Security Now! #714 - 05-14-19

Android 'Q'

This week on Security Now!

This week we look at a widespread problem affecting all WhatsApp users, many interesting bits of new arising from last week's Google I/O 2019 conference, a worrisome remotely exploitable flaw in all Linux kernels earlier than v5.0.8, the just-released hours ago new set of flaws affecting all Intel processors known as "ZombieLoad", a bit of miscellany and odds and ends, then we take a deep look into the significant security enhancements Google also announced in their next release of Android... "Q".

VintageBeef @VintageBeefLP I gotta wait until my oven installs new firmware before I can make dinner. #TheFutureIsNow

Please wait while we update your system to the latest firmware. When it completes, you will see the screen go blank as it restarts.

It may take up to 30 minutes. Please do not unplug the appliance or turn off the Internet connection.

Security News

Update Your WhatsApp App

The 1.5 billion systems running mobile WhatsApp, including iOS, Android and Windows phone, can be infected with a potent malware created by the Israeli NSO Group. Exploiting a mobile handset simply requires placing a WhatApp call. Answering the call is not required.

FaceBook Security Advisory:
https://www.facebook.com/security/advisories/cve-2019-3568

CVE-2019-3568
Description: A buffer overflow vulnerability in WhatsApp VOIP stack allowed remote code execution via specially crafted series of SRTCP packets sent to a target phone number.
Affected Versions: The issue affects:

- WhatsApp for Android prior to v2.19.134
- WhatsApp Business for Android prior to v2.19.44
- WhatsApp for iOS prior to v2.19.51
- WhatsApp Business for iOS prior to v2.19.51
- WhatsApp for Windows Phone prior to v2.18.348, and

The exact status of the vulnerability was a bit unclear. In one instance a WhatsApp representative said the vulnerability was fixed in updates released on Friday. But others were told by WhatsApp that Friday's fix was made to the company's servers to prevent attacks from working. Regardless, WhatsApp released across the board updates yesterday for all end users.

The NSO Group makes "Pegasus", an advanced system penetration malware that jailbreaks or roots a targeted mobile device that allows spyware to then examine private message history, activate the device's microphone and camera, and collect all manner of sensitive information.

Today the NSO Group will be facing a challenge in Israeli court regarding its ability to export its commercial software. The challenge comes from Amnesty International and other human rights groups who allege that it is being used to target human rights attorneys and others. The NSO Group claims that their potent software tools are only used by and for legitimate law-enforcement agencies.

Google's forthcoming change for "Same Site" Cookies.

Published last Tuesday, by Google's Mike West, is an IETF Network Working Group Internet-Draft titled: Incrementally Better Cookies.

https://tools.ietf.org/html/draft-west-cookie-incrementalism-00
https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies
https://web.dev/samesite-cookies-explained/

1. Introduction

The HTTP State Tokens proposal ([I-D.west-http-state-tokens]) aims to replace cookies with a state management mechanism that has better security and privacy properties. That proposal is somewhat aspirational: it's going to take a long time to come to agreement on the exact contours of a cookie replacement, and an even longer time to actually do so.

While we're debating the details of a new state management primitive, it seems quite reasonable to reevaluate some aspects of the existing primitive: cookies. When we can find consensus on some aspect of HTTP State Tokens, we can apply those aspirations to cookies, driving incremental improvements to state management in the status quo.

Based on conversations at [HTTP-Workshop-2019] and elsewhere, I'd suggest that we have something like agreement on at least two principles:

1. HTTP requests should not carry state along with cross-site requests by default (see Section 8.2 of [RFC6265bis]).

2. HTTP requests should not carry state over non-secure channels (see Section 8.3 of [RFC6265bis], and [RFC7258]).

Wow: "HTTP requests should not carry state along with cross-site requests by default." That just said that unless a cookie is explicitly declared as "cross site" it won't be returned by a web browser!

Traditionally, cookies could be tagged, when set, with either of two optional tags: HTTPONLY or SECURE. [[[EXPLAIN]]]

More recently, the "SAMESITE" tag has been added and it is supported by all current browsers except Internet Explorer.

A cookie's SameSite tag can have any one of three different settings: "None", "Lax" or "Strict". And if not set, it's default meaning would be "Lax".

