NAT Slipstreaming 2.0

Description: This week we examine another instance of a misbehaving certificate authority losing Chrome's trust. We cover a number of serious new vulnerabilities including an urgent update need for the just-released Gnu Privacy Guard; another supply chain attack against end users; a disastrous 10-year-old flaw in Linux's SUDO command; and, thanks to Google, some details of Apple's quietly redesigned sandboxing of iMessage in iOS 14. I'm going to share something that I think our listeners will find quite interesting about some recent architectural decisions for SpinRite, and then we'll conclude with a look at the inevitable improvement in NAT bypassing Slipstreaming.

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-804.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-804-lq.mp3

SHOW TEASE: It's time for Security Now!. Steve Gibson is here. We've got lots to talk about. Flaws in GPG you're going to want to fix right away. Same thing with SUDO, an exploit that's been around for - Steve holds up his 10 fingers. I counted 10 years. Wow. We'll also talk about BlastDoor, the new protection Apple snuck into Apple Messages. Google found out about it, though, and told the world. It's all coming up next. Steve will tell you on Security Now!.

Leo Laporte: This is Security Now! with Steve Gibson, Episode 804, recorded Tuesday, February 2nd, 2021: NAT Slipstreaming 2.0.

It's time for Security Now!, the show where we cover the security news of the world and help keep you safe online with this guy right here, Mr. Steve Gibson of the Gibson Research Corporation. Hello, Steve.

Steve Gibson: Leo. Great to be with you again for our 804th episode for 2-2-21, Groundhog Day. And maybe it's fitting that it's Groundhog Day because the title of today's podcast is NAT Slipstreaming 2.0.

Leo: Redux. Okay.

Steve: We've talked about that. We introduced the concept of NAT Slipstreaming in November, shortly after Samy Kamkar had figured out a way to, for a remote server, whether it was serving JavaScript or a malicious ad, anything that could run JavaScript in your browser could trick your NAT router into opening up a reverse mapping through its firewall on other ports, which had various deleterious effects.

Well, that was then. The browser makers immediately moved to keep that from happening. But I quote Schneier again lower in the podcast, reminding us that attacks never get weaker, they only ever get stronger. We're back now with 2.0, a far more powerful and worrisome attack. But we're going to wrap up with that. We're first going to examine another instance of a misbehaving certificate authority who has finally - and after five years of misbehavior, I would say it's time to lose Chrome's trust.

Leo: Oh, boy.

Steve: We're going to cover - yeah. That's never good. It's like, sell that stock short.

Leo: Oh, boy. Actually, Google's stock just went through the roof this morning, by the way.

Steve: Yeah, I saw, it did like a straight line.

Leo: Yeah, yeah. They had very good earnings, yeah.

Steve: We're going to cover a number of serious new vulnerabilities, including an urgent update need for the just-released Gnu Privacy Guard. GPG has a newly induced bad problem. We've got another supply chain attack, this time not aimed at corporations, but more at gaming end users. We have a disastrous 10-year-old flaw in Linux's SUDO command, believe it or not.

Leo: Yes. I got an update to SUDO yesterday, and I thought, hmm, yeah.

Steve: Good. Unfortunately, think of all the Linux systems out there that won't.

Leo: Oh, yeah, yeah.

Steve: And thanks to Google we have some details of Apple's quietly redesigned sandboxing of iMessage which they implemented in iOS 14. I'm also going to share something that I think our listeners will find quite interesting about some recent architectural decisions that I've just made in the last, well, actually it was something that was an aha moment yesterday about SpinRite. And then we're going to conclude with a look at the inevitable improvement in NAT bypassing slipstreaming.

Leo: Ooh.

Steve: So I think another interesting podcast for our listeners.

Leo: Ooh, as always. Okay, Steve. Picture of the Week?

Steve: Well, I think this is a pretty simple one. It was in my queue of...

Leo: It actually made me laugh.

Steve: Yeah. So we've got two guys sitting side by side, each at their own terminal. And the guy on the left says to the guy on the right, "So why are you going onto the Dark Web?" And we see the screen of the guy on the right that says "Dark Web," and it's a black screen that's got a skull.

Leo: Skeleton skull.

Steve: Right. And then below it says "Welcome! And have a great day!" Anyway, the answer to the first person's question is, "I forgot my password, so I need to go look it up."

Leo: Yeah, they know. That's a variant on the, oh, you lost your data? Don't worry, the NSA has a copy.

Steve: Exactly. Exactly. So okay. As we know, our certificate-driven web server system of trust is based upon a chain of trust, actually many chains of trust, where each chain is anchored by a signing authority's root certificate. That root cert contains their public key, which matches their secret key used to sign the certificates which are presented by web servers when people browse to them. So with that model, any web server presenting an unexpired identity declaration certificate that has been signed by any of the browser's trusted certificate authorities will be trusted. And of course the signature is verified because it can be verified against the public key which the browser has in its root store.

As we know also, those of us who have been listening to this podcast for a while, several times in the past few years we've covered the interesting and often fraught news of signing authorities either deliberately abusing or inadvertently failing to properly authenticate the identities of those whose certificates they sign. The inevitable result is that they lose the privilege of having their signatures trusted by the industry's web browsers, which effectively renders their signatures useless and therefore worthless.

And we're here again today. Google has just announced that they intend to ban and remove support from Chrome for certificates issued by the Spanish Certificate Authority Camerfirma, C-A-M-E-R-F-I-R-M-A, Camerfirma. That revocation of trust will go into effect once Chrome 90 hits the release channel in mid-April, so about 2.5 months from now. And at that point Chrome will no longer trust any Camerfirma signatures. So this of course means that none of the otherwise valid TLS certificates that have previously been signed by Camerfirma will be seen as valid by Chrome. So all of the web servers currently serving certificates signed by Camerfirma will be invalid for all Chrome users.

And as we know, once upon a time this wasn't taken so seriously, back at the beginning of the podcast. If that happened, you'd go, okay, fine, yeah, you know, I'll go anyway. But no. These days we're taking this all much more seriously. And I don't even know if you're able to force Chrome past an invalid signature on a cert. I should probably know that. But anyway, so since untrusting any certificate authority's root cert instantly ends that aspect of the CA's business, and also punishes their previous customers whose signed certificates no longer function, the decision to do this is always made only after

the faulting party has been warned many times, and only after it's become clear that, for whatever reason, their conduct is placing the greater Internet at risk.

So in this instance the final decision to drop trust for Camerfirma's certificates comes only after the company was given more than six weeks to finally explain a string of 26 incidents. And frankly, I'm going to share three of those. And those are - I would call them "compound failures" because, I mean, calling them 26 is really condensing the number. The incidents detailed by Mozilla, sort of stored there, date back to March of 2017. I've got a link in the show notes for anyone who's interested in more detail than I'm going to provide, but you'll be convinced here in a minute.

The two most recent problems of these 26, and again counting very generously, occurred just last month, in January, even while and subsequent to Camerfirma being notified that it was under probationary investigation the month before, in December of 2020. The incidents paint a clear and disturbing picture of a company that has failed, and not just once, but seems determined to fail to meet industry-wide agreement on the quality of their product and the security standards they must hold in return for the privilege of issuing TLS certificates for website operators, software makers, and enterprise system administrators.

As we know, and we've talked about - because it's sort of a fascinating aspect of the whole public key infrastructure. In a sense, traditional certificate authorities are printing money; right? You pay them hundreds of dollars, and they make sure that you're who you say you are. And then they press a button which spits out a signature which essentially, I mean, they are bearing some cost to run this bureaucracy and to verify your identity. But you're paying them for some bits which, you know, they're very special bits. It takes them to make those bits.

So, but that's the point. It's in return for printing money, their conduct has to meet standards. And looking over the list of Camerfirma's 26 transgressions, it appears very clear that they have failed over many years to deserve what was an initial presumption of trust that they were given by the industry's web browsers, assuming that this was a serious business that was going to take its responsibilities to heart. But that hasn't happened. And so that trust that was originally given is being rescinded.

Okay. So here's just three that made sense to share. Issue R - and I think they start at A. Issue R was titled "Failure to disclose unconstrained sub-CA." And this was named "DigitalSign." Now, okay. So remember that it's possible for a certificate authority not only to sign the end certificate for a web server, for example, but to sign a sub-certificate authority, that is, a subsidiary certificate. And "unconstrained" means that that sub-CA certificate is not constrained against doing its own signing. So essentially they were giving their trust to a third party and allowing them to do anything they wanted to, to sign any certificates. And so their trust would flow down to that sub-CA.

