SECURITY NOW!

Transcript of Episode #860

Trust Dies in Darkness

Description: This week we examine the consequences of paying ransomware extortion demands. How did that work out for you? We take a deep look into "Daxin," a somewhat terrifying malware from attackers linked to China. We take something of a retrospective look at Log4j and draw some lessons from its trajectory. We touch on some technical consequences of Russia's invasion of Ukraine, including which kitchen appliances Russia's servers are claiming to be, and the question of the possible consequences of the U.S. becoming involved in launching some cyberattacks at Russia. We have a piece of interesting listener feedback and the results of last week's next SpinRite development pre-release. Then we're going to take a look at the significant mistake Samsung made which crippled and compromised the security of all 100 million of their most recently made Smartphones.

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-860.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-860-lq.mp3

SHOW TEASE: It's time for Security Now!. Steve Gibson is here. News flash. Ransomware authors are not reliable. Shocking; isn't it? A new nation-state attack, we think from China. Why some Russian websites are saying, "Hey, I'm a teapot." And then the hundred million Samsung phones that have a fatal flaw, and what you can do about it. It's all coming up next on Security Now!.

Leo Laporte: This is Security Now! with Steve Gibson, Episode 860, recorded Tuesday, March 1st, 2022: Trust Dies in Darkness.

It's time for Security Now!. And you couldn't have a better time for this guy to show up, Steve Gibson of GRC.com, our security guru, our privacy mentor, our sensei of...

Steve Gibson: Security.

Leo: Security. Yeah, I said - I was trying to find a new word for security.

Steve: Yeah, it's a little alliterative.

Leo: Alliterative word for security. Safety. Our sensei of safety. He's here...

Steve: Oh, I like that.

Leo: Yes, to lead us down the path to righteousness.

Steve: Lead us from darkness.

Leo: Yes.

Steve: Today's podcast is from the first portion of a recently released and to be shown later this year paper from three security researchers at the University of Tel Aviv. Podcast title is "Trust Dies in Darkness."

Leo: Oh, boy.

Steve: It's a little gloomy. So we've got Security Now! #860 for this first day of March. So that means we're going to have one of those weeks, or a month where we get the earliest Patch Tuesday possible, which will be next Tuesday, March 8th. And we'll get there in due time. We're going to examine first the consequences of paying ransomware extortion demands. How did that work out? We're going to take a deep look into Daxin, frankly a somewhat terrifying malware from attackers linked to China. We take something of a retrospective look at Log4j and draw some lessons from its trajectory. We're going to touch on some technical consequences of Russia's invasion of Ukraine, including which kitchen appliances Russia's servers are claiming to be.

Leo: What? Okay. Okay.

Steve: I know. And the question of the possible consequences of the U.S. becoming involved in launching some cyberattacks at Russia.

Leo: Yes. I want to talk about that. I think that's a little interesting, the saber-rattling, yeah.

Steve: A real mixed blessing.

Leo: Yeah.

Steve: We also have an interesting piece of listener feedback that comes from last week, and the results of last week's next SpinRite development pre-release, which I mentioned was coming. It did. Then we're going to take a look at the significant mistake Samsung made which crippled and compromised the security of all 100 million of their most recently made smartphones.

Leo: Yeah.

Steve: Because they did it in darkness.

Leo: Yeah, yeah, yeah.

Steve: And that's where trust goes to die.

Leo: Yes. They thought they were doing everything right.

Steve: They did, but they didn't let anybody see. And so there are a number of things that we're going to talk about this week which are basically putting meat on some of the fundamental principles that I've been talking about on the podcast for years.

Leo: It's the old pseudorandom number generator problem once again.

Steve: And we do have a fun Picture of the Week.

Leo: We do.

Steve: Which we'll be coming back to shortly.

Leo: Yes, any moment now. I'm going to call you the Sensei of Assembly Language, that's what I'm going to call you. And now the Picture of the Week.

Steve: So, okay. So we have, I suppose, a programmer who has recently met his demise, presenting himself to the Pearly Gates. And, now, traditionally St. Peter has had a long scroll, which he's scrolled through to find out the disposition of this recently passed person. But this is 2022, so he has a laptop. Anyway, St. Peter is telling the programmer: "Says here you should be in hell; but since you coded in Assembly, we'll count it as time served."

Leo: Oh, now, you love your assembly language.

Steve: You just can't take that assembly away, period, nope. That's my language.

Leo: For you, hell is a high-level language.

Steve: And I visit there from time to time. I prefer assembly language.

Leo: What high-level languages do you write in?

Steve: Perl is one.

Leo: Oh, nice.

Steve: Yeah, Perl. I never got into PHP, but of course C. In fact, I'll be mentioning a little bit later that I implemented a cryptographic algorithm which I needed for SQRL in C, and open-sourced it because - and I wrote it in C...

Leo: Oh, nice.

Steve: ...so that everybody else would have access to it. So, you know, sort of whatever the need is. Gravity was written in C, and I've been maintaining it now for a few years when it famously, at the beginning of 2020, it stopped working because the programmers put in, if an article's date is 2020 or later, it must be bogus. Well, that was true in 1980...

Leo: Yeah, not so much now.

Steve: ...when, you know, Gravity was in use. But not so much now. So it needed a little...

Leo: I love C. Kernighan and Ritchie, that little slim C programming language book, one of my favorites of all time. It just...

Steve: Yes, yes.

Leo: It was so elegant and simple and beautiful. And I had come from, as most of us had, BASIC, you know, a little bit of BASIC. And when I found C, I was, you know, this is probably '83 or '84, very early on. I just fell in love.

Steve: Well, and the idea, too, that the language would just provide a minimum structuring environment for the library. That is, you know, most of C is in the functions which you call. And then there's just a little bit of glue, you know, some curly braces and some loops, looping structures and variable naming. But, you know, it just, they just...

Leo: I like that in a language. I think a language should be easy to grasp. That's one of the reasons I like Lisp and Scheme and those languages. The idea is you could teach it in a day. You can learn it in a day. Then there's a lot more. But languages like Perl, you know, where their philosophy is there's more than one way to do it, there are so many extra little - so much - who was it, it was Perlis who said: "Syntactic sugar will give you cancer of the semicolon." There's so much syntactic sugar. There's so much extra stuff.

Steve: I have the perfect example. For anybody who wonders what we're talking about in Perl, everyone's heard about like the "if" statement, you know, if this, then that. Do you really need "unless"?

Leo: No. No.

Steve: I'm not kidding. There's an unless.

Leo: Makes no sense. Oh, I know. Lisp doesn't even have do loops. They don't even have, I mean, there's no iteration. You don't need it because it's all there. And so the less you have to bend your mind to understand this, I think the sooner you get down to getting stuff done.

Steve: Yeah. Just, come on.

Leo: Unless. Maybe. Well, there's a maybe, too, by the way. You know there's a maybe clause. Let's not get into it.

Steve: No, no. If you really want to, and we have mentioned this once before on the podcast, but there was once an exercise some friends and I did of designing a language kind of backwards. So rather than having the go-to statement, you would have the come-from. And you can actually do that.

Leo: Come-from. Okay. Okay. The beauty of...

Steve: Yeah. That one will hurt you.

Leo: ...C, and your assembly, and Lisp, too, is you can do macros. So if you want to have a come-from, you can write it.

Steve: Oh, in fact, for example, I believe that language is for communication, not only with the computer, but for the programmer. So I have a macro, which the macro is zero, Z-E-R-O. And what it expands to is XORing P1 with P1. So anyway, so for example, in my code I say zero EAX, which zeroes the EAX register.

