Security Now! #701 - 02-12-19

Adiantum

This week on Security Now!

This week we look at Apple's most recent v12.1.4 iOS update and the two 0-day vulnerabilities it closed, as also examine the very worrisome new Android image-display vulnerability, we dive into an interesting "reverse RDP" attack, look at the new LibreOffice & OpenOffice vulnerability, we consider Microsoft's research into the primary source of software vulnerabilities, MaryJo gets an early peek at enterprise pricing for extending Windows 7 support, China and Russia continue their work to take control of their country's Internet, Firefox's resumption of its A/V warning release 65. We then update with a few more SQRL anecdotes, share a bit of listener feedback, then see how Google does the Cha-Cha with their new "Adiantum" ultra-high-performance cryptographic cipher.

The Ridiculously Inexpensive PDP11/70 Kit

Security News

Update your iOS devices to v12.1.4 if you haven't already

Aside from fixing the unattended Facetime eavesdropping bug, the 12.1.4 update also forecloses three additional flaws, two of which are being actively exploited in the wild as 0-day exploits.
https://thehackernews.com/2019/02/ios-security-update-facetime.html

Google's Project Zero researchers found the two 0-days, which they privately disclosed to Apple and Apple discovered and fixed a different flaw relating to Facetime.

- CVE-2019-7286: a memory corruption issue that could allow a malicious application to gain elevated privileges on the vulnerable Apple device.
- CVE-2019-7287: a memory corruption issue that could allow a malicious application to execute arbitrary code with kernel privileges.
- CVE-2019-7288: discovered by the Apple security team, this flaw is another FaceTime issue with Live Photos.

My phone needed a nudge. I had to go looking. So if your phone hasn't yet bugged you about the update, you should bug it. As with the recent updates, these apply to the iPhone 5S, and later, iPad Air and later, and iPod touch 6th generation.

And also, as before, these iOS troubles are largely shared with macOS. So it's time to update to Mojave 10.14.3 which fixes all of those iOS vulnerabilities for macOS, cluding the FaceTime issues.

Google / Android / February 2019 Updates...

A trio of surprisingly worrisome Android PNG image display bugs.
https://source.android.com/security/bulletin/2019-02-01.html

Google: The most severe of these issues is a critical security vulnerability in Framework that could allow a remote attacker using a specially crafted PNG file to execute arbitrary code within the context of a privileged process. The severity assessment is based on the effect that exploiting the vulnerability would possibly have on an affected device, assuming the platform and service mitigations are turned off for development purposes or if successfully bypassed.

We have had no reports of active customer exploitation or abuse of these newly reported issues. Refer to the Android and Google Play Protect mitigations section for details on the Android security platform protections and Google Play Protect, which improve the security of the Android platform.

"Framework:
The most severe vulnerability in this section could enable a remote attacker using a specially crafted PNG file to execute arbitrary code within the context of a privileged process."

The Hacker News framed it a bit less clinically and probably more realistically. Paraphrasing what they wrote:

Using an Android device? Beware! You need to be more cautious when opening an image on your smartphone — downloaded from anywhere on the Internet or received through messaging or email apps. Yes, just viewing an innocuous-looking image could hack your Android smartphone—thanks to three newly-discovered critical vulnerabilities that affect millions of devices running recent versions of Google's mobile operating system, ranging from Android 7.0 Nougat to its current Android 9.0 Pie.

Although Google engineers have not revealed any technical details, the updates mention fixing a "heap buffer overflow flaw", "errors in SkPngCodec," and bugs in some components that render PNG images.

We know that Google is working diligently to improve the Android update ecosystem. But what has already shipping has already shipped, and most of Google's efforts are necessarily forward looking. So that means that the bad guys know there are three, one of which is very bad, PNG image rendering flaws in Android since v7.0 Nougat, released on August 22, 2016 -- so two and a half years ago -- and every Android phone sold in the interim, until and unless patched, will be vulnerable to these flaws. Most phones sold in the interim will always be.

Those who have been listening to this podcast since the summer of 2015 will recall the Android "Stagefright" exploit which caused quite a bit of activity on both sides of the law. Given that this needs only to display a specially crafted image which can be received through any channel -- through IM -- even if secured -- eMail or web page... I have the feeling we haven't seen the end of this.

Reverse RDP Attack: Code Execution on RDP Clients
https://research.checkpoint.com/reverse-rdp-attack-code-execution-on-rdp-clients/

We've recently covered the news of exposed remote desktop (RDP) servers. Apparently, people believe that it's safe to leave an RDP server exposed and unmonitored to the Internet for anyone to connect up to and start guessing usernames and passwords -- or "credential stuffing" which is the term that's now in vogue for brute force guessing logon credentials.

But CheckPoint research became curious about the security of the CLIENT connected to RDP servers. The RDP protocol is extremely complex and was developed in the dark (as opposed to "in the light of day") as a proprietary protocol by Microsoft. This hasn't prevented its reverse engineering by others and several 3rd-party and open source cline clones now exist.

But "a complex protocol" should give any security-aware person pause... Since we know that complexity is the enemy of security, no one should be surprised to learn that the RDP clients are rife with exploitable flaws.

Checkpoint wrote:
Overview
Used by thousands of IT professionals and security researchers worldwide, the Remote Desktop Protocol (RDP) is usually considered a safe and trustworthy application to connect to remote computers. Whether it is used to help those working remotely or to work in a safe VM environment, RDP clients are an invaluable tool.

However, Check Point Research recently discovered multiple critical vulnerabilities in the commonly used Remote Desktop Protocol (RDP) that would allow a malicious actor to reverse the usual direction of communication and infect the IT professional or security researcher's computer. Such an infection could then allow for an intrusion into the IT network as a whole.

16 major vulnerabilities and a total of 25 security vulnerabilities were found overall.

