Security Now! #908 - 01-31-23

Data Operand Independent Timing

This week on Security Now!

This week we embark upon another two hour tour to answer some pressing questions: What happens if the vendor of the largest mobile platform begins blocking old and unsafe APIs, and can anything be done to prevent that? What new add-on is now being blocked by the dreaded Mark of the Web? Would you have the courage to say no after your gaming source code was stolen? Is any crypto asset safe, and what trap did our friend Kevin Rose fall victim to last week? How can Meta incrementally move to end-to-end encryption? Isn't it all or nothing? What other new feature did iOS 16.3 bring to the world, what's the latest government to begin scanning its own citizenry, and why aren't they all? Or are they? What spectacular success gives the FBI bragging rights, and why is Russia less than thrilled? What questions have our listeners posed? What's the possible value of making up your own words? How's SpinRite coming? What, is your favorite color? What have Intel and AMD just done to break the world's crypto? And what exactly did ChatGPT reply when it was asked by one of our listeners to explain an SSL certificate chain in the voice of a stoned surfer bro? Leo will present the answer to that in his dramatic reading once the answers to all of the preceding questions have been revealed during this week's gripping episode of Security Now!.

Security News

Android to start blocking old and unsafe apps

In a piece of VERY welcome news, in what will likely be a very effective move to reduce malware on the Android platform, Android 14 will begin fully blocking the installation of apps that target outdated versions of Android.

The guidelines for the Google Play Store have long ensured that Android developers keep their apps updated to use the latest features and safety measures of the Android platform. But this month the guidelines were updated to require all newly listed Play Store apps to target Android 12 at a minimum. Meaning, no use of older long-deprecated APIs and no longer secure features.

Until now, these minimum API level requirements only applied to apps that are intended for the Google Play Store. Should a developer wish to create an app for an older version, they could do so and simply ask their users to sideload the APK file manually. And if an Android app hasn't been updated since the guidelines changed, the Play Store would continue serving the app to those who have installed it once before.

But according to a newly posted code change, Android 14 is set to make API requirements strict to block the installation of all outdated apps. And this will include blocking users from sideloading APK files and also block app stores from installing those apps.

To minimize disruption, Android 14 devices will initially only block apps targeting very old Android versions. But over time the plan is to raise the bar to Android 6.0 (Marshmallow), with Google having a mechanism to progressively ramp it up over time. It will probably still be up to individual device makers to set their device's threshold for outdated apps, or perhaps whether to enable an "oldest limit" at all.

Google believes, with good reason, that this will curb the use of malicious apps on Android. The Google developer who's responsible for the change notes that malicious apps intentionally target older versions of Android to bypass certain protections which are only enforced on newer apps. Does this remind anyone of the protocol version downgrade attacks against SSL and TLS? As we've seen, it is often necessary to stop using and to prohibit the use of older obsolete and insecure technologies. The same is exactly the case here.

And even so, if, for whatever reason it is absolutely necessary to install a very outdated application, it will still be possible to do so through a command shell, by using a new flag. But given the extra steps required, it is much less likely that someone would do this by mistake and inadvertently install malware.

Microsoft to block Internet sourced Excel add-ins:

And speaking of blocking malware, Microsoft also plans to block the execution of Excel add-in (XLL) files inside Excel and other Office apps if the XLL files were downloaded from the Internet. Microsoft says it made this decision to "combat the increasing number of malware attacks in recent months" that have abused Excel add-ins to bypass email filters and execute malware on user devices. Right. Unfortunately, the use of ZIP file containment to avoid the dreaded "MOTW" — the Mark of the Web — has skyrocketed among those creating and distributing malware,

specifically because Microsoft has finally (after how many years?) Started restricting what MOTW-marked files can do. The reason I zipped Rob's LastPass vault script was to protect it from the Mark of the Web. The ZIP received the dreaded marking, but its contents were protected. So it's not that I don't think that what Microsoft is doing is useful, it's how anything that was this insecure was ever allowed to happen in the first place.

An example of saying "no" even when it may hurt...
Riot Games reported that it had received a ransom demand via email from a threat actor who hacked one of its employees to then gain access to one of its game development environments. Riot says the hacker is asking the company to pay ransom, or they will release the source code for the League of Legends and Teamfight Tactics games as well as the source code of a legacy anti-cheat platform. The company says it does not intend to pay the ransom and expects the leaked source code to "increase the likelihood of new cheats emerging."

I suspect that they also figure that there's really no way that the bad guys would actually honor their promise to destroy the gaming system source code. It's just too juicy and tempting. Besides, they're crooks! There's little reason to believe that the source would not eventually emerge onto the Internet.