Leo: And it's clearer to have that than the XOR.

Steve: Yes.

Leo: You know and I know the XOR is zeroing it.

Steve: Yes.

Leo: But by writing zero it's very clear when you're reading the code. The machine doesn't care.

Steve: Nope, same thing for the machine.

Leo: The machine sees the XOR, yeah.

Steve: Exactly.

Leo: Yeah, no, I love that. That's exactly right. You're writing for yourself 10 years from now, or even six months from now.

Steve: Yeah. The older we get, the shorter the duration.

Leo: Why did I do that? What am I doing here?

Steve: Okay. So one of the things any victim of a ransomware extortion wonders is whether the bad guys, who at that moment hold all the cards, will honor their promises to never come back for a double dip at the ransom trough. Last week, Help Net Security reported on the result of a global survey that was conducted by the cybersecurity firm Venafi (V-E-N-A-F-I), which highlighted the unsurprising lack of trustworthiness - yeah, who would have thunk - of ransomware bad guys. The report found that in most cases, once the initial ransom is paid, the extortion shock does not end. And the report puts some numbers to this unsurprising conclusion.

Venafi calls themselves "The world's most trusted machine identity management platform." And while I don't know about that, I saw that they specialize in, among other things, keeping an organization from being surprised by the expiration of their machine identifying certificates. I know that my favorite company, DigiCert, does the same thing. So I guess Venafi does something similar, although I don't think they make certs. Anyway, they are certainly aware of and in the enterprise space.

Their survey revealed that 18%, so about one in five, of victims who paid the ransom still had their data exposed on the dark web. 8% refused to pay ransom, and the attackers used every available extortion opportunity, like pushing them every way they could. 35% of victims paid the ransom, but nonetheless were still unable to retrieve their data. One in three paid the ransom, oops, sorry, decryption didn't work.

As for the ransomware actor extortion tactics, they summarized those, too. 83% of all successful ransomware attacks featured double and even triple extortion, the first extortion being over the data encryption, the second extortion over the public release of confidential or trade secret data, and the third extortion is a DDoS. We're going to attack you if you don't follow one of our previous extortions. So, I mean, it's a mess. 38% of ransomware attacks threatened to use stolen data to extort the victim's customers. 35% of ransomware attacks threatened to expose stolen data on the dark web. And 32% of attacks threatened to directly inform the victim's customers of the data breach incident, you know, and we've talked about this before, by way of embarrassing them.

So Venafi explained that the lack of credibility of ransomware actors' promises to their victims stems from several factors. First, most ransomware-as-a-service operations are short-lived, so they simply look to maximize their profits in the shortest possible period of time. As such, they don't care about long-term reputation. We've witnessed on this podcast a continual churn of ransomware-as-a-service names with a suspicion that new actors coming on the scene are relabeled old actors.

Of course, secondly in ransomware-as-a-service, the operators and their affiliates are separate entities, and many renegade affiliates don't follow the rules set by the core ransomware operators. Enforcing these rules is rarely considered a priority for the groups. Third, even if the data isn't leaked right away, the remnants of data breaches may be maintained for a long time in multiple threat actor systems and almost always find their way to the broader cybercrime community sooner or later. You know, you can imagine, if you're a bad guy, and you've got the goods on some company, and you've promised to delete it, well, why would you?

You could for a while, you know, maybe abide by your agreement. But again, really, we're talking criminals. So why expect noncriminal activity from bad guys? As Venafi's report underlines, paying the ransom is only motivating the crooks to return for more, as it sends the signal that the victim sees this as the easiest way out of trouble, which is just an illusion.

Venafi's vice president said: "Organizations are unprepared to defend against ransomware that exfiltrates data, so they pay the ransom. But this only motivates attackers to seek more. The bad news is that attackers are following through on extortion threats, even after the ransom has been paid. This means CISOs are under much more pressure because a successful attack is much more likely to create a full-scale service disruption that affects their customers."

And in a different but related report published by Proofpoint just yesterday, Monday, which presents the results from a survey of thousands of employees and hundreds of IT professionals across seven countries, they found that 70% of the surveyed participants reported having experienced at least one ransomware attack last year, in 2021, 70% of respondents. 60% of them opted to negotiate with the attackers, so 60% of the 70%, and many of them ended up paying ransom more than once.

Now, the only way that would work would be that the initial ransom is for both data encryption and a promise to destroy exfiltrated data. Then once that money is in hand, and the decryption keys have been provided, that second demand is made for the second half, even though the promise was nothing, you know, there would no exfiltration if you pay up. Well, so the money is received. Decryption keys happen. Then a demand is made, a second demand, for additional money. And the logic must be that getting an additional payment from a victim who has already paid once is easier. They've got a proven payment channel. They've figured out how to convert dollars into bitcoin or whatever. They've already invested something in ransom. So what's just a bit more to keep, they believe, to keep their data out of the hands of either the dark web or the public.

And really, why should such people as the bad guys not turn right around either way, if the victim pays up or not, and resell the data again on the dark web? Again, as I said, why delete an asset? It seems clear that that's what would be done. For a while there was, and we talked about it here, a working theory that the ransomware gangs were, to some degree, they had a motivation to be concerned about their reputation since it would tend to increase the likelihood of the ransom being paid if it was generally understood that, when ransom is paid, decryption keys are provided. But that was before that secondary incentive was added, because it was added later on, of data exfiltration and its extortion to prevent its subsequent release and/or sale.

And it was also before the ransomware-as-a-service model tended to blur the lines of who was doing the attacking and who was responsible for keeping their word. While the ransomware-as-a-service operators tightly control the unlocking of the encrypted data, we know that, it's the separate individual affiliates who obtain control of and get the pre-encrypted exfiltrated data. They hold that separately. Therefore, the RaaS operators have no say or control over the possibility of secondary extortion being perpetrated by the independent affiliates. So there's just been a lot of change in this side of the world.

And finally, whereas ransomware was a little-known novelty once upon a time, with only a few perpetrators, it's grown into a well-known and recognized form of cybercrime with too many villains now to count. So it's dramatically changed the calculus about any one gang's reputation. It just no longer matters. As a consequence of all this, all the data points to the only sane approach being to never capitulate to ransomware demands. I mean, unless you absolutely have no backups of your systems, and the data which has been encrypted absolutely has to be restored.

And, I mean, just at this point there's no excuse for not having backups of your systems. Don't pay the ransom, restore systems and data from backups, and immediately alert law enforcement authorities of the incident. To do otherwise, all the data suggests, and even common sense as we've seen how this terrain has evolved over the last few years, doing otherwise will just be futile. You'll end up losing money to the bad guys. They'll be happy to take your money. And you'll still be in trouble. So it just doesn't, at this point, makes no sense to pay up. There's just no logic behind it. Unless, again, as I said, understanding all of this you still make the calculus for some other reason that it's the only way out is to pay the ransom and then hope for the best.

Okay. Daxin. D-A-X-I-N. Yesterday, also yesterday, Symantec's Threat Hunter team, which is now part of Broadcom Software, posted really interesting details of what is by far the most technically sophisticated Chinese backdoor malware these researchers had ever seen. And, you know, it's one thing when script kiddies get something from GitHub and go scan and hope that they get lucky. The whole idea of malware being really, really good is unnerving, like on a different level.

Leo: Makes me think a government wrote it, not script kiddies; you know? It's a little scary.