Strict: If a same-site cookie carries the "Strict" attribute, the browser will only send cookies if the request originated from the website that set the cookie. If the request originated from a different domain than the domain of the current location, no cookies tagged with the Strict attribute will be sent.

Lax: If the attribute is set to Lax, same-site cookies are withheld on cross-site subrequests, such as calls to load images or frames, but WILL be sent when a user navigates to the URL from an external site, for example, by following a link.

And the default behavior if the flag is not set, which is the same as not supported by the browser, which is to include the cookies in any request, including cross-origin requests.

So, until now, no specification for the SameSite tag meant 'undefined' which meant that all cookies would always be sent.

Mark's proposal continues...

With those principles in mind, this document proposes two changes that seem possible to deploy in the near-term. User agents should:

1. Treat the LACK of an explicit "SameSite" attribute as "SameSite=Lax". That is, the "Set-Cookie" value "key=value" will produce a cookie equivalent to "key=value; SameSite=Lax". Cookies that require cross-site delivery can explicitly opt-into such behavior by asserting "SameSite=None" when creating a cookie.

2. Require the "Secure" attribute to be set for any cookie which asserts "SameSite=None" (similar conceptually to the behavior for the "__Secure-" prefix). That is, the "Set-Cookie" value "key=value; SameSite=None; Secure" will be accepted, while "key=value; SameSite=None" will be rejected.

We are currently at Chrome 74.

Chrome 76 will be introducing a new "same-site-by-default-cookies" flag. Setting this flag will shift the default behaviour of Set-Cookie to apply the meaning of SameSite=Lax if no other SameSite value is provided. So this is Google's move towards providing a more secure default and one that makes the intended purpose of cookies clearer to users.

With this flag enabled in Chrome, cookies without the SameSite attribute will be restricted to the same site the user is browsing. Cookies that need to be available in a third-party context will then need to be declared as such to the browser, and the user, by marking them as SameSite=None.

Mark West's second proposal "HTTP requests should not carry state over non-secure channels" proposes the elimination of non-SECURE cookies. Cookies simply won't be sent by any browser unless the channel is secure.

Everyone here knows how annoyed I have always been that cookies have long been abused for 3rd-party tracking when they were originally intended only to support 1st-party HTTP state.

But, of course the entire world of now glued together by 3rd-party cookies. So they are not going away anytime soon. But this incremental move does take a step toward making their intended use for cross-site linking clear and explicit. This initiative will give those who depend upon cross-site cookies time to simply add the "SameSite=None" term to their "Set-Cookie" headers and nothing will break. But... all of those cookies will then be explicitly labelled, which appears to be Google's point and intention.

And the default "Lax" case will be nice too. It will mean that unless explicitly instructed to send cookies for passive asset queries such as for images and other bits, if a browser has a cookie for that asset's site which is NOT marked as "SameSite=None", it will begin being treated as "SameSite=Lax" and will not automatically be sent. It's not a BIG deal, but it's an incremental improvement.

Google plans to add some anti-fingerprinting to Chrome.
Browser "Fingerprinting" as it's known, is the observation and combination of many individual weak browser identifiers. Any one identifier taken individually would never be useful for identifying a specific browser or user. But when many of these individually weak features are combined the whole, there may be many many fewer browsers having exactly the same "fingerprint." For example, the "User Agent" header often contains a bunch of version information. Although many other browsers will have exactly the same version information, many more will not. So now, with just this one bit of data, we have subdivided the world's entire web browser population into two sets. A much larger one that this browser is NOT a member of, and a much smaller set that does include this browser. It also turns out that browsers emit their query headers in distinctive sequences. So now we can further sub-divide the group we're in... and so on. Another often-probed signal is the exact type font collection installed on a machine. It's often the default. But when it's not it's another means of separating a system into a much smaller pool. Taken to its practical limit, identification can become quite narrow... And it IS being done on the web.

So Google's announcement that they're going to be blocking certain types of fingerprinting was interesting... and I'll be very interested to see what they do once it lands.

The first major browser to block fingerprinting was the Firefox/Tor Browser, which did so to thwart the deanonymization of its users. This feature was later incorporated into the mainstream Firefox browser as Mozilla was moving to adopt a more privacy-first approach which began back in late 2017.