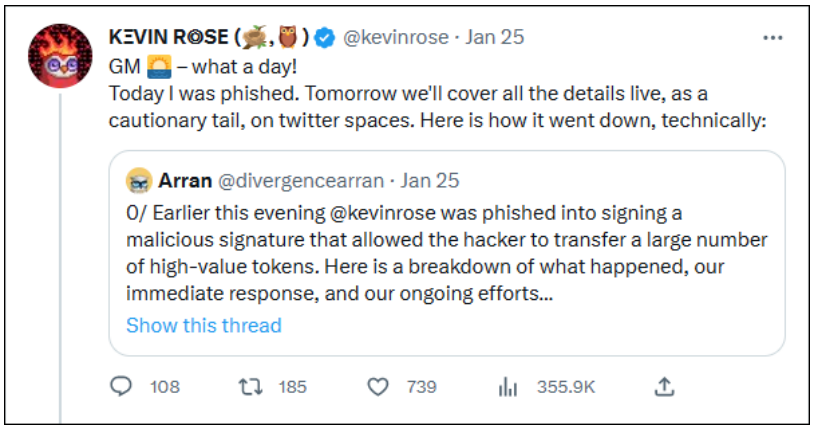
Hacked Wormhole funds on the move:
The CoinTelegraph reported that the threat actor behind that hack of the Wormhole cryptocurrency platform in February of last year, so nearly a year ago, recently moved $155 million worth of the $321 million in assets they stole from the company, according to the blockchain analysis platform CertiK. So that's nearly half of that $321 million that they stole from Wormhole. When Wormhole saw the funds move, they reiterated their willingness to pay a $10 million reward if the funds were returned to its wallets.

Okay, Leo. First, how can there possibly be so much cryptocurrency sloshing about? We keep talking about hundreds of millions of dollars here and hundreds of millions of dollars there. And there appear to be no end of random currencies and exchanges all loaded up with this cash value. And, my lord! Is anyone able to keep their assets to themselves? When I was thinking about this story it occurred to me that it was like someone had come up with the great idea of having banks BEFORE they had invented safes to protect the bank's assets from theft.

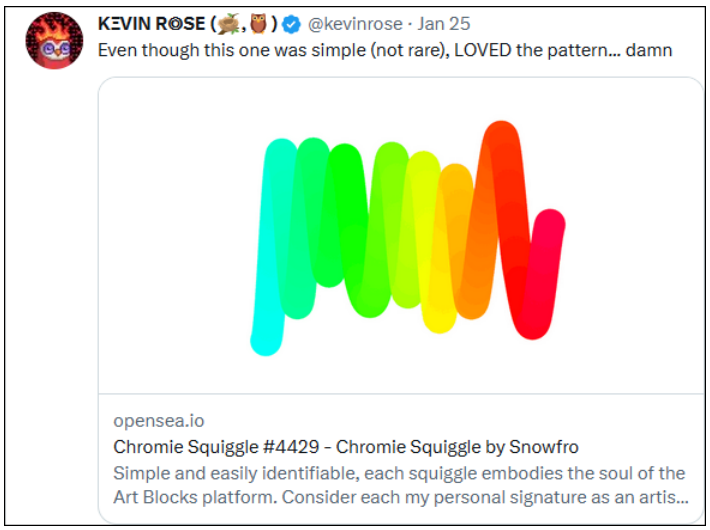
Anyway, not a huge news item, and just another tidbit to make us shake our collective heads.

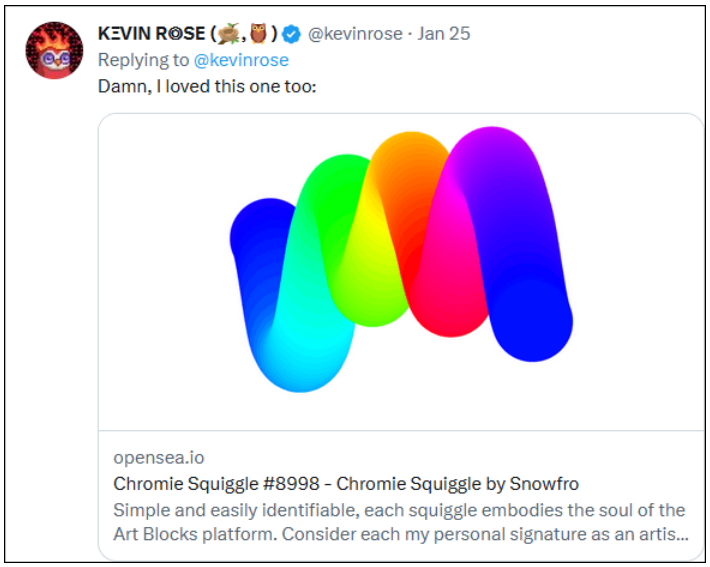
Kevin Rose Hacked:
Under the heading *"It really can happen to anyone"* last Wednesday came the news that the super tech-savvy gazillionaire founder of DIGG, now tech venture capitalist, NFT maven and really neat guy, Kevin Rose fell victim to a classic social engineering attack. Yes... It **really** can happen to anyone.