Steve: Yeah, yeah, yeah. Exactly. This is not, as we will see, Daxin did not come from no script kiddie. And we're only ever talking about stuff that Russia is doing, and China is doing. I just have to hope, because things are getting crazy in the world right now, you have to hope that our government is every bit as competent and has, like, over there they're doing podcasts about us and stuff that they're finding from the NSA, much as we're talking about China malefactors.

Anyway, their post, Symantec's post stated that the malware appears to be used in a long-running, and I'll put some dates on that in a minute, espionage campaign against select governments and critical infrastructure targets. Daxin, as they named it, allows an attacker to perform various - oh, I've got that way turned - turn that - sorry about that.

Leo: Is that Pebbles?

Steve: It was actually a four year old, the four-year-old son of someone on CompuServe.

Leo: Oh, cute. Oh, cute.

Steve: And I've had that sound file from the days of CompuServe.

Leo: Yeah, which means the kid's 38 now. But okay, fine.

Steve: Yeah, exactly. And does not want to be reminded of the way he sounded back then.

Leo: No, oh, man.

Steve: So Daxin, as they named it, allows an attacker to perform various communications and data-gathering operations on the infected computer, and it was seen in use as recently as November of 2021, so a few months ago, by attackers linked to China. Most of the targets appear to be organizations and governments of strategic interest to China. And further strengthening the Chinese Daxin link is the fact that other tools also commonly seen and associated with Chinese espionage actors have also been found on some of the computers where Daxin had been deployed.

Symantec's researchers were extremely impressed and said that, considering Daxin's capabilities and the nature of its deployed attacks, it appears to be optimized for use against hardened targets, allowing the attackers to burrow deep inside a target's network and exfiltrate data without raising suspicion.

And this is not all just theoretical. Thanks to Broadcom's membership in the Joint Cyber Defense Collaborative (JCDC), Symantec researchers worked with CISA, our Cybersecurity and Infrastructure Security Agency, to engage with multiple foreign governments targeted by Daxin, and assisted in its detection and remediation. So these guys have been on the front line with this thing. Symantec feels that this one will be worthy of multiple blog postings, but they provided sufficient meat for us in their first disclosure.

Daxin is a Windows kernel driver, which is a bit chilling due to the unfettered power that anything operating in the kernel has. And this makes it a relatively rare format for ransomware. It implements sophisticated communications functionality, which both provides a high degree of stealth and permits the attackers to communicate with infected computers on highly secured networks, including and specifically those where direct Internet connectivity is not available.

Its capabilities suggest that the attackers invested significant effort into developing communication techniques that can blend in unseen with normal network traffic on the target's network. Specifically, the malware avoids starting its own network services. Instead, it commandeers legitimate services already running on infected computers. It's capable of relaying its communications across a network of infected computers within the compromised organization network. The attackers can select an arbitrary path across infected computers and send a single command that instructs these computers to establish requested connectivity.

This use case has been optimized by Daxin's designers. By using network tunneling, attackers are able to communicate with legitimate services on the victim's network that can be reached from any infected computer. So it's essentially a backdoor allowing an attacker to perform various operations on the infected computer such as reading and writing arbitrary files. The attacker can also start arbitrary processes and interact with them. While the set of operations available to Daxin is quite narrow, this was deliberate because its real value lies in its stealth and its communications capabilities.

Okay. So listen to this. Daxin is capable of hijacking legitimate TCP connections on the fly. It does this by intercepting and monitoring all incoming TCP traffic for certain patterns. Whenever any of these patterns, in other words triggers, when any of these patterns are detected, Daxin disconnects the legitimate recipient of the traffic and takes over the connection. It has an entire TCP/IP stack that it brings along on its own. It then performs a custom key exchange with the remote peer, with both sides following complementary steps to perform that negotiation.

The malware can be both the initiator and the target of a key exchange. And as we know, the beauty of a public key exchange is that the exchange traffic need not be secret, and an eavesdropper cannot learn the key which is negotiated during the exchange. A successful key exchange opens an encrypted tunneled communications channel for receiving commands and sending responses. Daxin's use of hijacked TCP connections affords a high degree of stealth to its communications and helps to establish connectivity on networks with strict firewall rules.

So, for example, say that some organization maintains any sort of public-facing service, such as a website. Daxin would get into the web server, hook into its pre-encrypted and post-decrypted communications, monitoring them for a magic cookie. Once that magic cookie is seen, Daxin knows that the remote attacker, looking just like anyone else on the Internet, has connected and emitted what looks like any other web query. But at that point it takes over the connection from the web server, negotiates its own encryption key with the remote individual that sent the magic cookie, then switches to its own private encryption, tunneling its communications through the existing connection.

To anyone looking, this is just an HTTPS connection like a bazillion others. It's encrypted already, so this double layer of encryption goes unseen. But of course it's not like any other connection. Daxin's built-in functionality can be augmented by deploying additional components on the infected computer. It provides a dedicated communication mechanism for such components by implementing a device named \\.\, which is a standard Windows nomenclature for a device driver, TCP4. The malicious components can open this device to register themselves for communication. Each of the components can associate a 32-bit service identifier with the opened TCP4 handle. The remote attacker is then able to communicate through Daxin with selected components by specifying a matching service identified when sending messages of a certain type. The driver also includes a mechanism to send back responses.

So this basically, this sets up an entire private communications infrastructure. There are dedicated messages that encapsulate raw network packets to be transmitted via the local network adapter. Daxin then tracks network flows, such that any response packets are captured and forwarded to the remote attacker. In other words, it serves as a bidirectional network tap, allowing the attacker to establish and to tap communications with any legitimate services that are reachable from the infected machine on the victim's internal network.

One of Daxin's most powerful and unique capabilities is its ability to create multi-hop communications channels across multiple infected computers, where the list of nodes is provided by the attacker in a single command. For each node, the message includes all the details required to establish communication: the node's internal IP address, its TCP port number, and the credentials used during custom key exchange. When Daxin receives this message, it picks the next node from the list. Then it uses its own internal TCP/IP stack to connect to the TCP server listed in the selected entry.

Once connected, Daxin starts the initiator side protocol. If the peer computer is infected with Daxin, this results in opening a new encrypted communication channel. An updated copy of the original message is sent over this new channel, where the position of the next node to use is incremented. The process then repeats for the remaining nodes on the list. And if it sounds familiar, that's because it's a bit like onion routing, and it allows Daxin's influence to gradually permeate and penetrate deep inside a highly protected network environment as its remote attackers gradually map out and remotely explore the network environment they have infected.

The Symantec Threat Hunter team has identified Daxin deployments in government organizations, as well as entities in the telecommunications, transportation, and manufacturing sectors. Several of these victims were identified with the assistance of the Singapore-based PwC Threat Intelligence team.

While the most recent known attacks involving Daxin occurred in November of 2021, the earliest known sample of the Daxin malware dates from nine years ago in 2013. And that malware had already incorporated all of the advanced features seen in the most recent variants, with a large part of the codebase having already been fully developed at that time nine years ago. This suggests that the attackers were already well established by 2013, with Daxin's features reflecting their expertise back then at the time.

And Symantec believes that it goes back even further. An older piece of backdoor malware known as Zala (Z-A-L-A) contained a number of common structural features, but lacked many of Daxin's advanced capabilities. Daxin appears to build on Zala's networking techniques, reusing a significant amount of distinctive code, and even sharing certain magic constants with Zala. And they also share the use of a certain public library which is used to perform the kernel, the Windows kernel API hooking that is also common between variants of Daxin and Zala.

