Security Now! #860 - 03-01-22
Trust Dies in Darkness

This week on Security Now!

This week we examine the consequences of paying ransomware extortion demand: How did that work out for ya? 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 cyber attacks 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.

Says here you should be in hell, but since you coded in Assembly We'll count it as time served

Security News

Honor among thieves?

One of the things any victim of a ransom extortion wonders is whether the bad guys, who hold all of 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 conducted by the cybersecurity firm Venafi which highlighted the unsurprising lack of trustworthiness of ransomware actors. The report found that in most cases, once the initial ransom is paid, the extortion 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. So they are certainly in the enterprise security space.

Their survey revealed that:

- 18% (so about 1 in 5) of victims who paid the ransom still had their data exposed on the dark web.
- 8% refused to pay ransom and the attackers uses every available extortion opportunity.
- 35% of victims paid the ransom, but nonetheless were still unable to retrieve their data.

As for the ransomware actor extortion tactics, these are summarized as follows:

- 83% of all successful ransomware attacks featured double and triple extortion.
- (1st extortion is over data encryption, 2nd extortion is over public release of confidential or trade secret data, 3rd extortion is over DDoS.)
- 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.
- 32% of attacks threatened to directly inform the victim's customers of the data breach incident.

Venafi explained that the lack of credibility of ransomware actors' promises to their victims stems from several factors.

First, most RaaS 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 have witnessed a continual churn of RaaS names with the suspicion that new actors on the scene are relabeled old actors.

Secondly, in RaaS, 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 these groups.

Thirdly, 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 cyber-crime community sooner or later.

As Venafi's report underlines, paying the ransom is only motivating crooks to return for more, as it sends the signal that the victim sees this as the easiest way out of trouble, which is nothing but 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 customers.”*

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

The only way that would work would be that the initial ransom is for BOTH data decryption and a promise to destroy exfiltrated data. Then once that money is in hand, and the decryption keys have been provided, a second demand is made for the second half — the promise not to exfiltrate. The logic must be that getting an additional payment from a victim who has already paid once is easier. They've proven the payment channel, they've already invested something in ransom... So what's just a bit more to keep their data out of the hands of either the dark web or the public.

And, really, why would such people NOT turn right around, either way, even if the victim pays up, and resell the data again on the dark web? It seems clear that's what would be done. For a while there was a working theory that the ransomware gangs were, in fact, concerned about their reputation, since it would tend to increase the likelihood of the ransom being paid. But that was before the secondary incentive of data exfiltration and extortion to prevent its subsequent release and/or sale had appeared, 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, it’s the separate individual affiliates who obtain the pre-encrypted exfiltrated data. Therefore, the RaaS operators have no say or control over the possibility of secondary extortion being perpetrated by the independent affiliate.

And finally, whereas ransomware was a little known novelty with only a few perpetrators, it has grown into a well known and recognized form of cybercrime with too many villains to count. His has dramatically changed the calculus about any one gang’s reputation. It 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. Restore systems and data from backups, and immediately alert the law enforcement authorities of the incident.

To do otherwise will nearly always be futile because all other scenarios eventually lead to the same result. The only difference being the depletion of the victim’s funds, the enrichment of ransomware cybercriminals, and the enhancement of their motivation to continue.

Daxin
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. Their post stated that the malware appears to be used in a long-running espionage campaign against select governments and critical infrastructure targets.

"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 2021 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 commonly seen and associated with Chinese espionage actors were also 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 into a target's network and exfiltrate data without raising suspicions.

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 with Daxin and assisted in its detection and remediation. 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 unfettered power that anything operating in the kernel has. And this makes it a relatively rare format for malware. 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 is 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 communications capabilities.

Daxin is capable of hijacking legitimate TCP/IP connections. It does so by intercepting and monitoring all incoming TCP traffic for certain patterns. Whenever any of these patterns are detected, Daxin disconnects the legitimate recipient and takes over the connection. It then performs a custom key exchange with the remote peer with both sides following complementary steps. The malware can be both the initiator and the target of a key exchange. 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 communication 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 an encryption key, then switches to its own private encryption. To anyone looking, this is just an HTTPS connection like a bazillion others. But of course it's not.

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 "\\.\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 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.

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 bi-dirrectional network tap allowing the attacker to establish communication 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 to use 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 then 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. It's a little bit like Onion Routing and it allows Daxin's influence to gradually penetrate deep inside a highly protected network environment as its remote attackers gradually map out and remotely explore the network environment they have infected.