Checkpoint examined the following clients:

- mstsc.exe – Microsoft's built-in RDP client.
- FreeRDP – The most popular and mature open-source RDP client on Github.
- rdesktop – Older open-source RDP client, comes by default in Kali-linux distros.

Of "rdesktop" they noted: Since "rdesktop" is the built-in client in Kali-linux, a Linux distro commonly used by red teams for penetration testing, an intriguing reverse-attack scenario would be Blue teams installing organizational RDP honeypots to attack red teams try to connect to them through the RDP protocol.

CheckPoint wrote:

As is usually the case, we decided to start looking for vulnerabilities in the open source clients. It seems that it will only make sense to start reverse engineer Microsoft's client after we will have a firm understanding of the protocol. In addition, if we find common vulnerabilities in the two open sourced clients, we could check if they also apply to Microsoft's client. In a recon check it looked like "rdesktop" is smaller than "FreeRDP" (has fewer lines of code), and so we selected it as our first target.

We decided to perform an old-fashioned manual code audit instead of using any fuzzing technique. The main reasons for this decision were the overhead of writing a dedicated fuzzer for the complex RDP protocol, together with the fact that using AFL for a protocol with several compression and encryption layers didn't look like a good idea. (AFL: American Fuzzy Lop)

rdesktop / Tested version: v1.8.3

After a short period, it looked like the decision to manually search for vulnerabilities paid off. We soon found several vulnerable patterns in the code, making it easier to "feel" the code, and pinpoint the locations of possible vulnerabilities.

We found 11 vulnerabilities with a major security impact, and 19 vulnerabilities overall in the library.

The xrdp open-source RDP server is based on the code of "rdesktop" which led us to do a bit of additional recon on xrdp. Based on our findings, it appears that similar vulnerabilities can be found in "xrdp" as well.

Instead of a technical analysis of all of the CVEs, we will focus on two common vulnerable code patterns that we found.

I cannot describe the details of the vulnerabilities in an audio podcast format, but suffice to say that they spotted a repeated style of coding that was flawed in its approach and assumptions which could (and did) lead to, as they put it, "massive heap-based buffer overflows".

They wrote: By chaining together these two vulnerabilities, each pair found in three different logical communications channels, we now have three remote code execution vulnerabilities.

Detailing another instance of a vulnerability, they wrote:

CVE 2018-8795 – Remote Code Execution – Another classic vulnerability is an Integer-Overflow when processing the received bitmap (for screen content) updates. Although the bitmap "width" and "height" parameters are only 16 bits each, by multiplying them together with "Bpp" (bits-per-pixel), we can trigger an Integer-Overflow. Later on, the bitmap decompression will process our input and break on any decompression error, giving us a controllable heap-based buffer-overflow.

This calculation can be found in several places throughout the code of "rdesktop", so we marked it as a potential vulnerability to check for in "FreeRDP".

Of the FreeRDP client they wrote:

FreeRDP / Tested version: 2.0.0-rc3

After finding multiple vulnerabilities in "rdesktop", we approached "FreeRDP" with some trepidation; perhaps only "rdesktop" had vulnerabilities when implementing RDP? We still can't be sure that every implementation of the protocol will be vulnerable.

And indeed, at first glance, the code seemed much better: there are "minimum size checks" before parsing data from the received packet (which was the primary feature missing from rdesktop), and the code "feels" more mature. It is going to be a challenge. However, after a deeper examination, we started to find cracks in the code, and eventually we found critical vulnerabilities in this client as well.

We found 5 vulnerabilities with major security impact, and 6 vulnerabilities overall in the library.

Some additional sleuthing discovered that the RDP client NeutrinoRDP is a fork of an older version (v1.0.1) of "FreeRDP" and therefore probably suffers from the same vulnerabilities.

At the end of our research, we developed a PoC exploit for CVE 2018-8786.

Additionally: CVE 2018-8787 – Same Integer-Overflow – As we saw earlier in "rdesktop", calculating the dimensions of a received bitmap update is susceptible to Integer-Overflows. And indeed, "FreeRDP" shares the same vulnerability.

There is an Integer-Truncation when trying to calculate the required capacity for the bitmap updates array. Later on, rectangle structures will be parsed from our packet and into the memory of the too-small allocated buffer. (Whoops!) This specific vulnerability is followed by a

controlled amount (“bitmapUpdate->number”) of heap allocations (with a controlled size) when the rectangles are parsed and stored to the array, granting the attacker a great heap-shaping primitive. The downside of this vulnerability is that most of the rectangle fields are only 16 bits wide, and are upcasted to 32 bits to be stored in the array. Despite this, we managed to exploit this CVE in our PoC. Even this partially controlled heap-based buffer-overflow is enough for a remote code execution.

As I was reading through this research one thing kept pinging: These clients were written with the assumption that they would always and only be connecting to a well-behaved RDP server. They inherently took the server's provided parameters at face value since, afterall, it was a trusted Windows operating system they were connecting to, right?

So... What about Microsoft's own original RDP client? The one that's built into all of our Windows machines today??

They wrote:

We started by testing our PoCs for the vulnerabilities in the open-source clients. Unfortunately, all of them caused the client to close itself cleanly, without any crash. Having no more excuses, we opened IDA (Interactive DisAssembler) and started to track the flow of the messages. Soon enough, we realized that Microsoft’s implementation is much better than the implementations we tested previously. Actually, it seems like Microsoft’s code is better by several orders of magnitude, as it contains:

- Several optimization layers for efficient network streaming of the received video.
- Robust input checks.
- Robust decompression checks, to guarantee that no byte will be written past the destination buffer.
- Additional supported clipboard features.
- …

Needless to say, there were checks for Integer-Overflows when processing bitmap updates.

However, the CheckPoint researchers DID find some troubling flaws in Clipboard Sharing over RDP. It is possible for a malicious RDP server to use a classic path traversal attack to place files anywhere on the user's system limited only by the user's current privileges. Since users an change their own startup folders, malicious files could be placed there for eventual execution.

The researchers also found that the RDP server could dynamically monitor the client and capture anything the client copied to their clipboard -- like a complex admin password -- even if it was only briefly present.

THE DISCLOSURE TIMELINE:
16 October 2018 – Vulnerability was disclosed to Microsoft.
22 October 2018 – Vulnerabilities were disclosed to FreeRDP.
22 October 2018 – FreeRDP replied and started working on a patch.

28 October 2018 – Vulnerabilities were disclosed to rdesktop.
5 November 2018 – FreeRDP sent us the patches and asked for us to verify them.
18 November 2018 – We verified FreeRDP patches; gave them a "green light" to continue.
20 November 2018 – FreeRDP committed the patches to their Github as part of 2.0.0-rc4.
17 December 2018 – Microsoft acknowledged our findings. (See Microsoft's Response below.)
19 December 2018 – rdesktop sent us the patches and asked us to verify them.
19 December 2018 – We verified the rdesktop patches; gave them a "green light" to continue.
16 January 2019 – rdesktop committed the patches to their Github as part of v1.8.4.

Microsoft replied: "*Thank you for your submission. We determined your finding is valid but does not meet our bar for servicing. [In other words... We have MUCH bigger problems over here.] For more information, please see the Microsoft Security Servicing Criteria for Windows (https://aka.ms/windowscriteria).*"

CheckPoint writes: As a result, this path traversal has no CVE-ID, and there is no patch to address it.

Admittedly, the probable risk of connecting to a malicious Windows server is low for most users. But it's conceivable that there are high-value clients who have some reason to connect to not-fully-trusted Windows machines. So... All of this would be worth keeping in mind.

CheckPoint recommends updating to the latest releases of rdesktop and FreeRDP. And they suggest that if a user of Windows' native Remote Desktop has and reason not to trust the machine they're connecting to... Disabling the default-enabled clipboard sharing -- since Microsoft apparently has no intention of fixing it -- would be prudent.

LibreOffice and Apache OpenOffice RCE's
https://insert-script.blogspot.com/2019/02/libreoffice-cve-2018-16858-remote-code.html

And speaking of classic Path Traversal vulnerabilities...
Security researcher Alex Inführ has discovered a severe remote code execution (RCE) vulnerability in these two open source office suites that could be triggered just by opening a maliciously-crafted ODT (OpenDocument Text) file. The attack relies on exploiting a directory traversal flaw, which has since been assigned the CVE-2018-16858. This automatically executes a specific python library bundled within the software using a hidden onmouseover event.

To exploit this vulnerability, Alex created an ODT file with a white-colored hyperlink (so it can't be seen) that has an "onmouseover" event to trick victims into executing a locally available python file on their system when placing their mouse anywhere on the invisible hyperlink.

According to the researcher, the python file, named "pydoc.py," that comes included with the LibreOffice's own Python interpreter accepts arbitrary commands in one of its parameters and execute them through the system's command line or console.
Inführ provided a proof-of-concept (PoC) video demonstration showing how he was able to trick the event into calling a specific function within a Python file, which eventually executed the researcher's payload through Windows command line (cmd) without showing any warning dialog to the user.

Alex wrote that reporting the bug was, in his words, "kind of a wild ride." He said:

At first I reported it via the libreoffice bugzilla system. Apparently for security issues it is better to send an email to officesecurity@lists.freedesktop.org, but I did not know that. So my bugzilla report got closed but I convinced them to have another look. The bug was picked up and moved to a thread via officesecurity@lists.freedesktop.org. The issue was verified and fixed quite fast.

Timeline:
- 18.10.2018 - reported the bug
- 30.10.2018 - bug was fixed and added to daily builds
- 14.11.2018 - CVE-2018-16858 was assigned by Redhat - got told that 31.01.2019 is the date I can publish
- 01.02.2019 - Blogpost published

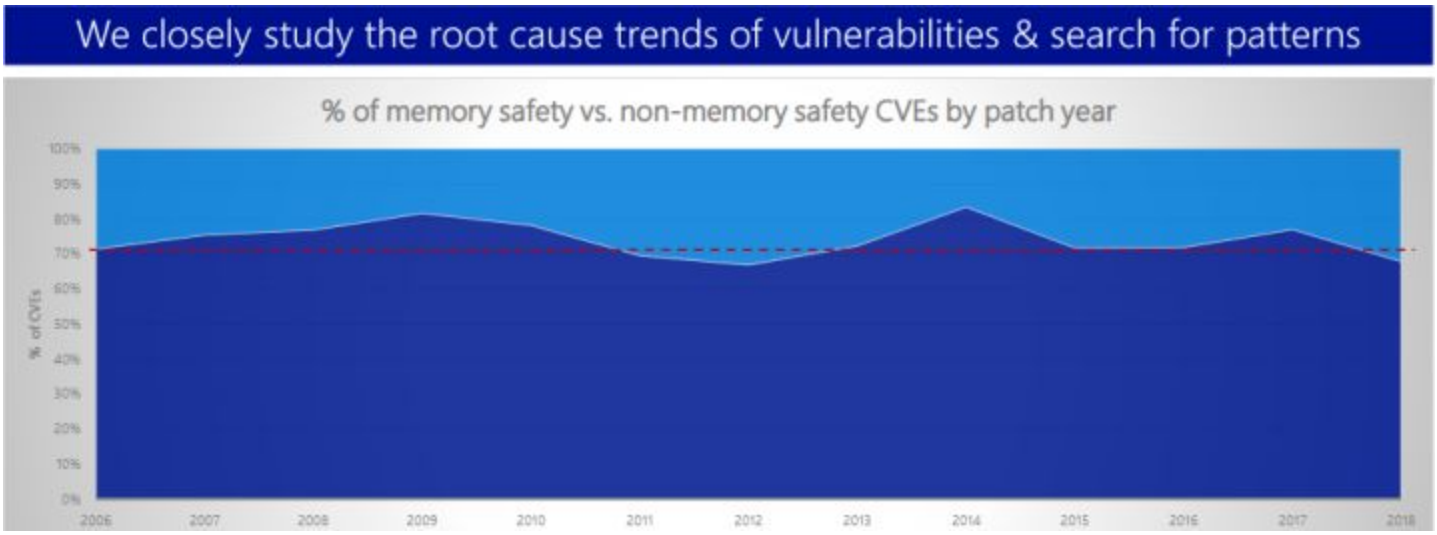
The path traversal is fixed in (I just tested these versions):
- Libreoffice: 6.1.4.2
- Libreoffice: 6.0.7

