The PACMAN Attack

Description: This week will, I expect, be the last time we talk about Passkeys for a while. But our listeners are still buzzing about it, and some widespread confusion about what Apple presented during their WWDC developers session needs a bit of clarification. While doing that, I realized and will share how to best characterize FIDO (which we're going to get) with respect to SQRL (which we're not). I also want to turn our listeners onto a free streaming penetration testing security course which begins Wednesday after next. Then we have a ton of listener feedback which I've wrapped in additional news. One listener's question in particular was so intriguing that I'm going to repeat it, but not answer it yet, so that all of our listeners can have a week to contemplate its correct answer. And although I wasn't looking for it, I also stumbled upon a surprising demonstration proof that we are, indeed, living in a simulation. When I share it, I think you'll be as convinced as I am. And finally, as suggested by this podcast's title, we're going to take a very deep dive into the past week's headline-capturing news that Apple's famous M1 ARM chips all contain a critical bug that cannot be fixed. Just how bad is it?

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-875.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-875-lq.mp3

SHOW TEASE: It's time for Security Now!. Steve Gibson is here. Some final thoughts about Passkeys and the potential problem with Passkeys. Also a free streaming pen test security course which begins next week. A lot of listener feedback. And then, finally, what is that flaw, the so-called unfixable flaw with Apple's M1? Steve explains the attack and why you probably don't need to worry about it. It's all coming up next on Security Now!.

Leo Laporte: This is Security Now! with Steve Gibson, Episode 875, recorded Tuesday, June 14th, 2022: The PACMAN Attack.

It's time for Security Now!, the show where we cover the latest security with this guy right here, the man in charge. Get ready, folks, this show is going to blow your mind. Steve Gibson, you promised me that; right?

Steve Gibson: Oh, we've got a good one today, Leo, for 875. I think that's sort of a good number to have an amazing podcast. Okay. So this week will, I expect, be the last time we talk about Passkeys for a while. But not quite yet. Our listeners are still buzzing about it, and some widespread confusion about what Apple presented during their WWDC developers session does need a bit of clarification. But while I was writing that up, I realized, and I'll share, how to best characterize what FIDO is, which we're going to get, with respect to SQRL, which we're not. But more importantly, what the WebAuthn issue

is. I mean, that's really the big transformative deal that having FIDO on the front end is going to drive, and it's where all the inertia is going to be.

Anyway, I also want to turn our listeners on to a free streaming penetration testing security course which begins Wednesday after next. Then we have a ton of listener feedback which I've wrapped in additional news. Oh, and one listener's question in particular was so intriguing that I'm going to repeat it, but not answer it yet so that all of our listeners can have a week to contemplate its correct answer.

Leo: I love those. All right. My pencil's poised over my note pad. All right.

Steve: It just needed it. I mean, it's like a, hmm. And although I wasn't looking for it, I also stumbled upon a surprising demonstration proof that we are indeed living in a simulation. And it's broken. That's the point of the proof.

Leo: Wait a minute. The simulation's broken?

Steve: Well, reality is.

Leo: Oh, okay.

Steve: And reality can't be broken unless there's a bug in the simulation. Anyway, when I share it, I think you'll be as convinced as I am. So kind of maybe we're going to rock everyone's world today.

Leo: Let's do it.

Steve: Finally, as suggested by this podcast's title - oh, which I forgot to mention is "The PACMAN Attack" - we're going to take a very deep dive into the past week's headline-capturing news that Apple's famous M1 ARM chips, and presumably the M2s coming soon, all contain a critical bug that cannot be fixed. We'll find out just how bad it is.

Leo: Yikes.

Steve: And Leo, I've got the weirdest Picture of the Week. You can't argue with it. You know? But it's - who printed this on a cable?

Leo: That's a good question, and I'm seeing it, so I know what you're saying. Well, we'll find out in a moment.

Steve: Okay. This is not my fault. I didn't do this.

Leo: Not your fault.

Steve: Okay. So what we've got here is an Ethernet cable. You know, CAT 5 or CAT 6. And it says, it's printed, very authentic-looking, I mean, I believe that this cable actually has this printed on it. It says "Cut here to activate firewall." And then it's got an arrow pointing to a little line, and you can see an arrow pointing from the other direction where it probably says the same thing, "Cut here to activate firewall," telling you that, yeah, I mean, it just must be somebody who is making cable with a great sense of humor, who said, okay, we'll just put something funny on it.

Leo: He's not wrong. He's not wrong.

Steve: No. And the person who sent this to me used a big red rectangle to highlight that phrase. Unfortunately, it overlays the top-level domain. I can see www...

Leo: We want to know where you get this, don't we.

Steve: Yes, yes. I was like, okay, where, what? So I can't see what the top-level domain is. It's www.bit dot something. Looks like maybe an "n" something. I don't know. I mean, I think the shortest TLD is two characters, so I think we're missing...

Leo: Maybe an "l"? Bit.nl, that's Netherlands; right?

Steve: That feels like, yes.

Leo: This feels like a Dutch joke.

Steve: It does.

Leo: Cut here to activate firewall.

Steve: Yeah, yeah.

Leo: I love it because they give you a line and arrows pointing at the line so just in case you're concerned about where you're cutting, you know exactly.

Steve: Yeah, well, because you wouldn't want to cut like in the middle of one of those words.

Leo: No, no. Cut the right spot.

Steve: Because, you know, who knows what would happen?

Leo: It's got to be made up. Don't you think it's made up?

Steve: You mean the whole picture was, like, faked?

Leo: Yes.

Steve: It's pretty good, Leo.

Leo: I'm going to go to bit.nl, and I'll find out for you. If I can order them, I'm getting you a whole roll. We'll call it the Security Now! - oh, yeah. You know what? It's a datacenter, bit.nl. It's in Dutch, but a datacenter's network managed hosting in the cloud. So I think they did this as a joke. I love it. I love it.

Steve: Very cool.

Leo: Yeah.

Steve: Okay. So I heard from many different listeners that during the WWDC developer presentation on Passkeys, Apple talked about synchronizing keys. So I listened carefully to the entire presentation. For anyone who's interested, the 34-minute presentation video is this week's GRC shortcut of the week. So that means it's https://grc.sc/875. As I said, it's about a 34-minute presentation, 33 or 34. And I believe that what these people thought they heard was Apple addressing the need for - or thought that what they heard being Apple addressing the need for the type of synchronization we talked about last week, syncing apps across non-Apple ecosystems like Android and Windows, is not what happened. I found no mention of anything like that anywhere in the presentation, nor is it anywhere in the developer docs, which I've also linked to in the show notes, for anybody who wants to jump right to it.

The types of Passkey sharing Apple supports is first and foremost using the iCloud Keychain, of course, to dynamically synchronize keys, these Passkeys, across an individual user's Apple ecosystem. So as we know, all Apple devices thus will remain synchronized. The other form of sharing Apple described uses AirDrop to pass logon credentials to another user. And AirDrop can also be used to permanently pass a Passkey to someone else for some site or service, permanently adding it to their Keychain to use from then on. So that's sort of like explicit, here's my Passkey, you can now use it to log on as me. But so far, from everything I've seen, Apple has in no way suggested that they will ever be synchronizing Passkeys with external non-Apple platforms. Nothing has been said so far either way. They haven't said they're not going to, but nobody seems to have asked that question, and it was not part of the developer presentation.

But Apple's example solution of using Airdrop to send a Passkey to another person's iDevice - like a friend of yours, a spouse, a child, a sibling, whatever - for their subsequent use, highlighted something that I think is important to understand. And this is where, in thinking about it, I realized what really makes SQRL different from FIDO and why. SQRL is a complete solution for secure remote logon; whereas FIDO and technically FIDO2 with its Passkeys, is a replacement for usernames and passwords. The two are not the same.

For example, take the case of giving someone else access to a site. If you give them your Passkey, which is Apple's solution, demonstrated during the developer presentation, then they are now "you" on that site in every meaningful way. When they authenticate, it's

you authenticating because they're using your Passkey. It's the exact equivalent of you giving them your username and password. And since they are you, they can see your settings, your private details, everything that you can see and do when you log in using that same Passkey. And since they are you, they're presumably also able to change the Passkey to lock you out. And they can presumably pass it along to others, unless Apple has realized that secondary Passkey sharing is a really bad idea and should be blocked, which would technically be possible. I don't know either way.

Leo: Well, this is the situation we're in right now. When Lisa needs the login to our Comcast account, I just send her my password and login.