On the 25th, Kevin tweeted a short message to his 1.6 million Twitter followers. He tweeted:



I won't delve into the details here since all of this just makes my eyes cross and makes me feel really old. But among other assets, Kevin apparently lost control of some of his favorite "squiggles", the lossage of which he laments in several later tweets once the breath of the theft has been determined:





Three days later, last Saturday, Kevin followed-up, tweeting:

Some other industry reporting that followed this event noted that the attacker made off with more than 40 NFTs and that the stolen assets were worth $2 million on Wednesday, when the theft occurred, $1.4 million early Thursday, and just $1 million by Thursday afternoon. Perhaps I'm not all that unhappy that I haven't chosen to waste – I mean invest – time in learning all about this weird world. Aside from the apparently uncontrollable lack of security, the whole thing doesn't feel like a financially stable ecosystem. But I know that Kevin is just having fun with it all.

Facebook will be moving more users into E2EE:

Meta has said that it plans to migrate more of its messenger users over to the end-to-end encrypted (E2EE) version of Facebook Messenger over the next several months. The company says users will be chosen at random, and users will be notified when their private conversations will be upgraded to the E2EE version. In addition, the company has also expanded the features of its E2EE version, which now also supports some of the features of the original Messenger app, such as link previews, chat themes, user active status, and support for the Android floating bubble mode.

Meta wrote: *"Over the next few months, more people will continue to see some of their chats gradually being upgraded with an extra layer of protection provided by end-to-end encryption. We will notify people in these individual chat threads as they are upgraded. We know people will have questions about how we select and upgrade individual threads, so we wanted to make clear that this is a random process."*

As I was putting this bit of news together I was thinking: *"Wait... How can you move some users to E2EE and not everyone at once? If you have E2EE then both ends must be E2EE-capable, right?"* But then Meta's comment clarified that. Apparently **all** messenger users already have E2EE messenger apps, they just cannot themselves enable its use for all of their communications. But Meta, selectively, can. So now what Meta said makes sense, writing: *"Over the next few months, more people will continue to see some of their chats gradually being upgraded with an extra layer of protection provided by end-to-end encryption."*

I suppose this is the way to slowly move the world there while dealing with anything unexpected that might arise. In any event, it sounds as though once Meta is confident that everyone can be moved to and using E2EE by default, that will happen and messenger will join the ranks of iMessage, Signal, Telegram and Threema who have all been offering full E2EE as a core feature.

iOS 6.3 and FIDO

Last week, when I mentioned that with the release of iOS 16.3 Apple's full iCloud encryption would be available globally, I forgot to mention that this release also allowed the use of third-party FIDO-certified Security Keys for Apple ID. Until now, although Apple has allowed

users to use various forms of two-factor authentication (2FA) methods to secure their Apple ID accounts, Apple has been slow to add support for hardware dongles. iOS 16.3 and macOS Ventura 13.2 are the first iOS and macOS versions which allow users to use FIDO-certified hardware security keys to log into Apple accounts. So, in case any of our iOS and macOS users have those keys and didn't have anywhere to stick them, now they do! Oh… and NFC tokens will also work with the phone's NFC capability.

Scan thy Citizenry

We've been following the growing and, I think, entirely sane and rational emerging practice of governmental security entities proactively scanning the networks of their citizenry, commercial enterprises and governmental services. Poland's CERT, known as CERT Polska, recently described their Artemis system as follows:

> *Artemis scans services exposed to the Internet to look for common vulnerabilities and configuration errors. Regular scanning of entities that fall under the constituency such as schools, hospitals, or local authorities, allows us to monitor and improve their cybersecurity. This is important because of the nature of these organizations – they are used by citizens on a daily basis and any incidents affect them as well.*
>
> *The scan results are not shared publicly – they are instantly forwarded to the administrators of the systems in question. The data is then used to address vulnerabilities and to detect similar issues in other parts of the infrastructure. As a part of the scanning process CERT Polska also verifies whether the identified vulnerabilities were fixed correctly.*
>
> *One important aspect of the created tool is that it enables administrators to easily distinguish scanning activity as conducted by CERT Polska. This helps minimize the unwanted effects like unnecessary attack mitigation. All relevant information is accessible to administrators on a dedicated page.*
>
> *The scanning results, aside from improving the security of a specific entity, help us to create a better view of the current cybersecurity landscape and designate our resources where they are needed the most at the moment.*

Again, I see nothing but upside to this, and all governments hopefully have similar undertakings underway. In this case of Poland's proactive scanning, here's what they wrote about their results so far:

> *The scanning process began on 2nd of January and has already produced some results. We've scanned close to 2000 domains and subdomains of local governments and we were able to detect few hundred websites based on outdated software. We've also dealt with numerous cases where configuration files that included passwords, backup archives and data records were publicly accessible. We've also found a few dozen of incorrectly configured directories that contained the page source code and in some cases access credentials.*

Not bad for a start and certainly easily justifiable in any budgetary meeting. The use of outdated

systems doesn't necessarily translate into exploitable vulnerabilities, but it can point to IT administrators who are not keeping their public-facing systems current. And that CAN lead to potential exploitation. At the very least, it creates some very much needed feedback and accountability. When a system gets hacked after an organization had been notified of a problem and for whatever reason chose to do nothing about it, the excuse of "we didn't know" will no longer fly.