Vulnerable:
- Openoffice: 4.1.6 (latest version)

I reconfirmed via email that I am allowed to publish the details of the vulnerability although openoffice is still unpatched. Openoffice does not allow to pass parameters, therefore my PoC does not work but the path traversal can still be abused to execute a python script from another location on the local file system.

To disable the support for python, the pythonscript.py in the installation folder can be either removed or renamed (example on linux /opt/openoffice4/program/pythonscript.py)

Microsoft says 70 percent of all security bugs are memory safety issues
And that percentage hasn't budged much, nor consistently through the past 12 years.



Speaking at the BlueHat security conference in Israel last week, Microsoft security engineer Matt Miller said that over the last 12 years, around 70 percent of all Microsoft patches were fixes for memory safety bugs.

As we know, these go by terms we use here all the time: buffer overflow, race condition, page fault, null pointer, stack exhaustion, heap exhaustion/corruption, use after free, or double free.

The reason, Matt explained, for this high percentage is because Windows has been written mostly in C and C++, two "memory-unsafe" programming languages that allow developers fine-grained control of the memory addresses where their code can be executed and where they data can be stored. Just a simple mistake in the developers' management of memory can lead to exploitable memory errors that attackers can exploit to obtains remote code execution or the elevation of their privilege within the system.

This all makes memory management and safety errors today's biggest attack surface for hackers, and they are the error attackers are capitalizing upon. Miller's presentation asserted that "use after free" and "heap corruption" vulnerabilities remain the preferred bugs for attackers searching for exploitable vulnerable code behavior.

So... No big or surprising news there, but it's interesting to put a number on the problem, and it's good that Microsoft is paying attention. I've been mentioning recently that it's somewhat surprising to see how deeply entrenched existing computing technology actually is -- meaning how reluctant we appear to be to really change anything fundamental, despite the fact that fundamental change is what we clearly need.

To first note my own hobby horse: When we use a username and password to logon to a computer or to a remote website we're using the same technology that mainframe terminals used in the early 1970's... which is now nearly 50 years ago!

And, similarly, coding has not appreciably changed through all that time. Oh, sure, there have been interesting academic experimental "memory safe" languages, but for many different reasons they never gain a foothold. And if the language itself is safe, they're still running on old school hardware architectures and being compiled or, more likely, interpreted by buggy interpreters running on old school hardware. So it never really gets anywhere.

And then there's programmer hubris -- never to be underestimated in scope and depth. No programmer ever thinks they're going to make a mistake. And when they inevitably do it's considered "a mistake" -- oops! -- and it's treated as though it won't ever happen again. But of course, it will and it does... over and over and over.

If this is going to change at some point, we need to rethink the whole thing. “The rethink” would probably need to come down from some ivory tower somewhere. But I also wonder "will it really ever change?” Is there really, truly, the motivation? People say they want security, but they don't insist upon it. They complain when something is insecure, but they cannot SEE security -- they don't even understand what security is -- it's utterly intangible. And we smugly lock the front doors on homes with easily broken glass windows. It’s well understood that Apple’s iOS based devices are significantly more secure than Android, yet most Android devices are much less expensive, so that’s what most people buy.

And as we've just noted, Apple has problems too. So not only don't and won't people actually pay more for more security, this industry still doesn't have true security to sell them… do we?

The cost of remaining with Windows 7 after this time next year…
https://www.zdnet.com/article/how-much-will-staying-patched-on-windows-7-cost-you-heres-the-price-list/

Windows 7 Extended Security Updates

	Windows Enterprise (add on)	Windows 7 Pro
Year 1 (Jan '20-Jan '21)	• $25/device	• $50/device
Year 2 (Jan '21-Jan '22)	• $50/device	• $100/device
Year 3 (Jan '22-Jan '23)	• $100/device	• $200/device

- ❑ Yearly price gives access to cumulative security updates over 12 months
- ❑ Purchases in years 2 and 3 require payments for of prior year(s)
- ❑ Ubiquitous availability in all programs and channels (VL, CSP)
- ❑ No minimum purchase necessary

Microsoft said last Fall that it would offer paid Windows 7 Extended Security Updates on a per-device basis for big customers willing to pay for them after the company ends Windows 7 support on January 14, 2020. Microsoft officials wouldn't talk about how much those updates would cost, beyond saying they'd get more expensive over time.

However, Microsoft has briefed some of its partners and salespeople about the cost of these Extended Support Updates (ESUs). And, as you'd expect, they're not cheap, especially for customers who may want to apply them on multiple PCs. They're even more expensive for customers using the Pro version of Windows than the Enterprise one.

Last Fall, Microsoft officials said they would provide Windows 7 Extended Security Updates for three years, meaning through January 2023. These will be security patches/fixes like the ones Microsoft is currently providing for free for Windows 7 users, as Windows 7 is still in "Extended" Support through January 14, 2020.