Steve: Right.

Leo: She doesn't have a separate one because it's the same account.

Steve: Right.

Leo: So that's the standard. Does SQRL solve that with some sort of shared access? No.

Steve: Uh-huh. Of course. Of course.

Leo: How could it? Because Comcast only has one login for my account.

Steve: Okay. So I don't know either way whether they're going to block secondary Passkey sharing. In any event, when you volunteer to give your Passkey to someone else, your site access has now escaped your control. And I agree with you, Leo. It's exactly the same thing.

We solved this with SQRL. If you want to allow someone to share some access to an account as a guest for example, sharing a Netflix account you obtain a one-time invitation token from the site and provide it to them. When they attempt to log into the site with their own SQRL ID, the site doesn't know them, so it prompts them to create a new account or to use an outstanding invitation if they have one. Their use of the invitation identifies them to the site as a guest of yours, enabling them to subsequently log into your account using their SQRL ID. Since they're using their SQRL ID - and we just sold a copy of SpinRite 6. Thank you, if you're a listener.

Leo: Woohoo.

Steve: Since they're using their SQRL ID, and guests are unable to request invitations for others, you, as the account owner, retain control. And you're able to rescind their guest access status at any time, which isn't possible otherwise, I mean in traditional username and password sharing. And this all scales seamlessly to enterprise use when hundreds of users might need to share access to common resources. It's called "Managed Shared Access," and it's part of the SQRL solution. It's already there. We have an online

demo with the entire solution working, and its operation is fully worked out and specified. And needless to say, there's a lot more to SQRL.

So as it stands, the FIDO2 Passkeys system is without question more secure than usernames and passwords. No doubt about it. It's definitely superior. But the FIDO designers were crypto people working to solve one small part of the much larger problem of practical real-world user authentication. They didn't think the whole problem through because that was never their charter. They could credibly say it wasn't their job. It wasn't. But even in a FIDO2 Passkeys world, that job still needs to be done. SQRL does it; but unfortunately, FIDO and Passkeys does not do it.

Unfortunately, this means that instead of being as revolutionary as it could have been, we get another half-baked solution. It's way better than what came before, but it missed the opportunity, which comes along so rarely, to address the practical needs of, and really solve, the network authentication problem. Rather than the true breakthrough that SQRL's adoption would have meant, we're going to get incremental progress. It's definitely progress. But because it wasn't really thought through as an entire solution, FIDO is basically a crypto hack. It also brings a whole new set of problems. If FIDO is to be our solution, we really do need some form of centralized Passkey storage and synchronization, not only within a vendor, but also across vendors.

Last week, someone calling themselves Captain Jack Xno mentioned @SGgrc in a tweet to someone else, so it appeared in my Twitter feed. Captain Jack wrote to this other person: "You may be excited about Passkeys, but SQRL was carefully developed over seven years by @SGgrc and solves problems you may not even realize you'd have." And he mentioned potentially cross-platform portability. And yeah, as we know, SQRL does that. And it does so much more.

Someone tweeting as @drnathanpgibson, I mean, that's his Twitter handle, said: "Hi Steve. Loved your detailed breakdown of Passkey. You mentioned waiting for password managers to start providing sync services for these FIDO2 private keys. I see that LastPass seems to be promising something 'later this year.'" And then he has a link to the blog. He says: "Do you know anything more about when this syncing might be coming to a password manager near me?"

And so I saw LastPass's blog post last Monday the 6th. It was a bit confusing. And, I mean, I spent some time trying to figure out what they were saying because they were at the same time also promoting the new use of what they called "No more passwords today." And what I understand is that apparently that's by the use of their own LastPass authenticator which would use the biometrics present on a handset. Thus you could unlock your LastPass vault without using a master password. Okay, so not that big an announcement.

But separate from that immediate announcement was indeed a forward-looking statement of their intention to support FIDO2 Passkeys. So that's not today, but at least one major password manager is taking aim at this problem. And if one does, they'll all need to. So I suspect that the biggest effect of Apple's, Google's, and Microsoft's support may be to induce websites to bring up their own support for WebAuthn, which is what's necessary on the back end. And so let's talk for a minute about...

Leo: By the way, that's what 1Password says they're going to do is support Authn.

Steve: Good. Good.

Leo: Yeah. So that's the way to do it; right?

Steve: Yes.

Leo: And then presumably there'd be an export/import feature from 1Password Manager or iCloud, I would hope.

Steve: I would - well, yes. Now, it'll be really interesting to see whether Apple allows a wholesale export.

Leo: Well, they say they are. They say there's an export feature. Yeah?

Steve: No. No.

Leo: They didn't say that.

Steve: No. They said there's a Passkey sharing, which is different. You use Airdrop to send one key to a phone...

Leo: No, no, I know that's not the same. But somebody told me. You've watched the presentation.

Steve: I watched the presentation. I read the developer docs. There's not a word about...

Leo: Oh, okay. There's no export.

Steve: About export. And if I'm wrong, listeners, please correct me. I'd much rather it be true that they're going to allow export than just be a curmudgeon and say, wow, they didn't get it right. So if anyone finds that there's an export of Passkeys from i-land, I want to know.

Okay. So WebAuthn. The heavy lift that FIDO will face and that SQRL would have faced was the need for backend web server support. As we know, it's surprising, always surprising how slowly things change. No question about it, it's going to take years. It's going to take all of the major web server platforms to build it in, then for all of the existing web servers to be upgraded to support it and then to do whatever they need to do on their end to actually give it a database and enable it and have the lights on. And since it's not clear that there's a huge benefit to them, since things are sort of working as-is, I think we can expect it's going to take a while. Think about how long it took for the "Logon with Facebook" and "Logon with Google" OAuth kludge to finally become popular. And it's still far from universal. It's around, you encounter it, but you certainly can't use it everywhere.

What you can use everywhere is the original fill-out-the-form username and password. The reason password managers were an overnight hit was that they provided an entirely

client-side solution which did not require any backend change to websites. Web servers didn't know or care how a user was arranging to provide their username and password. And they didn't need to care. But make no mistake, automated form-fill of username and password is and has always been a horrific kludge. The fact that kludges mostly work doesn't make them any less kludgy.

WebAuthn finally, and at long last, changes that. The adoption of WebAuthn, which was approved and formalized by the W3C consortium three years ago, back in 2019, represents a massive and long-needed update to username and password authentication. As I mentioned last week, FIDO and SQRL work essentially the same way. They both give a web server a public key. The web server generates a random nonce which it sends to the browser. The browser, holding the matching private key, which it never releases, signs and returns that nonce to the web server, and the web server uses the public key that it has on file to verify the returned signature. What WebAuthn is and does is provide all of the mechanics, definitions, protocols, specifications, and implementations of that new form of interchange between a web server and a web client.

Now, we're likely going to be facing a chicken-and-egg situation for some time. What kind of got lost amid the ballyhoo of Apple's announcement last Monday is the fact that you can't use it anywhere. I mean, it's like having SQRL, which we've had for years now. But you can't actually log in anywhere but GRC and a couple of our...

> **Leo:** TWiT.community also. You can use your SQRL there. I think you can. Well, I set it up. And then remember one of your guys in the forums was providing a backend server because you need a backend server for SQRL; right? Some sort of authentication thing.

Steve: Well, you can - I don't have one on any of my implementations. But you certainly could. You could federate the authentication to a backend server.

> **Leo:** There was something going on I couldn't do that he was doing. And then he let that slide, and so people were using SQRL to log into TWiT.community, couldn't use it, and then he said, oh, yeah, let me see if I can get - I don't know. I haven't kept up on that.

Steve: Then it just sort of died.

> **Leo:** Yeah, just sort of died out, yeah. But I needed something, some piece that he was providing.

Steve: Well, thank you for trying, Leo. It will be good that Apple and Google and Microsoft will all be supporting Passkeys, which is to say FIDO2 on the client side and WebAuthn on the protocol backend server side. That's the glue that makes this possible. But when iOS 16 arrives with its built-in Passkey support, you'll probably only be able to login to Apple.com, Google.com, and maybe Microsoft.com, due to the heavy lift of change that will be required on the back end.

But WebAuthn is the key. Period. It provides a complete replacement for the insecure mess of usernames and passwords that we've been living since the dawn of man. And interestingly, WebAuthn optionally supports SQRL's chosen 25519 elliptic curve.

Leo: Yay.