The extensive sharing and common codebase indicates that Daxin's developers at least had access to Zala's codebase. They believe that both malware families were used by the same Chinese-linked actor, which became active no later than 2009. So an additional four years before 2013, when Daxin was first seen. So needless to say, this is not something that anyone wants to have crawling around inside their networked machines. And you have to ask yourself, you know, how sure are you that you don't have it?

Leo: Not that it matters at all, but it's probably, because it's Chinese, Da Xin, Da being one word and Xin being pronounced "shin."

Steve: Ah.

Leo: But I think Daxin is fine. It's just that's probably Chinese, I would guess; right?

Steve: I bet it is, thank you.

Leo: Bet it is, yeah. If you said Da Xin, I would say, c'mon, man. C'mon, man. I know how to pronounce Log4j, that's for sure.

Steve: That's true.

Leo: Although some people say "log forge," which I think is wrong.

Steve: Oh. I think it is, too.

Leo: Yeah. Now, "Log Forge."

Steve: Yeah. So the chart, I've got a chart here in the show notes which begins on the 10th of December last year with a big spike. Then it drops a bit. And then it just sort of shows a nice mountain, you know, going up one side and coming down the other. There is a weird, very, very tall spike, kind of late in the game. It turns out that's been attributed to a one-day coordinated security scan by a security firm. So that's not attack traffic. But mostly what we see is a lot of initial interest. And then it kind of dies off.

Johannes Ullrich, the security guy over at SANS who documented this Log4j scanning activity, said: "Our sensors detected exploit attempts almost immediately," meaning with the original release of the Log4j news. And of course, as we know, Log4Shell exploit vulnerability scanning experienced a large cross-Internet spike in activity as threat actors around the world began searching the Internet frantically for Java apps that might have used the Log4j library and were testing the exploit to see what was vulnerable.

But with the advantage now that we have of hindsight, a couple months' worth, we can see that this spike in activity lasted for maybe about three weeks, until the end of 2021. So basically through December. And as we've noted here previously, one of the reasons for attackers losing interest in Log4Shell, the exploit of Log4j which was the library, was the complex nature of the Java ecosystem which resulted in the Log4j library being used and implemented in different ways across Java apps. That meant that a drop-in, dead simple universal exploit wasn't available; and that the first exploit that was published, which set off the gold rush scanning frenzy, turned out to be not so universal after all.

Bad guys had to reverse engineer individual Java apps, figure out how and where they were using Log4j, and then try different exploit variations to see what worked best. In other words, by definition not script kiddie-compatible. This entire process was and still is complex and time-consuming. And given that other new vulnerabilities - oh, look, there's something shiny - new vulnerabilities are being disclosed on a nearly daily basis, some of which are easier to exploit, Log4j was soon relegated to the "perhaps I'll get back to that eventually" category.

But that doesn't mean that these are not the droids you're looking for, so move along. The threat is still real. And if you look at that graph, the scanning has not gone down to zero. It's still kind of percolating along there. It's just very good news that the first example of an exploit was only the lowest of the hanging fruit. The fact that the rest of the fruit may well be, is still there but is well out of reach, casual reach, just means that it's moved into the longer term toolkit of the world's more serious actors.

So a case in point actually of exactly this was SentinelOne's recent report that one of Iran's state-sponsored groups was exploiting the Log4Shell vulnerability to compromise VMware Horizon servers recently. This was only a couple weeks ago. This confirms that not all attackers have lost interest in the vulnerability, and that the threat remains present for companies running unpatched Java systems.

So I just, you know, we talked about this having gone away. I just wanted to make sure that people understand this is going to be an issue. Unfortunately, it is beginning to take the shape of all these other problems that we have that never completely go away anymore. We understand from what Google's research showed that we talked about initially, that fixing the existing installed base of Java cannot possibly be an overnight event. Thanks to the complexity of new attacks, as an industry as a whole we did dodge a bullet. So we don't want to get cocky because the other thing that the chart shows, as I said, is that scanning has never returned back to zero. It's now holding constant.

Okay. As everyone knows, we're in the midst of an interesting time. Russia's President Putin has directed Russia's military to launch an attack against Ukraine. Many did not believe it would happen, and the world is still recovering from its shock. But happening it is. We're talking about this here because, as I mentioned briefly last week, the attacks have not only been conventional. They have also had a prominent and evolving cyber component. And while there has been a longstanding low-level cyber standoff between major global adversaries, and we'll talk a little bit more about that in a minute, we haven't yet experienced what any country might be able to unleash upon another.

And frankly, having listened to well-placed and well-informed experts in the U.S., everyone is a bit afraid to see what full-scale cyberwarfare might look like. We know what happens when you shoot a bullet, when you launch a missile, or roll a tank. Though the effects can be deadly, they are understandable and contained. But the presumption held by everyone is that all of the major powers have already deeply infiltrated each other's networks. And, I mean, that Daxin thing I was just talking about is a perfect example. It's like, that is just creepy, to think that something that powerful is scattered around. And apparently it is. And with this infiltration of each other's networks is the planting of cyber bombs and booby traps. And no one's really anxious to go first. So as I said, we're in the midst of an interesting time.

Last week, the day after I mentioned the previous Russian DDoS attacks against Ukraine's military and some of its banking infrastructure, additional DDoS attacks were launched against those same Ukrainian facilities. In response, the Ukrainian government issued a call to arms to local hackers, after which alleged "hacktivists" claimed credit for knocking the website of the Russian state-run news service RT News offline. And, you know, if we all just keep this at the level of DDoS attacks, we're going to be fine.

Then last Thursday Russian government websites went dark to some parts of the world after being targeted with a flood of web traffic via a DDoS attack which attempted to knock them offline. No one has claimed credit, so it's unclear who directed the attack, or if it was successful in disrupting sites. The reason there's some question about the success of the attacks is that the websites in question began claiming to be teapots. Now, presumably Russian teapots.

Leo: What?

Steve: We've talked about this before, but it's due for a refresher. A web server replying to an HTTP query begins its reply with a three-digit code to indicate the server's overall response to the query. Any response beginning with a "1" is informational. Responses beginning with a "2" indicate success; "3" indicates one of a number of redirection types, like temporary so don't remember the redirection, or this is permanent so please don't ask for that old URL again. Responses beginning with "4" represent a client error, that is, like asking for something bad; and "5" is a server error.

Within that framework, 200 is the most common code representing success, and everyone is familiar with the infamous "404 error," which means that the resource being requested was not found or does not exist on the server.

Back on April 1st of 1998, thus the traditional April Fools' Day, Internet RFC 2324 was offered to describe and standardize version 1.0 of the HTCPCP. HTCPCP is, of course, the lesser known Hyper Text Coffee Pot Control Protocol. In describing the rationale and scope of the proposed HTCPCP protocol, this RFC explains: "There is coffee all over the world. Increasingly, in a world in which computing is ubiquitous, the computists want to make coffee."

Leo: Sure they do, yeah, yeah.

Steve: Coffee brewing, yeah, I mean, it's self-evident; right?

Leo: Sure, yeah.

Steve: "Coffee brewing is an art, but the distributed intelligence of the web-connected world transcends art. Thus there is a strong, dark, rich requirement for a protocol designed 'espressoly' for the brewing of coffee."

Leo: Oh, dear.

Steve: "Coffee is brewed using coffee pots. Networked coffee pots require a control protocol if they are to be controlled. Thus this document specifies a Hyper Text Coffee Pot Control Protocol (HTCPCP), which permits the full request and responses necessary to control all devices capable of making these popular caffeinated hot beverages." Okay.

Leo: This is because of the Cambridge Coffee Pot; right? The very first webcam? Yeah.