The Hive ransomware organization takedown

In a very impressive piece of high-tech intelligence, Law enforcement agencies from the US and EU have seized servers and websites operated by the Hive ransomware gang. The US Department of Justice says the FBI secretly breached the Hive gang's infrastructure in July of last summer, from where agents retrieved more than 1,300 decryption keys over the past seven months. Of these, the FBI distributed 1,000 decryption keys to past Hive victims but also shared in real-time more than 300 new decryption keys to companies that had computers encrypted in ongoing Hive attacks last year. Officials say they prevented ransomware payments estimated at roughly $130 million, but they also notified many other companies when their networks were breached, even before Hive and its affiliates had a chance to deploy their ransomware and encrypt their data. Since June 2021, when the Hive gang launched its operation, the group is believed to have made more than $100 million from ransom payments. So, wow... nice going FBI!

And Russia reacts...

The Russian government has blocked access inside the country's borders to the websites of the CIA, FBI, and the US State Department's Rewards for Justice program. Russian officials with Roskomnadzor, the country's Internet watchdog, told Interfax they blocked access to the websites for (love this) "spreading fakes about the Russian military and discrediting them." However, the timing of this decision is coincidental and might be telling, as it came just hours after the State Department offered a $10 million reward for information on the Hive ransomware gang and its possible ties to a foreign government.

Hmmmmm... I wonder which foreign government might be feeling a bit guilty?

Errata

Thanks goes to our listener Edwin Rosales who wrote to Leo:

Hi Leo!

FYI, in Security Now episode 906, Steve erroneously conflated Symantec with Norton (Now Gen Digital, Inc.).

I pointed out to Steve neither Norton nor their parent company, Gen Digital, are affiliated with Symantec, ever since Broadcom acquired the Symantec brand and enterprise security assets back in 2019, but he may have missed my Twitter reply to his post.

I also mentioned the acquisitions/reorgs are a bit confusing to follow. After Symantec sold its

brand and enterprise assets to Broadcom, what was left was the consumer product, which they rebranded as "NortonLifeLock", which was again as Gen Digital last November (2022).

Closing the Loop

Person typing #22 / @pt22

*@SGgrc you mentioned diceware on SN. log2(7776) is 12.9 bits of entropy per word. yes they have fewer bits of entropy *per character*, but a 6-8 word (random!) diceware phrase (plus one capital, digit, and special) is the holy grail: memorable, easily typed, and secure.*

I get the attraction of the diceware idea. But 12.9 bits of entropy per word is not a lot of entropy. It's about equivalent to two randomly chosen characters worth of entropy. So, to get the 20 randomly chosen characters worth of entropy that are about what you need, we'd need to use ten diceware words. And that's quite a lot of typing.

I don't know why it never occurred to me to just share what I have always been doing. Somehow I've never talked about it, as if divulging my own personal system would make my own use of it more vulnerable. But that's not the case. If you've paid close attention to this podcast, you may have heard me mention some of the words I've made up through the years. But I've been quite careful to never mention any of the made up words I use for master passwords. Made up words are an intriguing compromise.

Let's consider the advantages of using your own, fun, made up words that you won't forget because they're auditory. Such non-words are easy to invent: They could be shorter, like bingle, borhog, zinkles, crample, zootram, simulax, and jubaloo. Or you could go with longer words like vorchhoggen, weiglestagen, rambloses, plakonkits or footremith. (I do want to point out that you will want to steer clear of anything having to do with framulators.) My point is, made up words have a lot going for them. Being phonetic, they are fun to say and memorable if you practice them a bit. Unlike diceware words, they're not going to be in anyone's dictionary, and since each one can have many characters, when you use several together you also get a self-padding longer haystacks-style benefit as well.

So, my best advice is to do what I have always done: Invent a few of your own fun words, add some capitalization and toss in a random special character between each word and you've got something that's memorable, with good entropy, that won't be found by any dictionary attack and won't be cracked within several lifetimes.
And if you're feeling a lack of inspiration or imagination, look on the Net under "fake word generators" and you'll discover that many of them already exist. You can use them as a starting point building your own set of personal words that you wind up settling on.

Magic by Barry / @BarryWallis

@SGgrc @leolaporte

Listening to the last SN had me come up with a new term. Instead of script-kiddies we now have chat-kiddies.

I liked Barry's thought. We do need a new term that's similar in concept to script-kiddies. But I think that the word "chat" isn't domain specific enough. So maybe "ChatBot-kiddies" ??

Bitwarden and PDKDF's

Last Tuesday: Bernd / @Quexten

Hi Steve,

glad to hear you mention changes towards Argon2 support in Bitwarden today!
Some clarification of why the scrypt pull request is closed now, in favor of argon2: I first implemented scrypt, because the libraries were more widely available (pure javascript in the browser, the already in-use "BouncyCastle" library on mobile).