As Microsoft officials said previously, the ESUs are for larger business and education customers only. They will be available to any Windows 7 Professional and Windows 7 Enterprise users with volume-licensing agreements. Office 365 ProPlus will continue to work on devices with Windows 7 Extended Security Updates through January 2023

China's cybersecurity law update lets state agencies 'pen-test' local companies
China draws up law that makes it perfectly legal to hack any internet-related company activating in its borders.
https://www.zdnet.com/article/chinas-cybersecurity-law-update-lets-state-agencies-pen-test-local-companies/

Last week we noted that the Japanese government had announced plans to scan their own public Internet space for exploitable vulnerability exposures. I wanted to follow-up by also noting that last November the Chinese government gave itself sweeping power to perform similar (and likely more intrusive) scans of its own Internet space. The newly enacted law allows the Ministry of Public Security (MPS), the same agency which also maintains China's Great Firewall and its nationwide facial recognition system and surveillance cameras network, to:

- Conduct in-person or remote inspections of the network security defenses taken by companies operating in China.
- Check for "prohibited content" banned inside China's border.
- Log security response plans during on-site inspections.
- Copy any user information found on inspected systems during on-site or remote inspections.
- Perform penetration tests to check for vulnerabilities.
- Perform remote inspections without informing companies.
- Share any collected data with other state agencies.
- The right to have two members of the People's Armed Police (PAP) present during on-site inspection to enforce procedures.

Meanwhile... Russia prepares to disconnect from the Internet.
Russia's internet contingency plan gets closer to reality.

Russian authorities and major Russian internet providers are planning to disconnect the country from the internet as part of a planned experiment. The ultimate goal is for Russian authorities to implement a web traffic filtering system like China's Great Firewall and to also arrange to have a fully working self-contained country-wide intranet in case the country needs to disconnect.

"Why do we need rest of world?"

The initial experiment, intended to occur before, but hopefully not on, April 1st, will serve to provide insight and feedback, and possible modifications to a proposed law to this end introduced in the Russian Parliament in December of last year.

A first draft of the law mandated that Russian internet providers should by able to ensure the independence of the Russian internet space (Runet) in the case of foreign aggression, to disconnect the country from the rest of the internet. Additionally, Russian networking firms would have to install "technical means" to re-route all Russian internet traffic to exchange points approved or managed by Roskomnazor, Russia's telecom watchdog. Roskomnazor will inspect the traffic to block prohibited content, make sure traffic between Russian users stays inside the country, and is not re-routed "needlessly" through servers abroad, where it could be intercepted.

The Russian government has been working on this project for years and in 2017, Russian officials said they plan to route 95 percent of all internet traffic locally by 2020. To pull this off, they'll need their own DNS. So authorities have built an in-country backup of the Domain Name System (DNS). It was first tested in 2014, and then again last year. It will be a major component of "Runet" when ISPs plan to pull the plug and disconnect the country from the rest of the world.

Firefox is again rolling out it's Release 65
The conflicting A/V vendors have back off of scanning into Firefox since Firefox has become aggressive about informing its users when anything --malware or anti-maleware-- is intercepting and decrypting the its user's traffic.

Since I'm using the traffic scanner built into Windows --Defender-- I get to use Firefox and still have all of my traffic scanned.

SQRL

- **Domsch:**
  Just wanted to chime in. I created my first sqrl ID months back without having a use for it. Of course i lost the restore codes. So after listening to SN-700 on my Commute yesterday and today, i set up a new Identity. Created on my Android phone with the Awesome Android App, signed on to the Forums in seconds, and 5 Minutes later, signed on to the forums on my work PC without any Problems. This is awesome! The biggest thing for me: It's independent of a password manager. The Password Manager extension is one of the big things keeping me with firefox. I'd really like to switch to falkon, but without my Passwords that's a no-go.

  So here is me hoping SQRL will find wide adoption on all Platforms, so i can finally leave my Password manager behind. Thanks Steve for creating this awesome authentication method. And especially for not going with the "Early-Bird, Alpha, Pre-Order, Green-Light" Approach that's so common nowadays but instead doing it right and delivering an actually 100% working Product!

- **CormaP:**
  I installed the Android SQRL client a week or 2 ago and have just used it (to register here) for the 1st time. I foolishly forgot my password for my identity.

  - So I reset the password using the Rescue Code - Painless
  - Logged in to the Forums using SQRL - Painless
  - Setup 2-factor using the LastPass authenticator - Painless

  All good so far. My question is, can I avoid having to reenter my password when logging in to a site (on my phone?) or can I just use the phones fingerprint sensor? Or is it simply because I have only logged in twice? Am I missing the paint/doing something wrong?

  Thanks, Cosma

(Only the latest Android Pie has secure fingerprint unlock. So, there, yes! And on iOS with thumb and face ID.)

- KenRS:
  Jeff (writing to Jeff Arthur about his SQRL client for iOS) Just trying your client on this forum and it's great to see it working. The identity import using the printed QR code worked perfectly. Once set up, I scanned the SQRL QR code appearing on my laptop screen with my iPad and was instantly logged in on the laptop - that part does seem like magic.

  Logging in "on-device" on the iPad also worked. The only thing that still is rough is signing in using the finger-print authentication on the iPad. Instead of logging in instantly I'm left with safari searching for "localhost" . Cancelling that leaves me logged in.
  Regards
  Ken

So... where is SQRL?: I am sharing these anecdotes while a great deal of work is underway behind the scenes. It has become clear to the authors of the other SQRL clients that this bread is just about baked, and that SQRL really does work and is actually going to happen. So work is proceeding at an increased pace. I'm working to flesh out the SQRL forum content so there's material there for newcomers, and at the same time I need to update SQRL's documentation so that those who want to evaluate its underlying technology and operation will be able to do so.

Closing The Loop

Dave (@darkmatter_0x00)
Hi Steve, great podcast. Quick question, a few episodes ago you mentioned a small firewall VPN device/router that can be bought, what was the name of it? Only has a few ports, and ran pfsense... maybe I'm wrong about the pf part. Thanks!