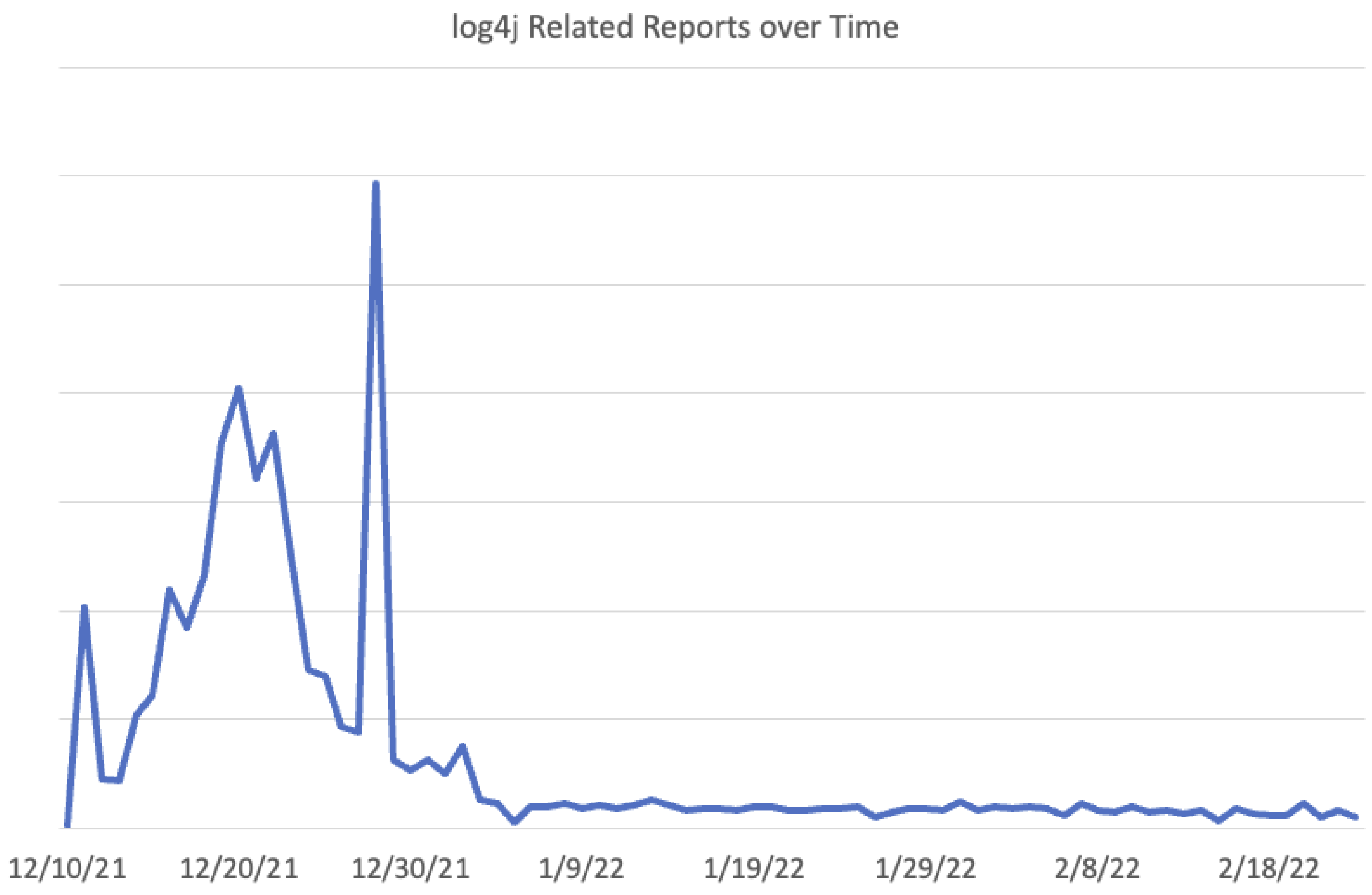
The Symantec Threat Hunter team has identified Daxin deployments in government organizations as well as entities in the telecommunications, transportation, and manufacturing sectors. Several of these victims were identified with the assistance of the Singapore-based PwC Threat Intelligence team. While 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. This suggests that the attackers were already well established by 2013, with Daxin features reflecting their expertise at that time.

And Symantec believes that it goes back even earlier. An older piece of backdoor malware known as Zala 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. And they also share the use of a certain public library which is used to perform kernel API hooking that is also common between variants of Daxin and Zala. The extensive sharing and common codebase indicates that Daxin designers 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.

Needless to say, this is not something that anyone wants to have crawling around inside their networked machines.

Whither or Wither: Log4j / Log4Shell



That large late spike in activity on 12/28 has been attributed to a one-day coordinated scan using IPs in 165.232.72/24 and 165.232.76/24, part of Digitalocean's network.