After the simple-to-implement scrypt support was done, I began work on the argon2 pull requests expecting them to take longer (and they did take a fair bit more work). Argon2 requires WebAssembly in the browser, and that was a concern in the forums in the last few years, but these days all browsers except IE (which hopefully no one uses) support it. Mobile support was also a bit more work, and finally changes in the server are necessary to account for argon2's extra parameters.

scrypt mainly relies on a [single] work factor which determines both memory and time complexity, while argon2 configures memory and time separately. Because of this, changes in the backend and communication are necessary to send these extra parameters [required by Argon2] (compared to PBKDF2's [simpler] iterations).

Since the initial argon2 pull request was complete before scrypt was reviewed and merged, and since multiple new key derivation functions seemed redundant, and argon2 is the newer, more crack-resistant function, we decided to close the scrypt pull request to focus on argon2.

OWASP also recommends argon2 over scrypt, scrypt over bcrypt and bcrypt over pbkdf2.

Anyways, after some back-and-forth with the Bitwarden team, we are close to getting support merged, which is exciting!

And then, Yesterday at 8:46am: Bernd / @Quexten

Final update on Argon2 in Bitwarden: Support has now been merged into their master branches for mobile, desktop/web and servers. Next release should feature argon2 as a new PBKDF option.

So, this is wonderful. And if this is an example of the agility that an open source approach and mentality can bring to security products where agility can be crucial then I'm sold.

Dennis Keefe, Financial Coach / @prosperuscoach

Steve, in regards to ChatGPT becoming so popular, what do you think would be the best career path to focus on over the next 5 years. I'm currently working on Linux SysAdmin certifications.

The use of these Large Language Model transformers is bringing us to the brink of something. But like most big game-changing somethings, we almost certainly do not yet fully understand the something that we're on the brink of. If you wanted to go back to school, there's likely to be big career opportunities in artificial intelligence. Just as all larger companies need a CEO and a CFO, a COO and a CIO, it may be that before long there will be CAIO positions at the top.

But my best advice for picking a career, any career, has never changed: first and foremost follow your heart over your wallet. Many people with fat wallets and thin lives. Find something you love and work to get really good at it. You'll enjoy the process of getting good at it and then you'll love being good at it.

One thing we know for certain is that the career opportunities in cybersecurity are very real. As we've discussed, they are likely not for someone who wants to punch out at 5pm every weekday, since computers never sleep and bad guys are often in faraway time zones. But it should be clear to anyone who follows this podcast that cybersecurity is a growth industry today.

Mark Sidell / @msidell

Steve, Bitwarden can store the TOTP seed for a web site. When you visit the site, it will automatically copy the current six-digit code to the clipboard, making it simple to paste the code into the site's MFA control. Would you ever use this Bitwarden feature? It would seem to reduce MFA protection to a single factor -- your Bitwarden master password.

I think that Mark is exactly right. One of our recurring observations is that just because something can be done doesn't mean that it should be done. I don't mind having Bitwarden offering this feature, but I would never consider using it for exactly the reasoning that Mark suggests. The entire benefit of the OTP Auth app running in my iPhone is that it is physically and logically disconnected from the website I'm authenticating to. If my password manager is able to fill-in my username, my password **and** my time-varying 6-digit token, then a useful aspect of the second factor is lost.

The counter argument to this, is that what the OPT token actually protects against is the theft and reuse of our static credentials. It makes one of our required credentials dynamic so that credential reuse (last week's podcast topic) is completely thwarted. When viewed from that perspective, having a password manager also able to provide the dynamic component of a set of credentials seems reasonable.

So I suppose that what makes me nervous about turning over my OTP generation to the same system that's holding all of my other credentials is the "all of my eggs in one basket" concern. As I've said, the first thing that went through my mind when I heard that the LastPass customer

vault backups were now in the hands of the bad guys, was that all of the sites I most cared about are setup with OTP tokens **that LastPass never had any awareness of.** That turned out to be a blessing. So I'm glad that my OTP token generation keys were never theirs to lose.

John /@PsyVeteran

Hey again Steve @SGgrc. I'm looking back through the podcast and show notes looking for the name of the VPN-like technology you guys talked about and reference every now and then. One of its features was much wider bandwidth than classical VPN technologies. Could you recall the name for me, I'm stumped. Warm regards and thanks for your great insights as always. John

Okay. I think that what John is asking about is what's now being referred to generically as "Overlay Networks" or sometimes "Mesh Networks." This was what we discovered early with Hamachi and there are now a number of similar solutions. The ones we've talked about in the past are Tailscale, ZeroTier and Nebula. For example, Nebula is an open-source, peer-to-peer mesh network created by engineers at Slack and open sourced after several years of internal use. Many things impress me about TailScale, including that it's a mesh network based on WireGuard, which is the right core and that after we talked about it here many of our listeners wrote to say that they were astounded by how easy it was to setup and use. And did I mention that it can be used for free for personal & hobby projects to connect up to 20 devices for secure peer-to-peer connections with single sign on and multi-factor authentication.