And there are situations where like a business entity, a subsidiary - we'll talk about one in a second, another one of theirs, MULTICERT. But this one, DigitalSign, you can do it, but part of the requirements are that you acknowledge that. You tell the industry that this is what you've done because obviously this is a very powerful thing. You're basically giving away the right to sign things. So in April of 2018 Camerfirma failed to disclose an unconstrained sub-CA, despite Mozilla requiring in November of 2017 that such disclosures be complete by April 15th of 2018. No explanation was provided by Camerfirma as to the cause of this omission, nor were effective controls provided that would prevent such Mozilla policy violations in the future. So that's one of the three.

So here's a big compound one. That was Issue R. Here's T: "Failure to disclose unconstrained sub-CA," and this one is MULTICERT. And this has date ranges from 2018 through 2020. And the issue reads: "In the course of resolving Issue R" - that's the one I

just mentioned - "it was further discovered in July of 2018 that Camerfirma had failed to disclose two additional sub-CA certificates" - they're just doing this whenever they want to - "operated by MULTICERT. Mozilla policy required that such sub-CAs be disclosed within one week of creation. Camerfirma's explanation at the time was that they failed to consider that the person responsible for disclosing into the CCADB" - the organization to which you need to disclose - "would be unavailable, and that the backup for that person would also be unavailable. They resolved this by adding a backup to the backup, in case the backup fails.

"However, the disclosure in the CCADB turned out to be incorrect and misleading, as Camerfirma disclosed that they operated the sub-CA when in fact it was externally operated. At the time, this was only detected because Microsoft had disclosed the CP/CPS they had for MULTICERT's root, which participated in Microsoft's root program. Camerfirma's explanation was that the person responsible for disclosing was overloaded, and that three people would be responsible for disclosures going forward." And this is all still T; right? This is one. This is considered one problem.

"In 2019, Camerfirma failed to provide correct audits for MULTICERT. Their explanation was that they only had one person responsible for" - I think this is in someone's garage in Spain - "one person responsible for communicating with their sub-CAs, and had failed to consider that the person responsible for communicating with sub-CAs and disclosing into CCADB would be unavailable. They stated that they intended to prevent such issues from recurring in the future by purporting to implement additional steps."

And finally, still Issue T: "In 2020" - that is to say, last year - "Camerfirma again failed to properly disclose sub-CAs operated by MULTICERT." So their previous assertions that they were going to fix these previous problems weren't fixed. "They erroneously reported them as covered by Camerfirma's CP/CPS. Their stated reason was because these new sub-CAs were not covered by the new audit from MULTICERT yet, although the expectations for how to disclose had been previously communicated by Kathleen Wilson." So again, they were told. They didn't do it.

And finally, Issue X, which was again about this MULTICERT sub-CA. There's mis-issuance from 2018 through 2019. "In August of 2018, within weeks of having received a cross-signed sub-CA from Camerfirma" - that is relating to Issue T - "MULTICERT issued several certificates that violated the ASN.1 constraint for organizationName field length. Camerfirma's response was that both they and MULTICERT would now regularly check crt.sh for certificate lint violations."

Then, later: "In October of 2018, it was discovered that MULTICERT had mis-issued 174 certificates with an incorrect qcStatements extension." So they're just not even - they're not doing this right. "In response to the report that was provided, MULTICERT revoked the certificates and then mis-issued new certificates with a validity period greater than 825 days" - which is illegal - "to replace those the following month. Further, after reportedly fixing the underlying issue, MULTICERT again mis-issued another certificate having a validity period greater than 825 days." Which, you know, their system shouldn't do. What, do they have someone typing it in manually? Oh, I'm sorry, we used the previous year calendar by mistake. I mean, really.

But anyway: "During the course of this investigation, MULTICERT also failed to revoke on the timeline required by the BRs" - the baseline requirements - "in order to give the customer more time to replace certificates." That's right. "In March of 2019 it was discovered that MULTICERT's certificates also had insufficient entropy, containing only 63 bits of entropy, rather than the required 64."

So again, those are just three - they call them three, that's more like 13 - of the 26 issues that have been compiled since 2017. And, okay, maybe having 63 bits of entropy

rather than 64, you know, you have a stuck bit, is not a big deal. But in the CA business, which as I said can be, if you behave yourself, a money-printing goldmine, it is about the details and about rule following. And every rule has been established for some good reason over time. They didn't just make up a bunch of rules because they were sitting around saying let's make this harder to be a CA. They're all there for a reason.

And if a CA is sloppy about something as simple as the amount of entropy in their certs, and of course that's an important thing to have, then it begs the question, what else are they doing wrong? And if their system can somehow issue certificates with illegal validity periods greater than is allowed, again, what else are they doing wrong?

So through the years we've discussed the walk of shame made by the StartCom subsidiary of WoSign - oh, and by the way, StartCom ended up doing some business with this Camerfirma group. I didn't share those mistakes, but they're there. And remember Diginotar, of course, and even Symantec. In each case, trust in those companies' certs was revoked. Diginotar filed for bankruptcy. And as we know, Symantec, whose certificate-signing name was essentially ruined by their own misconduct, sold their certificate authority business to DigiCert.

So at this point the other browser makers have been silent on Camerfirma. And I just think that's because Google is a first mover here. Chrome has clearly made the right decision, so I'd expect to see similar announcements being made by Apple, Microsoft, and Mozilla before long.

Leo: Oh, yeah. This list is from Mozilla, so clearly they're well aware of it. It doesn't seem that malicious. It seems sloppy; right? I mean...

Steve: Yeah. Agreed. Agreed. Although...

Leo: But you make an excellent point. Here's your opportunity to make a lot of money. Why be sloppy?

Steve: Oh, Leo. Could I have that business, please?

Leo: I know, I know. We just paid - I use DigiCert, among others, for our certs. And I just got another one. It's expensive. I spent more than a thousand bucks for the cert. It was a wildcard cert. And they were good. They called, you know, the business line. I had to figure out a way to answer the business phone, which was - that was an interesting...

Steve: Yeah, in fact, Sue does that for me. And so we have a standing deal. I say, okay, Sue, now DigiCert's going to be calling sometime this morning. Are you going to be around?

Leo: Make an appointment, yeah.

Steve: Because you need to answer and say, hi, I'm a breathing human.

Leo: Right, right. No, I was really glad that they went the extra mile, you know. It was inconvenient, but they did it. And this was a renewal. This wasn't even, like, they already had verified me. So yeah, you kind of want these guys to do it right. I really think this.

Steve: Well, and of course, as I've said before, there is now, of course, certs are also available for free; right?

Leo: Let's Encrypt.

Steve: The ACME Protocol with Let's Encrypt. It'll do that for you. But Let's Encrypt is also where all the malicious certs are.

Leo: Right.

Steve: That is, all of the soundalike, lookalike, Unicode hack domain names, you go to any of these things, they have, whoa, yeah, look, I've got TLS. Yes. Let's Encrypt is my certificate. So I would not be surprised if at some point there isn't an indication that is surfaced on our browsers about whether there was any validation of this assertion made by a human in the loop, or was it just automation? To be fair to Let's Encrypt, their goal was to encrypt the Internet. And they have done that.

Unfortunately, we rely on certificates for two things: not only encryption, but authentication. And that's the problem with Let's Encrypt is that, because it's a bot system, it's an automated certificate issuance, I mean, this could have been done 20 years ago if we wanted to. It's not hard. But that just means that there's no assertion of the identity behind the company. And that's why I'm happy to have my relationship with DigiCert is that at some point I wouldn't be surprised if there isn't something that says, okay, yeah, you're encrypted, but we're not sure who you're talking to.

Leo: It's also the case that there is no standard price; right? The prices vary.

Steve: True. And for what it's worth, the way DigiCert works, especially now that certs have had their maximum issuance lifetime renews...

Leo: Yeah, I got my new one-year cert; right.

Steve: Right.

Leo: But I bought two years.

Steve: Well, and that validation probably lasts you two years. That is to say they decoupled the validation of your identity from the issuance of certs so that - and this is what's been so handy for me. I'll, like, in the middle of the night on a holiday weekend I'll say, oh, you know, I really want forums.grc.com. And I go over to DigiCert and make myself a cert, which...

Leo: Because GRC.com has already been validated.

Steve: Correct.

Leo: Yeah, yeah.