The chart of Log4j scanning activity shown above, as viewed from a network of well-placed honeypots, reveals that the initial excitement has died way down, but that it hasn't died off completely.

Johannes Ullrich, ICS SANS fellow who documented this Log4j scanning activity said: “Our sensors detected exploit attempts almost immediately.” referring to the explosion of early first reports of the problem. And indeed, Log4Shell exploit vulnerability scanning experienced a large cross-Internet spike in activity as global threat actors began searching the Internet for Java apps that might have used the Log4j library and were testing the exploit to see what was vulnerable.

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

Bad guys had to reverse-engineer individual Java apps, figure out how and where they were using Log4j, then try different exploit variations and 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 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 are looking for, so move along. The threat is still real so 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 that fruit may be well out of casual reach only means that Log4j has moved into the exploit toolkit of the world's more serious actors.

And a case in point was SentinelOne's recent report that one of Iran’s state-sponsored groups was exploiting the Log4Shell vulnerability to compromise VMWare Horizon servers. This confirms that not all attackers have lost interest in the vulnerability and the threat remains present for companies running unpatched Java systems.

We understand from what Google's research showed, that fixing the existing installed base of JAVA cannot possibly be an overnight event. Thanks to the complexity of new attacks, as an industry we dodged a bullet. Let's not get cocky because the other thing that charts shows is that scanning has never returned to zero and it’s now holding constant.

“418 I’m a teapot”

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's 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, 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 US, everyone is a bit afraid to see what full scale cyber warfare might look like. We know what happens when you shoot a bullet, 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 have planted cyber bombs and booby traps. No one's really that 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 these 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. 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 the sites. The reason there's some question about the success of the attacks is that the websites in question began claiming to be teapots. Presumably Russian teapots.

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 this or permanent so please don't ask again. '4' is a client error 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.

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 HyperText Coffee Pot Control Protocol. https://datatracker.ietf.org/doc/html/rfc2324

In describing the rationale and scope of the proposed HTCPCP protocol, the RFC explains:

There is coffee all over the world. Increasingly, in a world in which computing is ubiquitous, the computists want to make coffee. 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. Coffee is brewed using coffee pots. Networked coffee pots require a control protocol if they are to be controlled. Thus, this document specifies a HyperText 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.

With that background, the protocol needed to provide for an error in the event that the HyperText Coffee Pot Control Protocol was inadvertently applied to a non-Coffee Pot.

The HTTP response 418 — beginning with digit '4' which is universally used to signify an error — was thereby defined to mean "I'm a teapot" 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. A combined coffee/tea pot, that is only temporarily out of coffee, should instead return an error 503.

The upshot of all this is that some websites 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 others began returning last Thursday. This is believed to be a geofencing measure, denying any further processing of traffic from IP addresses 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.

Will the US 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 in Ukraine. Within hours of the report, press secretary Jen Psaki said in a tweet that NBC got it wrong: "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.

But that source also said that most of the potential measures on the slate of possible cyberattacks (a slate that, 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.

Wow.

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 their rails, that's a significant capability that one would not want to waste, since it could probably never be done 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 noted before, is that all sides are keenly aware that no one is able to withstand a cyber attack. 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, 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 discussing a catastrophic mistake Samsung recently made 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.

Windows 11 Compatibility

Last Wednesday, while I was working away on SpinRite, I had Windows Weekly running in the background as I often do. Leo, Paul and Mary Jo were talking as they always do about what's on their mind regarding Microsoft, Windows and often the industry at large.

At one hour and 13 minutes into last week's Windows Weekly podcast #765, Paul very matter-of-factly stated something as ***"we know it for a fact"*** which brought me up short and yanked my attention away from my code. Wait! What?? We do?! So here, now, is 63 seconds from last week's Windows Weekly #765: https://media.grc.com/mp4/Windows-Weekly-765.mp4

This, of course, is what every one of this podcast's listeners have been putting up with me ranting about from the first moment 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 that 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 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 v1.2 for everything they wanted to protect with TPM 2.0.

Now, this may seem odd, but I would really 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 14, 2025 when support might be ending for Win10. I'll have a large stable of machines 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, will we all be able to upgrade our machines that don't currently qualify to run Windows 11? It's difficult to imagine not. Microsoft has really messed this one up.

Closing the Loop

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.

SpinRite News

SpinRite's 9th development pre-release went out last week 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. 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 BIOS for its bulk scanning when the system's firmware is somehow faster than my own hardware drivers. At this moment, because I just haven't had the chance to dig into it yet, we have an odd problem with older IDE drive cabling and the use of cable select rather than explicitly jumpering drives for master or slave operation, and there are a few very very edge cases which I'll be looking into next. 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.