So now Google has indicated that Chrome would be receiving anti-fingerprinting features as well. As Google phrased it "Because fingerprinting is neither transparent nor under the user's control, it results in tracking that doesn't respect user choice. This is why Chrome plans to more aggressively restrict fingerprinting across the web. One way in which we'll be doing this is reducing the ways in which browsers can be passively fingerprinted, so that we can detect and intervene against active fingerprinting efforts as they happen."

Also for Chrome: No more messing with the "Back" button!
We have all entered a search term into our favorite search site and received a list of search results.

We then examine the results and use our prior experience of various sites to decide which one, or ones, we wish to check out further.

So we select and click a link and jump to the target to have a look around. Perhaps that first guess doesn't have what we were hoping, but we recall that there were some other search results that looked hopeful. So we hit our browser's BACK button and... Nothing. We cannot go back to where we were. We're trapped.

Unfortunately, I've had this happen on Microsoft's developer site pages due to the redirection system they built. Microsoft has created quite a tangle with their attempt at a browser-based centralized identification and authentication system. Our browsers remember the path we took. So if we get redirected a few times -- essentially being auto-bounced forward -- then our browser's back button will only take us to the previous page that redirected us to where we were, or are, or... something. In the case of Microsoft's dumb redirection that clicking the back button multiple times quickly can often take me back faster than the browser will redirect forward... sort of like climbing up a hill of sand faster than the sand is coming down. So in those instances I'm able to escape.

But the nefarious behavior I'm talking about is definitely different. It's deliberate. It's designed and intended to thwart a visitor's attempt to easily leave a website we inadvertently stumbled into. This seems so dumb, since we're obviously ultimately going to win any such battle. It's such a desperate loser move. But it's definitely done.

My first recourse when I hit a site like that is to hold by browser's BACK button depressed. After a bit of a build in "are you sure" delay, you'll be rewarded with a list of that tab's previous pages in most-recently-visited-page-first order. So you scan down the list for the first occurrence of your search engine's logo, select that one, and you're back to choosing some other propitious looking link.

So what's happening? JavaScript allows pages to directly edit the "prior pages visited" stack, to push or pop entries from it. This allows an annoying website to push entries to itself into the stack, which will be subsequently "popped" by your browser when you push its back button. But since the current page pushed its own URL into the stack, popping the stack keeps you where you were, or are, or something.

Anyway... Google has decided that this sort of deliberate and sneaky behavior is NOT okay. Actually, it wasn't ever really Google. The participants in the Web Incubator Community Chapter (WICG) identified this annoying behavior back in 2016. WICG is a forum for discussing how to improve the web experience for users. But in November, the Chromium developers decided to look into fixing this.

The solution is a next-gen feature in the works that will tag entries on the URL stack by whether a user action caused the entry to be placed on the stack. That beautifully solves both the redirect chain and the stack-manipulation problems. If you think about it, what you want is to step back BEFORE anything that any automation did for or to you, automatically.

So we have a longstanding problem solved in a way that doesn't break anything else.

ALL Linux Kernels before v5.0.8 are vulnerable...

... to a very difficult to exploit, yet also very potent if it can be exploited, vulnerability that CAN be used for remote code execution.

All Linux machines running distributions using kernels prior to 5.0.8 are affected by a race condition vulnerability leading to a use after free in the kernel's net namespace cleanup.

The problem was found in Linux's Reliable Datagram Socket protocol which is very worrisome since that makes it remotely exploitable. Consequently, the vulnerability has been assigned a high 8.1 severity score by the NIST and it can be abused by unauthenticated attackers without interaction from the user.

However, the attack is quite complex, leading to a low exploitability index of 2.2. So the overall impact score is limited to 5.9.

However, the flaw comes with high confidentiality, integrity, and availability impact which makes it possible for would-be attackers to gain access to all resources, modify any files, and deny access to resources after successfully exploiting the vulnerability.

The Linux kernel developers issued a patch for the CVE-2019-11815 issue in late-March and fixed the flaw in the Linux kernel 5.0.8 version released on April 17.

Individuals are not in much danger, but Internet facing Linux servers would do well to update their Kernels as soon as possible.

Zombie Load
https://zombieloadattack.com/
https://zombieloadattack.com/zombieload.pdf
https://www.cyberus-technology.de/posts/2019-05-14-zombieload.html

<quote> Watch out! Your processor resurrects your private browsing-history and other sensitive data.