Based upon its specification and the amazing experiences of our users, I'd say check out TailScale.

steven lacey / @infosecsocial

So @SGgrc - during SecurityNow: 905 "1" you mention the lack of an API for password changes. Is there any chance this could be implemented in a safe and secure way?

SpinRite

On the SpinRite front, we had a very productive week dealing with various oddball edge cases. I'll share two examples:

We discovered that there are some BIOSes whose USB support uses the available 32-bit'ness of the processor while not preserving the high 16 bits that they modify. This would be okay if only 16-bit client programs were running on those machines, since those clients would be unaware that they are actually operating on a 32-bit processor. But SpinRite is now working with multiple 16 megabyte buffers within a flat 32-bit address space. So it has grown into a 32-bit application running within a 16-bit DOS real mode environment. API functions which are called are required to preserve any registers that they use. But we discovered that the USB functions on some AMD motherboards are not doing so. So now SpinRite is protecting itself from that behavior and a

handful of mysterious behavior has been resolved.

In a second example, a Canadian tester named Andre sent me some drives, one of which was reliably causing SpinRite to crash. I had a similar drive that was also misbehaving, but SpinRite would not crash for me. The drives arrived yesterday morning so I plugged-in the culprit and at long last recreated the crash that Andre, and others, had been able to independently reproduce. I saw when it was happening, went there in the code, watched the problem occur, saw that something was modifying the stack, found the problem and fixed it. No more crashes.

So that's where SpinRite is now. It's done and it's working. 629 current SpinRite owners have been testing it and I'm happy to say that most of them are bored. Bored testers is what you want. But not all of them are bored yet, and I want all testers to be bored because they're unable to find any wacky system or damaged drive that causes SpinRite trouble. Once we're there it'll be done.

Data Operand Independent Timing

Last Wednesday the 25th, Eric Biggers, a software engineer at Google on the Platform Encryption Team, brought a significant cryptographic security issue to the attention of the well-trafficked OSS Security list in a posting titled: "Data operand-dependent timing on Intel and Arm CPUs." Now, admittedly, in any other venue than ours that might sound dry. But not here.

We were recently talking about data dependent timing, which is a huge issue for cryptographic security. We were talking about it in the context of the scrypt PBKDF algorithm which might suffer from a side-channel attack (as the initial version Argon2 also did) due to the fact that the value of the user's password directs the functioning of the algorithm. In other words, if the algorithm operates in any way differently depending upon the data that it's processing – specifically anything that must remain secret, like the user's password or the algorithm's secret key – then it's theoretically possible to reverse that process by observing the algorithm's behavior from afar, things like power usage, execution timing, cache hits and misses, branch prediction traces – whatever – to figure out what data must have been given to the algorithm in order for it to behave in the way that was observed. And we've seen how astonishingly clever researchers have turned out to be.

One of the reasons the Rijndael cipher won the competition to become the AES standard, was that its operation was beautifully independent of the secret key it was operating upon. No jumps or branches were taken or not based upon the algorithm's secret key. You don't want to have any "secret dependent" behavior. But... What if the timing of the instructions themselves was dependent upon the data that the instructions were processing?

A traditional example of this, that may be familiar to old coders, is CPU multiply instructions. Binary multiplication is inherently a complex process. So inside a processor, multiplication was traditionally an iterative process with the number of clock cycles required varying widely depending upon the data that was being multiplied. The show notes contains a table showing the number of clock cycles required by various early Intel CPUs, from the original 8088/86, the 268, 386 and 486. In even the later 486, a 32-bit multiply will require anywhere between 13 and 42 clock cycles, where that count is entirely dependent upon the data that's being multiplied:

MUL - Unsigned Multiply

```
Usage:  MUL     src
Modifies flags: CF OF (AF,PF,SF,ZF undefined)

Unsigned multiply of the accumulator by the source.  If "src" is
a byte value, then AL is used as the other multiplicand and the
result is placed in AX.  If "src" is a word value, then AX is
multiplied by "src" and DX:AX receives the result.  If "src" is
a double word value, then EAX is multiplied by "src" and EDX:EAX
receives the result.  The 386+ uses an early out algorithm which
makes multiplying any size value in EAX as fast as in the 8 or 16
bit registers.

                          Clocks                Size
Operands         808x  286   386   486          Bytes

reg8              70-77   13   9-14  13-18         2
reg16            118-113  21   9-22  13-26         2
reg32              -      -    9-38  13-42        2-4
mem8          (76-83)+EA  16  12-17  13-18        2-4
mem16     (124-139)+EA  24  12-25  13-26        2-4
mem32              -      -   12-21  13-42        2-4
```

So we come back to the question: What if the timing of the instructions themselves was dependent upon the data that the instruction was processing?

Which leads us into what Eric wrote:

Hi,

I'd like to draw people's attention to the fact that on recent Intel and Arm CPUs, by default the execution time of instructions may depend upon the data values operated on. This even includes instructions like additions, XORs, and AES instructions, that are traditionally assumed to be constant-time with respect to the data values operated on. For details, see the documents from each CPU vendor.