Steve: Yeah. With that background, the protocol needed to provide for an error in the event that the Hyper Text Coffee Pot Control Protocol was inadvertently applied to a non-coffee pot.

Leo: Yes.

Steve: The HTTP response 418 beginning with the digit "4" which is universally used to signify an error made by the client was thereby defined to mean "I'm a teapot."

Leo: Oh, that's great.

Steve: And thus not the appropriate target of the HTCPCP protocol. The formal definition of error 418 says: "The 'HTTP 418 I'm a teapot' client error response code indicates that the server refuses to brew coffee because it is permanently a teapot."

Leo: This is an April Fools' Joke, but it's a real - but they implemented it.

Steve: Yes. Yes. "A combined coffee/tea pot, that is only temporarily out of coffee, would instead return an error 503." You know, just to be particular about that. And in fact, Leo, there was a move some years ago to remove the "418 I'm a Teapot" from Java. And as I recall, a 15 year old started, like created a website, like saved the 418 code.

Leo: Oh, man.

Steve: Which generated so much momentum that 418 was formalized and reserved, and the other languages that were considering pulling it all decided, nope, we're going to leave 418 in there.

Leo: If you go to Google.com/teapot, you actually get a 418. So even Google's in on this one.

Steve: So the upshot of all this is that some websites use, actually do use the 418 I'm a Teapot response for requests they do not wish to handle, such as automated queries. And that is the response that the mil.ru Russian website and a few others began returning last Thursday.

Leo: Wow.

Steve: This is believed to be a geofencing measure, denying any further processing of incoming IP traffic outside of Russia. So at least in this case the sites in question are not down due to DDoS, they are administratively blocked to non-Russian Internet traffic. And, clearly, you're a teapot.

Leo: Very funny. Very funny. Yeah, I saw a number of people initially saying, oh, these sites are down, these sites are down; and others saying use a VPN exiting in Russia, and they're still up. It's just geofencing.

Steve: Yup. We can't get to them. And in fact we did talk at some length some time ago about Russia having deliberately experimented, and it was a little spooky at the time, with their own DNS system, rather than relying on the...

Leo: Oh, they have their own GPS, GLONASS.

Steve: Right. Well, but normally DNS relies on the global root servers, and DNS won't work without the root servers. Well, they've implemented an internal DNS facility that would allow Russia to go off the Internet and become a self-sustaining, you know, it wouldn't all just go down and dark for everybody. I mean, without connectivity to the rest of the world you have a rather proscribed Internet. But still it would be up and running. And vital services and things presumably would be using Russki DNS as opposed to regular DNS.

So on the question "Will the U.S. Attack?," last Thursday NBC News headlined their story "Biden has been presented with options for massive cyberattacks against Russia" with the subhead "The options presented include disrupting the Internet across Russia, shutting off power, and stopping trains in their tracks." Whereupon the White House immediately denied the report that President Biden has been presented with an arsenal of ways to launch massive cyberattacks against Russia, attacks which would be designed to disrupt Russia's ability to sustain its military operations with Ukraine.

Within hours of the report, Press Secretary Jen Psaki said in a tweet that NBC got it wrong. She tweeted: "This report on cyber options being presented to @POTUS is off base and does not reflect what is actually being discussed in any shape or form," she said. However, NBC subsequently made clear that they felt their reporting was quite well sourced, accurate, and noted that the White House had not indicated what about the story was incorrect.

NBC's sources, which were cited as "two U.S. intelligence officials, one Western intelligence official, and another person briefed on the matter," told NBC that no final decisions had been made as of earlier Thursday. One of those sources said the possibilities range from the aggravating to the destructive: "You could do everything from slow the trains down to have them fall off the tracks," said the source, who'd been briefed on the matter. In other reporting there were passing references to messing with train track switches. So I think that gives us some further information.

But that source also said that most of the potential measures on the slate of possible cyberattacks -a slate again Press Secretary Psaki said was inaccurate - would not be destructive but would rather be designed to be disruptive, hence falling short of an act of war by the United States against Russia, according to NBC. To which I say wow to all that.

One of the problems with that sort of interference is that it would almost have to leave footprints. If President Biden were being provided with these options, he would also certainly be told that this would inevitably be tipping our hand to Russia about at least the cyber capabilities that were used because they would be exposed. If the U.S. is really able to shut down Russian rail transportation, or run their trains off the rails, that's a significant capability that one would not want to waste since it could probably never be used again. So the question is, is this the time to use and inevitably expose such a capability, assuming that it could not be used again?

And it would definitely aggravate Russia and almost certainly trigger a response. The main trouble, as I've noted before, is that all sides are keenly aware that no one is able to withstand a cyberattack. Today, everyone depends crucially upon their networks, yet everyone's networks are a true mess. Everyone has instances of crappy software still running which was written long before security was a consideration.

And as we've seen, and as this podcast chronicles ad nauseam, even today, when network security is given extensive lip service, we're still unable to get it right. In a few minutes we're going to be talking about a catastrophic mistake Samsung recently made, or made actually a while ago and had continued, which cripples the security of 100 million of their Android handsets. We still don't have the structures in place to get security right today. The only good news is we're not alone. No one else does either, and we're all using the same crappy software. As a consequence, no one dares to pull that trigger.

Okay. Last Wednesday I was working away on SpinRite. I had Windows Weekly running in the background, as I often do. Leo, you, Paul, and Mary Jo were talking as you guys always do about what's on your minds regarding Microsoft, Windows, the Enterprise, and often the industry at large.

BEGIN CLIP WINDOWS WEEKLY 765

PAUL: Yeah, so today, in case you thought things hadn't changed, Microsoft arbitrarily - and by the way, we know for a fact arbitrarily decided on Windows 11's hardware requirements. Right? And they did it to help the PC industry.

Leo: So all the TPM 2.0 stuff, the 8th Generation Intel...

PAUL: 8th Gen, exactly. Completely arbitrary.

Leo: It runs fine on the earlier stuff.

MARY JO: So, you know, it's very interesting to me they are not blocking the people from running it; right? Like there are so many who do it.

PAUL: But you know what, though, if they did, I think there would have been a lawsuit and a class action lawsuit. And one of those things that would have come out is what I just said, that those hardware restrictions are completely arbitrary, and that people who run Windows 11 on non-supported hardware are in fact not endangering the community. They're not, you know, there's nothing bad going on at all, and that Microsoft just did this arbitrarily or artificially, I should say, to help drive PC sales.

MARY JO: Yes.

END CLIP WINDOWS WEEKLY 765

Steve: Okay. Now, I mean, so I stopped coding SpinRite. It's like, my mouth hung open. For Paul to just say so matter-of-factly, "We know it for a fact." Now this, of course, well, not that we know it for a fact, but our listeners, this podcast's listeners have been putting up with me ranting about this from the first moment that this issue arose because from a software engineering, computer science standpoint, which is the only thing I care about and the only thing that matters, whether Microsoft wants it to be true or not, it had to be that Microsoft was deliberately and calculatingly lying to the entire world.

What Microsoft was saying about their "uncertainty" over which systems Windows 11 could be safely run on simply could not be true. It was not possible. It wasn't computer science. It had to be management-driven, marketing-inspired gibberish. Recall that Security Now #835 was titled "TPM v1.2 vs 2.0." During that podcast, because this thing just was driving me crazy, we looked feature by feature through the differences between the two, searching for any rational computer science justification for Microsoft to require Windows 11 to have TPM 2.0 over 1.2 while Windows 10 was working quite happily with version 1.2. We found none. Not one. What we found was that Microsoft was already using TPM 1.2 for everything they wanted to protect with TPM 2.0.

