The MMU Side-Channel Attack

Description: This week, Leo and I discuss the completely cancelled (amid a flurry of serious problems) February Patch Tuesday; that it's not only laptop webcams that we need to worry about; the perils of purchasing a previously-owned Internet-connected auto; Chrome changing its UI making certificate inspection trickier; the future of Firefox add-ons; that Win10's lock screen is leaking the system's clipboard; a collection of new problems for Windows; an amazing free crypto book online from Stanford and New York University; pfSense and Ubiquiti follow-ups; a bit of geek humor and miscellany - and a deep dive into yet another sublime hack from our ever-clever friends led by Professor Herbert Bos at the University of Amsterdam.

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-600.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-600-lq.mp3

SHOW TEASE: It's time for Security Now!. Steve Gibson is here with a couple of fun photos, two Pictures of the Week - well, a cartoon and a picture. How processors are made. Oh, that's exciting. We also have a look at a new exploit that is unfixable on modern hardware. It's the side-channel MMU attack. Well, we'll find out what that is. Get your propellers and your propeller hats ready. It's time for a geek-out episode of Security Now!.

Leo Laporte: This is Security Now! with Steve Gibson, Episode 600, recorded Tuesday, February 21st, 2017: The MMU Side-Channel Attack.

It's time for Security Now!, the show where we cover your security and privacy online with the Explainer in Chief himself. Look, I can hold his head in my hand. Steve Gibson is here from the Gibson Research Corporation. I'm actually virtually holding your head in my hand. Steve joins us via Skype each week to talk the latest security news. And today it's an exploit. You like to talk about exploits; right?

Steve Gibson: Well, yes. So first of all, Episode 600.

Leo: Oh, wow, I forgot.

Steve: Yeah.

Leo: Wow.

Steve: Yeah, yeah, yeah. We've hit 600.

Leo: We're doing almost exactly 52 a year, with a few exceptions.

Steve: Yeah. Yeah.

Leo: So this is our - that can't be right - 12th year?

Steve: You know, it's going to really foul me up if we have to go to four digits. So I think we ought to just plan on 999, if that's where we're...

Leo: We'll stop in eight years? We'll stop? That'll be it? It'll be over?

Steve: I think we'll be done. We'll be fried by that point.

Leo: We'll be in our late 60s. There'll be a new president eight years from now.

Steve: That's probably the case, well, in all likelihood that's true.

Leo: Life will be very changed. I mean, imagine what we've seen in eight years of technology evolution.

Steve: Oh, yeah, yeah, yeah. I mean, during the 12 years of this podcast we've seen, you know, we've gone from the quaint little email macros...

Leo: Honey monkeys, or what was that, yeah.

Steve: Right.

Leo: Melissa.

Steve: To what we're going to discuss today is yet another piece of sublime engineering from our friends who are led by Professor Herbert Bos at the University of Amsterdam. They're the people who gave us Flip Feng Shui and Drammer. And they've figured out how to, in JavaScript in a browser, leverage the innate functioning of the memory management hardware which is present in all contemporary CPU architectures in order to break, to robustly break address space layout randomization.

Leo: ASLR. Oh, wow.

Steve: Yeah. So anyway, so this is, okay, you may be able to walk and chew gum at the same time, but you're not going to be able to listen to this podcast, the latter half of this podcast, AND walk and chew gum. So choose two out of those three.

Leo: This is what we call a propeller hat episode.

Steve: This is going to be, yes, we're going to lose some people. I don't want to. And I'll take this easy.

Leo: Take it as a challenge, my friends. Can you continue to listen to this show?

Steve: Yeah, so stop chewing or stop walking. Maybe this is where you put your car on auto drive because, boy, this is going to take - this is going to need everything you've got.

Leo: Holy moly.

Steve: But I think everyone's going to enjoy it. And we have a bunch of interesting news. We've got Microsoft's February Patch Tuesday changed from "delayed" to "entirely cancelled," amid a flurry of serious problems. They updated, saying, oh, we're not even going to do it this month. Wait till March.

Leo: What? They canceled it?

Steve: They completely canceled it. So, I mean, and it's not because there wasn't anything in need of fixing. There's like some serious problems. We've got the SMB zero-day flaw still out there flapping in the breeze, and a couple other things that we'll talk about today.

It turns out that it's not only laptop webcams that we need to worry about. Oh, and a really interesting story, actually from an IBMer through the RSA Conference, about the perils of purchasing a previously owned, Internet-connected auto.

Leo: Whoa.

Steve: Yes. Chrome 56 changed its UI, making certificate inspection more difficult. We've got a workaround for that, at least for Windows. I don't know if it works on other platforms. Some news about the future of Firefox add-ons, which is a definitely mixed blessing. It turns out that Windows 10's and also 8.1's lock screen is leaking the system's clipboard data, and Microsoft says they don't care.

Leo: Oy.

Steve: There's the collection of new problems for Windows. An amazing free crypto book online from Stanford and New York University professors that we'll talk about. Some quick follow-ups to pfSense and Ubiquiti that we've been talking about recently. A fun bit of geek humor, a little bit of miscellany, and then a deep dive into yet another sublime hack from our ever-clever friends from the University of Amsterdam, who are that team that Herbert Bos leads.

This just came out of - actually it came out of, well, it went public last Tuesday and just totally ate up my Twitter feed. All of the press got it wrong of course because, as I said, this is a wind-up your propeller. So we'll talk about that. And so many people were tweeting, saying, do you know about this? Do you know about this? And it's like, yeah, I haven't been able to say anything about it because it was embargoed until they made news. And so I've been waiting to talk about it because, again, it's just, wow, another beautiful piece of work. So I think we've got a great two hours for our listeners.

Leo: Wow. All right. Let's hit the week's news. I guess that's next.

Steve: Well, we do have our Picture of the Week.

Leo: Oh, of course we do.

Steve: Which is a fun little cartoon. It shows a married couple, husband and wife. He's in front of his laptop, and she's sort of looking over his shoulder. And he says, in the caption, "Of course this website is safe. As an extra measure of security, they make you sign in with your Social Security number, mother's name, your bank account, home address, phone number, and date of birth."

Leo: What could go wrong?

Steve: How could a bad guy ever find out all those things? So of course it's secure.

Leo: Yeah.

Steve: That's right.

Leo: And then they store it.

Steve: Yeah.

Leo: For future reference.

Steve: That's right. Just to make sure you don't change anything. So as we mentioned at the top of the show, the February 2017 Security Update was formally canceled. In an update that Microsoft Security posted later that Tuesday, when we referred to it a week ago, they formally said "Update on 2/15," so it was Wednesday, the day after, we will deliver updates as part of the planned March Update Tuesday, March 14, 2017. So don't be holding your breath, anybody. I had had it on my mind because I know that there are so many significant pending issues. And the presumption was that, well, in fact, they originally said that, you know, they were a little obscure. It was like, uh, something has just recently come to our attention that we need to deal with, so we're going to be delaying the update. Well, yeah, a month.

And actually it may have been just the fact that they've established this monthly cycle, that there's too much blowback, even at the expense of delaying a fix for something important, too much blowback about an out-of-cycle patch that they said, well, you know, it's going to upset too much of corporate customers for us to just drop this thing a week or two late. So we're just going to stay with the plan, stay with the normal Patch Tuesday cycle. I mean, because I imagine they must have these problems fixed by now, but they just - they must have decided it's too much of a problem for us to do it out of cycle.

Leo: I mean, we don't know, obviously.

Steve: We don't know, yeah.

Leo: They haven't said.

Steve: Oh, yes. I mean, some of these problems, they're just - they're not difficult to fix. I can't figure out what's going on. So there's a U.S.-based industrial cybersecurity firm named CyberX Labs in Framingham, Massachusetts. Their blog disclosed the results of some of their research, which was interesting. Their posting was titled "CyberX Discovers Large-Scale Cyber-Reconnaissance Operation Targeting Ukrainian Organizations." And in their blog they wrote: "CyberX has discovered a new, large-scale cyber-reconnaissance operation targeting a broad range of targets in the Ukraine." And it's funny, too, because I think I heard you, Leo, talking about Ukraine versus The Ukraine?

Leo: Yeah.

Steve: And, like, why does...

Leo: They don't like "The Ukraine" anymore.

Steve: Yeah. And I thought, okay, so I'm trying to say Ukraine, not The Ukraine.

Leo: It's hard. We've been trained for years to say The Ukraine.

Steve: Yes.

Leo: You don't say The Italy or The France.

Steve: Precisely. Or The America. But we do say The United States.

Leo: Oh.

Steve: So I was sort of wondering if somehow its history of The Ukraine is connected to The United States.

Leo: I think it's connected to The Soviet Union and not wanting to be seen as a region of the Soviet Union, but instead a whole country.