After Meltdown, Spectre, and Foreshadow, we discovered more critical vulnerabilities in modern processors. The ZombieLoad attack allows stealing sensitive data and keys while the computer accesses them.

While programs normally only see their own data, a malicious program can exploit the fill buffers to get hold of secrets currently processed by other running programs. These secrets can be user-level secrets, such as browser history, website content, user keys, and passwords, or system-level secrets, such as disk encryption keys.

The attack does not only work on personal computers but can also be exploited in the cloud.

Make sure to get the latest updates for your operating system!

ZombieLoad: Cross Privilege-Boundary Data Leakage

May 14 2019

ZombieLoad is a novel category of side-channel attacks which we refer to as **data-sampling attack**. It demonstrates that faulting load instructions can transiently expose private values of one Hyperthread sibling to the other. This new exploit is the result of a collaboration between Michael Schwarz, Daniel Gruss and Moritz Lipp from Graz University of Technology, Thomas Prescher and Julian Stecklina from Cyberus Technology, Jo Van Bulck from KU Leuven, and Daniel Moghimi from Worcester Polytechnic Institute.

In this article, we summarize the implications and shed light on the different attack scenarios across CPU privilege rings, OS processes, virtual machines, and SGX enclaves, and give advice over possible ways to mitigate such attacks.

Implications

A short summary of what this security vulnerability means:

- By exploiting the CPU's so-called bypass logic on return values of loads, it is possible to **leak data across processes, privilege boundaries, Hyperthreads**, as well as values that are loaded inside **Intel SGX enclaves**, and **between VMs**.
- Code utilizing this exploit works on Windows, Linux, etc., as this is not a software- but a hardware issue.
- It is possible to retrieve content that is currently being used by a Hyperthread sibling.
- Even without Hyperthreading, it is possible to leak data out of other protection domains. During experimentation it turned out, that ZombieLoad leaks endure serializing instructions. Such leaks do however work with lower probability and are harder to obtain.
- It is an *implementation detail* what kind of data is processed after a faulty read.
- Using Spectre v1 gadgets, potentially any value in memory can be leaked.
- Affected software:
  - So far all versions of all operating systems (Microsoft Windows, Linux, MacOS, BSDs, …)
  - All hypervisors (VMWare, Microsoft HyperV, KVM, Xen, Virtualbox, …)
  - All container solutions (Docker, LXC, OpenVZ, …)
  - Code that uses secure SGX enclaves in order to protect critical data.
- Affected CPUs:
  - Intel **Core** and **Xeon** CPUs
  - CPUs with Meltdown/L1TF mitigations are affected by fewer variants of this attack.
  - We were unable to reproduce this behavior on non-Intel CPUs and consider it likely that this is an implementation issue affecting only Intel CPUs.
- Sole operating system/hypervisor software patches do not suffice for complete mitigation:
  - Similar to the L1TF exploit, effective mitigations require switching off SMT (Simultaneous MultiThreading, aka Hyperthreads) or making sure that trusted and untrusted code do not share physical cores.

Miscellany

SQRL: Just added support for Controlled Folder Access...

- Appeared in the infamous October 2018 Fall Creator's Edition.
- Disabled by default. Anti-Ransomware. Protects the user's created content.

Working on the final SQRL features, capabilities and implementation document.

2nd SQRL Presentation: OWASP LA Monthly Dinner Meeting - Wednesday, May 22, 2019
OWASP: Open Web Application Security Project
Signal Sciences · 8520 National Blvd · Culver City, CA
Wednesday, May 22, 2019 -- 6:15 PM to 8:15 PM
https://www.meetup.com/OWASP-Los-Angeles/events/259755502/

blog.grc.com

Android "Q"

With "Q", Google moves Android security forward significantly:
From Google I/O: 14 Android OS modules to get over-the-air security updates in real-time
Google announces a new way for delivering Android security updates for core OS components.

Stephanie Cuthbertson, Senior Director for Android: "Your regular device gets regular security updates already but you still have to wait for the release, and you have to reboot when they come. We want you to get these faster. Even faster. And that's why in Android Q we're making a set of OS modules updateable directly over the air, so now these can be updated individually as soon as they're available and without a reboot of the device."

In a project known internally as Project Mainline, Google's developers have spent the past year working to split several OS core components into separate OS modules. These modules, despite encompassing a core service of the Android OS, will work like Android apps which are able to receive security updates on-the-fly through the Google Play Store.