Now, it may seem odd, but I would have no problem at all if Microsoft had just told the truth. If they had said that they don't want Windows 11 to run on older machines because they want to require people to purchase new hardware when they upgrade their software, I couldn't have cared less. That would have been fine with me because it would have been the truth. Of course, it would have infuriated everyone else.

Now, I don't know what I will do on October 14th of 2025 when support might be ending for Windows 10. I'll have a large stable of machines, as will many people, that would happily run Windows 11 just fine. And by then we can hope that all of the nonsense changes they've been making to Windows 11 will have settled down. By that time we'll all be able to upgrade our machines that don't currently qualify to run Windows 11. Will we be able to upgrade them? Will Microsoft decide, okay, gee, guess what, surprise, it works just fine? It's difficult to imagine not. And there's just no other way to look at this other than Microsoft having really messed this thing up. This is just sad. But anyway, it was very refreshing, Leo, to hear the gang over on Windows Weekly say yeah.

Leo: Yeah, yeah, yeah,

Steve: Everyone knows that.

Leo: Yeah.

Steve: I was like, oh. Well, that's good because it had to be true. Wow. I had a piece of closing-the-loop feedback. Someone sent me: "Hi, Steve. How can I get the AS number of my bank? NSLOOKUP doesn't do it. I want to check before I enter my password that the ASN hasn't changed. A little bit of personal self-help to avoid a BGP attack on my money."

And that was a good question. Of course, he's talking about last week's topic which was the BGP attack where some very clever guys managed to reroute some traffic to a different location. And it does, it sort of begs the question, what could someone do? Okay. And the answer is, first of all, just to clarify, individual entities like a bank or like GRC or like TWiT do not have AS numbers.

I actually regret not getting one for myself back like when I first got my GRC Internet connectivity because I could have back then. There were plenty of IPv4 addresses. Nobody thought they were going to expire. You have to have at least a Class C network in order to be an AS. So had I obtained a Class C network, I would own my own block, something something something dot something something something dot something something dot star. I would own a Class C network that would be mine. I would be an AS. And so when I went from Verio over to Level 3, I would have transported that IP space with me. And the routing tables would have changed so that the packets bound for my block of IP addresses now went to Level 3, where they used to go to Verio. My IPs would have never changed.

Instead, I'm not an AS. That never occurred to me. I didn't think of it. I've got a block of IPs. And when I did move to Level 3, they all had to change. DNS had to propagate some. It's not the end of the world, but it's just not quite as cool. But the point is individual entities don't have them. Networking companies do. So what would change would be the certificate.

And that's where GRC's certificate fingerprinting comes in. What had to happen for that BGP attack to get pulled off as it was, was the moment the IP addresses were rerouted, they applied with an entity supporting the ACME automation API, like, for example, Let's Encrypt. They applied for a certificate for the domain they were hijacking. And at that point all of the traffic was coming to them. So Let's Encrypt was able to verify that they were the apparent owner of that domain and would instantly give them a certificate.

But it wasn't the certificate of the bank; right? It wasn't the actual certificate for whatever it was, developers dot something dot something. It was a different certificate that had just been minted for that because the actual owner of that domain is protecting their certificate as one of their most prized possessions.

So to answer this listener's question, if you were really concerned, you would look at the certificate of those few entities where you absolutely want to make sure you never have a connection spoofed, and look at the so-called thumbprint of the certificate, which it must change if a different certificate is issued. It does change, even like for me, every couple years, now every year for all of us. When our certs change, we get a different thumbprint. The certificate, even if it's issued from the same CA, it will have a different timestamp. It'll have, I mean, it is a different key. So everything about it is different. The hash will be different. The thumbprint will be different. So that would set off a false positive when that domain rekeys. So it's a little less useful to do that.

But anyway, I thought that was sort of an interesting question. How would you protect yourself against being the victim on the user end of a BGP attack? And it would be that the authentic certificate could not be the one you're connecting to. And so that's what to check.

Leo: Good to know.

Steve: I'll just mention before we take a break that SpinRite's 9th development pre-release went out last week, as I expected it would, and I'm pleased to report that it was a significant win. Reports of it successfully working across a number of systems where previous releases had not were posted to confirm that the things I believed were fixed were indeed fixed. After that we found and I fixed a problem with the BIOS on a Supermicro motherboard, and we also found and I fixed a new bug that I had introduced with the new mini-benchmark which directs SpinRite to use the firmware for its bulk scanning when the system is somehow faster than my own hardware drivers. We've seen that a little bit. It's not a huge win, but we want SpinRite to be as fast as possible. And if that means using an existing adapter's own firmware, so be it.

At this moment, because I just hadn't had the chance to dig into it, we do have a couple odd problems that have been found, so I'll be digging into that this evening. But it feels as though we're getting very close to having these last few bits resolved, with SpinRite able to operate robustly on every piece of hardware anyone has tested, which is around 300 people, generally each with multiple machines.

And I have to admit I'd been holding my breath on this one since the immediately previous release was December 23rd. And since I had re-engineered SpinRite's hardware interrupt handling, made many other improvements, introduced the mini-benchmark and the awareness of the possibility of system firmware being faster, everything was working on my systems, but none of it had been widely tested by our development community. So today I'm feeling quite relieved to have our testing gang synchronized and up to date with SpinRite's latest code, which has now largely been proven.

So as I said, I'll be back to it this evening, get the few remaining problems fixed. And at that point, SpinRite's all-new hardware interface platform will be in place, not only for 6.1, but also for 7 and beyond. And it really is the case that now, or at that point, most of the hard work will be behind us. And it won't be that much longer before it will push past the point where testing stops now and SpinRite will actually be running out on the surfaces of these drives. So, whew. What a project. But worthwhile.

Leo: Working hard. I love it. Now, Truth Dies in Darkness.

Steve: Yeah. The academic research paper written by a trio of researchers at the University of Tel Aviv will be presented during the upcoming USENIX Security 2022 symposium. That's this summer. Their paper is titled "Trust Dies in Darkness: Shedding Light on Samsung's TrustZone Keymaster Design."

The paper is extremely interesting due to the specific mistakes it finds and highlights and describes in some detail. But it carries, and this is what I found so refreshing, it makes very explicit a far more important and overriding message that this podcast's listeners have been hearing from me for years, which is that important, high-volume, widely used, critical technologies cannot remain proprietary and secret. They simply can't. As a global society we must change the way we think about these things. Somewhere around 100 million Samsung Android smartphone users cannot be allowed to carry critical security technology that no one else has ever been allowed to examine. But that's the world we have today.

My favorite example that we've frequently discussed here is voting machines. How is it conceivable that closed, proprietary, inherently secret, electronic voting machines were ever allowed to be deployed and used within the United States? How? The U.S. could have easily used its buying power to require the use of an open design or fully independent public examination and auditing of these machines. The idea that the value added is not the proprietary hocus pocus firmware, you know, we've got that wrong. The value-added should be the hardware implementation which uses generic, widely vetted, and completely transparent software. But that's not what we do.

Okay, as for today's latest example of how this fallacy of secrecy has landed the very well meaning, but misguided, Samsung in hot water, in the abstract of their 20-page paper, the researchers explain. They say: "ARM-based Android smartphones rely on the TrustZone hardware support for a Trusted Execution Environment (TEE) to implement security-sensitive functions. The TEE runs a separate, isolated TrustZone Operating System (TZOS), in parallel to Android. The implementation of the cryptographic functions within the TZOS is left to the device vendors, who create proprietary undocumented designs." Which turns out to be these guys' hobby horse just as much as it is mine.