Non-constant-time instructions break cryptographic code that relies on constant-time code to prevent timing attacks on cryptographic keys -- which is most cryptographic code. This issue may also have a wider impact on the ability of operating systems to protect data from unprivileged processes.

For Intel, processors with Ice Lake and later are affected by this issue.
The fix for this issue is to set a CPU flag that restores the old, correct behavior of data-independent timing: DIT (Data Independent Timing) on Arm, and DOITM (Data Operand Independent Timing Mode) on Intel.

Linux v6.2 will enable DIT on Arm, but only in the kernel. Without any additional patches, userspace code will still get data-dependent timing by default.

No patch has been merged to enable DOITM on Intel processors. Thus, as-is, it's not really possible to safely execute cryptographic algorithms on Linux systems that use an Intel processor with Ice Lake or later. (I'd guess that the same is true for other operating systems too; Linux is just the one I'm looking at.) To fix this issue, I've proposed a Linux kernel patch that enables DOITM globally.

I consider this issue to be a CPU security vulnerability; it shares many characteristics with other CPU security vulnerabilities such as Meltdown and Spectre. However, Intel and Arm do not seem to consider it to be a security vulnerability. No CVEs seem to have been assigned yet.

- Eric

First of all, CVEs are not generally assigned to things that are deliberate and by design, as everything Eric is complaining about, is. It's not fair to compare this to Spectre and Meltdown which shocked and rocked the computing world five years ago, in January of 2018. But, being today's podcast topic, you can bet that there are some interesting details here to share.

After encountering Eric's posting, I started digging. And the fairest characterization would not be to say, as Eric did, that Intel doesn't seem to consider this to be a security vulnerability, only that they have decided to turn this over to developers.
First, I was curious about when this suddenly became a problem. Intel writes:

For Intel® Core™ family processors based on microarchitectures before Ice Lake and Intel Atom® family processors based on microarchitectures before Gracemont, neither of which enumerate IA32_UARCH_MISC_CTL, [which is an internal control register] developers may assume that the instructions listed here operate as if DOITM is enabled. Intel Core family processors based on Ice Lake and later, such as Tiger Lake, Lakefield, and Rocket Lake will explicitly enumerate DOITM. Intel Atom family processors based on Gracemont and later will also enumerate DOITM.

Translating that into English, Intel's earlier processors executed all of their their instructions in constant time. So they were inherently safe to use, regardless of OS, kernel, userland or anything else. Instruction timing **did not vary** based upon the data that the instruction was processing. If the data in two registers were added, XORed or multiplied, the instruction always took the same amount of time regardless of what was in the registers being used.

Then that changed.

What must have happened, is that Intel realized that there was a way to optimize and speed up the execution of some instructions depending upon their data. Here's an off-the-cuff example I've made up to highlight the idea:

One of cryptographers' most favorite instructions is the XOR, where one of the instruction's two datums conditionally inverts the bits of the other. XOR is also known as carry-less multiplication since the operation is similar to multiplication but where adjacent bits do not carry an overflow into the next most significant bit. Thus adjacent bits do not affect one another. And that's significant since that means that adjacent bytes don't affect one another either. Now suppose that Intel's internal microarchitecture contains lots of granular execution engines, as, in fact, we know it does. So imagine that the work of performing a 32-bit XOR could be subdivided into four separate 8-bit XORs with each 8-bit XOR being handled by a different execution microengine. Then we observe that with an XOR, anytime either of the bytes being XORed is zero, no data is changed and the XOR has no effect, thus there's nothing for that little microengine to do, and it

could, instead, be made available to work on other instructions. So in this little synthetic example, we see how a 32-bit XOR which encountered non-zero data in all four bytes of both arguments would need to enlist the help of four 8-bit microengines, whereas the same 32-bit instruction presented with bytes of zeroes in either argument would leave those microengines free to work on other instructions. In this fashion, the effective execution time of such a processor's 32-bit XOR becomes dependent upon the data it's XORing.

Okay. So, was Intel going to pass up the opportunity to make their chips go faster by arranging for some instructions to go faster some of the time? Hah! No way! But... Whoops! A side effect of this is that it would screw up the longstanding assumptions made by cryptographers that the execution speed of instructions was independent of the data being processed.

So what did Intel do? They couldn't permanently break everyone's crypto, sending us all back to the dark ages. So they added a "mode" called DOITM—Data Operand-Independent Timing Mode. **The controversy is that, by default, it's off.** This means that suddenly the behavior of Intel's newer chips **has** changed – they are no longer safe for crypto! They got somewhat faster by being able to finish some instructions more quickly, but the side effect of this is that they have also became insecure processors for performing cryptographic operations. Unless this new DOIT mode is explicitly and deliberately turned on, it will be off. And what Eric is lobbying hard for within the Linux kernel is to immediately turn the darned thing on, permanently and globally.