Steve: Yeah, with its special properties that allow for non-random deterministic private key synthesis. That's the key behind SQRL, as I've mentioned, if you'll pardon the pun. So it might be possible, someday in the future - I'm not going to do it, we've got a bunch of SQRL developers now, they'll do this - to transparently run a modified SQRL solution to use SQRL-style deterministic Passkeys on the server infrastructure that FIDO built. So that would be cool. And perhaps indeed the only kind of progress that can practically be made in today's world is incremental.

So the fact that everyone's excited, and it's going to be available in our clients, and when web servers start adding Authn support, their UI, their login UI will show, hey, if you've got a Passkey, use it. And so you'll give them, the first time you login as you, using old-school login, then you'll have your client generate a passkey which they'll hold onto, and you can then use that to log in in the future.

But the other point I wanted to make was that what I really did with SQRL, and this is exactly to the point you were making with you and Lisa, is I didn't just stop at replacing usernames and passwords. I solved the whole problem. I mean, as you heard me say back then, every single question anybody could ask about what if this, what if that, blah blah blah blah, I had SQRL solve the problem and provided all kinds of forward-looking solutions that would be handy to have. We're not going to get it yet. Maybe we'll get it eventually.

Leo: Yeah. I realize that Discourse, which is the forum software I use - XenForo you don't have to do anything. I think it's built in. But Discourse has to use an OAuth2 provider. So Jose C. Gomez, who I think is probably still around in your forums, had set up a SQRL OAuth2 provider.

Steve: That's very clever. Basically an OAuth gateway for SQRL.

Steve: Gateway, exactly.

Leo: But it's gone, and so it's a 502 Bad Gateway.

Steve: Bad gateway. Bad gateway.

Leo: Bad gateway. So unfortunately everybody who used SQRL that set up an account on TWiT.community I guess is out of luck.

Steve: [Buzzer sounds]

Leo: So, sorry, I apologize. I don't know enough to run an OAuth2 server myself.

Steve: Well, and as we know, I've moved on. So I did what I could, and maybe we'll get little pieces of SQRL over time.

Leo: Admittedly, this is not perfect. But, you know, the perfect can be the enemy of the good. And Passkey is much better than what we're doing right now.

Steve: Yes.

Leo: So, yes, there could be something better.

Steve: I guess my only argument is that making a change on the back end requires everybody to make the change.

Leo: Right.

Steve: And it's such a heavy lift that, if we had just - if we were going to make a change, let's make it a good one. I mean, let's make it one that actually solves all these other use cases for which people are like, you know, sharing passwords, and what if a bad guy did get a hold of your key and then were able to use it to lock you out of an account. That doesn't work with SQRL. Some person who gets your SQRL identity cannot lock you out of accounts where you are registered with SQRL. It's just that there's so much there. And that's where the time went. But I got it out of my system, and now I'm back to SpinRite.

Leo: Do you know what the lift is, well, you must because you've watched this video now, how hard it would be - let's say I wanted to implement Passkeys on a site.

Steve: So no mortal will do that.

Leo: It's that hard.

Steve: Yeah. And so but it'll be a library. For example, I googled "FIDO2 Passkeys IIS." Obviously Microsoft server. Zero hits. There's nothing for Microsoft's web server.

Leo: So just it's like with SQRL. XenForo supported SQRL, so it was easy for you to implement. Discourse does not. It requires an OAuth2 backend.

Steve: Actually, XenForo did not support SQRL.

Leo: Oh, it didn't. But somebody wrote something.

Steve: That was Rasmus Vin.

Leo: Rasmus wrote that.

Steve: Our wonderful PHP guy who built -

Leo: So he hacked XenForo to do that.

Steve: Yeah, well, actually XenForo has a beautiful add-ons architecture. So he was literally able to create an add-on that any XenForo user, any XenForo site could just do it. And of course none have because what's a SQRL?

Leo: Well, you're reliant, just as I am with Discourse, I would be reliant on Discourse adding that capability. And somebody who's using WordPress, they would need a plugin. The good news is that the web is mostly dominated by a handful of servers.

Steve: Yes. Very few servers, and they will all get WebAuthn plugins, or modules in the Apache case, for example. And so you just, you know, add the WebAuthn module and configure it, and you're probably good to go.

Leo: Yeah, right. Well, that day I hope comes soon.

Steve: It'll be interesting to see. It's not, and this is the problem, there was a benefit to Google and Facebook doing the login with Google Login with Facebook OAuth hack because, as we know, they track you as you pass by. They know who you are and where you're logging in. So this federated OAuth login is, you know, they had a benefit. They had a reason for doing it. It's not clear to me what the benefit will be to web servers and websites adding Passkeys. Things are working the way they are now. So we'll see.

Leo: Yeah, yeah.

Steve: And as I said, not everybody is doing the logon with Google and Facebook. You certainly can't use it universally. You see it here and there. And it's kind of like being able to buy something with PayPal. It's like, oh, good, I can use that here. But other places, oh, you need my credit card number [grumble].

Leo: WordPress does have a SQRL plugin. Did that get a lot of uptake?

Steve: Yeah.

Leo: It did.

Steve: Yeah, yeah.

Leo: I think that's got to be key. You've got to have those easy switches.

Steve: Yes, got to be a drop-in solution. So last Wednesday a company known as Offensive Security, who are the people behind Kali Linux, announced that they'd be live streaming their "Penetration Testing with Kali Linux" course sessions on Twitch later this month. I mention this because a subset of the course's material all of the streamed content will be open to the public on Twitch at no charge. And I felt sure that some of our listeners would find that interesting. For those who don't know, Kali Linux is a Debian-based Linux distro which is targeted toward facilitating information security tasks such as Penetration Testing, Security Research, Computer Forensics, and Reverse Engineering.

The course in question, PEN-200, is a paid course which helps students prepare for the Offensive Security Certified Professional (OSCP) certification exam. Before the pandemic it was only available in person. But during the COVID response live training was suspended and Offensive Security moved their courseware online for remote instruction. So today a 26-part, 13-week, twice-weekly hour-long per event online course will be offered to prepare students for the OSCP certification. Non-enrolled viewers are welcome to audit the course. You of course won't get any credit for it, but there's the information. There's an FAQ in the show notes, or a link to the FAQ in the show notes, which details everything you'll need to get going.

I grabbed the two most relevant Q&As from that page. First one is where and when will it be happening? And their answer is the OffSec Live: streaming sessions are currently planned to start June 22nd, 2022, so that's Wednesday after next at from 12:00 to 1:00 p.m. Eastern, so that's currently 9:00 to 10:00 West Coast, and run through December 7, 2022, so pretty much the rest of the year.

The OffSec Live: PEN-200 streaming sessions will consist of twice-weekly, hour-long interactive presentations. They said: "We'll also be streaming a weekly office hour where we will address any questions a student may have regarding the PEN-200 course and prior streaming sessions." And those, I would imagine, will be only for enrolled people.

And how much will it cost? And they said the OffSec Live: PEN-200 Twitch streaming sessions will be free and open to the public. So just a heads-up for our listeners.

Before I introduce the proof that we are living in a simulation - just wait. You laugh now. Just wait. I did want to mention that Surfshark has followed ExpressVPN in pulling out of India. I heard Andy mention on MacBreak Weekly that he logs into a VPN in India.

Leo: Alex, yes.

Steve: Oh, Alex, yeah.

Leo: So he can watch the Pittsburgh Steelers football game.

Steve: Yes. And the good news is, as I mentioned last week, at least in the case of ExpressVPN, you can still get an IP from India, even though it's being hosted in some other country.

Leo: Right. Some very clever little shenanigans they must be pulling there.

Steve: Yup, a hack.

Leo: Yeah.

Steve: And Leo, let's take our second break, and then the proof that we're living in a simulated reality.

Leo: Okay, officially the best tease in the history of all time.

Steve: You cannot un-know this, so be careful. This is going to change things.

Leo: I remember I went to see the movie "The Matrix" without any prior knowledge; right? And I go in, I watch it, I come out, my eyes are like this because I think, oh my god, I'm in a simulation. Because that's the whole premise. So I'd have to say my life was never the same since. So I'm ready.

Steve: And once upon a time...

Leo: Are we going to want the blue pill here, or the red pill? It's up to you.

Steve: Once upon a time the fact that Elon agreed with us had some value. Not so much anymore.

Leo: No.

Steve: No one really cares what he thinks.

Leo: No. Get ready. Your red pill moment is coming up. Okay. I'm ready to have my mind blown. I'm taking the red pill, Steve. Go ahead.