They said: "In this work, we expose the cryptographic design and implementation of Android's hardware-backed Keystore in Samsung's Galaxy S8, S9, S10, S20, and S21 flagship devices. We reverse engineered and provide a detailed description of the cryptographic design and code structure, and we unveil severe design flaws. We present an Initialization Vector (IV) reuse attack on AES-GCM that allows an attacker to extract hardware-protected key material, and a downgrade attack that makes even the latest Samsung devices vulnerable to the Initialization Vector reuse attack. We demonstrate working key extraction attacks on the latest devices.

"We also show the implications of our attacks on two higher level cryptographic protocols between the TrustZone and a remote server. We demonstrate a working FIDO2 WebAuthn login bypass and a compromise of Google's Secure Key Import. We discuss multiple flaws in the design flow of TrustZone-based protocols. Although our specific attacks only apply to the around 100 million devices made by Samsung, it raises the much more general requirement for open and proven standards for critical cryptographic and security designs."

Okay. So a couple of things. ARM's Trusted Execution Environment (TEE) is their version of Apple's Secure Enclave, or the PC's TPM, or what Intel refers to as their Trusted Execution Technology (TXT). In every case, this notion of sequestering secrets and secret operations is a tacit acknowledgement of our generic inability to secure any larger attack surface. There's no actual computer science reason why the entire computer system cannot be secure. Except that we've tried over and over and over, until we finally just gave up, recognized and accepted the reality that as an industry we don't know how to do that yet. We cannot secure our own systems. It's beyond us.

So all these various enclaves of whatever name they take are an attempt at encapsulation. The thinking is, if we can just separate the system's most critical and crucial operating secrets, and the limited set of operations we need to perform with those secrets, from the rest of the wild and woolly and totally beyond our control system, then at least we can keep those secrets secret.

What these guys found, at tremendous expense to themselves because this was all locked tightly away and considered to be proprietary, and they've demonstrated, is that around 100 million of Samsung's Android-based smartphones contain a fundamental and horrifyingly simple and obvious in retrospect design flaw which can be readily exploited to completely compromise any Samsung smartphone's most tightly held secrets.

We talked a long time ago about what's known as "authenticated encryption." The idea is that if you want to secure a secret, you need two things. You need to encrypt it for secrecy, and you also need to somehow detect any tampering with the secret. So you need both encryption of the plaintext and cryptographic authentication that the message has not been modified. Our longest time listeners will recall the industry's wrestling back and forth with the question, should you encrypt first, then apply authentication to the encrypted result? Or should you authenticate the plaintext first, and then encrypt that whole authenticated result? Does it matter? Many early systems didn't think it did. So they tossed a coin or perhaps did what was easiest. But either way, they often got it backwards.

There's only one right answer. The only right answer is that when both encrypting and authenticating, you always encrypt first and apply authentication last. The reason is the first thing you want to do upon receiving the result for decryption is to authenticate that the received package has not been tampered with. You never decrypt first, then authenticate. And if anyone's wondering how we got to 860 episodes of this podcast, and why my hair is no longer black, it's because we spent a lot of time back in the old days closely examining those questions and the security doing it wrong could compromise.

SQRL, as I mentioned at the top of the show, needed authenticated encryption to protect its user's key material and the client's settings. Windows is a nonsecure operating system, and the SQRL identity that's stored in Windows is just a file on the file system, with no protection. So SQRL needed to protect that file. And Samsung was in the same situation. They have so-called "blobs" that are on the file system that is this keying material.

So like AMD and Samsung, AES-GCM Advanced Encryption Standard Galois Counter Mode is the authenticated encryption algorithm I chose for SQRL. So did Samsung. And as I mentioned, I wrote my own implementation of it in C since I wanted to, and did, open source it. AES-GCM is elegant because it simultaneously encrypts and authenticates as it goes. It will absolutely positively not decrypt and authenticate anything that's been altered after it was encrypted with this AES-GCM because it's both. And that's what I needed for SQRL. AES-GCM is, however, a bit brittle.

Coincidentally, we were talking recently about brittle encryption and mistakes being made in cryptographic implementations. Last week was how the guys who wrote the Hive ransomware got their encryption wrong. There are many instances in cryptography where something must only be used exactly once. Of course the famous one-time pad has its uses restriction in its name. Last week's XOR keystream reuse was what hit and bit the Hive guys.

And AES-GCM requires the use of a so-called Initialization Vector, the IV. The IV can be anything. And it's never a secret because it's required to reverse the process. So it's always out in full view. But the crucial requirement that makes AES-GCM somewhat brittle, that means you need to use it with care, with full awareness of what could go wrong, is that much as with an XOR keystream, AES-GCM's Initialization Vector must never be reused.

I don't have it in the show notes, but in Samsung's API, they allow the caller of the API to provide the initialization vector. And that allows you to call the API multiple times, providing the same initialization vector, thus forcing reuse against every rule of AES-GCM. And so without breaking Samsung's rules because Samsung allowed the caller of their API to break the rules, it is possible to get this trusted execution environment to do things always with a fixed IV, and that results in well-known leakage problems for AES-GCM.

So now you can guess and understand the crucial and obvious mistake that Samsung's engineers made. The Tel Aviv researchers wrote: "ARM is the most widely used processor in the mobile and embedded markets, and it provides TEE hardware support with the ARM TrustZone. TrustZone separates the device into two execution environments." And again, as I said, it's because we have to. We've just given up trying to make Android or Windows or anything else truly secure. So let's just, you know, let's create a little enclave where we can be absolutely sure we do it right. Unfortunately, Samsung did create an enclave. They just didn't do it right at the enclave boundary.

So these two execution environments for ARM: A non-secure REE (Rich Execution Environment), in other words Android, where the "Normal World" operating system runs; and a secure TEE (Trusted Execution Environment) where the "Secure World" operating system runs. The REE and the TEE use separate resources - separate memory, separate peripherals - and the hardware enforces the protection of the Secure World, they wrote.

They said: "In most mobile devices, the Android OS runs the nonsecure Normal World. As for the Secure World, there are more choices. Even among Samsung devices, there are at least three different TrustZone Operating Systems (TZOSes) in use." They said: "The Android Keystore provides hardware-backed cryptographic key management services through a Hardware Abstraction Layer that vendors such as Samsung implement. The Keystore exposes an API to Android applications, including cryptographic key generation, secure key storage, and key usage," in other words, encryption and signing.

"Samsung implements the HAL through a Trusted Application called the Keymaster TA" - or Keymaster Trusted Application - "which runs in the TrustZone. That Keymaster TA performs the cryptographic operations in the Secure World using hardware peripherals, including a cryptographic engine." So cool; you know? High tech, hardware enforced, everything you could possibly want.

"The Keymaster TA's secure key storage uses blobs." And that's what I was talking about, blobs out in the normal file system. They say: "These are 'wrapped' encrypted keys that are stored on the REE's" - Rich Execution Environment, in other words Android - "file system. The wrapping, unwrapping, and usage of the keys are done inside the Keymaster TA using a device-unique hardware AES key." So per-device hardware never exposed to the outside world. "Only the Keymaster TA should have access to the secret key material. The Normal World should only see opaque key blobs.

"Although it is critical to rigorously verify and test such cryptographic designs, real-world TrustZone implementations receive relatively little attention in the literature." They wrote: "We believe that this is mainly due to the fact that most device vendors do not provide detailed documentation of their TZOS and proprietary TAs" - like Keymaster - "and share little to no information regarding how the sensitive data is protected.