Bldg35 (@Bldg35)
Hi, just wanted to say thanks for planting the idea in my head, regarding pfSense. I've got my SG-1100 on-line now. Reconfigured my Asus RT-AC5300 to AP/AiMesh mode. I was surprised that I didn't lose any WiFi features, such as Guest node isolation.

It didn't take too long to get an OpenVPN server running in pfSense, for remote access. Also, for some reason, internet access feels noticeably faster. For instance, my 10 ring security cameras now come up on my phone, much faster in live view, and recorded videos. Not sure why that is? But, I'll take it!??

Andy Blak (@Andy_Blak_)
Hey Steve!
Recently, you mentioned the Netgate pfSense products on SN. I purchased one after the show for my home network. Though I'm having trouble configuring it. Do you have any recommendations for who could assist with this? Netgate charges $900+ minimum for their support (I called them). Just looking for a more reasonable option.

Many thanks, Andy

Adiantum

(add-eee-ant-tum)

Google does the Cha-Cha!

https://thehackernews.com/2019/02/fast-adiantum-file-encryption.html
https://cr.yp.to/chacha.html
https://security.googleblog.com/2019/02/introducing-adiantum-encryption-for.html
https://opensource.google.com/projects/adiantum
https://www.blog.google/technology/safety-security/adiantum-encryption-safer-devices/
https://www.zdnet.com/article/googles-adiantum-is-a-new-form-of-encryption-for-your-mobile-device/
https://threatpost.com/google-boosts-encryption-for-low-end-android-devices/141658/

How can crucially-important full-disk encryption be offered on inexpensive commodity hardware lacking any encryption acceleration?

Think about this for a moment: Many of us now take for granted the fact that our devices -- at rest -- are fully encrypted, meaning that the mass storage our devices carry cannot simply be read out as a file system by any either good or bad guys who want to know what's in there.

But as we've mentioned before, this is not reality for many users of lower-end Android Smartphones who, wanting full security, activate their device's whole-disk encryption only to suffer such a reduction in performance that they are effectively forced to back out of that and decrypt their drive and run with their device unencrypted.

The trouble is, our chosen best-of-class cipher, the Rijndael cipher which was finally chosen to be the AES (Advanced Encryption Standard) is wonderfully secure, but only at the significant cost of per-byte encryption performance. And it's bad enough, that back in 2010 when Intel introduced their new Intel Core processor family "Westmere" that new silicon, and all silicon since, has incorporated six new instructions known as AES-NI (for "AES New Instructions" because all the cooler names were already taken) to specifically speed up the use of the AES cipher on Intel architectures. It's not that AES cannot be done with out them. It can, of course. But each of those six instructions is specific to what the AES cipher algorithm needs, so that each is able to replace a great many more standard instructions to get the same job done much more quickly.

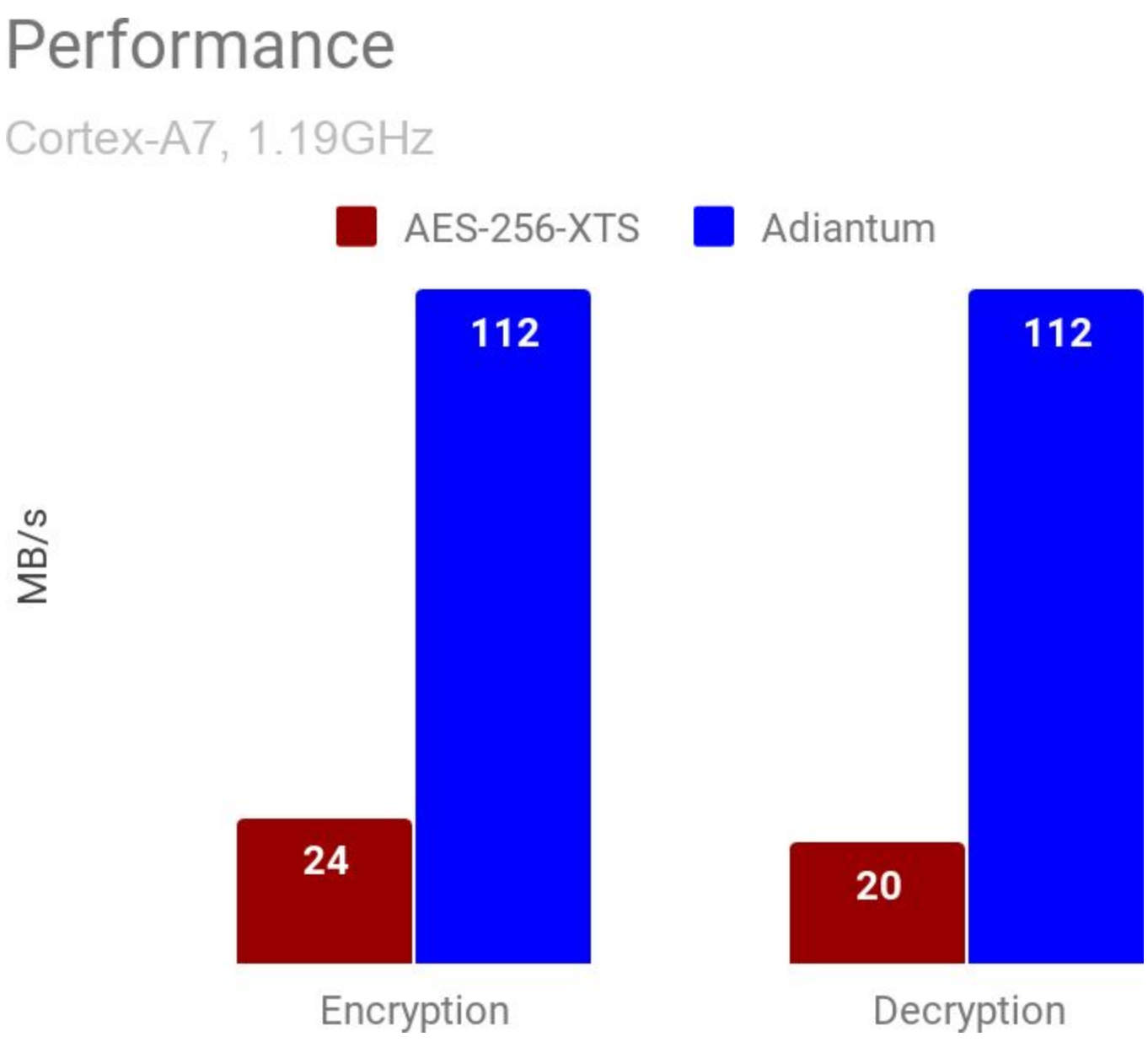
The point being... AES rocks, but not so quickly without some help.

And also, as we've discussed before, the reason encrypting the entire drive is so sensitive to the performance of its encipherment is that "bulk enciphering" is happening continuously, on the fly, for every byte of data read and written to the drive. The cipher is "in-line" and cannot be bypassed.

So, the bright lads at Google decided to take aim at this problem to see how they might improve upon the situation. Since it's simply not possible to speed up the Rijndael cipher without

hardware they knew that they would need to replace Rijndael with something else. This is feasible in any closed system where the cipher is only for internal use. For example, when transacting over TLS there is no choice other than to use AES since it's used in an "external standard" for communication with other standards-following servers. But internally, deep inside an Android device, Google is 100% free to choose anything that works.

And what works... is ChaCha!



On ARM Cortex-A7, Adiantum encryption and decryption on 4096-byte sectors is about 10.6 cycles per byte, around 5x faster than AES-256-XTS.

I smiled when I saw ChaCha since it is the brainchild of the same person who designed the 25519 elliptic curve cipher that is entirely responsible for enabling SQRL's solutions. That person is Daniel J. Bernstein.

Google's Online Security Blog for last Thursday, February 7th, is titled:

“Introducing Adiantum: Encryption for the Next Billion Users”

Here's what Google wrote, which says it perfectly:

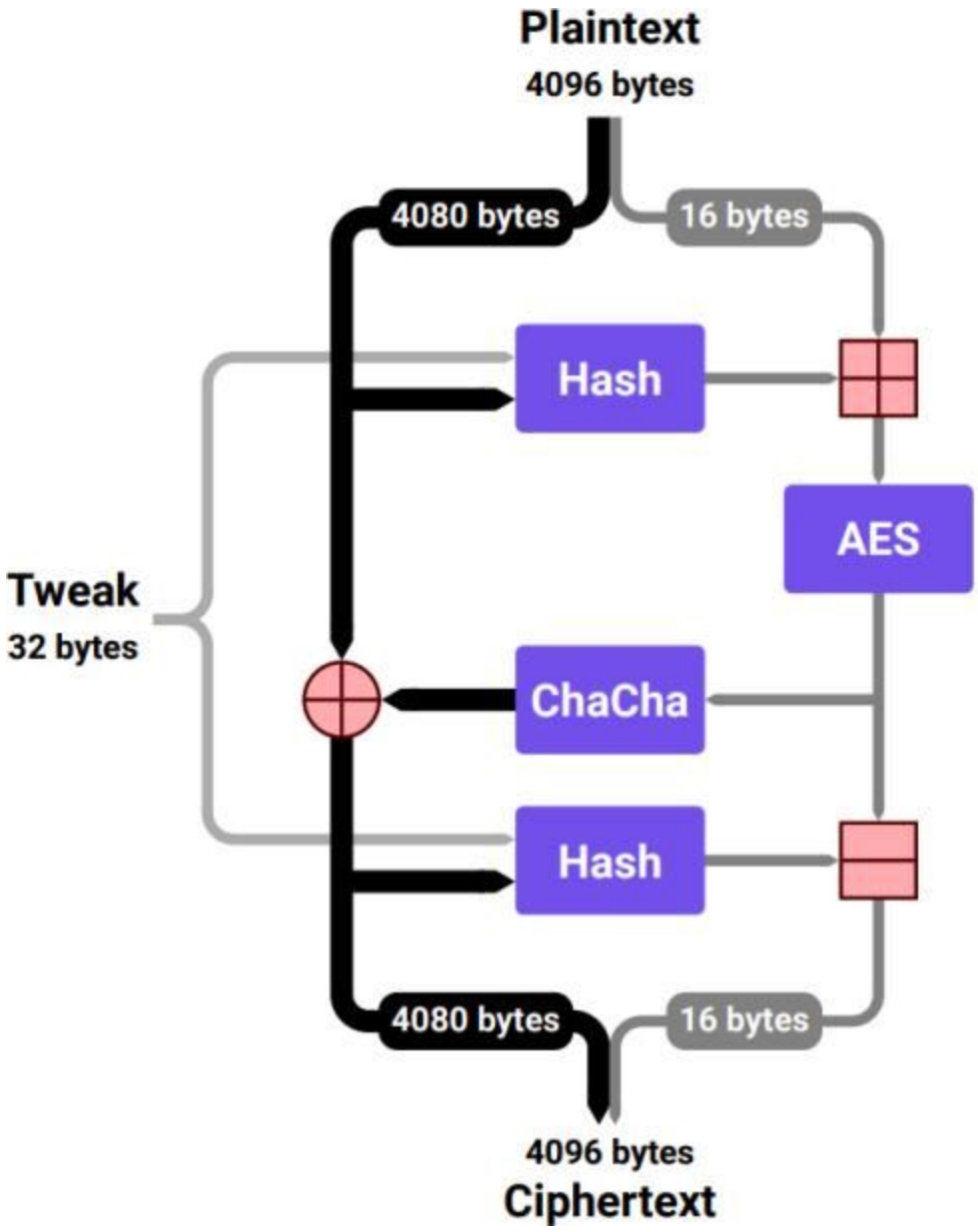
Today, Android offers storage encryption using the Advanced Encryption Standard (AES). Most new Android devices have hardware support for AES via the ARMv8 Cryptography Extensions. However, Android runs on a wide range of devices. This includes not just the latest flagship and mid-range phones, but also entry-level Android Go phones sold primarily in developing countries, along with smart watches and TVs. In order to offer low cost options, device manufacturers sometimes use low-end processors such as the ARM Cortex-A7, which does not have hardware support for AES. On these devices, AES is so slow that it would result in a poor user experience; apps would take much longer to launch, and the device would generally feel much slower. So while storage encryption has been required for most devices since Android 6.0 in 2015, devices with poor AES performance (50 MiB/s and below) are exempt. We've been working to change this because we believe that encryption is for everyone.

In HTTPS encryption, this is a solved problem. The ChaCha20 stream cipher is much faster than AES when hardware acceleration is unavailable, while also being extremely secure. It is fast because it exclusively relies on operations that all CPUs natively support: additions, rotations, and XORs. For this reason, in 2014 Google selected ChaCha20 along with the Poly1305 authenticator, which is also fast in software, for a new TLS cipher suite to secure HTTPS internet connections. ChaCha20-Poly1305 has been standardized as RFC7539, and it greatly improves HTTPS performance on devices that lack AES instructions.

However, disk and file encryption present a special challenge. Data on storage devices is organized into "sectors" which today are typically 4096 bytes. When the filesystem makes a request to the device to read or write a sector, the encryption layer intercepts that request and converts between plaintext and ciphertext. This means that we must convert between a 4096-byte plaintext and a 4096-byte ciphertext. But to use RFC7539, the ciphertext must be slightly larger than the plaintext; a little space is needed for the cryptographic nonce and message integrity information. There are software techniques for finding places to store this extra information, but they reduce efficiency and can impose significant complexity on filesystem design.

Where AES is used, the conventional solution for disk encryption is to use the XTS or CBC-ESSIV modes of operation, which are length-preserving. Currently Android supports AES-128-CBC-ESSIV for full-disk encryption and AES-256-XTS for file-based encryption. However, when AES performance is insufficient there is no widely accepted alternative that has sufficient performance on lower-end ARM processors.
To solve this problem, we have designed a new encryption mode called Adiantum. Adiantum allows us to use the ChaCha stream cipher in a length-preserving mode, by adapting ideas from AES-based proposals for length-preserving encryption such as HCTR and HCH. On ARM Cortex-A7, Adiantum encryption and decryption on 4096-byte sectors is about 10.6 cycles per byte, around 5x faster than AES-256-XTS.

Unlike modes such as XTS or CBC-ESSIV, Adiantum is a true wide-block mode: changing any bit anywhere in the plaintext will unrecognizably change all of the ciphertext, and vice versa. It works by first hashing almost the entire plaintext using a keyed hash based on Poly1305 and another very fast keyed hashing function called NH. We also hash a value called the "tweak" which is used to ensure that different sectors are encrypted differently. This hash is then used to generate a nonce for the ChaCha encryption. After encryption, we hash again, so that we have the same strength in the decryption direction as the encryption direction. This is arranged in a configuration known as a Feistel network, so that we can decrypt what we've encrypted. A single AES-256 invocation on a 16-byte block is also required, but for 4096-byte inputs this part is not performance-critical.



Cryptographic primitives like ChaCha are organized in "rounds", with each round increasing our confidence in security at a cost in speed. To make disk encryption fast enough on the widest range of devices, we've opted to use the 12-round variant of ChaCha rather than the more widely used 20-round variant. Each round vastly increases the difficulty of attack; the

7-round variant was broken in 2008, and though many papers have improved on this attack, no attack on 8 rounds is known today. This ratio of rounds used to rounds broken today is actually better for ChaCha12 than it is for AES-256.

Even though Adiantum is very new, we are in a position to have high confidence in its security. In our paper, we prove that it has good security properties, under the assumption that ChaCha12 and AES-256 are secure. This is standard practice in cryptography; from "primitives" like ChaCha and AES, we build "constructions" like XTS, GCM, or Adiantum. Very often we can offer strong arguments but not a proof that the primitives are secure, while we can prove that if the primitives are secure, the constructions we build from them are too. We don't have to make assumptions about NH or the Poly1305 hash function; these are proven to have the cryptographic property ("ε-almost-Δ-universality") we rely on.

Adiantum is named after the genus of the maidenhair fern, which in the Victorian language of flowers (floriography) represents sincerity and discretion.

~ ~ ~

The reason ChaCha is so much faster than AES on general purpose processors lacking specific support for AES is that ChaCha is based on the ARX pattern (Addition-Rotation-XOR) which are all CPU-friendly instructions. By comparison, AES uses binary fields for the S-box and Mix-Columns computations, which are generally implemented as look up tables to improve performance. But the fact that AES uses lookup table with an index derived from the secret makes implementations vulnerable to cache-timing attacks. ChaCha20 is not vulnerable to such attacks. (AES implemented through AES-NI is also not vulnerable.)

Using traditional XTS mode for whole disk encryption, a one-bit change to the plaintext results in only a 16-byte change to the ciphertext, which reveals more to the attacker than necessary. By comparison the Adiantum cipher mode is a “super pseudorandom permutation” over the whole sector. This means that **any** change to the plaintext of the sector results in an unrecognizably different ciphertext sector and vice versa.

Also…
Adiantum was designed in 2018 by Paul Crowley and Eric Biggers at Google specifically for low-powered mobile devices running Android Go. It's available in the Linux kernel since version 5.0. Originally Google was planning to use Speck but decided against it due to the cipher's ties with National Security Agency (NSA).

~ 30 ~