Steve: Right. So anyway, I was self-conscious about the use of "the" throughout their blog posting. And I thought first, before I realized they were in the U.S., I thought, oh, well, maybe this is a definitive non-U.S.-biased source of where "the" should be used.

Leo: I have been told in no uncertain terms, there's no "the" there.

Steve: Interesting.

Leo: It's Ukraine.

Steve: Interesting. So range of targets in Ukraine is what it should say, not the Ukraine.

Leo: That's correct.

Steve: In Ukraine. Okay, well, so forgive me for any "the's" that are in their blog posting, but I didn't put them there.

Leo: We'll send them a letter, an angry letter.

Steve: It's good to know that, definitively, no more "the."

Leo: Right.

Steve: So they say: "Because it eavesdrops on sensitive conversations" - and here's the point, or one of the most scary factors - "by remotely controlling PC microphones in order to surreptitiously 'bug' its targets, and uses Dropbox to store exfiltrated data, CyberX has named it 'Operation BugDrop.' CyberX has confirmed," they write, "at least 70 victims successfully targeted by the operation in a range of sectors including critical infrastructure, media, and scientific research. The operation seeks to capture a range of sensitive information from its targets, including audio recordings of conversations, screenshots, documents, and passwords." So this is a comprehensive trojan of some sort.

"Unlike video recordings, which are often blocked by users simply placing tape over the camera lens, it is virtually impossible to block your computer's microphone without physically accessing and disabling the PC hardware." Which, as we know, is true. Maybe disable the media driver, the hardware-level driver to access the microphone. But that's not obvious. And covering the microphone hole, if you could even find it, with a piece of tape, well, that might muffle the audio a bit, but it's not like blocking it the way you can sticking tape over the camera.

So they said: "Most of the targets are located in the Ukraine, but there are also targets in Russia" - not The Russia - "and a smaller number of targets in Saudi Arabia and Austria. Many targets are located in the self-declared separatist states of Donetsk" - I didn't practice pronouncing this before.

Leo: Don't look at me for help, buddy.

Steve: "...Donetsk and Luhansk, which have been classified as terrorist organizations..."

Leo: I believe that's "the Donetsk and Luhansk."

Steve: I think so, "...by the Ukrainian government. CyberX is keeping the identities of the affected organizations private, but give us a high-level peek, saying: 'Examples of Operation BugDrop targets identified by CyberX so far include: a company that designs remote monitoring systems for oil and gas pipeline infrastructures; an international organization that monitors human rights, counterterrorism, and cyberattacks on critical infrastructure in the Ukraine; a scientific research institute; an engineering company that designs electrical substations, gas distribution pipelines, and water supply plants; and editors of Ukrainian newspapers." So, you know, this really does sound - we don't have any reason to believe it's Russian backed. But it sure seems like it might be.

"Operation BugDrop," they write, "is a well-organized operation that employs sophisticated malware and appears to be backed by an organization with substantial resources. In particular, the operation requires a massive backend infrastructure to store, decrypt, and analyze several gigabytes per day of unstructured data that is being captured from its targets. A large team of human analysts is also required to manually sort through captured data" - like, listen to all of that - "and process it manually and/or with Big Data-like analytics.

"Initially, CyberX saw similarities between Operation BugDrop and a previous cyber-surveillance operation discovered by ESET in May 2016 called Operation Groundbait. However, despite some similarities in the Tactics, Techniques, and Procedures" - we have a new acronym, TTPs, Tactics, Techniques, and Procedures - "used by the hackers in both operations, Operation BugDrop's TTPs are significantly more sophisticated than those used in the earlier operation."

And they go into some additional details that leads us, I mean, they're like, there are connections to Duqu and Stuxnet and high-level work. So no clear attribution for this. But a serious, persistent campaign that was discovered by these guys. So props to them. And it's just nice, even though it's not affecting us, and there's nothing we can do about it, it's sobering to recognize that this kind of high-budget, large data-focused campaign is present, probably all over the place in the world. This is just an example of that.

Graham Cluley, who is a well-known security blogger, provided some nice coverage for a story that many outlets picked up. CNN he quotes at one point. And this is from a presentation that Charles Henderson of IBM gave at the RSA Conference recently. He's the global head of X-Force Red at IBM.

And so Graham wrote: "Not too long from now it will be pretty much impossible to purchase a new car which is not connected to the Internet in some fashion. Many modern purchasers are more swayed by the gadgetry bells and whistles their car includes than its performance, and within a world where everything seems to have an associated smartphone app, why should vehicles be any different? If most new cars," he writes, "are going to be Internet-enabled, then you know what that means? Yup. Second-hand cars are going to be increasingly 'smart' as well as vehicles are sold on after a few years." I don't know how that parses.

He says: "Oh, and yes, as I'm writing on We Live Security, it should go without saying that it also means security threats. This point was brought home last week at the RSA Conference in San Francisco, where IBM's Charles Henderson described how - over two years after he had traded it back in to the original authorized dealer - he was still able to access his old car via a smartphone app. Despite deauthorizing" - and again, this is not a newbie. This is head of X Force Red at IBM, speaking at the RSA conference. "Despite deauthorizing all associated accounts, satellite radio, garage door openers, resetting the Bluetooth, as well as surrendering all the keys at the time of sale" - so this guy Charles Henderson completely did everything he knew that he could to divest himself of access to the car - "he discovered that his mobile app never forgot his old car.

"The app allowed Henderson to track the geolocation of the car, adjust its climate control, send its SatNav systems new directions, and even trigger its horn. But perhaps most alarmingly of all, the app also gave Henderson the ability to remotely unlock the vehicle." And this is two years after it was gone from him.

"Fortunately," writes Graham, "the IBM researcher isn't one of the bad guys. But it's easy to imagine how a car thief or stalker would exploit such a feature. As Henderson explained to CNN: 'The car's new owners would have no clue that they were potentially at risk.'" Quoting him in his interview with CNN: "The car," he said, "is really smart, but it's not smart enough to know who its owner is, so it's not smart enough to know it's been resold. There's nothing on the dashboard that tells you 'the following people have access to the car.'"

"It turns out that, although Henderson took more effort than probably most people in ensuring that he had wiped the car's knowledge of him and associated accounts before trading it in, that wasn't enough. As the researcher explains, that's because a full factory

reset of the unnamed vehicle does not revoke access by the smartphone app. The information still lurks in the cloud and can only be wiped by a factory-authorized dealer. One has to wonder," writes Graham, "how often that occurs. Henderson's own investigation discovered four major vehicle manufacturers were allowing previous owners to access cars from a mobile app.

"This is the Internet of (insecure) Things at work again. In the rush to add bells and whistles," writes Graham, "features are not being properly thought through" - as we well know on this podcast - "and security is not uppermost in manufacturers' minds. Until more effort is made by vendors to integrate the Internet in a safe way into the myriad of devices that surround us, we are going to hear more and more stories of security breaking down like this."

And it's like something that had never occurred to me. I mean, I guess it's - I don't know how you flush that. Maybe you delete the app and then reinstall it? But a lot of these apps, you know, they use the cloud to keep themselves synchronized. So a really interesting oversight on the fact of, apparently, four major car manufacturers are still allowing this to be possible.

So I mentioned also that Chrome 56 has a UI update that took out a feature that I imagine many of our listeners would have appreciated, or had appreciated in the past. It used to be that, when you were at a secure site, and you wanted to check the certificate, for example, you'd like to know if there's a man-in-the-middle attack or a middlebox is filtering your connection, that is to say, is the certificate actually from eBay or Gmail or your bank? Or is it from some manufacturer of a TLS-intercepting proxy that's in your connection? The only way to know is to look at the certificate and see what it says.

It used to be that you could click on the padlock in Chrome, and you would get a little pop-up where it would give you the domain. And I did it to myself this morning, www.grc.com. Then underneath it says, "Your connection to this site is private." And then it says "Details," which is a hyperlink. You click on that. And in all versions of Chrome prior to 56, that is, up to 55, that then presents you with the certificate information. And that certificate information is the Windows UI under Windows, and it's the Mac UI on the Mac because Chrome has traditionally used the existing secure infrastructure in the underlying OS to perform its TLS stuff.

Well, the ability to do that is gone in Chrome 56. If you click on the padlock, you no longer have an option to view the certificate. Which, again, they probably simplified it for most users. Maybe this is a consequence of some rearchitecting of the way they handle security. I didn't take time to dig more deeply into this. You can still find it on all versions of Chrome 56 under the three dots or three bars menu. You go there and look under More Tools, and then Developer Tools, and then click on the Security tab, which you may not even be able to find because it's way off to the right, and it gets truncated with the little chevron. So you may have to click the chevron. Then you'll find Security Overview. And then there, there is a View Certificate button. So they made it a lot more involved, at least for Windows users.

There's a shortcut, however, which is not documented anywhere that I could find. And that's Ctrl-Shift-I. So for our listeners, who tend to be certificate-aware, if you are using Chrome 56 and later, Ctrl-Shift-I, it still won't pop up the original convenient certificate display. But that jumps you immediately to the proper page, the Security Overview page in the Developer Tools area, and does allow you to pretty quickly get a hold of the certificate. So Ctrl-Shift-I for some reason.

So Firefox has been moving forward with their plan to evolve the browser. And over the

year of 2017 they have a succession of updates planned which take us from 51, where I am today, up to 57. And unfortunately there are deliberate and side effect consequences to some of the things that they're going to be doing. They'll be putting the finishing touches during 2017 up through November on their plan, which they put in motion in 2015, to replace their original add-ons API. Theirs was an ad hoc, you know, their own system, which had the advantage of more flexibility than the Chrome approach. Chrome's system and the Chromium-based browsers - Chrome, Vivaldi, Opera, and others - they're based on a WebExtensions API.

So the mixed blessing is Firefox has been moving to the WebExtensions API. But the existing Firefox extensions are not WebExtensions APIs. So as they move through this year to WebExtensions, as a consequence of other things they're doing, the ultimate upshot, by the time we get to Firefox 57 in November, is the entire existing Firefox add-ons ecosystem dies. Which makes me nervous because, for example, there are not yet equivalent extensions in Chrome that I depend upon for Firefox. There are various workarounds. And largely they're the same.

So what Firefox gains from this, of course, is a single WebExtensions API that it will then share with the other Chromium-based browsers. So that's cool. That means that - and it's great for developers that then can only need to write one extension which will then run, not only on Chrome, but also on Firefox. So Firefox inherits the extension API of what has now become with Chrome the world's number one browser, which long-term is probably good for them. It does create some pain for we Firefox users who in the meantime, our days are numbered, maybe, with some of our favorite extensions. We know that things like uBlock Origin, that's already on Chrome. LastPass is already on Chrome. Many things are.

I really like my hierarchical tabs. And I know that there are a few tab solutions. Nothing that I've seen yet replaces what I've got on Firefox. So I can hope that maybe some of the existing add-on developers will choose to move their work over to WebExtensions, which would also give them access to the number one browser in the industry, which would be useful. So on Firefox 53, which is slated for release on April 18, there will be no new legacy add-ons, as they call them, will be accepted any longer.

So they'll support existing ones, but no new ones will be accepted at addons.mozilla.org. However, developers and users will be able to update preexisting legacy add-ons, and Firefox will start running with multiprocess support for most users. I tried to turn mine on, thinking, well, maybe that's a good thing. But it's been disabled by one or more of the legacy add-ons that I have. So that's one of the problems is that there has been some conflict with the behavior that Firefox brings with multiprocessor support and its incompatibility with legacy add-ons, which is one of the other reasons that they're beginning to move away. It's like, okay, they're just going to have to force the issue.

So any Firefox add-ons that are confirmed as multiprocess-incompatible will be disabled in Firefox with Firefox 53 on April 18th. So this incompatibility is being flipped. Right now the incompatible add-ons are able to disable the multiprocess feature. With Firefox 53, it goes the other direction. Multiprocess feature stays, and incompatible add-ons, they're disabled. So that's an important takeaway. Then over Firefox 54, 55, and 56, Firefox will be expanding its sandbox security features to incorporate multiprocessor support and also expanding. Right now that's only two process. You have sort of a UI process and then everything else. And that will be expanding from two to three and more processes in order to create more robustness and to deliver more performance.

And then on Firefox 55, although there's no firm date set for that, the legacy add-ons will stop working due to this migrating change. And then, finally, with Firefox 57, which is

due to release on November 14th, that's the end of the line for old Firefox add-ons. If developers don't migrate their code from the old add-ons API to the new WebExtensions API by November 14th, their add-ons will stop working for good. So that means that, with Firefox 57, it will only run add-ons built to the new WebExtensions API. For some reason, addons.mozilla.org will continue to list the legacy add-ons. But if you've been updating your Firefox and kept it current, it won't run them any longer. And they have not said at what point they will stop providing the legacy.

Oh, there is one possible reason, and that is there are some of the Firefox derivatives, derivative browsers, like Firefox ESR and Pale Moon. They may still provide a safe harbor for the legacy add-ons, and so you'd be able to get them, for that reason, from there. So Mozilla says that the reason old legacy add-ons will stop working in Firefox 57 is because they plan to remove from Firefox 57 a workaround that allowed non-multiprocess code to run on the new Firefox engine. So it's just that's over at that point. And removing the workaround means that no code will run in Firefox if it hasn't been optimized for multiprocessor support.

And at this point it would be crazy for a legacy add-on developer to update their add-on. They might as well just recode for the WebExtensions API, which, again, it's somewhat more limited. There are some things that it cannot do which may end up being restrictive for the sake of security and so not a bad thing anyway. So anyway, a little bit of a bumpy road for the next year. I don't, at this point, my problem with Chrome is that it's just so memory hungry. And I'm still operating within a 32-bit, 4GB, actually 3GB under Windows, constrained address space. So with Chrome burning up as much memory as it does, I'm still using Firefox, and happy to do so. But I know that we have a lot of Firefox users among us, and so I wanted to let everybody know what the future looks like for 2017 in Firefox.

Leo: Steve Gibson, Leo Laporte. We're talking security and the Windows Clipboard.

Steve: So, okay. Earlier this year a Norwegian MVP, Oddvar Moe, made a rather shocking discovery that went mostly under the radar. He found that on Windows 10 there is a way to easily read clipboard contents, that is, the system's global clipboard, from the screen of a locked machine without requiring any form of authentication. So this is a problem. In any enterprise environment, any coworker could easily go through a bunch of PCs at lunchtime, harvesting potentially juicy information such as passwords, without leaving any traces. So how do you do this? It's pretty simple. The Win+L locks the workstation. So you can imagine employees locking their workstation with Win+L and then going off to lunch. Well, from the lock screen, Win+ENTER starts the Narrator. Then CapsLock+F1 opens Narrator Help, and Ctrl-V pastes the clipboard into the narrator and exposes its contents.

Leo: It starts reading you the clipboard.

Steve: Yes.

Leo: Wow. All of this without unlocking.

Steve: All of this without unlocking. And in the write-up it said this can also be achieved

through the WiFi selector UI on the lock screen. So there are a couple ways to do this. And it not only affects Windows 10, but it's confirmed to be effective on 8.1 According to Oddvar, Microsoft does not consider this to be a security issue because it requires physical access. Okay. So there are some mitigations, and I have a solution for our listeners. Possible mitigations include disabling these features through the appropriate Group Policy settings, or using a cute little utility called ClipTTL.

And as we know from our discussion of the Internet and packets, TTL often stands for Time To Live, as is the case here. It's a small utility, written by Firas Salem, to protect against this and other cases of accidental clipboard pasting. And since it is the case often that the clipboard can contain some sensitive information because we use it as a little transport carrier from one place to another, ClipTTL is an automatic clipboard wiper which prevents accidental or malicious pasting.

Firas is a Brussels, Belgium-based security consultant whose work we've covered before. He also wrote that RCC utility which many of our listeners were interested in because it finds potentially rogue certificates in Windows and Firefox. It looks for any certificates that are not part of the baseline installation. So he's at TrustProbe.com. I've got the links in the show notes. The utility that automatically wipes your clipboard after a set about of time is ClipTTL. But there has been some additional people saying, what was that utility that lets me audit my trust store? And that's RCC, also by the same guy. And in fact he's got a whole bunch of interesting bits of freeware that are sort of related and in this environment that I would recommend people check out. He's a good guy.

Leo: That's why LastPass somehow, I don't know how it does this, it's kind of magical...

Steve: Yes.

Leo: ...allows one, and one only, paste; right?

Steve: Correct. And...

Leo: It clears the clipboard after you paste.

Steve: Yes, it does.

Leo: How does it do that?

Steve: And it's caught me out a couple times, where it's like, I was assuming it would - like I needed it still to be there.

Leo: And paste it several times, right.

Steve: Yeah, like for example I would have it generate a password, put it on the

clipboard, then I would paste it into a web app, but I also then wanted to make my own private copy.

Leo: Right.

Steve: And so I'd go over and open Notepad and try to paste it, and it was gone. It's like, ooh.

Leo: There must be a command or some sort of pasteboard command that does that.

Steve: Oh, yeah, yeah. Yeah, there is. Well, for example, if you've ever exited an app, and you'll get a popup from Windows saying the app that you've just...

Leo: We're going to clear your clipboard, yeah, yeah, yeah.

Steve: Yes, the app that you've closed just left a whole bunch of stuff there.

Leo: Yeah, that's embarrassing.

Steve: Like, uh, no, I don't want that.

Leo: No, no, no, no.

Steve: Get rid of that, please.

Leo: Yeah.

Steve: So on November 16th of 2016, a little over 90 days ago, a problem was reported to Microsoft. And again, I don't know what's going on up in Redmond. But this was Project Zero gave Microsoft, and confirmed receipt, a formal notice of a worrisome problem that had been found, actually a number of worrisome problems, in a commonly used API known as SetDIBitsToDevice. Actually, that's a friend of mine, that API. I used it way back in the days of 16-bit Windows when I wrote ChromaZone, that crazy screensaver toolkit app that's what I used to teach myself the Windows API.

And I was using it just recently. SetDIBits, DI stands for Device Independent Bits. So it's a way of supplying bitmap information to a device and, for example, like a way of programmatically drawing a bitmap. I use it in SQRL, for example, in order to display the QR codes for transfer between devices. So it's a way for me to take an algorithmically generated binary QR code and display it. You SETDIBITSTODEVICE.

Well, it turns out that the ever-present metafiles, speaking of Honey Monkeys and the

very early episodes of this podcast, we covered the controversial escape, which I'm convinced Microsoft originally had in the Windows metafile on purpose. But then a decade later it didn't look nearly as benign as it originally did when security wasn't such an issue. Well, it turns out that after metafiles we got enhanced metafiles. And metafiles are interpreted, essentially interpreted API calls. So when you process a metafile, you are actually reading the API and its parameters.

So what the Project Zero guys discovered was that there were some mistakes in the verification of parameters to the SETDIBITSTODEVICE API, and actually a number of others. So that was on November 16th, 2016. Microsoft was informed. Well, last Tuesday, February 14th, which was supposed to be Patch Tuesday, but as we know wasn't for whatever reason, that was the 90-day expiration on the Project Zero timer. And all the details of this went public last Tuesday. So what went public is a full documentation of the problem, including a proof of concept, which demonstrates...

Leo: I can just see them at Microsoft, ready to push the button on the update. This comes out, and they say, "Did you know about this? I think he told us, didn't he?"

Steve: I mean, you almost - it almost feels like a communication screw-up of some...

Leo: It must be. They ignored it for three months.

Steve: Yes. Yes. And it's not a hard problem. So reading from this, "The proof-of-concept file attached here," they say in their disclosure, "consists of a single EMR_SETDIBITSTODEVICE record" - so excluding the enhanced metafile header and end-of-file records - "which originally contained a 1x1 bitmap. The dimensions of the DIB, the device-independent bitmap, were then manually altered to 16x16, without adding any more image data. As a consequence, the 16x16, 24 bits per pixel" - which is to say 3 bytes per pixel - "bitmap is now described by just 4 bytes," which of course is no longer necessary. It needs, what is it, 16x16 is going to be 256. Three bytes, that's 768 bytes. But the API only gives it four, which they say is "good for only a single pixel. The remaining 255 pixels will be drawn based on data from the heap, which may include sensitive information, such as private user data or information about the virtual address space."

The author of this said: "I have confirmed that the vulnerability reproduces both locally in Internet Explorer and remotely in Office Online via a .docx document containing the specially crafted EMF file." So what this means is that the abuse of this API allows private data from the heap to be turned into an image which can then be captured and exported. So this is not good. And it's been public for a week. And Microsoft will be updating this and hopefully fixing this in March. So until then, good luck to us. And it's all public now.

Leo: Yeah.

Steve: Project Zero also having been busy, spent some time looking at the Windows NVIDIA driver. Oliver Chang posted a nice little summary, saying: "Modern graphic drivers are complicated [yeah, indeed] and provide a large promising attack surface for [what he writes] EoP [which is] elevation of privilege, execution opportunities, and sandbox escapes from processes that have access to the GPU [such as] the Chrome GPU

process. In this blog post," he writes, "we'll take a look at attacking the NVIDIA kernel mode Windows drivers, and a few of the bugs that I found." Yeah, 16. He says: "I did this research as part of a 20% project with Project Zero, during which a total of 16 vulnerabilities were discovered."

And I won't go through them all because they've been fixed. And in fact they were fixed impressively. Describing NVIDIA's response, he wrote: "The nature of the bugs found showed that NVIDIA has a lot of work to do. Their drivers contained a lot of code which probably should not be in the kernel, and most of the bugs discovered were very basic mistakes. One of their drivers, NvStreamKms.sys, also lacks very basic mitigations, [for example, like] stack cookies, even today." So a lack of even automated security mitigations that could have easily been turned on with a proper compilation.

"However, their response," he writes, "was mostly quick and positive. Most bugs were fixed well under the deadline, and it seems" - unlike Microsoft's - "and it seems that they've been finding some bugs on their own internally. They also indicated that they've been working on re-architecting their kernel drivers for security, but were not yet ready to share any concrete details."

So he concludes, saying: "Given the large attack surface exposed by graphics drivers in the kernel and the generally lower quality of third-party code, it appears to be a very rich target for finding sandbox escapes and elevation of privilege vulnerabilities. GPU vendors should try to limit this by moving as much attack surface as they can out of the kernel."

And of course this has been a constant refrain of ours. One of the arguably, in retrospect, worst things Microsoft did was to move GDI, the Graphics Device Interface, from userland, from being a DLL in user mode, where it was subject to the same restrictions as the programs which invoked it. Unfortunately, back at a time when processors were having a hard time keeping up, and Microsoft graphics performance was lackluster, Microsoft decided, okay, there's just too much ring transition between GDI and the kernel. So they moved GDI into the kernel.

And this is - that's the reason that, for example, we then had all of these problems with graphics vulnerabilities, where a JPEG can take over Windows. It's because the JPEG is being interpreted in GDI, which is now in the kernel, which makes any escape from GDI subject to having ring zero root privileges, essentially. So what they should have done is kept it out in userland, in which case, even if you did find a problem, it would be like a bug in the user's software, rather than a bug in the kernel. So anyway, it would be great if - again, it's just sort of - the problem is it's easier to keep it all as one lump and just stick it all in the kernel.

The problem then is all of that code has to be perfect. If instead you go to the effort of bifurcating this job so that the only code in the kernel is what has to be there, then you keep most of the bulk of the work in the user space, where it's under ring three and subject to all of the protections that our state-of-the-art hardware provides. So anyway, we'll hope that this is growing pains. But I was impressed with NVIDIA's response. They jumped right on it, and they fixed all 16 problems. I did see a summary of them and scan them, and they were just like, fixed, fixed, fixed, fixed, fixed, fixed, fixed, right down the line, and did so quickly. And they said that they're working to make things better.

So I have a treat for our listeners: toc.cryptobook.us. That'll take you to the table of contents of a 580-page tour de force, freely downloadable PDF, graduate course in applied cryptography. We've talked about this Victor Shoup in the past. He's a professor of security and cryptography at New York University. And Dan Boneh is a professor of computer science and electrical engineering at Stanford. He's also the co-director of the

Stanford Computer Security Lab. These two profs are authoring this book. It is at 0.3 version, not yet finished, but already really comprehensive.

Describing it in its preface, they said: "Cryptography is an indispensable tool used to protect information in computing systems. It's used everywhere and by billions of people worldwide on a daily basis. It's used to protect data at rest and data in motion. Cryptographic systems are an integral part of standard protocols, most notably the Transport Layer Security (TLS) protocol, making it relatively easy to incorporate strong encryption into a wide range of applications.

"While extremely useful, cryptography is also highly brittle. The most secure cryptographic system can be rendered completely insecure by a single specification or programming error. No amount of unit testing will uncover a security vulnerability in a cryptosystem. Instead, to argue that a cryptosystem is secure, we rely on mathematical modeling and proofs to show that a particular system satisfies the security properties attributed to it. We often need to introduce certain plausible assumptions to push our security arguments through.

"This book is about exactly that: constructing practical cryptosystems for which we can argue security under plausible assumptions. The book contains many constructions for different tasks in cryptography. For each task we define a precise security goal that we aim to achieve and then present constructions that achieve the required goal. To analyze the constructions, we develop a unified framework for doing cryptographic proofs. A reader who masters this framework will be capable of applying it to new constructions that may not be covered in the book.

"Throughout the book we present many case studies to survey how deployed systems operate. We describe common mistakes to avoid, as well as attacks on real-world systems that illustrate the importance of rigor in cryptography. We end every chapter with a fun application that applies the ideas in the chapter in some unexpected way." And it goes on. Then the book goes on for 580 pages. I browsed through it. Everything is there: symmetric crypto, public key, hashes, you name it. Over the years, many of our listeners have asked for recommendations of books. And of course Schneier's "Applied Cryptography" is still the bible, but it's not free. This one is.

They're not finished. I think they said they have three major parts, and they have all of part one and half of part two finished. But what's there already is very comprehensive and worthwhile. So again, TOC dot - it's got its own domain: toc.cryptobook.us. And there's a link there that will take you to crypto.stanford.edu, where you can find it. So links are in the show notes. The show notes are already posted publicly at GRC.com/sn. I've got the header for this Episode 600 is already there with the link to the show notes. So everybody should be able to find it without any trouble. And it looks great.

So if anyone is looking for something to just browse through casually or to really roll up your sleeves, I know that from the past a lot of our listeners are interested. This looks like the place to go. And this thing is only going to get better with time as the profs continue to flesh it out. And this one, this is - it had a 0.1, 0.2, and 0.3. This 0.3 was just released in December of 2016, so a few months ago. And they're working on finishing it. Looks like it's going to be a nice piece of work.

And they did note, I didn't describe it, but that there is a deliberate hierarchical nature, that is, you can skip the proofs. You can skip a lot of the formalism and still get a lot out of it, just by reading sort of the introductions to each section and the goals of the various algorithms, without getting down into the weeds. The weeds are there, but you don't need to bother with them. And they explicitly have written it this way so that there is a

softer, gentler side. It's not just for people who want to do formal proofs for crypto. There is sort of an upper layer that will still give you a lot.

And I wanted a couple pfSense and Ubiquiti follow-ups. I got a tweet from a Dan Moutal, whose Twitter handle is @scruffydan. He said: "Listening to this week's Security Now!," he said, "and the EdgeRouter also allows you to restrict UPnP by IP address." Now, that's something I didn't know, so I wanted to share that. We've talked about the Ubiquiti EdgeRouter X, which we like so much because it is a true router with independent ports where each port can be its own subnet, which allows you to, for example, create an IoT region. We were talking about, last week, I talked about how pfSense, part of the UPnP UI on pfSense absolutely has UPnP ACLs, Access Control Lists, to allow you to explicitly restrict what ranges of IP or specific devices on your LAN have access to Universal Plug and Play, because you really don't want that to give blanket permission. You'd rather only provide it where you have to.

In fact, back when we were having connection problems, not it turns out due to anything we were doing, but because the relay server at Microsoft was bogged down that afternoon, one of the things that I did was thinking, well, maybe Skype now requires UPnP. It certainly would like to have it. Even though I had manually statically mapped a port and given Skype permission to use that statically mapped port, I thought, okay, well, maybe it just - it's not using that anymore, and it needs to use Universal Plug and Play. So I for that reason enabled Universal Plug and Play, but only permitted the machine that I use Skype on for this podcast to have access to it. So again, I'm practicing what I'm preaching here.

The point is that Dan brings up is that, if you do have a Ubiquiti EdgeRouter, as many of our listeners now do because it's $49 and a great bargain for that price, you do need to use the command-line interface. It's not surfaced in the web UI, but the capability is there. He says: "You need to use the CLI and the UPnP2 service to create the UPnP ACL." And the router is so popular, and there's so much stuff on the 'Net, that I'm sure, users, if you google for UPnP2 service, UPnP ACL, Ubiquiti EdgeRouter, you'll probably be able to find some how-tos and guides and things to help you with the command-line interface. But the point is, it's in there. And that would allow you without any additional hardware to just use the Ubiquiti EdgeRouter X and enable UPnP for those devices on your network that need it.

Somebody else tweeted, he asked: "What was the name of the new pfSense box that you mentioned on SN-599." last week. And he said: "I don't think you mentioned the name, just that it had two ports." And so that is the SG, I'm sure no relation, -1000. So if you go to pfSense under Products, you'll find the cute little two-port box, the two-port router that is running pfSense. It's $149. What I don't know, and I didn't dig in, is what its bulk throughput is. That is, it does have a pair of gig interfaces, so a gig link to your upstream DSL or cable modem or wherever you're getting your bandwidth from; and a gig link then, probably to a switch, which would then fan out to go to and access other things. The beauty is that this cute little thing, I mean, it's like the size of a matchbook, I mean, it's just big enough to hold two 10-base-T ethernet connectors. So it's very cute and tiny, and very low power.

The beauty of it is it's running pfSense, which, as we know, I mean, that's the universal toolkit for this. I mean, you can run proxies, you can run OpenVPN VPNs - I couldn't figure out if I was missing another word there - VPNs for point-to-point links. And just you know, any other kind of security - full firewalls, traffic shaping, bandwidth limiting, rate limiting, I mean, it's everything you could want. The question is, will this little box actually run your data at a gig? Now, it doesn't really have to unless you have a gig upstream bandwidth. But will it run it at 300Mb? I just don't know. So it's one thing to

have the links capable of a gig. But the question is, what is the packet rate throughput through it? That I don't know. But that's what it is, it's the SG-1000 for $149 from Netgate.com, which is where the pfSense.org page for their products takes you when you click on it. And a beautiful little machine.

And finally someone tweeted: "Hi, Steve. I'm listening to SN-599. You can use the Ubiquiti EdgeRouter X with the small pfSense box for network segmentation." And so that's another point that I hadn't made. I was talking about using that with a VLAN-aware switch because the pfSense box has full support for VLANs, virtual LANs. And what that essentially means is that different ranges of IPs could be tagged as members of VLAN 1, VLAN 2, VLAN 3. Then when that goes to a VLAN-aware switch - you want to make sure that a little dumb switch is VLAN-aware - then it would perform the network segmentation. So this person was tweeting to me, he just observed that the Ubiquiti EdgeRouter X will also support network segmentation. So I wanted to make sure everyone knew that, too. So we're ending up with a nice little not-very-expensive toolkit of components that allow us to structure a very robust and safe network.

A number of listeners reminded me of miniLock.io. And this followed up from our conversation, Leo, last week about PGP and...

Leo: AES Crypt.

Steve: And GPG and Keybase.

Leo: And Keybase [crosstalk], yeah.

Steve: Right. And they reminded me that I had talked about miniLock some time ago because it was originated by our friend Nadim Kobeissi, who did Cryptocat and then did miniLock.io. It's audited, peer reviewed. And following on from what you said, Leo, and that is the notion that public key crypto makes much more sense for sharing encrypted content than symmetric key crypto. And you're right, this provides a simple means to do that. It is available as a Chrome add-on, so anyone with the most popular browser in the world, Chrome, can easily use this. It uses your email and a secret passphrase to generate what they call a "miniLock ID."

MiniLock IDs are small because it also uses the Bernstein curve, the Curve25519, which allows us to use much shorter and more convenient keys than normal public key, traditional RSA public key crypto allows at the same level of security. So it makes them easy to share online. So anyone can use your ID, which again you generate with your email address and secret passphrase, to encrypt files to you, that is, encrypt them for your eyes only; and only you who have the matching private key to your miniLock ID, which is essentially your matching public key, then are able to decrypt them. So it's been audited. Matt, Matthew Green - in Maryland; right? That's where Matt Green is?

Leo: Yeah, he's at Johns Hopkins in Maryland.

Steve: Johns Hopkins, right. He's gone over it and was involved and helped with this. It's been audited independently and peer reviewed, developed by experts and so on. So, for example, the user flow would be - we have the typical roles in crypto, Alice and Bob.

Alice wants to send a scan of her passport to Bob. Sending it over email would compromise her personal information.

So Alice decides to first encrypt the scan using miniLock. Bob opens miniLock and enters his email address and his own personal secret passphrase. miniLock displays his miniLock ID, which is tied to and actually generated from his passphrase and is persistent. He sends Alice his miniLock ID, which is a relatively short ASCII string, and he can tweet it to her. He can email it to her. I mean, it can be sent over a public channel, does not need to be kept secret because all that is his public encryption key, essentially.

Alice drags and drops her passport scan into miniLock, which again is a Chrome extension, and enters Bob's miniLock ID as the recipient. She clicks the encrypt button and sends the resulting .miniLock file to Bob. Once Bob drags the encrypted file into miniLock, it automatically detects it as a miniLock-encrypted file destined to Bob, and decrypts and saves the passport scan on his computer. So no third-party software, just uses Chrome, uses Curve25519 for short and tweetable keys, and is a superior way to send encrypted content between two people who both have Chrome. So thank you to our listeners for reminding me that we talked about this once before, and I just forgot to bring it up again last week.

Leo: I just did it. It's great, yeah.

Steve: Cool, cool. Also last week our title was HTTPS Insecurity. And the link that I tweeted, and I don't think I had it in the show notes, or maybe I did, I think I had it in the show notes, but it turns out that the document disappeared, and a whole bunch of people - actually I was very impressed with how many of our listeners wanted to dig deeper and follow up. And they weren't able to because the document was gone. So it's back. When I saw that, I grabbed it, as I still had a copy from my own reading of it last week. So I stick it up on GRC and tweeted it a few days ago. It is also now in the show notes. So anybody who isn't in Twitter or missed my tweet can, in this week's show notes, down here in the notes is the link to grab that paper, if you're still interested.

[https://media.grc.com/misc/HTTPS-Interception.pdf].

And then we have two little bits of lighter side. So we have a riddle: Why are assembly programmers always wet?

Leo: Why, Steve?

Steve: Because they work below C level.

Leo: Oh, geez. With the letter "C," obviously, yeah.

Steve: That's right. Yes, we do. And then the follow-on funny Picture of the Week, if you're a little geeky, is a photo that shows how computers are made in our show notes.

Leo: I guess I should show that, huh?

Steve: I think you should, just so that it...

Leo: Oh, because you can't really describe it, can you.

Steve: No, you really...

Leo: Yeah, you just have to - you've got to see it to be there. And if you're listening on audio, just download the show notes. Steve tweets the URL out every week at @SGgrc, or you can go to GRC.com/securitynow and get the show notes, and you'll see the image.

Steve: Right. And I did tweet this in my feed a few hours ago. I got a lot of people thought it was fun and funny.

Leo: It's a small IC and some resistors with squiggly tails, that's all.

Steve: That's right. Use your imagination. And under the picture is a link to the full size original image, which is really large. So anybody who wants a high-resolution copy, you are welcome to it.

Leo: Funny. Resistance is fertile.

Steve: Oh, that's really good.

Leo: Chickenhead21 gets credit for that one in the chatroom.

Steve: That's wonderful. Wonderful. And I have a quick note to share from a Bob McKinstry in St. Louis, Missouri. He wrote: "Hey, Steve. I'm a listener of Security Now! since year one" - so 12 years - "and a SpinRite owner. A few days ago, one of the residents in our training program asked how he could get help with his Dell laptop, as our IT department does not support machines they don't own. The story is classic: After a power outage, his laptop would not boot into Windows. It would blue screen, reboot, rinse and repeat.

"Without hesitation, I told him about SpinRite. I lent him my trusty SpinRite CD, told him he'd have to purchase SpinRite if it did fix his machine. I told him he had to be patient and let the program do its work. So he fired up SpinRite and went to bed. When he woke in the morning, he was good to go. Just like that. SpinRite," he writes, "is a great product. Another drive saved. Another happy customer. Thanks for all that you do with GRC, and thanks for all the great podcasts that you've done with Leo on the TWIT network." And Bob, thanks for sharing your success.

Leo: All right, Steve Gibson. Time to talk - what is MMU?

Steve: That's the Memory Management Unit.

Leo: Ah, okay.

Steve: Yes. So we do have - I got a tweet while you were talking about ITProTV.

Leo: Yes?

Steve: From Mike Roberts, who also had something clever to add. He said: "Who said resistors are passive?"

Leo: All right, all right.

Steve: Because, of course, unlike a transistor, resistors and capacitors are considered passive components.

Leo: Oh, man, that's geeky.

Steve: No, these resistors are - they're getting busy, as they say. Okay. So the press coverage of this. I won't embarrass the specific magazines or the specific websites by calling them out. But one's headline was "A Chip Flaw Strips Away a Key Hacking Safeguard for Millions of Devices." Uh, no. It has nothing to do with a chip flaw. These guys have figured out how to very cleverly leverage a fundamental operational characteristic of all modern processor architectures to penetrate the randomization of all address space layout, ASLR. Another one wrote "New ASLR-Busting JavaScript Is About to Make Drive-by Exploits Much Nastier." Uh, no. It doesn't make them nastier, it makes them significantly more possible and likely.

Someone else, one of the other major coverages was "JavaScript Breaks ASLR on 22 CPU Architectures." Uh, no. It breaks it on all contemporary CPU architectures. In one of the papers they did, they applied some of their research to 22 different CPU architectures, but that's all the ones they had around. So until something changes, this thing just busts it. And, finally, "A Simple JavaScript Exploit Bypasses ASLR Protections on 22 CPU Architectures." Well, okay, it says a "simple JavaScript exploit." There's nothing whatsoever simple about this.

Now, I hope I didn't scare everyone away with my admonitions at the top of the show to not try to walk and chew gum while listening to this. It's not that crazy. But these guys are. As I mentioned, this is the group at the University of Amsterdam who gave us, and we covered in Episode 576, Flip Feng Shui. And our listeners since then will remember that they came up with just a "sublime," is the word I used at the time, a sublime hack, where they were able to leverage the fact that virtual machine environments consolidated identical pages of virtual memory in order to share pages that were the same. They figured out a way to use the bit flipping of Rowhammer in order to get a shared page with sensitive information from an adjacent running VM. And that was Flip Feng Shui that we described.

Then later, in Episode 583, they gave us Drammer, which was a new attack exploiting Rowhammer on Android devices, and essentially cut through all of the Android devices that had been looked at. There were some newer ones had some mitigations against it. But in general it was devastating at the time. This one, there are some potential mitigations we'll talk about. But again, this is just sublimely clever. So in the show notes I've got links to the two research papers that back this up. I don't think these links will be ephemeral as the one from last week was. So they're linked on their site. They have a history of keeping these things present into the future. If they do go away, I can provide them, but I'd rather that people went there. So their site is www.vusec.net/projects/anc. That's the short name of this attack, ANC.

Okay. So what they've got, and I'll explain exactly how it works, is a working means to allow arguably the sloppiest, least low-level code that we use, namely JavaScript in a browser, to determine the virtual address of important regions of memory in the operating system, which is exactly what address space layout randomization is designed to hide, to obscure, and to prevent.

Okay. So there isn't a harder problem to solve than doing this in JavaScript. They have native code that can do it. And unfortunately, malware could take advantage of this if it wanted to, also. So, I mean, this is a big deal. So the press coverage was wrong in the details but right in the overall concept. And the bad news is, as we'll see, what this leverages is a low-level fundamental fact of the way our processes are designed on purpose, which leaks information in an exploitable way about where things are in the operating system's address space. And I don't know what's going to happen. There are mitigations that can be applied to the browser to make this less feasible. But there's nothing you can do at the low level. There's nothing you can do for native code running on the system. And I'll explain why.

So backing up a little bit, address space layout randomization. One of the early mitigations for buffer overruns, which are such a problem, the idea with a buffer overrun was that an Internet-connected client or even a JPEG, anything that where an attacker had some way of providing data, whether as a fake image or as a packet to a server, if the hacker could provide data, that data would be accepted in some kind of buffer that you open the image, and so that JPEG file gets read into a buffer. Or it's received over the Internet. It goes into a communications buffer. What the early attacks were, they leveraged some means of executing that data, executing the data in the buffer.

And so the first bright idea that occurred to people is, wait a minute, let's add a feature to the processors where segments of memory can be marked as "no execute," that is, you can put anything you want to in there, but you cannot set the processor's instruction pointer to an address in there. There will be a flag on that page of memory saying execution is prohibited. And so it just - it simply can't be executed. So that was called DEP, Data Execution Prevention. And that was good. And Microsoft gently moved us into it because we wanted to make sure that applications were DEP compatible. And so initially the OS used it, and then only parts of the OS, and then users could turn it on for specific applications. And we sort of, you know, we gradually adopted data execution prevention. And it was a good thing.

So now attackers are prevented from providing their own data to somehow jump to and execute. But they're clever blokes, as we know. So they said, hey, you know, we can't provide our own code, but there's already a lot of code in the system. So what if we jump to the end of a subroutine that does a little bit of work and then comes back to us. The way subroutines work, you normally jump in at the top of a subroutine. And where you came from is put on the stack. And when the subroutine exits, it pops the stack and returns to the instruction after the one that caused you to execute the subroutine. So

that's why it called a subroutine. It's you're doing something, you say, go do this, and then come back to me when you're done.

Well, what they realized was they may not want all that the subroutine does. But toward the very end, there might be something useful going on. So nothing makes you only jump in at the top. You can typically jump in anywhere. And we've talked about mitigations to prevent that, to allow only jumping to the top of subroutines, but there's work going on about that, too. But so that's the way Data Execution Prevention was worked around. It's like, okay, if you won't let us provide our own code, we'll just go jumping around in the operating system, using code that's already there, to get our work done.

So then in this cat-and-mouse game the next flash of mitigation was, ah, the problem is the same code for a given OS is always loaded in the same place. Like when the OS boots, it loaded itself in at the bottom, and everything else went on top. But we realized it didn't have to go at the bottom. There's nothing, because of the way memory management works, we could scatter the code around within the OS's virtual address space, and we could do it randomly so that every boot of the OS, every instance of booting all over the planet, would end up just scattering, scatter-shotting the various modules of the operating system into random addresses. So now the hacker has a problem because there's code they may want to execute, but they don't where it is.

Now, there have been some interesting mistakes made by older APIs, well, not even older APIs, but APIs that weren't written with this hide-their-own-address problem. For example, there are APIs, like functions you could call in the operating system, where you say give me a buffer containing something. And that function allocates in itself some buffer space and returns a pointer to the buffer. And in doing so it gives up its own address. So there were APIs, there were functions that were leaking, that were found to be leaking their own addresses. And that was used for a while in order to, again, to get around the fact that ASLR randomizes where everything is.

Okay. So that's where we were. We had data execution prevention, and then we added address space layout randomization so that code that was essentially blind, it came in and is running in the OS, and there's that boundary between it and the operating system where all it can ask is the operating system to do things for it. But it can't see where they're being done. So that was good news. Okay. These guys figured out a way around that, and thus the problem.

Modern Memory Management Units, all modern architectures have it. This is this whole virtual memory versus physical memory thing, the idea being that there is a mapping between the virtual memory addresses and physical addresses. Meaning that the instructions and the programs and the jumping and accessing and reading and writing what looks like physical memory may be physical memory, but the address you're using is a sort of a pseudo address, a virtual address which the processor's hardware, the Memory Management Unit, maps into physical memory. So, for example, 64-bit processors, contemporary 64-bit processors like the Intel x64 family, they have a 64-bit virtual address space. Now, okay, because that is an insane memory space.

Okay, remember, 64 bits - 32 bits is the Internet; right? So that's 4.3 billion. So 64 bit is, we already know that's going to be 4.3 squared billion billion, which is something like 17 billion billion. Well, okay, that's more memory than anything's going to have for a long time. So Intel said, okay, we're not even going to bother with all 64 bits. We'll use 48. So on 64-bit Intel processors the virtual address space is 48 bits. There is what's known as page-level granularity, meaning that memory is allocated in pages, which happen to be 4K for various reasons. It's convenient. That's the typical cluster size on a

hard drive. So when you need to load a page, you load a cluster. So 4K sort of was a nice size. 4K is 12 bits. That is, it takes 12 bits to address 4K.

So in a 48-bit address, the lower 12 bits is the offset in the page. That is, 12 bits gives you 4K bytes, so it's the location, the byte location within the 4K page. So that's 12 bits out of 48, leaving 36 higher order bits. Those are broken into four sets of nine bits. Nine bits, since eight bits gives us 256 combinations, nine bits gives us 512. So each of those four sets of nine bits is considered a pointer into a system global page table. So the highest order nine bits of the 48-bit address, that actually is a pointer to a 512-entry table which is the first page table. And its location, its absolute location in physical memory is pointed to by a register in the processor called CR3, Control Register 3.

So the highest nine bits is the offset into that table, which contains a pointer to the second-level table, which the second most significant nine bits provides the offset into. That entry points to the third-level table, where the third most significant nine bits provide the offset into that table; which points to a fourth-level table, which the fourth and least most significant nine bits of the upper 36 provides the offset to; which believe it or not finally points to the actual page in physical memory.

Leo: Does some of this go back to the memory segmentation of the early x86?

Steve: Yeah. It's like that on steroids to the power of infinity.

Leo: Because I remember when I was doing assembly language on the Motorola 68000 it was such a joy because the memory space was flat, had big enough registers you could directly address anywhere in the memory. You didn't have to do this segmentation thing.

Steve: Yes. And as a programmer of 16-bit code, I was, I mean, a lot of SpinRite still is 16 bits because everything I've been writing for the last couple decades has been 32 bits. So there's more and more 32-bit code coming into SpinRite. But the original stuff had segments, and I was always having to, like, be loading the segment...

Leo: It's so annoying.

Steve: Oh, it is. And so for me...

Leo: So why do they still do it? The registers are 64-bit now. They could address all that memory easily, directly.

Steve: Ah, yes. It's because you give the processes the illusion of 3GB, or however much memory you want to. But many different processes are sharing it, and it doesn't actually exist. That's what a page fault is. A page fault is an attempt to map a page which is not currently in physical memory. And so that's where swapping comes from. So what happens is, if that page is marked as swapped out, then the process thinks it has the memory, and it's completely hidden from it. It says, "Give me this byte at this address."

Well, so the processor looks up through this crazy four-level hierarchy of page tables and goes, whoops, we swapped that out last week, and no one's asked for it recently. So it has a pointer to where it is. It reads it into physical memory and then adjusts the page tables to point to that. So the code has no idea any of that happened. Oh, and the virtual memory manager, which is where that's going on, it's scrambling stuff up all over the place. Stuff's being swapped out, brought back in. Processors are being put to sleep and woken back up. So this is all, I mean, it's incredible that this all works. And it all works because it was developed carefully over time, and it's sort of a process of evolution, sort of like a human body. It's amazing that it all works as well as it does.

So all of these levels of indirection, this four stages of hierarchy provides a huge amount of flexibility to the operating system. But imagine the overhead. I mean, we're talking every single instruction fetch, every single data write or read has to be mapped through this system. Well, you couldn't get off the ground if there wasn't some way to speed that up. So the first thing there is, is something called TLBs, Translation Lookaside Buffers. The TLB, it's got a funky name, Translation Lookaside Buffer. But that's like the cache for all of that work.

So the Translation Lookaside Buffers provide a shortcut that maps virtual memory to physical memory so that you don't have to follow this four-layer linked list, essentially, every single time you want to do anything with physical memory, whether it's execute an instruction or read or write data. The problem is there's only so much TLB space available. Processes are busy. And the TLB starts out empty in the beginning when you boot up. So it's only as a consequence of doing all of that fetching through all these crazy page tables that you end up maturing the TLB so that it's got a useful working set of mappings in order to convert virtual addresses to physical addresses.

Okay. So if the TLB does not have the mapping, there is an LRU flush, a Least Recently Used replacement policy which - so if an entry hasn't been used for a while, it gets pushed out of the TLB by what's been going on more recently. So that's standard caching logic. So the processor looks through the TLB, hoping to find a match for the virtual memory range that it's having to do something with, anything with - execute an instruction, read or write data. If it doesn't find it, now it's like, oh, crap. I need to do the lookup.

So, now, these page tables exist in main memory. And if there's anything we know about DRAM it's not only can you hammer it, but it is slow, it is excruciatingly slow compared to the contemporary speed of today's processors, which have just shot ahead, I mean, with dizzying speed. What, 4.3 GHz, 4.3 billion cycles per second. Meanwhile, DRAM's technology has kept it back. So it's way faster than a hard drive, but it is agonizingly slow compared to the processor's appetite.

So what's been developed is Level 1, Level 2, and Level 3 caches. The Level 1 cache is the closest to the execution and data units. And it's actually split into Level 1 instruction cache and Level 1 data cache, meaning that instructions have their own Level 1 cache, and data reads and writes have their own. That's to prevent either one from knocking the other one's data out of the cache. Then the Level 2 cache is below the Level 1 cache and feeds the Level 1 cache. And that's shared between instructions and data. But both Level 1 and Level 2 caches are per core. So if you have a quad core chip, each of the four cores has its own Level 1 data and instruction cache and Level 2 cache.

However, the entire chip of multiple cores shares a much larger, like on the scale of 8MB, Level 3 cache on a large chip; whereas, for example, the Level 2 are more like 32K. They're big and very fast and accessible to the core. But Level 3 cache, still on the chip, is more like 8MB. And then that's - if it's not in the Level 1 cache or the Level 2 cache or

the Level 3 cache, then the system has to go out and ask for it from DRAM, which in pulling it into the chip, it leaves it in the caches, where it can then be found from there on.

Okay. So if a mistake was made, a fundamental architectural mistake, it's that it never occurred to anyone. And I don't mean just Intel because AMD has the same problem, call it a vulnerability, as does the ARM architecture. They all do this. The Memory Management Unit with this crazy multiple hierarchy of page tables, in order to get the speed it needs, if the Translation Lookaside Buffer has had its entry knocked out, uses the caches. It uses the same set of caches as the program. And that's the problem. What this means is that Memory Management Unit activity, which is by design hidden from the code, that's all down in the OS. You could argue it's even below the OS. It's in the hardware.

But the point is, programs do not need to know their physical addresses. They see their virtual addresses. And they should not see them. It's hidden by design. But the mistake that every processor makes is that this Memory Management Unit, for the sake of performance, is sharing the caches with executing processes. And we know how an executing process can determine if something is cached or not. If something is cached, and it's able to determine the length of time it took to get it, the cache is so much faster that it will get an almost instant answer. In fact, it is instant, if the cache is close to the core. If it's not in the cache, it forces a DRAM read which is measurably slower. So this allows operating code to probe what's in the cache and is not.

And what these guys so cleverly figured out is that, if something as crude and coarse and granular and high-level and interpreted, well, JavaScript is not as interpreted as it always is because we're trying to speed it up so much, but still, given the limited resources of JavaScript running in the browser, these guys figured out how to essentially use the fact that memory management was sharing the data cache of the code that was running in the browser to probe and reverse-engineer the contents, the recent history of the Memory Management Unit's use; and, for example, to turn in the virtual address of the heap for the process, which is memory which should otherwise have no known value.

So we talked about timing and the vulnerability of our systems to timing attacks in the past. And as a consequence, browsers - well, for example, there were several attacks where the time required to decompress data was used by adding data to headers and making lots of queries, JavaScript was able, by accurately measuring the time it took to perform this, was able to reverse-engineer the contents of the unknown data because the compression would compress the added data, if it was the same as the unknown data, and that would change the length of the compressed result. And so very clever scripting was able, with enough queries, to figure out the unknown data compared to what was known.

Browsers years ago in response to those attacks which we covered on the podcast, they decided, okay, we're not going to give JavaScript super granular timing information. The browsers, both Firefox and Chrome, reduced the resolution of the API, which captures the current instant in time from infinity. It used to be the resolution of the clock. So it was nanoseconds. They reduced it to five microseconds. So they deliberately removed resolution so that JavaScript could not determine as accurately as it originally could what time it was. So once again, not to be daunted, our clever engineers said, okay, fine. We don't have accurate timing. But is the five microseconds itself accurate?

And it turns out that Chrome and Firefox differ a little bit. Chrome is a little trickier. They fudge the actual timing event. It still has five microseconds' resolution. But they delay it a couple different amounts in order to, like, further obscure the actual moment of that

five microseconds event. Firefox didn't do that. So, for example, in Firefox the way you solve this problem is you query the API until the timer ticks. So that gives you tick alignment. Then you perform the operation that you want to time, in this case a read or a write of a certain location because you're wanting to probe the cache. And as soon as you get control back from that operation, you start doing something, you start counting how many times you can go in a loop until the next timer tick.

And so what happens is essentially the five microsecond resolution puts fenceposts. And you align yourself with one fencepost at five microseconds, do something, and the moment you get back, you spin until the next fencepost, until the next value, the next five microsecond tick occurs. And so the number of spins that you are able to accomplish, that allows you to infer, by comparing cached and uncached counts, spin counts, essentially, whether the unknown timing was short or long. So that's one of the clever things these guys did. There's another approach using WebWorkers, which essentially extends the JavaScript API to a multithreaded environment, where you're able to say, you go off and do this. Oh, and you also - the workers, in order to coordinate, they have a region of shared memory, memory that they can both see.

So the WebWorker, you give it the task of simply spinning up a counter in the shared memory as fast as it can. Now you have your own timer, which you're able to use, which is, again, much higher resolution than the deliberately blunted resolution of the JavaScript timer in order to determine how long something took to occur. So these guys worked around the deliberate timing blunting that the browsers added to JavaScript. They realized that, because all the current processor architectures share a common cache, which is available to operating code, with the Memory Management Unit - which they have to cache, otherwise the whole system would just come to a grinding halt. If the page tables only were non-cached, nothing would ever get off the ground.

So but the problem is they could have been given their own cache. Instead, they shared that caching, which made good sense in terms of resource utilization for the processor, but it was a critical architectural mistake in this context. If you absolutely do not want programs to know where stuff is, you can't let the programs have any visibility into the memory management. And as a consequence of the fact that page tables are stored in main memory and are cached in a common cache which can be probed by timing-sensitive code, unfortunately we no longer have the protection that ASLR was designed to provide. So bravo to these guys. And I don't know what this means. I mean, the browsers could further work to fudge things. But as I said, there's nothing you can do to prevent this at the native code level.

So, I mean, and that's really where the big problem is with malware that people download by mistake and are relying on system-level protections to keep that code from being able to elevate its privilege to root or kernel level. This presents bypass opportunities. This is all in the public domain. It's open source. There's a bunch of utilities these guys created, both JavaScript and native code that demonstrates it. One of the papers is this 22-architecture demo that they provide, where essentially they did a generic memory mapping unit reverse-engineering code because these inner levels of detail don't need to be known by the software. So they're undocumented across these various chip families. The high level, how you talk to the Memory Management Unit, everybody knows that. But the inner details are - it's not that they're such a big secret. But it's like, well, it's just the hardware architecture. Nobody really needs to know how it's all wired together. To pull this attack off, you have to know.

So these guys have a tool that allows them to reverse engineer, under automation, to reverse engineer and determine the exact sizing of every level of caching in the hierarchy and the nature of the cache replacement policies within the chips. Just a phenomenal,

beautiful piece of engineering, and something that will probably drive some changes in the architectures of our chips moving forward. The bad news is there's nothing we can do about the entire install base of chips, of chip hardware that we have now. We're in for an interesting ride.

Leo: Not that, once you get the memory table, the rest is cake.

Steve: Correct.

Leo: I mean, you have to inject code somewhere in that memory table. You have to figure out where the next jump is, that kind of thing.

Steve: Correct, yes. And that's why I was critical, for example, in Ars Technica, who said new ASLR-busting JavaScript is about to make drive-by exploits much nastier. No, no, it's not. It makes them significantly more possible, and perhaps more likely, because this has the potential for defeating address space layout randomization, which we really are depending upon.

Leo: Well, I mean, there's been other issues with ASLR. It often breaks things. Does everything use it now?

Steve: Yeah, it's becoming very widespread. The most...

Leo: Because back in Windows 7 it was optional.

Steve: Yeah, they attacked Linux because Linux's ASLR is 28 bits of entropy, whereas Windows is only, I want to say - 28 bits in Linux. I think it's 24 bits in Windows. And so they tackled the harder of the two platforms.

Leo: Good for them. Good for them.

Steve: And they made this work both in Firefox and Chrome on Linux.

Leo: Wow.

Steve: Yeah, beautiful piece of work.

Leo: Yeah. It's quite elegant, that little timing thing.

Steve: Yup.

Leo: Very clever. Well, there you have it. Don't know what the upshot is.

Steve: Okay. Everybody can start chewing gum again.

Leo: Take your propeller hats off. Start chewing and walking. It wasn't as bad as you said. Or maybe just because I've had to already wrap my head around address segmentation for so long, which is a terrible thing.

Steve: Well, and remember, these things, as we know, exploits only get worse. They never get better. And so this is a major wedge in a technology that, to a large degree, a mitigation we've been depending upon.

Leo: Right, right.

Steve: Beautiful piece of work.

Leo: And to be fair, we didn't have ASLR till recently. Right?

Steve: Correct, yes.

Leo: So it's not like - it's going to be back to the old days. It's not like it's, oh, everything's, you know. This is a relatively new technique to make you more secure. And there are other things protecting you, I would hope.

Steve Gibson is at GRC.com. And someday we're going to answer your questions, maybe even next week, so get them out and put them there at GRC.com/feedback, or tweet him at @SGgrc. When you get to GRC.com, you'll note many fine and fabulous things. Don't get too distracted by those until you buy a copy of SpinRite, the world's finest hard drive maintenance and recovery utility. It's right there. Then after that you can try ShieldsUP! and Perfect Paper Passwords, the Password Generator, read about SQRL and the Healthy Sleep Formula. There's so much stuff there: GRC.com.

Also the show, of course, 64Kb audio, human-written transcripts by Elaine Farris, all at that site. We have audio and video at our site, TWiT.tv/sn. We also have a survey we'd love you to take. Just a week left in the survey, so if you can get to it before the end of the month, take a couple of minutes of your time, TWiT.tv/survey. It lets us know a little bit more about you in aggregate. We don't take any, you know, we don't want your personal information. We're not going to try to figure out who you are. We just want to know what percentage is men, what percentage is women, what percentage have college educations, what's the income. And some of that's for us for programming purposes, but I think we know you pretty well for that. Mostly it's for us to tell advertisers because they always want to know that stuff.

Steve: Right, right.

Leo: "Well, who's listening?" I don't know. They're good people. Everybody, all the other advertisers like them. They want that stuff. And so it helps us a lot to monetize the network. And if we don't do that, well, we got no network. So TWiT.tv/survey. If you would, it really helps out.

We have caps now, by the way. We have TWiT logo caps and TWiT.tv caps in the TWiT Store, TWiT.tv/store. We don't make a lot of money on it. It's not really a profit thing. It's more because, well, people have asked us. "We want T-shirts. We want mugs. We want hats." And now we've got really nice hats, actually. I'll wear one next time. That's at the TWiT Store, TWiT.tv/store.

Steve, let's do this again, maybe, I don't know, I'm thinking next Tuesday, about 1:30 Pacific, 4:30 Eastern?

Steve: How about for Episode No. 601?

Leo: 601.

Steve: Yeah.

Leo: It was a good 600. Kind of a nice show, as always. Congratulations on 600. Well done. Twelve years.

Steve: Yes, in our 12th year. Okay, my friend. I will talk to you next week.

Leo: Thanks, Steve.

Steve: Thanks.