"To advance and motivate this research area, we decided to use the leading Android vendor Samsung as a test case. We reverse engineered the full cryptographic design and API of several generations of Samsung's Keymaster TA and asked the following questions: Does the hardware-based key protection of cryptographic keys remain secure even when the Normal World is compromised? How does the cryptographic design of this protection affect the security of various protocols that rely on its security?"

Okay. And we know from the abstract at the top of their paper that their reverse engineering efforts paid off handsomely. They were also, of course, responsible for their pre-disclosure to Samsung. In that they wrote: "We reported our IV reuse attack on S9 to Samsung Mobile Security in May of 2021. Then three months later, in August 2021, Samsung assigned CVE-2021-25444 with high severity to the issue, and released a patch that prevents malicious IV reuse by removing the option to add a custom initialization vector from the API. According to Samsung, the list of patched devices includes the S9, the J3 Top, J7 Top, J7 Duo, TabS4, Tab-A-S-Lite, A6 Plus, and A9S.

"Also we reported the downgrade attack on S10, S20, S21 in July of 2021. Then three months later, in October 2021, Samsung assigned CVE-2021-25490 with high severity to the downgrade attack and patched models that were sold with Android P OS or later, including the S10, the S20, and S21. The patch completely removes the legacy key blob implementation."

And finally, in their conclusion, they scolded much as I have, writing: "Vendors including Samsung and Qualcomm maintain secrecy around their implementation and design of TZOSes and TAs. As we have shown, there are dangerous pitfalls when dealing with cryptographic systems. The design and implementation details should be well audited and reviewed by independent researchers and should not rely on the difficulty of reverse engineering proprietary systems."

In other words, this is classic Crypto 101. As we know, the great breakthrough in cryptography came when we switched away from using proprietary and secret non-keyed encryption algorithms that just scrambled the bits in some magic way to using public and publicly scrutinized keyed cryptographic algorithms, making the algorithm public and the keys secret. This same breakthrough principle must also be applied more deeply to the wider implementation and use of these now public algorithms. Their implementation needs to be public just the same as the algorithm itself, that is, what you do with the algorithm obviously matters. It's clear that the precise way these algorithms are used is every bit as important to the security of the entire system. And thus the headline of their paper: "Trust Dies in Darkness." And as an industry we need to shine a light.

So I thought it was interesting that these guys are coming to the same conclusion, which is sure, you've got, you know, here's Samsung trading on beautiful, academically designed, seriously vetted algorithms, and then keeping what they do with them secret. All these academics had to do was look at the fact that the AES-GCM was the algorithm being used, and maybe that wasn't obvious. Maybe Samsung never says it. They had to figure that out.

And once they did that, they realized, oh, my god, Samsung is letting users specify the initialization vector. Why? That should be designed, it should be obtained pseudorandomly, or maybe pure randomly because there's probably a high-quality hardware random number generator as part of this trusted execution environment. So get a really long, completely unpredictable, never going to happen twice, initialization vector and have the TEE export it along with the result so that it can be provided later, rather than having the user provide it. It's nuts. You would never do that with AES-GCM, and Samsung did.

Leo: Is this attack, can it be done through an app or over the air?

Steve: Yes.

Leo: It can.

Steve: Yes. All it would take would be to run an app which uses the trusted execution environment to exercise the keys for its own purposes. So apps do absolutely have access to the security API in Android. And so this would allow them to provide initialization vectors which would eventually leak the keying material that the whole point of this whole thing is to keep secret. And it's a little worrisome that this problem has been around for so long, and they're only patching from P OS and later on S10, S20, and S21. So once again we're in this problem of older Samsung devices that are no longer being maintained and no longer getting firmware updates, having what is now a well-known, fully documented, critical security flaw.

Leo: So, yeah. The S22 is not vulnerable, apparently, the newest Samsungs.

Steve: Yes. They do stop at 21.

Leo: Yeah. Android Pie is three years old, Android 9. And so there probably are a few older Samsungs that you can't update. What should owners of, say, a Galaxy S4, 5, 6, 7, what should they do? Should they be very careful about what they download, I guess, yeah?

Steve: Yeah, exactly. It would be something that you get into your computer that would, I mean, this is classic state actor or classic NSS Pegasus.

Leo: Oh, NSO Group.

Steve: NSO, exactly.

Leo: So, okay. So Grayson's mom, not being a dissident or journalist or opposition candidate, probably doesn't have to worry about the NSO Group.

Steve: No. What this does not do is simply dump the key. You've got to have something running, working, like doing encryption over and over and over and finding the biases in the...

Leo: So this is a nontrivial exploit, then.

Steve: Yes.

Leo: Okay.

Steve: It is a nontrivial exploit.

Leo: So highly unlikely, in other words, as an average or normal user. I mean, remember, phones weren't routinely encrypted until relatively recently. I mean, I remember when you had to turn it on. I think iPhones might have always been.

Steve: Remember that it used to slow things down.

Leo: Yeah, you didn't want to turn it on.

Steve: So be like, oh, yeah, don't turn that on. It'll slow down your phone.

Leo: So this is no worse than that. In fact, it's considerably better than that because if all your data's unencrypted, all a malicious actor would have to do is read the drive.

Steve: I guess the only thing that I would say is that once upon a time we weren't doing things that assumed encryption.

Leo: That's true.

Steve: If we didn't have it.

Leo: Right.

Steve: And so we've gone so much further now. We've got bitcoin wallets. People have a bitcoin wallet in their phone. And they're presuming that that stuff is going to be kept encrypted.

Leo: Well, and that's a good point. I mean, that is something that somebody might well be highly motivated to get to, yeah.

Steve: To get, yup.

Leo: So that's, I mean, still requires some real sophistication, though.

Steve: The analogy would be that this provides a way to mine the phone's key. It would be key mining.

Leo: Key mining, yeah.

Steve: Is what this allows.

Leo: All right. So Grayson's mom has got it going on. She's okay. All right. He was worried. Grayson's a regular listener to the show. So good news. He's a good son, that Grayson. So is Steve Gibson. Who is Gib? Whose son are you? Is a guy named Gib?

Steve: Oh, Leo, he's long gone. And I think he - hopefully he went to heaven because he was certainly not programming in assembly language.

Leo: You've spent your time in hell already. That is the man, the myth, the legend, Steve Gibson, still writing in x86 ASM. In fact, that's how he wrote SpinRite, the world's finest mass storage recovery and maintenance utility. 6.0 is available at GRC.com, his website. If you buy it today, you'll get an upgrade free to 6.1, and you can also participate in the development.

While you're there, you can also get a copy of this show. There's 16Kb audio. These are the two unique versions of the show that he has, the 16Kb for the bandwidth-impaired and really nice transcripts written by Elaine Farris. She actually is a court transcriptionist, so she's writing it all down and putting it in a transcript. All of that's at GRC.com. While you're there, browse around. There's all sorts of other stuff besides SpinRite. It's a really fun site to take a look at.

You can also get a copy of the show at our website, TWiT.tv/sn. Download audio or video from our site. That's our unique version is the video. There's also a YouTube channel dedicated to it. You could subscribe in your favorite podcast player.

We are going to be back here at our regular time, 1:30 Pacific, 4:30 Eastern, 21:30 UTC, next Tuesday and every Tuesday. Patch Tuesday this week, this coming week. So join us if you want to want to watch the show live, live.twit.tv. Chat live at irc.twit.tv or inside the Club TWiT Discord. Thank you, Steve. Have a great week. See you next time.

Steve: Will do, my friend. Stay cool, and see you on the 8th.