Steve: So I stumbled upon a proof that we are, indeed, living in a simulation. My proof that we are in a simulation is that I have identified a clear and incontrovertible bug that exists within the simulation itself.

Leo: A glitch.

Steve: Yes. It's a bug because it defies reality, which is after all what the simulation is intended to simulate.

Leo: Yeah.

Steve: And what makes this so compelling is that all of us are aware of this bug, but we all just shrug it off without it ever occurring to us that that is what it is. So here's the bug.

Leo: I'm ready to have my mind blown. Okay.

Steve: It's a clear failure in the rules of probability. If the rules of probability were being properly simulated, when you attempt to plug in one of those original USB-A style rectangular plugs, the plug's orientation would be wrong only half the time.

Leo: Yeah?

Steve: But we all know that it's not 50/50. Everyone's experience is that the first time we attempt to plug in one of those, it is almost always wrong.

Leo: Always. Always.

Steve: Always. Isn't it? It is. Leo, it is always wrong. It cannot be always wrong. But it is.

Leo: I have the Leo Laporte corollary to that. The toast always falls butter side down.

Steve: Well, see? I think the more we think about this, the more we're going to realize things are not all as they seem.

Leo: You got me.

Steve: I mean, right? I mean, isn't one of those G-D USBs always wrong?

Leo: Always upside down, yup.

Steve: Yes. It should be 50-50. It must be 50-50. And it is not. No one's experience is that it's 50-50. So we've been given a clue that we've not been paying attention to.

Leo: It's a glitch in the matrix. You found it.

Steve: It's a bug. It's a bug in probability. And that must mean that there's a failure in the simulation.

Leo: That's why cats always land on their feet.

Steve: Now, Leo, I have something that may be of interest to you. I have found, I stumbled upon, a valid use for facial recognition.

Leo: Okay.

Steve: It's in the show notes. It's the Smart Pet Door. The other day I encountered an absolutely valid and very clever application for facial recognition which I doubt that anyone would have a problem with, unlike recognizing people. The bad news is that it's called "Petvation," which has just got to be about the worst name that anyone has ever come up with. Petvation. My god. I'm sure there must be a wonderful name for such a product. Hopefully some listener will think of it and tell these poor Kickstarter people.

So what is it? Though you'd never know it from the name, it's a little automated doggie door, or cat door, which employs facial recognition of the household canine or feline.

Leo: Now, see, you're not a pet owner, so I know that you don't know how serious the need is for this. We have a pet door. And we actually had to chip our cats so that the pet door, when they get close, it uses RFID, and it goes chink and opens. Because otherwise raccoons, turkeys, vultures, wolves...

Steve: Apparently ducks.

Leo: Ducks.

Steve: I heard about apparently ducks.

Leo: Anything could use that portal.

Steve: Yup. So, I mean...

Leo: But I would love the face recognition. That'd be great.

Steve: Isn't that great?

Leo: Yeah.

Steve: So it's an automated doggie door or cat door, employs facial recognition. As they approach the door, their identities are verified. And only if they are known to it will it raise the barrier and allow them to pass. It's brilliant.

The text which accompanies this horribly named but otherwise wonderful-looking innovation notes that: "It can also recognize the most common unwanted guests raccoon, wolf/coyote, bear, deer..."

Leo: Bear? How big is your dog door?

Steve: How big is that dog door, yeah.

Leo: "...squirrel, rabbit, duck, boar, skunk, fox, rats, boar, snake, chickens, and more and deny them entry to your home."

Leo: Well, well, now.

Steve: So it's unclear to me, in reading that description, why explicit recognition of unwanted pets is important if it's truly being discriminating about the identity of the family's pet.

Leo: Sure. Well, you don't want to let the neighbor's cat in. I mean...

Steve: Well, what if you did have a pet duck? I mean, that's probably happened.

Leo: That's a good point, yeah.

Steve: So you'd probably have to turn off the duck eliminator. But in any event, perhaps that's just to make sure that nothing unwanted can get in, even if it's like some clever animal wearing a dog mask to spoof the identity of your pooch.

Leo: But really I think dogs - maybe I'm biased, but I think a lot of dogs look alike, and a lot of cats look alike.

Steve: Yeah.

Leo: Can you really do this accurately?

Steve: I don't know. I don't know.

Leo: I'm sending this to Lisa.

Steve: And you're right that chipping animals is a solution.

Leo: That's perfect, yeah.

Steve: Get them a little short-range RFID.

Leo: Exactly.

Steve: I should note that I stumbled upon this over on Kickstarter when I was following up on a recent communication from Ed Cano. That was Ed's 48th communication. Ed is the originator of the SandSara, which you and I are still waiting for, Leo.

Leo: Oh.

Steve: It may finally actually ship. Apparently, it has shipped, and they're working their way through customs at the moment.

Leo: What is it? It's been so long I forgot.

Steve: I know. The SandSara, for those who don't recall, is a nifty-looking tabletop bed of fine-grained sand through which a steel ball bearing is rolled by a magnet located underneath the bed.

Leo: Oh, yeah, I remember this. Yeah, yeah, yeah.

Steve: And I've got the link to it in the show notes. If you want to click on it, you'll see some animated reminders. The magnet is moved by a pair of stepper motors using a quite ingenious mechanism. This podcast discovered this Kickstarter project back in March of 2020, just as COVID was descending upon the world. Ed's original timeline anticipated fulfillment during August that same year, so it appears that it'll finally be happening just shy of two years late.

Leo: Yeah, that's pretty par for the course, unfortunately, with this stuff.

Steve: But I have no complaint with Ed's approach. His communication has been good and consistent; and, boy, is he a man after my own heart. He's proven himself to be an absolute perfectionist.

Leo: He should be. He raised 18 million Mexican dollars.

Steve: Yes. He has 2,000 supporters, 2,000 backers. Though it appears that we'll be receiving this nifty-looking gadget nearly two years late, it really does appear that what we'll be receiving will have been well worth the wait. As I mentioned, his project has just shy of 2,000 backers, and I know that many of them are among our listeners since Ed told me so. When we discovered this, we got quite excited at the time. So our collective wait may finally be nearing an end.

Leo: What is 18 million pesos in American money?

Steve: I don't know. It was a couple hundred bucks, I think.

Leo: Yeah. I think it's, yeah, yeah, it was. It's expensive. He says 19 pesos to the dollar. So, yeah, that's right, they're about a nickel each. So he raised a lot of money. What is 20 million nickels worth to you?

Steve: He didn't run out of money. Apparently he pulled this thing off. I thought of it because Lorrie, just think she will love this. It's just, you know, it's like the perfect - and he's worked on making it quiet because it was like you could hear the stepping motors buzzing. And so no, not anymore.

Leo: Yeah, it's got to be quiet, yeah.

Steve: And like these beautiful round wood, I mean, he just went through like fire in order to get this thing done. But they're shipping. So I just wanted to mention that. And that's how I found Petvation, the worst-named thing ever. The engineering looks good. But, boy, they're not much for marketing.

Leo: By the way, it's almost - it's $874,000. It's a lot of money he raised.

Steve: Yeah.

Leo: So I'm glad he didn't run with it.

Steve: No, he did not. He is a good guy. And if he were to do something else, I wouldn't put it on my calendar, but I'd probably give him more money because he does come through.

Leo: Yeah.

Steve: Okay. We've got some closing-the-loop feedback from our listeners, a bunch of interesting stuff. Robert Wicks, who tweeted from @bertwicks, he said: "You called it, Steve. This just popped up on my ancient i7-4770 system that has no TPM." And he sent me a screenshot that says: "Windows Update. Windows 11, version 22H2 is ready, and it's free. Get the latest version of" - yeah. Pinch your nose. "Get the latest version of Windows with," and it says, and it's a link you can click, "a new look, new features, and enhanced security."

Leo: Wow.

Steve: Because we always say that.

Leo: That's a fourth-generation i7, so it's just the TPM. This is definitely ineligible.

Steve: Yeah. It's got like smoke signals coming off of it. So Peter G. Chase also tweeted from @PchaseG, he said: "Ha ha. My only slightly ineligible-for-WIN-11-upgrade PC

processor is now magically eligible, and Windows is pestering me to do it. Nothing changed on my end. Must be them. You said they might miraculously change the requirements, but I still don't want Windows 11."