The only real question is: Is Eric overreacting? Intel, for their part, writes this:

> *"Software can enable data operand independent timing operation on a logical processor by setting DOITM to 1. Setting DOITM to 1 may impact performance, and that impact may increase in future processor generations. Users should evaluate their threat model to decide whether this is a significant threat to their applications and then ask the operating system to only deploy DOIT mode to applications that they deem necessary."*

In other words, sure, you can have old-school constant-speed instructions if you really think you need them, but we're hereby abandoning that model in the interest of performance today and probably more so in the future; so now it's going to be up to you.

So we're back to trying to get some sense for how bad the problem is, and how much cure we need to pour over it. Thomas Pornin, the author of the BearSSL SSL/TLS library has some nice real-world reality-check perspective about the threats and challenges of constant-time crypto. Thomas writes:

> *In 1996, Paul Kocher published a novel attack on RSA, specifically on RSA implementations, that extracted information on the private key by simply measuring the time taken by the private key operation on various inputs. It took a few years for people to accept the idea that such attacks were practical and could be enacted remotely on, for instance, an SSL server. In an article from Boneh and Brumley, seven years later in 2003, they conclude that:*
>
> *"Our results demonstrate that timing attacks against network servers are practical and therefore all security systems should defend against them."*

Since then, many timing attacks have been demonstrated in lab conditions, against both symmetric and asymmetric cryptographic systems.

This requires a few comments. First, while timing attacks work well in research conditions, they are extremely rarely spotted in the wild (I am not aware of a single case). Timing attacks usually require many attempts to gather enough samples for statistics to reveal the sought timing difference; as such, they tend to be somewhat slow and not very discreet. This does not mean that timing attacks are not real or do not apply, only that the state of the security of many systems is such that typical attackers have easier, faster ways in.

Another important point is that when timing attacks apply, they are all-encompassing: if the context is such that secret information held in a system may leak through external timing measures, then everything the system does may be subject to such leaking. This is not limited to cryptographic algorithms. Research on timing attacks tends to focus on secret keys because keys are high-value targets (a key concentrates a lot of secrecy), and cryptographers talk mostly about cryptography; however, even if all cryptographic algorithms in your system are protected against timing attacks, you are not necessarily out of trouble in that respect. In BearSSL I am doing my part, by providing constant-time implementations for all operations that are relevant to SSL; but slapping a constant-time SSL implementation over existing software is not sufficient to achieve general timing immunity. This is only a good start.

Timing attacks are a subset of a more general class of attacks known as side-channel attacks. A computer system runs operations in a conceptual abstract machine, that takes some inputs and provides some outputs; side-channel attacks are all about exploiting the difference between that abstract model and the real thing. In the context of smart card security, for instance, power analysis attacks (in particular Differential Power Analysis, that compares power usage between successive runs) have proven to be a great threat. Timing attacks still have a special place in that they can be applied remotely, through a network, while all other side-channel leakages require the attacker to be physically close to its target.

Constant-time implementations are pieces of code that do not leak secret information through timing analysis. This is one of the two main ways to defeat timing attacks: since such attacks exploit differences in execution time that depend on secret elements, make it so that execution time does not depend on secret elements. Or, more precisely, that variations in execution time are not correlated with secret elements: execution time may still vary, but not in a way that can be traced back to any kind of value that you wish to keep secret, in particular (but not only) cryptographic keys.

Okay. So where does all this leave us? With Linux we have an open process and transparency. We don't have that with Windows or Apple. And even though Apple has gone ARM, ARM is doing the same thing, calling theirs DIT. If history repeats, it's likely going to take some of those clever researchers demonstrating that Windows needs to take this seriously before Microsoft will ponderously go about doing so.

It appears that the future of our processors will be one of super-optimized variable-speed instructions with many of those instructions able to finish their work faster depending upon what that work is. But since doing this is anathema to crypto security, crypto implementations will need to be updated to flip on constant time mode while they do their work, then flip it back off afterward. I imagine that someone like BearSSL's Thomas Pornin will quickly add this awareness to his library, as will everyone else who has any sense.

Since actual real-world timing attacks have only been shown in the lab, does this mean that they are impractical? We know that crypto is where the truly paranoid are welcome. Linux may be panicking a bit depending upon the degree of true performance hit arising from flipping the switch into always on mode. It does appear that wrapping timing sensitive crypto in constant-time mode will be the right solution rather than simply wrapping the entire operating system in it.

This is all far too juicy for the world's security gurus to pass up. That band of merry researchers at the Ben-Gurion University of the Negev must be rubbing their hands together in delight. I suspect that we'll be reading a paper of theirs before long. So, this is now on everyone's radar and I'm sure we'll be circling back to this intriguing topic in the future.

For now, it's wait and see ... but for ChatGPT:

ChatGPT Statement of the Week

Biocow (tweeting from @biocow) tweeted the output from ChatGPT when asked to *"Explain an SSL certificate chain like a stoned surfer bro"* And Leo, for this one, I think you should do the honors: