountered it in reading, where law enforcement is just making sweeping requests for cell phone location information, just asking a court to say, hey, we're able to have this. So from now on they'll need to show probable cause before getting it. And I think that's probably a good thing.

Leo: And there you go. Although the disclaimer, you're not an attorney, so consult your attorney. This is not a legal judgment. This is Steve's reading of it all.

Steve: That's what the Supreme Court just said, yeah.

Leo: Yup. Again, consult your attorney. I'm sure they talked about it on This Week in Law in great detail on Friday. Some interesting results came out this Friday morning, yeah.

Steve is the host of the show, does it every Tuesday at about 1:30 Pacific, 4:30 Eastern, 20:30 UTC. If you want to tune in and watch live, you can. All you've got to do is go to TWiT.tv/live. You should also chat if you are watching live at irc.twit.tv. If you can't watch live, you can get on-demand versions. Steve has his version, audio version, plus full transcripts, which is really nice to read along as you listen, at his website, GRC.com. You'll also find all the other stuff Steve does, including his SpinRite, the world's finest hard drive recovery and maintenance utility. You'll find it at GRC.com.

We have audio and video at our website, TWiT.tv/sn. And of course you can always subscribe. Every podcast app in the world has Security Now!. All you've got to do is search for Security Now! and add it, and you'll get it the minute it's available every Tuesday afternoon. Steve, thank you.

Steve: Okay, my friend.

Leo: You said it was a light day, but it wasn't that light.

Steve: No. We'll be back next week with more news.

Leo: I'll see you then.

Steve: Thanks, buddy. Bye.