Okay. So what happened? Leo, under the category of "You just can't make this stuff up," we have the explanation for what Robert and Peter and many others experienced last week. The Verge covered the story last Thursday under the unbelievable headline - well, if it weren't Microsoft it'd be unbelievable: "Microsoft has accidentally released Windows 11 for unsupported PCs."

Leo: Well, thank you, Microsoft.

Steve: And it runs just fine.

Leo: Thanks so very much.

Steve: Paraphrasing from what The Verge wrote, they said: "Microsoft released the final version of its next big Windows 11 update (22H2) to Release Preview testers on Tuesday, and accidentally made it available to PCs that are not officially supported. Oops." They actually wrote "oops," not me.

Twitter and Reddit users were quick to spot the mistake, with hundreds of Windows Insiders able to upgrade their Windows 10 machines on older CPUs. Microsoft, as we know, has strict minimum hardware requirements for Windows 11, leaving millions of PCs behind, so the mistake will once again highlight the company's controversial upgrade policy. Which is to say that it runs just fine everywhere, even though we don't want to give it to you.

So they said: "Windows 11 officially requires Intel 8th Gen Coffee Lake or Zen 2 CPUs and later, with very few exceptions. While there are easy ways to install Windows 11 on unsupported CPUs, Microsoft doesn't even let its Windows Insiders officially install beta builds of the operating system on unsupported PCs, so this mistaken release is rather unusual. Microsoft is aware of the mistake, and it's investigating." Oh, well that's comforting.

Says Microsoft: "It's a bug, and the right team is investigating it." As opposed to having the wrong team investigate it, which would just add insult to injury, presumably. And that is what Microsoft's official Windows Insider Twitter account tweeted. "So if you managed to install Windows 11 on an unsupported PC and were expecting only Release Preview updates for Windows 10, you should be able to roll back the unexpected upgrade in the settings section of Windows 11," said The Verge. So anyway, got a kick out of that one.

And speaking of large numbers, Jeff Parrish, tweeting from @kb9gxk - sounds like he might be a ham, don't you think? Is that a ham designation, kb9gxk? Or does it have to be k9 something?

Leo: I don't know.

Steve: I don't know. He says you can go to math.tools/calculator/num... - and I didn't get the rest of the link - to convert that number. Remember the number was, I don't even remember now, 2^768 from last week.

Leo: Yes.

Steve: It was three sets of 256-bit encrypted.

Leo: It was big, yes.

Steve: Well, he says, it's large, just the first couple of numbers. And then he lists them. But somebody else sent me the entire thing, of course. We have that kind of listener.

Leo: I am not reading this. Are you going to read this?

Steve: No, no. But I'm going to go up from the bottom.

Leo: Okay.

Steve: Because I stumbled on something that was sort of interesting. So it ends in 38. Then we've got - we have 52,000. Then we have 139 million, 611 billion, 696 trillion, 116 quadrillion. Okay. Then we have 17 quintillion. Yeah, I guess I am reading it. 886 sextillion.

Leo: He's so good, he's reading it backwards, folks.

Steve: In high heels. 256 septillion. 555 octillion.

Leo: This is all Latin. You know, sept, oct, non. Go ahead.

Steve: Yeah. Well, and what happened was later I got to sept, oct, and nov. And it's, wait a minute, those are months: September, October, November.

Leo: That's right, yeah.

Steve: Yeah. So anyway, we got nonillion. Then we've got decillion. Then we have undecillion. Then duodecillion, tredecillion, quattuordecillion - these are not very imaginative at this point - quindecillion, sexdecillion...

Leo: That's why. It's orderly so you can figure it out.

Steve: Septendecillion. Then we have octodecillion and finally - oh, no, novemdecillion.

Leo: Novemdecillion, yeah.

Steve: And finally one vigintillion.

Leo: Vigintillion.

Steve: Vigintillion.

Leo: That's because, yeah, that's 20, yeah.

Steve: Wow. Anyway, thank you, those of you who followed up on the 2^768. We will never forget ye.

Leo: And Gumby I think told us he tried to do it in Emacs, and it quit at quadrillion. What a wimp. So never mind, Emacs.

Steve: So Roy Ben-Yosef tweeting at @WizardLizardRoy is his Twitter handle. That one wasn't free when he went for it. He said: "Hi Steve. Just heard your piece on the New South Wales driver's license. And maybe I'm missing something, but isn't it all client-side? They can do all sorts of fancy crypto and whatnot. Can't I just write my own app that looks the same" - with a dancing flower - "and shows whatever I want? Is there no, or shouldn't be, server-side variation somehow?" He says: "Long-time listener since 2011. You got me into the cyber security business since 2012. Thank you."

Leo: Nice. Wonderful.

Steve: And I'll follow that immediately with Ryan's tweet because he's his company's PKI expert. Ryan tweeted, actually he sent via email, GRC.com/feedback, it was in the mailbag: "Hello Steve. Regarding the New South Wales Digital Driver's License, I work as my company's PKI expert." You know, that's Public Key Infrastructure.

Leo: There's a job called PKI Expert.

Steve: Oh, yeah. And he knows all about WebAuthn, I'm sure. He says: "I liked your solution to the DDL problem." You know, I talked about how you could just solve this with a certificate. He said: "I wanted to let you know that your proposed PKI-based solution can also prevent spoofing of the driver's license QR code. Dancing flowers are all well and good, but could be spoofed if you were motivated enough," as our previous writer noted.

He said: "The way to prevent copying the QR code and passing it off as your own lies in exactly the type of certificate that you issue. If you issued a signed DDL certificate like you suggested, with the Key Usage attribute of digitalSignature," he said, "(and the

Enhanced Key Usage attribute of emailProtection), then your DDL can sign arbitrary messages. This is how encrypted S/MIME email works. In our case, the DDL [Digital Driver's License] could sign your name and age and include the signature in the QR code."

In other words, he came up with a very clever extension. The certificate that is the DDL itself could be used for signing. So that QR code could constantly be changing locally on the screen, each of those changes dynamically refreshed and signed by the certificate which is the DDL because it's a cert. So the point is that's a very clever extension of the idea.

He said: "But couldn't this QR code be copied?" He says: "Well, the next step is to have the person verifying the ID check the timestamp to see exactly when the QR code was signed. You could have the user push a button or enter a PIN in the app." And I had it just doing it automatically because you could do that, too. "This also lets you customize your trust. A police officer's scanning device could trust a QR code for 30 seconds, a bouncer's device could trust a QR code for 1 to 2 minutes, so you can pull up your QR code while you're in line and keep the line moving." He really thought this through.

"The chances of being able to predict the exact 30, 60, or 120-second window that you'll need to produce the QR code means that setting the clock ahead on your friend's phone and pre-generating a QR code are vanishingly small. Live long and prosper. Ryan." So Ryan, very nice piece of work on that.

Leo: Nice, yeah.

Steve: Tom Davies was a little grumbly. He said: "It always grates me a bit when you dunk on GDPR for 'ruining browsing.' Please have a look here." And he sent me to www.coventry.ac.uk. He says: "As you can see, it's perfectly possible to be GDPR-compliant only forcing a single interaction on the user. What has 'ruined browsing'" - he has again in quotes - "is not GDPR, but web hosts' reliance on tracking users to the point where they would rather ruin your browsing experience than allow you to opt out." He says: "I've not dug into it that much, but I'm sure we've all seen examples where it's clear that sites are deliberately making it extremely difficult to do anything other than accept the default 'please track me' option. It really seems that if you don't want to be tracked, these sites do not want you to visit."

So I was curious to see what Tom was referring to. And Leo, you brought it up. So I went over there. What Tom means - because we have a bar at the bottom with three options. What Tom means is that it's possible for a website to request to set a cookie in order to no longer need to notify about setting cookies. Okay. And just for the record, since Tom says this is grating on him, we've previously made it clear that the strict privacy regulations that had been in place long before the GDPR, but that everyone was happily ignoring thus no annoying cookie warning banners until the GDPR was enacted to give those regulations some teeth. I certainly don't disagree with the bias toward tracking that Tom has, but it really was the enacting of the GDPR that's what is making our lives miserable because now it's not just a regulation everyone ignores. It's something that everyone has to pay attention to.

Leo: Well, what really is the problem in my opinion is this notion that all cookies are bad. My site uses one cookie, a cookie to remember your preferences between light and dark mode. But because it uses one cookie, it's not tracking you, it's just you're saving on your computer your setting...

Steve: Right, your preferences.