I had been holding my breath on this one, since the immediately previous release was December 23rd and since then I had re-engineered SpinRite's hardware interrupt handling, made many other improvements, introduced the mini-benchmark and 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.

I'll get back to it this evening and we'll get the few known remaining issues fixed. At that point, SpinRite's all new hardware interface platform will be in place, not only for v6.1 but also for v7 and beyond. But most significantly... most of the hard work is now behind us.

Trust Dies in Darkness

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. Their paper is titled: “Trust Dies in Darkness: Shedding Light on Samsung’s TrustZone Keymaster Design.”

https://eprint.iacr.org/2022/208.pdf

This paper is extremely interesting due to the specific mistakes it finds, highlights and describes in detail. But it carries, and 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 cannot. As a global society we MUST change the way we think about these things. ~100 million Samsung/Android smartphone users cannot be allowed to carry critical security technology that NO ONE ELSE has ever been allowed to examine.

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 full independent public examination and auditing of these machines. The idea is that the value added is NOT the proprietary hocus pocus firmware — 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, so back to 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:

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.

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 reversed-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 AESGCM 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 (IV) 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 ~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 that 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. We can't secure our own systems. It's beyond us.

So all of 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 wooly and totally out-of-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 secrets — and have demonstrated, is that ~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. Many early systems didn't think it mattered so they tossed a coin or perhaps did what was easiest. But either way they often got it backwards. The only right answer is that when both encrypting and authenticating, you always encrypt first and apply authentication last, because 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's no longer black, it's because we spent a lot of time back in the old days closely examining those questions and the security they could compromise.

SQRL needed authenticated encryption to protect its user's key material and the client's settings. And like AMD and Samsung, AES-GCM — Advanced Encryption Standard Galois Counter Mode — is the authenticated encryption algorithm I chose. And 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 its encryption. Which is what I needed for SQRL.
AES-GCM is, however, a bit brittle. Coincidentally been talking recently about brittle encryption

and mistakes being made in cryptographic implementations... like last week 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. Last week's XOR keystream reuse was what bit the Hive guys. And AES-GCM requires the use of a so-called initialization vector, or 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 is that much as with an XOR keystream, AES-GCM's Initialization Vector MUST never be reused.

So now you can guess and understand the crucial and obvious mistake that Samsung's engineers made. The Tel-Aviv University researchers wrote:

ARM is the most widely used processor in the mobile and embedded markets, and it provides TEE hardware support with ARM TrustZone. TrustZone separates the device into two execution environments:

1. *A non-secure REE (Rich Execution Environment) where the "Normal World" operating system runs.*

2. *A secure TEE (Trusted Execution Environment) where the "Secure World" operating system runs.*

The REE and TEE use separate resources (e.g., memory, peripherals), and the hardware enforces the protection of Secure World.

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 (TZOS) in use.

The Android Keystore provides hardware-backed cryptographic key management services through a Hardware Abstraction Layer (HAL) that vendors such as Samsung implement. The Keystore exposes an API to Android applications, including cryptographic key generation, secure key storage, and key usage (e.g., encryption or signing actions). Samsung implements the HAL through a Trusted Application (TA) called the Keymaster TA, which runs in the TrustZone. The Keymaster TA performs the cryptographic operations in the Secure World using hardware peripherals, including a cryptographic engine.

The Keymaster TA's secure key storage uses blobs: these are "wrapped" (encrypted) keys that are stored on the REE's file system. The "wrapping", "unwrapping", and usage of the keys are done inside the Keymaster TA using a device-unique hardware AES key. Only the Keymaster TA should have access to the secret key material; the Normal World should only see opaque key blobs.

Although it is crucial to rigorously verify and test such cryptographic designs, real-world TrustZone implementations received relatively little attention in the literature. 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 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 reversed-engineered the full cryptographic design and API of several generations of Samsung's Keymaster TA, and asked the following questions:

Does the hardware-based 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?

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 in their pre-disclosure to Samsung. They wrote:

We reported our IV reuse attack on S9 to Samsung Mobile Security in May 2021. [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 IV from the API. According to Samsung, the list of patched devices includes: S9, J3 Top, J7 Top, J7 Duo, TabS4, Tab-A-S-Lite, A6 Plus, A9S.*

We reported the downgrade attack on S10, S20 and S21 in July 2021. [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 S10, 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 TZOSs 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: The great breakthrough in cryptography came when we switched from proprietary and secret non-keyed encryption algorithms which scrambled the bits, to public and publicly scrutinized keyed cryptographic algorithms. Make the algorithm public and the keys secret. This same breakthrough principle must also be applied to the wider implementation of these public algorithms. 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" We need to shine a light.