Once a security update is available, Google says it will push the update to all devices supporting this mechanism. The device will stop that particular OS component, apply the update, and restart the component without having to shut down the rest of the OS.

The 14 modules that can be updated in this fashion are: ANGLE, APK, Captive portal login, Conscrypt, DNS resolver, Documents UI, ExtServices, Media codecs, Media framework components, Network permission configuration, Networking components, Permission controller, Time zone data and Module metadata.

They are all internal core services, so lacking any user-facing surface. But they are often the components that are the most security troubled.

The Verge, who did some reporting on this, learned that individual device makers will be able to opt out of using this new feature (but why would they?). As would be expected, Mainline is only supported on Android Q, and only handsets that will be shipping with Android Q installed by default will be able to use it. Devices running previous versions of Android which are then updated to Q will not be able to use Mainline's on-the-fly upgrade features.

On Android Q devices where the phone maker chooses for whatever reason not to support the Mainline features will continue to receive whole-system security updates in the traditional fashion -- in one big update, either over-the-air from the phone maker or mobile carriers.

Besides this improved system for security updates, Android Q also comes with 50 other improved privacy and security features, which Cuthbertson described as the main focus of this release.

This includes support for TLS v1.3, MAC address randomization, increased control over location data, and support for new more granular settings allowing users to check which apps have access to a particular permission -- providing the option to revoke an app's access if desired.

What’s New in Android Q Security / Last Thursday the 9th
https://android-developers.googleblog.com/2019/05/whats-new-in-android-q-security.html?linkId=67173930

Encryption

Storage encryption is one of the most fundamental (and effective) security technologies, but current encryption standards require devices have cryptographic acceleration hardware. Because of this requirement many devices are not capable of using storage encryption. The launch of Adiantum changes that in the Android Q release. We announced Adiantum in February. Adiantum is designed to run efficiently without specialized hardware, and can work across everything from smart watches to internet-connected medical devices.

Our listeners will recall from our coverage of this back at the start of February that "Adiantum" uses the ChaCha20 stream cipher in a secure length-preserving mode. Unlike AES, which does not perform well on processors lacking some hardware support for it (Intel has added specific enhancements called AES-NI - for "New Instructions"). The bit-level manipulations required by AES require the use of many simple instructions. But ChaCha20 only uses basic instructions that are fast on all processors. This makes it an ideal cipher for lower-end systems.

<Google continued>: Our commitment to the importance of encryption continues with the Android Q release. All compatible Android devices newly launching with Android Q are required to encrypt user data, with no exceptions. This includes phones, tablets, televisions, and automotive devices. This will ensure the next generation of devices are more secure than their predecessors, and allow the next billion people coming online for the first time to do so safely.

However, storage encryption is just one half of the picture, which is why we are also enabling TLS 1.3 support by default in Android Q. TLS 1.3 is a major revision to the TLS standard finalized by the IETF in August 2018. It is faster, more secure, and more private. TLS 1.3 can often complete the handshake in fewer roundtrips, making the connection time up to 40% faster for those sessions. From a security perspective, TLS 1.3 removes support for weaker cryptographic algorithms, as well as some insecure or obsolete features. It uses a newly-designed handshake which fixes several weaknesses in TLS 1.2. The new protocol is cleaner, less error prone, and more resilient to key compromise. Finally, from a privacy perspective, TLS 1.3 encrypts more of the handshake to better protect the identities of the participating parties.

Platform Hardening

Android utilizes a strategy of defense-in-depth to ensure that individual implementation bugs are insufficient for bypassing our security systems. We apply process isolation, attack surface reduction, architectural decomposition, and exploit mitigations to render vulnerabilities more difficult or impossible to exploit, and to increase the number of vulnerabilities needed by an attacker to achieve their goals.

[I like the phrasing of that. It's a sober and realistic expression of the truth of the challenges facing any highly-targeted platform. And Android is arguably the #1 most targeted platform in the history.]