Leo: That's all it's doing. Because it's doing that, I have to put up a stupid-ass cookie thing. And I think it's wrong. And it's absurd. And it has the exact opposite effect by inuring people to these messages. It's making them forget about - it's just like, well, yeah, click. How many quadrillion vigintillion useless clicks have there been?

Steve: Oh, yes, clicks, clicks.

Leo: Waste of resources because that has to be transmitted. No, it's a perfect example of a misguided law that accomplishes exactly the opposite of its intent. By the way, pat on the back because today Firefox announced what should have been the case all along, that third-party cookies are entirely blocked. They're going to sandbox all cookies unless the originating site is asking for it.

Steve: By default, yes. Nice.

Leo: Yeah, and that's the problem is third-party cookies. First-party cookies are not problematic, I would submit. In any event, certainly my light/dark mode cookie is not problematic.

Steve: No.

Leo: It's incredibly infuriating. And it may be that our mentions of this grates on you, but that ain't nothing compared to how these cookie warnings grate on every other person in the globe because we have to do it because the EU decided to do it. It damaged the EU's credibility beyond belief. It's a terrible idea. It had all sorts of negative effects. It makes the EU a laughingstock. Thank you.

Steve: Yeah. Philip Le Riche, he said regarding last week's podcast, SN-874: "The only essential diff between Passkeys and SQRL is that SQRL uses computed private keys. So could SQRL be made to register its private key with a FIDO2 website and respond to a FIDO2 auth challenge? Then SQRL would work on any FIDO2 site." He said: "(But please finish SpinRite first.)"

Leo: Yes.

Steve: So he anticipated what I talked about at the top of this episode. The use of deterministic private keys may be possible under WebAuthn. If so, then yes, SQRL's single-key approach could transparently replace the big-bucket-of-keys approach that is used by FIDO without sacrificing any security. And as we now more know, there is in fact much more to SQRL than just the difference between one master key that synthesizes Passkeys on the fly versus randomly generated Passkeys, all of which you have to store. As I demonstrated at the top of the podcast, SQRL really goes all the way to solving authentication. What we have with Passkeys and FIDO2 is a very robust replacement of

usernames and passwords. So definite improvement there. And as for SpinRite, everyone should know that I'm done, done with SQRL.

Leo: What?

Steve: Except for talking about it here. I've handed it off and over to a very competent group of developers who know it now just as well as I do. And they hang out over at sqrl.grc.com. So no one need worry for a nanosecond that I will again become distracted by SQRL. That's not going to happen.

And somebody with a clever handle, lyntux, L-Y-N-T-U-X, lyntux, he said: "I wanted to remind you, @SGgrc, that a YubiKey can only hold up to 25 credentials each. That's not a problem, since there aren't that many FIDO2 websites out there. But that could change. That's another reason why SQRL is better."

Yes. And the fact that YubiKeys have a 25-credential limit also beautifully demonstrates that the FIDO project's work was never aimed at mass use and adoption. They were originally focused upon using super-secure hardware dongles for authentication to just a few crucial sites. That system was never able to scale. So when it failed to get off the ground, they relaxed their requirements and produced FIDO2.

I'm going to skip one because we're...

Leo: Yeah, it's my fault we started late. I apologize.

Steve: No, no, no, it's okay. Oh, I guess I pronounce his name Juho Viitasalo. He said: "Hi. I tried the latest SpinRite..."

Leo: I'm pretty sure that's not how you pronounce it, but it's going to have to do.

Steve: Yeah. Sorry about that. I'll just say that his Twitter handle is @juhov.

Leo: Yeah, that's easier, yeah.

Steve: "I tried the latest SpinRite EXE from your dev directory, but could not run it in Windows 7. I was hoping to burn an ISO that way. I have a failing SSD I need to massage. Can you tell how to use these executables you have on your server? Thanks."

Leo: Oh, boy.

Steve: Yeah. And Greg got a bunch of email from our listeners. And so I'm sure that he's referring to the reference we recently made to grc.com/dev/SpinRite. And I asked you, and you did, to pull up the directory listing just to sort of show how much work we did on this first phase of SpinRite. That directory contains a gazillion incremental SpinRite development releases. But they are DOS executables.

Leo: Oh, boy.

Steve: Not Windows EXEs. So they definitely will not run under Windows. But moreover, even if you did run them under DOS, they do not yet move the user past the system inventory stage, where SpinRite is figuring out what you've got and how it can best work with it. That's where all of our collective work thus far has focused. Once I do have something that's operational, all existing SpinRite licensees will be able to obtain their own copies from GRC's server. But we're not quite at that stage yet. So I apologize for the confusion.

Leo: I'm guessing Juho thought he'd found a source of free SpinRite, which this is not.

Steve: Maybe, although he was expecting to burn an ISO, and the normal SpinRite that you run on Windows will do that. So it sounds like he has some understanding of the way it normally works.

Leo: He just wanted to upgrade. Right, right.

Steve: Okay.

Leo: Could you use the old SpinRite for an SSD? Yeah; right?

Steve: Yes.

Leo: Yeah, okay.

Steve: Okay. If anyone listening to this podcast has only been paying half attention, although this has been so great, I don't know how that's possible, now is the time to bring your entire focus to bear on this next question, our final question from a listener. Because it wins the award for the best question maybe ever. And I'm going to leave it hanging and unanswered until next week. So everyone will have the opportunity this next week to ponder the question and its answer. And Leo and everyone in the Discord and chat rooms, please resist the temptation to blurt out any spoilers for everyone else who wants to ponder this very intriguing question. Here it is.

Erik Osterholm tweeted: "Hi Steve. Long-time listener. I'm a little confused on the key length discussion. I always thought that encrypting twice merely doubled the effective strength. Here's my reasoning: Imagine you have an algorithm and a key that you can fully brute force in one day. If you add one bit to the key, you double the key space, and therefore it takes twice as long to brute force (or two days).

"If you instead encrypt twice, then it takes one day to decrypt the outer ciphertext, at which point you get back to the first output of ciphertext. Then in one more day you can brute force the first/inner ciphertext. Like in the first example, this takes two days. It seems to me that these are equivalent. Is there something I'm missing?" So there it is. Everybody think about that. And we'll answer the question at the top of next week's show.

Leo: Hmm. His logic seems sound.

Steve: It seems very sound.

Leo: Hmm.

Steve: So no spoilers, anybody. Don't blurt it out. Just smile if you think you've got it.

Leo: Ain't gonna blurt it out 'cause I don't know it. You're safe with me, Steve.

Steve: So let's take our last break, and we're going to delve into the so-called "PACMAN Attack," which leverages an unpatchable and unfixable flaw which exists across Apple's custom M1 chipset family.

Leo: [Sigh]. Okay, Steve. Let's move on.

Steve: So by far the biggest headline-grabbing news this past week was the teaser that Apple's much-vaunted M1 chips contain a critical bug, or bugs, that cannot be patched or repaired. First of all, only part of that is true. The reason Apple is being singled out is that their M1 ARM architecture chips are the first to use a cool new feature of the ARMv8.3 architecture known as Pointer Authentication. Everyone else plans to use it, too; they just haven't gotten their newest chips out yet.

The heading of the 14-page research paper published by four researchers at the MIT Computer Science & Artificial Intelligence Laboratory reads: "PACMAN Attacking ARM Pointer Authentication with Speculative Execution." So once again we have the spectre - pun intended - of speculative execution raising its ever ugly head, this time in a new context, a new feature of the ARMv8.3 architecture initially represented by Apple's proprietary M1 chip. Okay? So what is it?

PACMAN is a novel hardware attack that can bypass the benefits of an emerging feature known as Pointer Authentication (P-A-C, PAC, so that's where the PAC comes from PACMAN). The authors of this work present the following three contributions, they feel: First, a new way of thinking about compounding threat models in the Spectre age, meaning hardware and software, not only one or the other. Second, reverse engineering details of the M1 memory hierarchy. There's almost nothing available publicly from Apple, so they had to just figure it out the hard way to pull this off in practice. And three, a hardware attack to forge kernel PACs, that is, kernel pointer authentications, from user space on M1.

They wrote: "PACMAN is what you get when you mix a hardware mitigation for software attacks with microarchitectural side channels. We believe the core idea of PACMAN will be applicable to much more than just PAC." So these guys believe that they've uncovered another entire class of exploitable microarchitectural flaws which are introduced by attempts to mitigate software attacks. We'll talk about Pointer Authentication and what it is in detail in a second.