Steve: Correct. And so I'm able to do that. But I could even reissue a GRC.com cert if I wanted because my point is every so often, separate from issuing, they perform the "Are you still you" dance. And that bumps their calendar forward. And then within that window you can do anything you want to.

Leo: That's interesting. I didn't know that, yeah.

Steve: And I mean, like, night or day, yeah.

Leo: That's in response to Let's Encrypt because they really do want to be a little bit easier; right.

Steve: They have to be, yeah, exactly. They have to make that tradeoff. Okay. So after Camerfirma is gone - and presumably, like I said, if I had stock I'd sell it - there will still be plenty of certificate authorities for browsers and consumers both to trust. So it's not like we don't have enough of them. Everyone will probably well remember the day I clicked on the list of certs in my root store, just about fell over when it was more than 400 because I remembered when there were five of them.

And so the best thing that probably comes from this, from this sort of banishing, is the sobering reminder to all the other CAs, even non-top tier, I mean, DigiCert is like top tier. I'm always - sometimes I'll look at some good website cert, wondering, and there it is, "Signed by DigiCert." And I think, yup. So Camerfirma, okay. But these non-top tier guys, I mean, here Camerfirma's like, well, the person who we needed to do this had a backup, but I guess the backup needs a backup because, you know, it's like, what?

Okay. So the point is, behave yourselves. If you've got a growing list of, like, mistakes over in Mozilla's collection pile, you ought to start thinking about taking it seriously because at some point the boom is going to be lowered on you, and you're not going to be able to print money anymore. Turns out it's not a right, it's a privilege.

Okay. So hopefully people who are using GPG are on some list somewhere, and this will be old news. But an urgent update is needed of the recently released Gnu Privacy Guard. Version 1.9.0 was recently released, just a few weeks ago, on the 19th of January. Libgcrypt is an open source crypto library as part of GPG, one of GPG's modules. Last Thursday our old friend Tavis Ormandy, whom we haven't heard from recently, of course of Google's Project Zero, publicly disclosed the existence of a heap buffer overflow in libgcrypt which was due to an incorrect assumption in the block buffer management code.

Tavis wrote - and I just hit the spacebar, and I went way down. Tavis wrote something. I'm sure he did.

Leo: I can read it. Here, I'll be Tavis Ormandy.

Steve: Very good. Actually, I found it. But you want to read it?

Leo: No. I was just going to give him some sort of nerdy voice, which is probably not the best thing to do.

Steve: Oh, thank you, Leo. He said: "Just decrypting some data can overflow a heap buffer with attacker-controlled data. No verification or signature is validated before the vulnerability occurs." He said: "I believe this is easily exploitable."

Leo: He said: "I believe this is easily exploitable." How about that?

Steve: I'm sure that's what he said. Sounded like Daffy Duck, maybe.

Leo: I am nervous now because I do use GPG. So I'm going to have to check and make sure I'm on the...

Steve: Make sure you're at 1.9.1.

Leo: 1.9.1, okay, yeah.

Steve: So Tavis forwarded his findings to the developers responsible for libgcrypt. And as soon as the report was received, the team published an immediate notice to users: "[Announce] [urgent] Stop using Libgcrypt 1.9.0!"

Leo: Oh. Oh, boy.

Steve: Yeah. "In the advisory, principal GnuPG developer Werner Koch asked users to stop using v1.9.0 as a new release had begun to be adopted by projects including Fedora 34 and Gentoo. A new version of libgcrypt, v1.9.1, was released in a matter of hours" - because it wasn't a hard problem, it was just a bad problem - "to address the severe vulnerability which was even too new to have had a CVE number assigned." That's the way you want your security folks to operate. Like, it's already done by the time the CVE gets issued.

In an analysis of the vulnerability, cryptographer Filippo Valsorda, whom we've referenced in the past, suggested that the bug was caused by memory safety issues in C - oh, imagine that, has anyone ever heard of that? - and may be related to efforts to defend against timing side-channel attacks. So what we have is an instance of an attempt to mitigate one potential threat, creating a new, very real vulnerability where none existed before. So as a consequence, users who had upgraded to libgcrypt 1.9 are being urged to download the patched version as quickly as possible. In other words, a trivial problem to exploit. The GPG developers agreed with Tavis's assessment of the bug's severity. They said: "Exploiting this bug is simple, and thus immediate action for 1.9.0 users is required."

Leo: You'd need physical access to the machine, though; right?

Steve: No, no, no.

Leo: Oh, you wouldn't.

Steve: This is a send-somebody-email exploit.

Leo: Oh, god. Okay.

Steve: Yeah, yeah, yeah.

Leo: I have 1.9.0 on my Mac here, so I'm upgrading it right now.

Steve: Yeah, good. So thank goodness, or thank Google, for Tavis.

Leo: Yes.

Steve: But you have to wonder what would have happened if Tavis had not been on the ball? How long would this newly introduced serious flaw have lingered within the GPG code? It's clear that when a skilled developer examined the code, Tavis - without the inherent bias of the code's author, and as we've often said, it's very difficult to see one's own mistakes - the problem was apparent. Tavis just saw it. So that suggests that malicious coders, whom we know are unfortunately every bit as talented as the world's best, might also be scouring the world's open source, looking for exactly such flaws.

So to me this suggests two things. First, we really need to appreciate how brittle our current coding and code is, and try much harder to just leave things alone that are already tried, tested, and true. Stop messing with code that works. Stop trying to make things better that are just fine the way they are. Really. The second thing is that we really need to reconsider our coding practices. We need implementation languages that inherently do not allow these sorts of mistakes to be made in the first place. Such languages exist. But they're not macho and fun to use. Consequently, a highly critical security-oriented cryptographic library for the widely used GPG is written in C, the lowest level, most macho language, with the worst security track record.

Leo: Okay. I might argue that assembly is even lower level macho.

Steve: Actually, I'll be getting to there in a second.

Leo: Okay.

Steve: That could be owing to the fact that so much code is still being written in C, but it's also owing to the fact that the language provides exactly zero protection from doing anything the programmer wants.

Leo: By design.

Steve: Yes. Which is why it's appealing to cowboys.

Leo: Right, right. Everybody loves C.

Steve: Yes. And yes, I know. I wrote SQRL in assembler. So who am I to speak about the need to use safe, high-level languages when security matters.

Leo: Do you have static type checking in assembler? I don't think so.

Steve: I'm just saying, Leo, everybody else should do it.

Leo: Yes.

Steve: Everybody else.

Leo: None of you are good enough to be using assembler.

Steve: And that's of course the problem; right?

Leo: Right.

Steve: Everybody thinks that everybody else should do it.

Leo: I'm good enough, yeah. I can do it.

Steve: With a foreseeable outcome that still today no one does.

Leo: Yeah. Everything should be rewritten in Rust except for SpinRite. I'm just saying.

Steve: So good luck and god bless and be sure to update your GPG code.

Leo: I just did, 1.9.1.

Steve: Nice.

Leo: Thank you, MacPorts. A good example is SUDO. You were talking about how long this could have gone undetected? SUDO was nine years, I think; right?

Steve: Ten.

Leo: Ten.

Steve: It was 2011.

Leo: Yeah. Undetected. And it allowed a local user to escalate to root. That's not good.

Steve: Oh, yes. We're getting there. We're getting there.

Leo: Oh, sorry, I didn't want to preempt you.

Steve: No. So we have another interesting supply chain attack and a little bit of interesting takeaway, I think. So the company BigNox, B-I-G capital N-O-X, is a Hong Kong-based company which publishes, among other products, an Android emulator for PCs and Macs called NoxPlayer. The website claims that they have over - that is, the BigNox website - that they have over 150 million users spread through more than 150 countries and speaking 20 different languages. Though the BigNox follower base is predominantly Asian, based on what ESET found - I'll get there in a second - the NoxPlayer is generally used by gamers to play mobile games on their PCs.

So the discoverers of something fishy going on were our friends and recent TWiT network sponsor ESET. They spotted a highly targeted surveillance campaign involving the distribution of three different malware families via tailored - and here it comes - malicious updates to selected victims in Taiwan, Hong Kong, and Sri Lanka. ESET spotted the first signs of something going on last September, and the compromise continued until "explicitly malicious activity," as they put it, was uncovered just last week, which prompted ESET to report the incident to BigNox. ESET wrote: "Based on the compromised software in question and the delivered malware exhibiting surveillance capabilities, we believe that this may indicate the intent of intelligence collection on targets involved in the gaming community."

So to carry out the attack, the NoxPlayer update mechanism served as the vector to deliver trojanized versions of the software to users which, upon installation, delivered three different malicious payloads, including, for example, the Gh0st RAT, you know, "RAT" as in Remote Access Trojan, which is used to spy on its victims, capture keystrokes, and gather sensitive information. Separately, they found instances where additional malware like the Poison Ivy RAT was downloaded by the BigNox updater from remote servers controlled by the threat actor.

So ESET wrote: "Poison Ivy RAT was only spotted in activity subsequent to the initial malicious updates and downloaded from attacker-controlled infrastructure." Now, unfortunately, BigNox in Hong Kong was not very helpful when they were contacted by

ESET, who wrote of this. ESET said: "We have contacted BigNox about the intrusion, and they denied being affected. We have also offered our support to help them past the disclosure in case they decide to conduct an internal investigation." In other words, ESET was, you know, this is now a page. I've got the link in the show notes here to their full disclosure. So this was going to put BigNox in the spotlight.

So anyway, for one thing, if anyone hearing this is a user of BigNox's NoxPlayer, you probably need to take responsibility yourself, because it doesn't sound like BigNox is prone to, for making sure that you didn't receive any malware. This link in the show notes contains some IOCs, Indicators of Compromise, which anyone who's worried can check for on their own system. The intrusions appear to be gaming world-centric and highly targeted. So it doesn't look like it's a big widespread campaign.

ESET said: "In comparison to the overall number of active NoxPlayer users, there is a very small number of victims. According to ESET's telemetry, more than 100,000 of our users" - meaning ESET's users - "have NoxPlayer installed on their machines. Among them, only five users received a malicious update..."

Leo: Wow. That's surprising.

Steve: Yeah, "...showing that Operation NightScout," as they termed it, "is a highly targeted operation."

Leo: Ah.

Steve: Yes. Yeah. And again, as we know, the more people you infect, the greater the opportunity for discovery. And but it was probably the infection of those five users that overlapped with ESET's being in their system watching what was going on that put ESET onto this behavior in the first place. So anyway, those victims, as I mentioned, are in Taiwan, Hong Kong, and Sri Lanka. They said: "We were unsuccessful finding correlations that would suggest any relationships among victims." On the other hand, they've got, you know, all they're seeing is those five that happened to be using ESET. Who knows overall because BigNox is saying that they've got 150 million users, not 100,000. So it may also be that ESET's overlap with targeted victims wasn't very big.

They said: "We were unsuccessful finding correlations that would suggest any relationships among victims. However, based on the compromised software in question and the delivery malware exhibiting surveillance capabilities, we believe this may indicate the intent of collecting intelligence on targets somehow involved in the gaming community." And then in their posting details and diagrams, the precise operation of the NoxPlayer update system is explained. And after that they conclude: "We have sufficient evidence to state that the BigNox infrastructure" - and then we have some domain names, res06.bignox.com - "was compromised to host malware, and also to suggest that their HTTP API infrastructure at api.bignox.com could have been compromised. In some cases, additional payloads were downloaded by the BigNox updater from attacker-controlled servers."

So the malware didn't always come from BigNox itself. Sometimes somehow there was a URL change, and in fact that's what they said: "This suggests that the URL field provided in the reply from the BigNox API was tampered with by attackers." And, you know, they said HTTP API. I didn't look closely enough to see whether it was actually HTTPS. Of course, if they have an unencrypted API, then anyone anywhere in line could intercept and tweak the URL in the reply field.

So what we have is a much smaller scale version of the now-infamous SolarWinds intrusion which one way or another leveraged the software updating channel to quietly slide malware into a victim's machine. And this sort of put me in mind of things you and I used to say, Leo, which we've sort of revised ourselves since. But once upon a time I believed that I could successfully take responsibility for not getting my own system infected. You know, we've talked about this through the years. Don't download anything that a website tells you you need. That's never going to end well. Don't click on links in sketchy email. Be very wary of anything you download from a third-party download repository. Whenever possible, go to the source. Get the thing you want directly from the publisher and so on.

But today we all need to face the fact that we no longer have that control over our own machine's destinies. Many of the tools, utilities, and widgets that we use are being periodically updated. For me, Notepad++ comes immediately to mind because it is constantly wanting to update itself. Not only is that annoying, since it seems to be working just fine for me, but it's also inherently dangerous. If their build process or their update server were to get compromised - and we've covered stories in the past where that has happened to well-meaning organizations - in a very short time, and remember it even happened to Lenovo at one point, in a very short time a huge number of Windows users would be infected because their instances of Notepad++ said, oh, I've got a better, shinier version. And everyone would say, oh, got to have that. Maybe it's better somehow.

The point is that the fact that they have this leverage over an install base of Notepad++ users puts them in the crosshairs of the bad guys. It makes them a high-value target. Bad guys would love to get their malware just automatically downloaded into all of the Notepad++ users in the world. So for me, the solution is Windows Defender, which I now no longer recommend only to others. I look at it, and it's the green little happy house with the flag on it down in my tray. I depend upon it, as well. And since at the moment I'm sitting in front of a Windows 7 machine, I'm thankful to Microsoft for continuing to update Win7's Defender, even though they have otherwise abandoned Windows 7 and wish I wasn't still using it - well, I and nearly half of all the other desktops that are running Windows in the world.

So I was recently stating that any responsible company needs to be performing continuous network intrusion detection surveillance because defenses have become too porous compared to the external pressure that exists to get in. And in an exact analogy at the personal level, I believe that today's end users must deploy their own local surveillance within their machines. It's no longer the case that our actions explicitly invite stuff in. And so if we don't do that, nothing will bite us. Anything that we've got which is saying oh, there's an update available, click here to get it, you already trust that thing because it hasn't bitten you before. But at any moment it could.

So I'm a 100% now subscriber to the idea that you need something that is vigilant in your system that is looking at everything that goes on. And I know Defender's working. I have, like, some old C drives archived on a semi-offline drive, and sometimes I'll open them and search for something that I know I had then. And back then I had a directory of well-marked malware. And sure enough, Defender pops up and says, what the hell are you doing? And it's like, okay, relax, it's okay, this is just, you know, I'm a security guy. I do research. I have this stuff for a reason. But so it's just sort of nice when it goes off and I go, oh, yeah, that is true. I forgot about that directory. Maybe be a good time to delete it, as a matter of fact.

So anyway, I thought it was interesting that here we've got at the user level individuals being targeted and surveilled, which has put me in mind of the shift that we've had. Here I'm arguing that everything needs to update itself; right? I don't know. Notepad? Just

leave my Notepad alone. It works just fine. In fact, I think maybe I should turn off automatic updates because I've just convinced myself that this is a real danger.

Apple has quietly put iMessage in a sandbox in iOS 14. The new code was discovered, reverse engineered, described and documented by Google Project Zero's Samuel Gross. And I thank Elaine for the pronunciation. The first time I talked about him I said "Grobe" because it's G-R-O and he has like a funky...

Leo: That's S-S.

Steve: Exactly. And so Elaine, always alert to these things, sent back a little note saying, uh, Steve, that's an S-S in German. It's like, oh.

Leo: It's a special symbol there for that, Ja.

Steve: Samuel Gross of Google's Project Zero wrote: "On December 20th, Citizen Lab published 'The Great iPwn,' detailing how 'Journalists Were Hacked with Suspected NSO Group iMessage Zero-Click Exploit.'" He said: "Of particular interest is the following note: 'We do not believe that the exploit works against iOS 14 and above, which includes new security protections.'"

Now, he said: "Given that it is almost now exactly a year ago since we published the Remote iPhone Exploitation blog post series" - and we covered that at the time. They were amazing and extensive. He said: "...in which we described how an iMessage zero-click exploit can work in practice and gave a number of suggestions on how similar attacks could be prevented in the future, now seemed like a great time to dig into the security improvements in iOS 14 in more detail and explore how Apple has hardened their platform against zero-click attacks." And of course that means that your phone could receive iMessages which, without you doing anything, just the receipt of the iMessage could be enough to give the attacker either access to your phone that you don't want them to have or some information that allows them to incrementally get to a point where they're able to succeed.

Now, what's cool about this, what Google has done, that is to say, reverse engineering iOS 14, is that Apple is famously mum about this. They just say, "Oh, it's better, but we want more." So Samuel provided more. He said: "The content of this blog post is the result of a roughly one-week reverse engineering project, mostly performed on an M1 Mac Mini running macOS 11.1, with the results, where possible, verified to also apply to iOS 14.3, running on an iPhone XS."

Now, I'll just stop there to note that, remember, it's virtually impossible to look inside an iPhone at this point. But what we do know is that Apple is reusing the same code systems, both in the iPhone and in macOS. And macOS is far more open to poke around in. And so we've talked about this happening before, but I wanted to remind our listeners that what we're now beginning to see is the iPhone's behavior being inferred from and then later verified from the behavior in the much more accessible macOS.

So Samuel said: "Due to the nature of this project and the limited timeframe, it is possible that I have missed some relevant changes or made mistakes interpreting some results. Where possible, I've tried to describe the steps necessary to verify the presented results and would appreciate any corrections or additions." He says: "The blog post will start with an overview of the major changes Apple implemented in iOS 14 which affect the security of iMessage. Afterwards, and mostly for the readers interested in the

technical details, each of the major improvements is described in more detail while also providing a walkthrough of how it was reverse engineered." And that's of course way more than we need to get into here. The main things that Apple did are of interest.

He says: "At least for the technical details, it's recommended to briefly review the blog post series from last year for a basic introduction to iMessage and the exploitation techniques used to attack it." Okay. So the four things Apple did, he said: "Memory corruption-based zero-click exploits typically require at least the following pieces: One, a memory corruption vulnerability, reachable without user interaction and ideally without triggering any user notifications. Second, a way to break ASLR remotely." Right? Address space layout randomization remotely. You need to gain information about that. "A way to turn the vulnerability into a remote code execution, third. And then, fourth, likely a way to break out of any sandbox, typically by exploiting a separate vulnerability in another operating system component, for example, a user space service or the kernel."

And certainly in years past, when we've talked about the way the Pwn2Own attacks have succeeded, it was often by doing what he was referring to, chaining vulnerabilities. One vulnerability gets you here. Then from there you're able to use a second one to move to a different location and so forth in a chain.

Okay. So he describes three things that Apple has done to essentially zap all four of those things. The first is the BlastDoor - great name - the BlastDoor Service. He said: "One of the major changes in iOS 14 is the introduction of a new, tightly sandboxed BlastDoor service which is now responsible for almost all parsing of untrusted data in iMessages," he says, and he gives an example of a function inside. He says: "Furthermore, this service is written in Swift, a mostly memory safe language which makes it significantly harder to introduce classic memory corruption vulnerabilities into the code base."

Leo: Woohoo.

Steve: So of course this is exactly what I was just talking about.

Leo: Swift is not C.

Steve: It's not C, and it's not assembler, exactly. And it is exactly this. That's the way you want to write your code to parse untrusted data and write that service in Swift so that the service itself will be highly resistant to attack. So then in his presentation he shows us a rough diagram of iMessage message processing pipeline, which is this big grid of arrows pointing everywhere.

And then he continues: "As can be seen, the majority of the processing of complex untrusted data has been moved into the new BlastDoor service. Furthermore, this design with its seven-plus involved services allows fine-grained sandboxing rules to be applied." He says: "For example, only the IMTransferAgent and apsd processes" - whatever those are - "are required to perform network operations. As such, all services in this pipeline are now properly sandboxed, with the BlastDoor service arguably being sandboxed the strongest."

The second thing of the three that Apple did, rerandomization of the dyld - that's probably dynamic load - shared cache region. He says: "Historically, ASLR on Apple's platforms had one architectural weakness. The shared cache region, containing most of the system libraries in a single prelinked blob, was only randomized per boot, and so

would stay at the same address across all processes." He says: "This turned out to be especially critical in the context of zero-click attacks, as it allowed an attacker, able to remotely observe process crashes, for example through timing of automatic delivery receipts, to infer the base address of the shared cache and as such break ASLR, a prerequisite for subsequent exploitation."

He said: "However, with iOS 14, Apple has added logic to specifically detect this kind of attack, in which case the shared cache is rerandomized for the targeted service the next time it is started, thus rendering this technique useless. This should make bypassing ASLR in a zero-click attack context significantly harder or even impossible," he says, "apart from brute force" - which would mean guessing, just guessing blindly and hoping you get lucky about the cache's location - he says, "depending upon the concrete vulnerability."

And finally, third, exponential throttling to slow down brute force attacks. He said: "To limit an attacker's ability to retry exploits or brute force ASLR, the BlastDoor and imagent services are now subject to a newly introduced exponential throttling mechanism enforced by the launch daemon, causing the interval between restarts after a crash to double with every subsequent crash," he says, "up to an apparent maximum of 20 minutes." He says: "With this change, an exploit that relied on repeatedly crashing the attacked service would now likely require in the order of multiple hours to roughly half a day to complete instead of a few minutes."

And anyway, so then for the remainder of his disclosure Samuel lays out the details of what he found. I've got the link in the show notes for anyone who really wants to, you know, is curious about exactly how Apple did these things. It seems to me in today's climate it is this sort of anticipatory design for security that's needed. After the troubles that surfaced in iOS 13, Apple did not simply patch those individual flaws and leave everything else alone. Instead, they fundamentally rearchitected the entire iMessage processing system to preempt entire classes of next-generation attacks. And it's the right thing to do. iMessage has an exposure to the world. It's the way unsolicited stuff can come into your phone.

And so rather than assuming that, oh, those were the last bugs we're ever going to have, Apple said, okay, those are not the last bugs we're ever going to have. We keep fixing them. So let's change the architecture. Let's look at, if we have a bug, how can we keep it from compromising the user? And that's what they did. And one could argue that it should have always been that way. But what we've been witnessing is a slow but sure, constantly rising threat level. Through the life of this podcast, the 15 years that we've been doing this, we've watched the threat level go from people putting scripts in email because they thought it was funny to nation-states deeply infiltrating U.S. corporate and government networks surreptitiously. I mean, it's a whole new game.

So Apple has been in the middle of it all, through all of this. And without question, lessons are being learned, hopefully by the entire industry. And that's what I think makes Samuel's sharing of what Apple won't tell us so valuable. If Apple hides this innovation, then the rest of the industry isn't able to go, ooh, yeah, maybe we should do that, too. To me, there's little doubt that future systems will be designed with these sorts of mitigations built in from the get-go. But only if those who are implementing them are willing to share what's going on. There's no way that bad guys knowing this is being done is going to help them.

The way Apple did this, it is limiting the amount of and rate of knowledge that a bad guy can obtain. And mitigation really is what this is because Apple is saying, okay, we keep trying to fix problems. Bad guys keep finding them. Let's change the system so it doesn't matter if they find them. We're still going to do our best to fix them, but let's add a really strong effective sandboxing around iMessage, which is what they've done. So props to

Google and Samuel for taking the time to do this reverse engineering and then share it with everybody else. It raises the bar of what a responsible provider needs to do to protect their own users. So bravo.

And Leo, as you said earlier, for the past 10 years the command all of us who have used Linux and Unix know as SUDO turned out to only be pseudo secure. Last week a very serious bug came to light in the command - this is where you do not want it - the command line parser of Linux's powerful SUDO command. As we know, SUDO allows non-root users to temporarily elevate their rights to obtain some root privileges. This is typically done for performing maintenance and various system-level things that your typical user, even if you're the only user on the system, the modern security architecture says everybody should be running with reduced rights all the time, and you only selectively and briefly raise your rights in order to do stuff. And that's essentially what Windows provided with the whole UAC architecture where you've got to say yes. The screen goes dark, and you get a dialog, and then you say yes, I want to do this. And that then switches your credentials briefly in order to allow you to do that.

So Qualys discovered the problem. They wrote: "The Qualys Research Team has discovered a heap overflow vulnerability in SUDO, a near-ubiquitous utility available on major Unix-like operating systems. Any unprivileged user can gain root privileges on a vulnerable host using a default SUDO configuration by exploiting this vulnerability."

They said: "SUDO is a powerful utility that's included in most, if not all, Unix- and Linux-based OSes." FreeBSD has it, so it's everywhere. "It allows users to run programs with the security privileges of another user. The vulnerability itself has been hiding in plain sight for nearly 10 years. It was introduced" - it'll be 10 years in July - "introduced in July of 2011 with commit 8255ed69 and affects all legacy versions from 1.8.2 to 1.8.31p2 and all stable versions from 1.9.0 to 1.9.5p1 in their default configuration."

They said: "Successful exploitation of this vulnerability allows any unprivileged user to gain root privileges on the vulnerable host. Qualys security researchers have been able to independently verify the vulnerability and develop multiple variants of exploit, and obtain full root privileges on Ubuntu 20.0, Debian 10, and Fedora 33. Other operating systems and distributions are also likely to be exploitable." And yeah, anything that was packaged in the last 10 years.

"As soon as the Qualys research team confirmed the vulnerability, Qualys engaged in responsible vulnerability disclosure and coordinated with SUDO's author and open source distributions to announce the vulnerability." So when that happened, the maintainers of SUDO followed up last Tuesday, the 26th of January, saying: "A serious heap-based buffer overflow has been discovered in SUDO that is exploitable by any local user. It has been given the name Baron Samedit by its discoverer." I have no idea...

Leo: No, that's a play on the voodoo god Baron Samedi.

Steve: Ah. Thank you, Leo.

Leo: He was evil. That is hysterical. Baron Samedit.

Steve: Baron Samedit, yeah. "The bug can be leveraged to elevate privileges to root, even if the user is not listed in the SUDOers file. User authentication is not required to exploit the bug." So the concern here, beyond this, I mean, I'm sure that our listeners who maybe have Linux deployed in environments where there might be unauthorized

users with access to it who should not be able to gain access, you're going to want to update your instance of SUDO immediately.

But the concern is that the Internet is full of systems - and this speaks to your point, Leo, you made before - which quite naturally depend for their security upon the proven strength of the Unix/Linux account privilege model. And this now well-known flaw punches right through that. There are doubtless countless systems where an unprivileged user account is easily available with minimal authentication requirements.

Leo: Not good.

Steve: Such systems are utterly dependent upon their security for that unprivileged account remaining unprivileged. But any Linux system that went online anytime after July 2011, 9.5 years, will contain this flaw until and unless it is updated. And we know that Linux is embedded everywhere. And everything we've seen demonstrates that most of those systems will never be updated. They're everywhere. So to repurpose a now-famous phrase, "Stand back and stand by."

Leo: It's an Evil Maid attack, though; right? Again, you'd have to be at the keyboard to do that. Most people - you can't...

Steve: No.

Leo: No?

Steve: No. You don't need to be at the keyboard. Any online account, any non-root online account can do this. So it is exploitable.

Leo: But you're at the keyboard of that online account.

Steve: Sure.

Leo: You can't get onto my Linux server unless you have a login, and that's not using SUDO. That's using SSH or something like that, which is much more secure.

Steve: Well, but you could also see scenarios where the privilege model is relied upon.

Leo: Right.

Steve: There is a low-privilege login.

Leo: If you have a user account, yeah, absolutely. Every university is just going to be vulnerable. Yeah, yeah, yeah.

Steve: Exactly. Exactly.

Leo: I'm hoping - I'm just logging into update. I forgot, see, this is a perfect example. I updated all my desktop Linux machines. But I've got my server in the other room, which I never log into, I haven't updated in a long time. Got to update that; right? Yup. Yikes.

Steve: And you're also right that this is a goldmine maybe for initial network intrusion, but definitely for post-intrusion lateral movement within an organization.

Leo: Yeah. Once you're on that system, right, now you can escalate, yeah. Wow.

Steve: Yeah. Okay. So I want to share a progress report that I wrote yesterday for my blog over on GRC's forums. And I've had this experience before. Anybody who's done any journaling probably has. I did a lot of journaling when I was in high school. And I would open my journal and decide I wanted to write about some personal subject. And even though I went in thinking I knew everything about it, invariably a paragraph or two in I would think of something that had never occurred to me before. And I decided that the act of making myself write it out longhand, it kept those ideas in my brain longer, and it gave them a chance to associate with other new ideas.

So this literally - I had a major, like, eureka event yesterday about the future of SpinRite. And I sort of started out what I have here in my notes as that blog posting. But then I expanded upon it because there was a lot more context. So I started by saying: "February 1st Progress Report. Much has transpired since my previous January 3rd announcement that work on SpinRite 6.1 has commenced. To plan for what SpinRite 6.1 would and would not be, I needed to know how I was going to solve SpinRite's imperative need to boot on UEFI systems that no longer offer a traditional BIOS fallback. So while beginning to work on SpinRite, I was also searching for any means to add BIOS support to a system that doesn't currently have any.

"The firmware which boots a system is intimately tied to its hardware. That's why systems need firmware in the first place. This means that it's not possible to simply run any BIOS on whatever hardware happens to be present. So I was hoping that someone, somewhere, might have created a UEFI-to-BIOS translation layer which would allow for BIOS calls to be translated into their UEFI equivalent. That would allow SpinRite to load such a translation layer, then boot the BIOS-dependent DOS, and then reuse everything that SpinRite already is on any UEFI system.

"But, unfortunately, nothing like that exists; and no one has ever created such a thing because the need for something like that only exists due to SpinRite's unique and continuing dependence upon the legacy of 16-bit DOS. So that search provided some needed clarity. The only way to move SpinRite forward would be to finally, after 35 years, end its dependence upon the BIOS and DOS. So I began looking around for the optimal environment which would host SpinRite's future."

And I'll spare everybody the blow-by-blow. That's all over in the spinrite.dev newsgroup where all of this conversation was happening earlier in January. But for what it's worth, we looked at everything. Linux; using Windows PE, the Pre-Execution Environment; the ReactOS; various other hobby OSes; and many of the various embedded OSes like VxWorks. In the end, I settled upon an extremely lightweight real-time operating system for embedded applications which is called the "OnTime RTOS32."

What I love about it is essentially it is able to host Windows applications, within limits, in an embedded environment. So it allows executable code to be written, tested, and debugged under Windows, which is still my preferred development environment; and then that PE format executable is repackaged for operation under its own loader and environment. And it supports booting from either a BIOS or UEFI firmware. But what that meant was that, in order to get to UEFI, I would need to dramatically change the operating environment.

Operating under DOS and the Intel processor's real mode, as SpinRite has always done, means that two 16-bit registers are always being referenced in order to access memory. And the way this is done is a bit funky. The bits in a 16-bit segment register are left shifted 4 bits to create a 20-bit value whose lower 4 bits are always zero. Then a 16-bit offset register or value is added to that 16-bit result. So the 16-bit offset register means that you can access a 64K, anything in a 64K region at one time, and the 16-bit segment register determines where that accessible 64K byte block is located with a granularity of 16 bytes.

So together this is the way the Intel real mode has always worked. This is what allows for addressing of 1MB of memory because that's what 20 bits gives you. So it's a funky way of synthesizing a 20-bit address using two 16-bit values. And this of course is where the Intel 8080 and 8086's 1MB memory limit came from. So Intel's later chips have 32-bit registers, like in Windows, and in any other, like Linux, Unix, I mean, any 32-bit OS uses a simple flat memory model where 32 bits directly allows you to specify the address. Period. You're done. And as we know, 32 bits allows the addressing of 4GB. That's 4.3 IPv4 addresses on the Internet. Those are 32-bit addresses.

So back when I wrote SpinRite, initially in 1985, I embraced segmentation. That's the way the Intel chip works, and that's all we had back then. So, for example, in SpinRite, when you rotate through SpinRite's screens, each of those screens is referenced by a 16-bit segment with each screen starting at offset zero within its own segment. On the one hand, it's messy; but it can be incredibly efficient and economical if you design to it. And it is Intel's legacy, and I designed to it.

So the problem moving forward is that only the Intel 16-bit real mode, or the 16-bit virtual 86 mode, has the hardware that performs this segmentation math. So there's no future there. If the chip is not in one of the 16-bit modes, you don't have segmented memory. So for the past three weeks or so my plan had been to finish SpinRite 6.1, and that that would be the final release of SpinRite to run in a 16-bit DOS environment, and that only made sense because any further investment in the 16-bit environment would not be portable to the future. So it just doesn't make any sense to invest more time and energy there.

My plan had been, and I described this over in GRC's newsgroup where we discussing this, to just bite the bullet and start over in a 32-bit flat memory environment, reimplement SpinRite from scratch, probably as a text mode command line utility, the way ReadSpeed is right now, get it all running, get all the functionality there, and then later move it into a graphical environment. The aha moment that I had just yesterday was - and now I'm like, I'm back in SpinRite. I've got the new memory manager is running. It's got simultaneous access to all of the system's memory. I've merged a bunch of the code from SpinRite and ReadSpeed. So that's how I'm spending my days now, and I'm making really good progress. And maybe it's the comfort with it that allowed me to see this.

But just it hit me. I'm not stuck using segmentation. I mean, I am with SpinRite's current code. But I can port 16-bit SpinRite into 32-bit land because all the segmentation is explicit in SpinRite. And so my plan now, after 6.1 is finished and published, and everyone's able to use it and play with it, I'm going to then port the 16-bit SpinRite as it

is into a 32-bit environment. It'll be hosted on this OnTime RTOS32. It'll look exactly as it does now because that's the way to get it done fast. And that will then give us a SpinRite that can boot on either DOS or UEFI, and it will be 32/64-bit code that can then continue to grow in the future.

So anyway, an interesting set of constraints that had to be worked through. But I've been thinking about it now for about a day, and I'm really happy with having sort of found a way out of this corner that SpinRite's age had painted me into. So I'm very excited to be making great progress with 6.1, and to have a way of not having to scrap everything in order to move over to UEFI and then continue working on the additional drivers that I want to add to it. So anyway, I just thought our listeners would find it interesting.

Slipstreaming 2.0.

Leo: All right. It's Groundhog Day, after all.

Steve: It is. It was Episode 792 of Security Now!, which we recorded last November the 10th, titled "NAT Firewall Bypass." And perhaps we should have added a 1.0 to it in anticipation of the technique's inevitable evolution. Thus today's podcast bears the title, "NAT Slipstreaming 2.0." I'll begin by reminding everyone where we were in November, and quickly place that first-generation exploit into context against today's second-generation exploit.

The original NAT Slipstreaming attack relied on a victim within an internal network, tucked safely, they thought, behind their NAT router, who would click on a link or just visit a website which was running JavaScript, would then run JavaScript on their browser, as every website and ad now does. That would open an incoming path, using some very clever packet hacking, open an incoming path from the Internet to the victim's device. So that was bad enough back then that all of our browsers quickly moved to block outbound access to a few additional remote port numbers through which the traffic had to go in order to trigger the NAT router's Application Layer Gateway, which I'll get to in a second.

So today's upgrade of NAT Slipstreaming extends this attack by allowing the remote attacker to trick the NAT into creating NAT traversal mappings to any device on the internal network, not only to the victim's device, which originally and unwittingly did something, clicked a link, visited a site with a malicious ad or whatever. As we know, many devices located on our internal LANs may be fine there, but they were never intended to be exposed to the Internet. How many times have I talked about services that should never under any circumstances poke their heads out onto the Internet?

So this is precisely what v2 of NAT Slipstreaming allows. Many embedded devices have minimal local security, if any, because they correctly presumed that they would only ever be accessible to local users. An example is an office printer, which can be controlled through its default printing protocol or through its internal web server. And we've talked about how printers, the firmware in printers could be subverted, and malware can actually live in a printer until it's rooted out.

Well, it turns out this provides a means for the malware to get into the printer in the first place. Or maybe an industrial controller that uses an unauthenticated protocol for monitoring and controlling its functions. Or an IP camera that has an internal web server displaying its feed, which is only available, supposedly, internally and maybe has default credentials which are not a problem because it's never going to be accessed from the outside. This second-generation variant of the NAT Slipstreaming attack can access these

types of interfaces from the Internet, which results in attacks that range from a nuisance to potentially sophisticated ransomware attacks.

Okay. So in addition to network interfaces of devices that are unauthenticated by design, many unmanaged devices may also be vulnerable to publicly known but currently unpatched vulnerabilities; right? I mean, there's like all kinds of vulnerabilities that exist in devices that just no one ever gets around to patching. These could be exploited by an attacker who is able to bypass the NAT firewall to initiate traffic that can trigger known weaknesses on devices behind the NAT.

So to gain some sense for this, in a recent study, Armis's researchers found that 97% of industrial controllers which were vulnerable to URGENT/11, which is something we talked about at the time, were left unpatched more than a year after the critical vulnerabilities were first published. So there were initially 100% of industrial controllers that were vulnerable to this. The URGENT/11, to remind our listeners, was a set of 11 zero-day vulnerabilities which were discovered in the VxWorks embedded operating system, the most embedded OS today. And of those 100%, 97 are still vulnerable a year later. They just - they didn't get fixed. Embedded OSes are not going to get updated.

So those things are sitting there with well-known vulnerabilities. The only thing protecting them is that bad guys out on the Internet can't access the embedded OS. This thing provides them a means to do so. 80% of Cisco devices which were vulnerable to the CDPwn vulnerability still are, nearly a year after those critical vulnerabilities were published. So again, you definitely don't want to allow a NAT router to be abused using this new more powerful slipstream approach.

So the original discoverer of this, Samy Kamkar, he described the summary of attack which he discovered, the original one. He said: "NAT Slipstreaming allows an attacker to remotely access any TCP/UDP service, bypassing the victim's NAT/firewall just by having the victim visit a website or causing their browser to run JavaScript contained within an online ad."

Okay. So as we know, strict NAT traversal rules are straightforward, simple, and elegant. They amount to only accept incoming return traffic from remote IPs and ports that recently received outbound traffic. That's all there is to it. Right? So traffic is seen going out of the NAT router. The NAT router makes a note of it, actually creates an entry in a mapping table such that when return traffic comes back from the same IP and port that it went to, the router knows who to send it to back on the inside of the LAN. And as I've often said years ago, this forms a sort of a beautiful firewall, automatically. Unsolicited traffic just hits the NAT and, because it wasn't expected, and it isn't part of a communications that was initiated from inside, it goes nowhere.

But Samy's original inspiration was to observe that those simple and straightforward NAT traversal rules, or really that one simple and straightforward traversal rule, often needs to be bent in order to accommodate the needs of more complex network protocols. Works great for email and for web surfing. Not so, for example, in the case of Active FTP, which is an example we used before, where the outbound control connection tells the remote FTP server what port to connect to it inbound.

For that to operate transparently through NAT, because the remote FTP server is going to try to initiate what will look like an unsolicited connection to the NAT router, it requires that the NAT router be looking inside the outbound FTP traffic to actually read out of the packet the outgoing port specification which the remote FTP server's going to receive, and then automagically reroute the remote FTP server's new and technically unsolicited connection through to the proper expected port on the client machine of the LAN. So in other words, it's got to get very involved in the protocol.

So collectively these are all known as ALGs, Application Layer Gateways. Application Layer because it is at the application that the packets are carrying. The network layer would be the packets themselves. The application layer is what's in the packet envelopes. And so the ALGs require the router to be aware of the content of the packets passing through it, rather than just the fact of the packet. It can no longer just look at the destination IP and port. So under the, in my case, I looked at this last night, under the NAT Passthrough tab of the WAN section of one of my Asus routers, it provides for PPTP, L2TP, IPSec, RTSP, H.323, SIP, PPPoE Relay, and FTP. Those are the Application Layer Gateways that this very capable router is able to provide.

Conservative as I always am, when first setting up that network I immediately disabled all of those ALG passthroughs since I have no need for them. Or so I thought. Consequently, I was quite puzzled when the Verizon LTE Network Extender - I've talked about this once before on the show. Lorrie and I are somewhere with a really bad cell coverage, so they just can't use Verizon at all. It's like, no bars. So I got one of those little femtocell boxes. But I noticed that when I hooked it up and plugged it in, it wouldn't connect. Just could not find its network.

So digging through the FAQs, I noticed that it mentioned its use of IPSec. And I thought, ah, I know what's wrong. So I went back into the router, turned on just the IPSec NAT traversal passthrough, and it came right up on the network. So it is the case that you might find that you need some of these things. But again, conservatives' security best practice is absolutely positively turn off things you don't know you need. And as it happens, as a consequence of that approach, that network that I configured was never vulnerable to v2 of this NAT traversal attack.

Okay. So what happened with the first and second generations of NAT Slipstreaming? It turns out that there is an even more insidious Application Layer Gateway present in our routers than Samy had originally exploited. Researchers at Armis were intrigued by what Samy originally found. They had done some research before, but hadn't really wrestled it to the ground. When they saw Samy's posting, they immediately dusted off what they had done before and pushed it across the finish line. And of course, as we know, I quoted Bruce Schneier at the beginning of the show, "Attacks never get less powerful. They only grow more powerful."

The guys at Armis said: "Building upon Samy's ideas, and combining them with some of our own, led us to the discovery of the new variant. This new variant is comprised of the following newly disclosed primitives. Unlike most other Application Layer Gateways, these ALG functions in our routers, the H.323 ALG, when supported" - and it is widely - "enables an attacker to create a pinhole in the NAT Firewall to any internal IP, rather than just the IP of the victim that clicks on the malicious link.

"Then WebRTC TURN" - that's the Traversal Using Relay around NAT - "WebRTC TURN connections can be established by browsers over TCP to any destination port. The browser's restricted ports list" - that is, this thing that the browsers did in November to prevent this - "was not consulted by this logic, and was therefore bypassed. This allows the attacker to reach additional Application Layer Gateways such as the FTP and IRC ALGs using ports 21 and 6667, respectively, that were previously unreachable due to the restricted ports list. The FTP ALG is widely used in NATS and firewalls." So their point was, first of all, if you have H.323, that's a powerful Application Layer Gateway for abuse. If you don't, then the support for WebRTC TURN can be abused to bypass the restricted ports list that are currently protecting our browsers from initiating this kind of traffic.

And they said: "This also defeated the browser mitigations introduced shortly after Samy first published the NAT Slipstreaming attack which added the SIP port 5060 to the restricted ports list, but did not block the port from being reachable via a TURN

connection." And they said: "H.323 is a VoIP protocol similar to SIP, which is also quite complex. For a network of VoIP phones to function properly, while having a NAT somewhere inside the topology, an H.323 Application Layer Gateway is required. The H.323 ALG needs to translate IP addresses that are contained within the application layer H.323 protocol packets, and open holes in the NAT in certain scenarios." Actually it's when call forwarding features are used.

They said: "Most ALGs only need to worry about at most two addresses to translate within a session, the IP address and ports of both sides of the TCP connection. However, H.323 is a telephony protocol and supports call forwarding. Therefore, in this case, one party within the session can refer to a third-party IP address belonging to the VoIP phone that the call should be forwarded to. Most H.323 ALGs support this feature."

So the upshot of all this is that a single H.323 packet sent over TCP port 1720 that initiates call forwarding can open an external pinhole through the NAT to any TCP port of any internal IP on the network. Thus NAT Slipstreaming just got a whole lot more powerful. The Armis blog about this contains full details for anyone who wants more. I've got the link in the show notes. They tested many routers and found virtually all of them to be vulnerable to full port and IP exploitation, meaning that they were able to open remote attacker access to any IP and port behind the NAT. Since all of these attacks bounced scripts and network packets off of our web browsers, they then responsibly disclosed what they had found to all browser vendors.

So in a quick timeline, November 11th of 2020 was the coordinated disclosure of the new variant initiated with Chromium, Mozilla, and Apple. So I think our podcast on the v1 was November 10th. So the day after, the Wednesday, after we did that podcast, these guys had already jumped on this, and they began a coordinated disclosure. On January 6th, so last month, Chrome release 87 contained the patch mitigating the new attack variant. On the 7th, the next day, Microsoft Edge inherited it from Chrome, so Edge 87 got the same patch. On the 14th, a week later, Safari released v14.0.3 beta that contains a patch, and it has since gone of course mainstream. And on the 26th of last month Firefox released 85 that contains a patch against the attack. And this was kept quiet and embargoed until all of our mainstream browsers were protected.

So recalling from our coverage of this last November, our browsers responded to the first slipstreaming revelations by blocking HTTP and HTTPS access to ports 5060 and 5061. Now, as a consequence of Armis's second-generation exposition, all of the browser vendors realize, holy crap, that doesn't even begin to be sufficient. So now all of the browsers are blocking outbound HTTP, HTTPS, and FTP access to ports 69, 137, 161, 1719, 1720, 1723, and 6566 TCP ports. Google said at the time, I mean, now they're acknowledging.

They said: "The NAT Slipstream 2.0 attack is a kind of cross-protocol request forgery which permits malicious Internet servers to attack computers on a private network behind a NAT device. The attack depends on being able to send traffic on port 1720 (H.323). To prevent future attacks, this change also blocks several other ports which are known to be inspected by NAT devices and may be subject to similar exploitation."

So this is sort of a fool me once, shame on you, but they're not going to only block 1720 this time. They're going to say, okay, what other ports are NAT routers looking at, and let's just shut this whole thing down. So that's been resolved by our browsers. But I would argue that this is a perfect time to look at the configuration of your routers. Because I think the other lesson here is to always turn off any unneeded features of your router. You wouldn't have thought that these things could be abused. Turns out Samy figured out how to do it.

As a consequence of my very conservative approach, my own network wasn't ever vulnerable to any of this because, as I said earlier, I had already disabled all that. Then one of them bit me. The lack of IPSec, it turns out, well, I have a need for it. So I turned only that one back on, and everything is good again. And I'll keep in mind in case I do something, I don't know, I don't use any VoIP, but I would need to turn one of those on if I did. So it's good that our browsers are protecting us better if your router doesn't have abuse prone Application Layer Gateways in the first place.

Leo: And, you know, you can turn them on if you need them.

Steve: Right.

Leo: Yeah. So if something breaks, turn it back on. But I think it's a good idea. Turn everything off and wait and see what happens.

Steve: Exactly, yeah.

Leo: It's true, you know, this is true everywhere. Same thing on a Network Attached Storage. You might be tempted to turn on bunch of services you don't use. Don't. Turn them off.

Steve: Right, exactly, exactly,

Leo: You know what everybody should do is go over to ShieldsUP!. Just check. See what services your router's saying hello to. Knock on the door. You want everything stealthed. I've learned that over the years.

Steve: Yup.

Leo: That's where you can get a copy of this show while you're there: GRC.com. Steve's got 16Kb audio, 64Kb audio. It's the sole and only source for 16Kb audio and transcriptions, beautifully written by Elaine Farris, so you can read along, even with the "Gross." And she's got the, what is that called, the eszett, something, she's got that funny - did she put the little funny "ss" in there?

Steve: Yeah. Yeah. And in fact I copied it from wherever it was that I saw his name.

Leo: Copy and pasted in, yeah.

Steve: And put it into my own show notes so I could be official also.

Leo: Hard to find it, yeah.

Steve: So I could be official also.

Leo: Yeah. I recognize it from Strasse, the German word for street. You see it in Strasse.

Steve: Ah.

Leo: All that at GRC.com. While you're there, pick up a copy of SpinRite. We're getting to the end of the development cycle for 6.1.

Steve: Oh, yes, yes, yes.

Leo: I can feel it. But get 6.0. You'll get a free upgrade to 6.1, and you can participate in the final beta tests and so forth. The world's best hard drive maintenance and recovery utility, getting better all the time. Soon we'll be saying the world's best hard drive and SSD recovery utility. I guess we could say that now, frankly.

Steve: It refuses to die, Leo. It just will not die.

Leo: A program that will not die. You can get 64Kb audio and video files at our website, TWiT.tv/sn. There's a YouTube channel. If you're a YouTube kind of person, subscribe to the Security Now! YouTube channel. Best thing in my opinion would be to get your favorite podcast client and subscribe there. And we're asking, everybody is asking for reviews these days. If you leave reviews on Apple's podcast platform, that's always helpful for people, "What is this Security Now!? I never heard of this." Scratch scratch scratch. Just say "Just download it and listen." And then you'll want to listen to all 803 other episodes.

Steve: And you won't need your Sominex.

Leo: Well, you might need your Vitamin D and your zinc. But you won't need the melatonin. There you go. No. It doesn't put you to sleep. It's fascinating.

Steve: No. Doesn't put me to sleep.

Leo: It's fascinating. No, you work hard on this, I know. I appreciate it. Steve can be messaged on Twitter: @SGgrc. If you follow him there, you'll get updates to when the show's available. He puts show notes up there. But again, he can take messages there or at GRC.com/feedback. It's been a fun day. Thank you, Steve. Go back to your knitting. I know you're [crosstalk].

Steve: Now, when we last talked you were going to get four inches of rain. Did you get...

Leo: Got it. Got it, yeah.

Steve: Wow.

Leo: Yeah. And now it's a beautiful clear sky. It rained yesterday, too. We got a lot of rain. The big atmospheric river missed us by a few miles to the south. But we still got a lot of rain. The main thing we're looking for, and probably you, too, is snow pack. I don't know about Southern California, but in Northern California all of our water is up there in the Sierra, and we have to get that runoff in the spring. And so we want heavy snow, and we got that.

Steve: Yeah. Yeah.

Leo: We got that, yeah.

Steve: Yay. That's good.

Leo: Thank you, Steve. Have a great week.

Steve: Okay, buddy.

Leo: We'll see you next week on Security Now!.