But here's what the researchers wrote for the summary Abstract at the top of their paper. They said: "This paper studies the synergies between memory corruption

vulnerabilities and speculative execution vulnerabilities. We leverage speculative execution attacks to bypass an important memory protection mechanism, ARM Pointer Authentication, a security feature that is used to enforce pointer integrity. We present PACMAN, a novel attack methodology that speculatively leaks PAC verification results via" - and I'll be explaining all this in a second - "via microarchitectural side channels without causing any crashes. Our attack removes the primary barrier to conducting control-flow hijacking attacks on a platform protected using Pointer Authentication."

In other words, pointers are what the system uses to, well, obviously to point to things; but, for example, to control its flow of control. So if you're able to change the pointers and not have that change detected, then you're back where you were being able to, without this protection, obviously, being able to, if you're able to get down into the kernel, to do things.

So they said: "We demonstrate multiple proof-of-concept attacks of PACMAN on the Apple M1 SoC [System on a Chip], the first desktop processor that supports ARM Pointer Authentication. We reverse engineer the TLB" - the Translation Lookaside Buffer, which is the way memory is organized these days - "on the Apple M1 SoC and expand microarchitectural side-channel attacks to Apple processors." Meaning before it had been Intel stuff; right? So now Apple. "Moreover, we show that the PACMAN attack works across privilege levels, meaning that we can attack the operating system kernel as an unprivileged user in user space."

And I loved how they set this up and framed this work in their paper's short introduction, so I want to share it, too, just two paragraphs. They said: "Modern systems are becoming increasingly complex, exposing a large attack surface with vulnerabilities in both software and hardware. In the software layer, memory corruption vulnerabilities such as buffer overflows can be exploited by attackers to alter the behavior or take full control of a victim program. In the hardware layer, microarchitectural side-channel vulnerabilities such as Spectre and Meltdown can be exploited to leak arbitrary data within the victim program's address space.

"Today it is common for security researchers to explore software and hardware vulnerabilities separately, considering the two vulnerabilities in two disjoint threat models. In this paper, we study the synergies between memory corruption vulnerabilities and microarchitectural side-channel vulnerabilities. We show how a hardware attack can be used to assist a software attack to bypass a strong security defense mechanism. Specifically, we demonstrate that by leveraging speculative execution attacks, an attacker can bypass an important software security primitive called ARM Pointer Authentication to conduct a control-flow hijacking attack."

Okay. So what is ARM Pointer Authentication? We've talked, and Leo was just talking about canaries; right? We've talked about related attack mitigation measures in the past. Remember stack cookies? They're a form of canary. The idea with stack cookies is that the system places little bits of unpredictable crypto hashes on the stack as a means for detecting when a buffer overrun has just occurred. So before executing a return from a subroutine, which is when the overwritten attack code would get executed, the bit of code performing the return first checks the stack for the proper cookie, the proper canary, and only executes the return if the value found at the location matches what's expected.

The idea is that attackers don't have nearly sufficient visibility into the system to know what the cookie should be, so their attempt to overrun a buffer on the stack is inherently somewhat brute force. You know, kind of a blunderbuss approach. The cookie allows their overwriting to be proactively detected before the contents of the stack is trusted and acted upon.

What the ARMv8.3 architecture introduces is a clever means of adding a tiny authentication hash - actually it can be too tiny, as it turns out, we'll get there in a second - adding a tiny authentication hash into ARM pointers as a means for detecting any unauthorized changes to that pointer. As we know, the ARM architecture is 64-bits wide. That means that it will have 64-bit pointers. But 64-bits is - I can't even pronounce this number. We now know we could.

Leo: It's a vigintillion.

Steve: It's a virgintillion, yes. It's 18,446,744,073,709,551,616.

Leo: It's a big number.

Steve: That's how many 64 bits gives you. Said another way it's, get this, 18.446 billion gigabytes of pointer space. 18.4456 billion gigabytes of pointer space in 64 bits. Since no one has nearly that much RAM in 2022, we'll see what happens 10 years from now, that means that a large number of the high-order bits of 64-bit ARM memory pointers are always going to be zero. So the clever ARM engineers realized that they could set a boundary point in their pointers. To the right of the boundary is a valid pointer which is able to address as much memory as a system has. And to the left of the boundary, using all of the bits, the high order bits, which are always going to be zero because they can't ever, because the system doesn't have enough memory for them to ever come into use, those can now be cleverly used to validate the bits on the right.

Here's how the MIT guys describe this. They said: "Memory corruption vulnerabilities pose a significant security threat to modern systems." Right. We're talking about that pretty much all the time on this podcast. They said: "These vulnerabilities are caused by software bugs which allow an attacker to corrupt the content of a memory location. The corrupted memory content, containing important data structures such as code and data pointers, can then be used by the attacker to hijack the control flow of the victim program. Well-studied control-flow hijacking techniques include return-oriented programming (ROP)" - which we've discussed at length in the past - "and jump-oriented programming (JOP).

"In 2017, ARM introduced Pointer Authentication (PA for short) into ARMv8.3 as a security feature to protect pointer integrity. Since 2018, Pointer Authentication has been supported in Apple processors, including multiple generations of mobile processors and the recent M1, M1 Pro, and M1 Max chips. Multiple chip manufacturers, including ARM, Qualcomm, and Samsung, have either announced or are expected to ship new processors supporting Pointer Authentication. In a nutshell," they wrote, "Pointer Authentication is currently being used to protect many systems and is projected to be even more widely adopted in the upcoming years. Pointer Authentication makes it significantly more difficult for an attacker to modify protected pointers in memory without detection. Pointer Authentication protects a pointer with a cryptographic hash. This hash verifies that the pointer has not been modified, and is called a Pointer Authentication Code, or PAC for short."

Okay, so this is really clever. It essentially makes 64-bit pointers self-authenticating such that any change to any of the pointer's 64 bits will trigger a protection fault and cause the operating system to immediately terminate the offending program. The Achilles heel of this approach is that there are not a large number of authentication bits available. In the case of macOS v12.2.1, which extravagantly uses 48 of the total 64 bits for valid pointers, only 16 bits remain available for the authentication role. As we know, 16 bits is

a famous number. It's 64K. That's the total number of possible combinations that can be taken by any 16-bit value. Again, here's what the researcher's describe.

They said: "Considering that the actual address space in a 64-bit architecture is usually less than 64 bits" - yeah, less than that 18 billion gigabytes - "for example, 48 bits on macOS 12.2.1 on M1," they said, "Pointer Authentication stores the Pointer Authentication Code (PAC) together with the pointer in these unused bits. Whenever a protected pointer is used, the integrity of the pointer is verified by validating the PAC using the pointer value. Use of a pointer with an incorrect PAC will cause the program to crash.

"With Pointer Authentication in place, an attacker who wants to modify a pointer must correctly guess or infer the matching PAC of the pointer after they've modified it. Depending on the system configuration, the size of the PAC, which ranges from 11 to 31 bits, may be small enough to be brute forced. However, simple brute forcing approaches cannot break Pointer Authentication. The reason is that every time an incorrect PAC is used, the event results in a victim program crash. Restarting a program after a crash results in changed PACs, as the PACs are computed from renewed secret keys every time the program runs. Moreover, frequent crashes can be easily captured by anomaly detection tools."

The breakthrough that these guys achieved was finding a way to successfully brute force probe for the proper authentication code given a modified pointer. By doing that, they're able to make up their own PAC for whatever pointer value they need. So, the final piece of description I'll share from their paper is the PACMAN Attack.

"In this paper, we propose the PACMAN attack, which extends speculative execution attacks to bypass Pointer Authentication by constructing a PAC oracle. Given a pointer in a victim execution context, a PAC oracle could be used to precisely distinguish between a correct PAC and an incorrect one without causing any crashes." In other words, exactly what's needed for brute forcing. They said: "We further show that with such a PAC oracle, the attacker can brute force the correct PAC value while suppressing crashes and construct a control-flow hijacking attack on a Pointer Authentication-enabled victim program or operating system.

"The key insight," they said, "of our PACMAN attack is to use speculative execution to stealthily leak PAC verification results via microarchitectural side channels. Our attack works relying on PACMAN gadgets. A PACMAN gadget consists of two operations: First, a pointer verification operation that speculatively verifies the correctness" - or not, typically - "of a guessed PAC; and, two, a transmission operation that speculatively transmits the verification result via a microarchitectural side channel." So these guys have gotten very clever. But then bad guys are, too.

"The pointer verification operation is performed by an authentication instruction, one of the new instructions introduced in ARMv8.3, which outputs a valid pointer if the verification succeeds, and an invalid pointer otherwise. The transmission operation can be performed by a memory load/store instruction or a branch instruction taking the output of the pointer as an address. If a correct PAC is guessed, the transmission operation will speculatively access a valid pointer, resulting in observable microarchitectural side effects. Otherwise, the transmission step will cause a speculative execution due to accessing an invalid pointer." They said: "Note that we execute both operations on a mis-speculated path. Thus, the two operations will not trigger architecture-visible events" - god, that's clever - "avoiding the use where valid guesses result in crashes."

So they've basically taken what we've learned in the last few years about microarchitectural side effects, the idea that subtle variations in branches taken and not

taken can be sensed across process because they're left in the hardware that everyone shares, and you're not going to cause a crash because they're deliberately arranging that these tests occur in the speculation path which the processor never executes. So the side effects are generated by the speculation portion of the engine which is executing ahead of where the processor is actually going. The processor doesn't take that path, so no mistake, no crash occurs. No exception fault is triggered. Yet it still leaves a trace, a footprint in the microarchitecture which they're able to sense. This is just so cool.

Okay. So now we bottom-line it. What does all this actually mean for those relying upon the security of Apple's M1-based devices? In their own FAQ, the researchers provide some reality-check answers. Does this attack require physical access, they ask themselves? "No. We actually did all our experiments over the network on a machine in another room. PACMAN works just fine remotely, even if you have unprivileged code execution." Should I be worried? "As long as you keep your software up to date, no. PACMAN is an exploitation technique. On its own it cannot compromise your system. While the hardware mechanisms used by PACMAN cannot be patched with software features, memory corruption bugs can be."

Can I tell if someone is using PACMAN against me? They replied: "Much like the Spectre attack our work is based on, PACMAN executes entirely in the speculative regime and leaves no logs. So probably not." They ask, why do you need a software bug to do the PACMAN attack? And they answer: "PACMAN takes an existing software bug (memory read/write) and turns it into a more powerful primitive (pointer authentication bypass). For our demo, we add our own bug to the kernel using our own kernel module. In a real-world attack, you would just find an existing kernel bug. Our demo only uses the kernel extension for adding a software bug. The entire attack runs in user space. So an attacker with an existing bug could do the same attack without the extension that their proof of concept provides." Finally, Is PACMAN being used in the wild? And they replied: "To our knowledge, no."

Now, Apple's response didn't offer much. They officially said: "We want to thank the researchers for their collaboration as this proof of concept advances our understanding of these techniques." Yeah, uh-huh, I bet. "Based on our analysis, as well as the details shared with us by the researchers, we have concluded this issue does not pose an immediate risk to our users and is insufficient to bypass device protections on its own."

Okay. That was carefully worded to be exactly factually correct. And it is. Generally speaking, the common M1 chip user does not have anything to worry about more after this than they did before. But the authors never suggested that this could be used in any form of standalone attack. What this does mean is that successful attacks against current and future ARM chips equipped with Pointer Authentication will need to be more complex in order to also defeat this new extra layer of protection. Because the protection is there, and it's good. But the attacks are going to have to be more complex to defeat it, and they can. As is, Apple's M1 chips based on the ARMv8.3 architecture are thus significantly more difficult to successfully attack than other systems which lack this additional, though imperfect, layer of protection.

And I'll close with this observation: Apple's allocation of 48 bits for their pointers, which affords access to more than a quarter million gigabytes of RAM, is obviously excessive in the extreme. Before this paper was published, Apple's engineers probably thought, what the hell, we have 64 bits of pointer space, so we'll use the lower 48 for pointing and the upper 16 for pointer authentication. Assuming at the time that it was not possible to brute force probe for the correct value of those 16 spare bits, Apple's engineers likely thought that there would only be one chance in 65,536 of an attacker correctly guessing the proper hash for a modified pointer; and a wrong guess, the first wrong guess, any wrong guess, would cause the victim program to be immediately terminated.

But now we know that 16 bits - now we know that 16 bits is not sufficient. So Apple could choose to significantly strengthen their Pointer Authentication by tightening down on the active pointer space, maybe even setting it up dynamically based upon the amount of RAM a system actually contains. A large system, even a large system with 64 gigabytes of RAM requires pointers to consume only 36 bits. 36 bits lets you address 64GB of RAM. That leaves 28 bits for authentication, which is 4,096 times more difficult and time consuming to brute force, with no other overhead or cost to the system. In any event, the sky is once again not falling, and all is well.

Leo: Good. Could it fall in the future?

Steve: Well, when you consider that every time we attempt to plug in a USB plug it fails, it could be that the attacker will guess right the first time.

Leo: Yes, it could be.

Steve: Even though they've got one chance in 65,536. I mean, I've never plugged in a USB plug correctly the first time. It just it never happens.

Leo: Never happens.

Steve: Which demonstrates, like that black cat, remember, that kind of flickered. Two cats walked by the matrix, and that represented a glitch in the matrix?

Leo: Yeah.

Steve: We've got a glitch here. You know, the best thing about USB Type C is that this may actually be an attempt to cover up this bug.

Leo: Hiding the glitch. I think you're right.

Steve: Now it doesn't matter. You will get it right the first time, every time. And with time those old Type A connectors, they'll fade away, and this bug in reality that made it very possible, I mean, somewhere someone is thinking, boy, those humans are dumb. Every time they try to plug in those USB things...

Leo: They never notice.

Steve: No. We can't change the code in the simulation now. It's running. We cannot patch it on the fly. So we just have to put up with this bug in USB Type A connectors. But now those are going to fade into the past. We're going to get Type C, and nobody will even know.

Leo: Ah. See? Brilliant. Brilliant. That matrix, it knows what it's doing. Well, thanks for joining us on the red pill edition of Security Now!. Mr. Steve Gibson lives at @SGgrc on the Twitter if you want to leave him a DM. He also has a little website he calls GRC.com, the Gibson Research Corporation. What he's researching right now, SpinRite 6.1. SpinRite 6 is current, world's best mass storage maintenance and recovery utility. 6.1 is imminent. And if you buy 6.0 now you'll get 6.1 when it comes out. You can also participate in the development of 6.1. You can also find this show there. He has a couple of unique versions, a 16Kb audio version. Why don't you do 8-bit? Why not just go all the way, do 4-bit audio?

Steve: We would call that the Minecraft edition.

Leo: Yeah, 8-bit audio. 16-bit audio, which is smaller, obviously, than the normal 64Kb audio. He also has transcripts which are actually very handy for searching and reading. And of course the 64Kb audio, the standard version. That's all at GRC.com. While you're there, check out all the wonderful goodies Steve makes available to you for free. And of course his bread and butter, SpinRite.

We have audio and video on our website, TWiT.tv/sn. There's a YouTube channel with a video dedicated to Security Now!. Just search YouTube for Security Now!. You'll also be able to subscribe because it's a podcast. As soon as we're done editing this thing, cleaning it all up, polishing it up, we'll put it out on the Internets, and you can get it just by subscribing in your favorite podcast player. If your podcast player has ratings, leave us a five-star, would you? Let the world know. Everybody needs to know.

Steve: I mean, this was clearly a five-star episode.

Leo: I don't know how you can get any better than this one. But we'll find out next week, that's for sure. We do this show every Tuesday. We try to do it around 1:30 Pacific, if Leo isn't a laggard, a slug. 1:30 to 2:00, something like that, Pacific. 1:30 is 4:30 Eastern, 20:30 UTC. The livestreams are at live.twit.tv. There are chatrooms. The IRC chatroom is free and open to all at irc.twit.tv. I will be back here next week. I know you will, too, Steve, for Episode 876. More revelations.

Steve: And the answer to the big question, why is double encrypting not only twice as strong, but exponentially as strong?

Leo: Takes one day to do the outer shell, one day to do the inner shell. It should take the same amount...

Steve: Seems right to me.

Leo: Should be one plus one; right?

Steve: Seems right to me.

Leo: What do the boffins say? One times one?

Steve: No, it would be 2^256 times stronger.

Leo: A lot better.

Steve: Way better.

Leo: Why is that?

Steve: Be like summing the keys, like having a 512-bit key instead of a 257-bit key. I know.

Leo: That's a head scratcher.

Steve: I know.

Leo: Get to work on that. Do you want people to DM with you their thoughts?

Steve: They can certainly DM me.