<Google>: In Android Q, we have applied these strategies to security critical areas such as media, Bluetooth, and the kernel. We describe these improvements more extensively in a separate blog post, but some highlights include:

- A constrained sandbox for software codecs. (We'll be focusing upon that detail in a second)
- Increased production use of sanitizers to mitigate entire classes of vulnerabilities in components that process untrusted content.
- Shadow Call Stack, which provides backward-edge Control Flow Integrity (CFI) and complements the forward-edge protection provided by LLVM's CFI.
- Protecting Address Space Layout Randomization (ASLR) against leaks using eXecute-Only Memory (XOM).
- Introduction of Scudo hardened allocator which makes a number of heap related vulnerabilities more difficult to exploit.

Authentication

Android Pie introduced the BiometricPrompt API to help apps utilize biometrics, including face, fingerprint, and iris. Since the launch, we've seen a lot of apps embrace the new API, and now with Android Q, we've updated the underlying framework with robust support for face and fingerprint. Additionally, we expanded the API to support additional use-cases, including both implicit and explicit authentication.

In the explicit flow, the user must perform an action to proceed, such as tap their finger to the fingerprint sensor. If they're using face or iris to authenticate, then the user must click an additional button to proceed. The explicit flow is the default flow and should be used for all high-value transactions such as payments.

Implicit flow does not require an additional user action. It is used to provide a lighter-weight, more seamless experience for transactions that are readily and easily reversible, such as sign-in and autofill.

Another handy new feature in BiometricPrompt is the ability to check if a device supports biometric authentication prior to invoking BiometricPrompt. This is useful when the app wants to show an “enable biometric sign-in” or similar item in their sign-in page or in-app settings menu. To support this, we’ve added a new BiometricManager class. You can now call the canAuthenticate() method in it to determine whether the device supports biometric authentication and whether the user is enrolled.

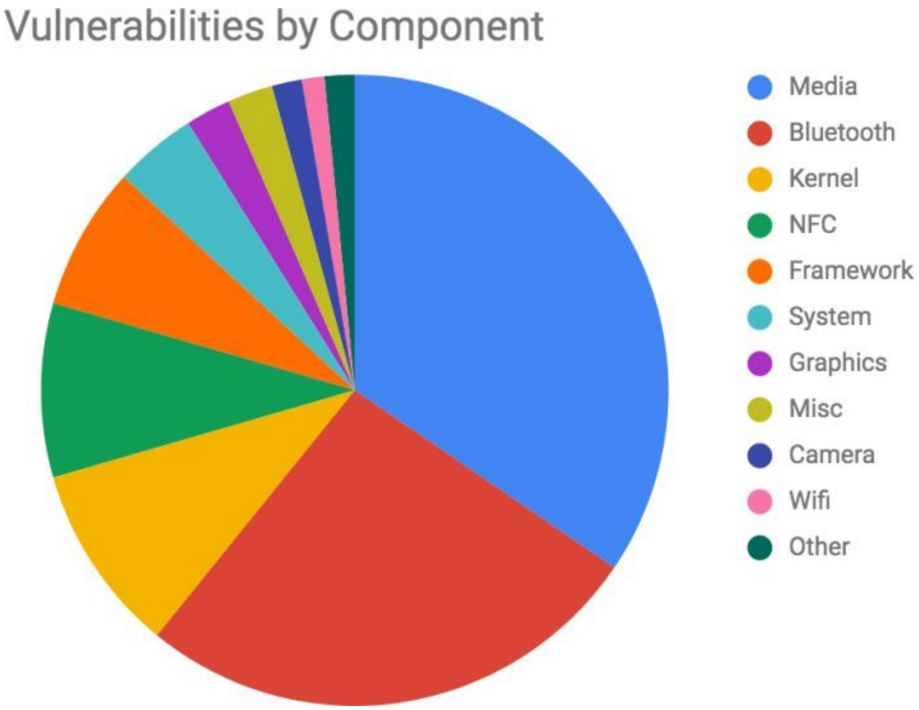
What’s Next?

Beyond Android Q, we are looking to add Electronic ID support for mobile apps, so that your phone can be used as an ID, such as a driver’s license. Apps such as these have a lot of security requirements and involves integration between the client application on the holder’s mobile phone, a reader/verifier device, and issuing authority backend systems used for license issuance, updates, and revocation.

This initiative requires expertise around cryptography and standardization from the ISO and is being led by the Android Security and Privacy team. We will be providing APIs and a reference implementation of HALs for Android devices in order to ensure the platform provides the building blocks for similar security and privacy sensitive applications. You can expect to hear more updates from us on Electronic ID support in the near future.

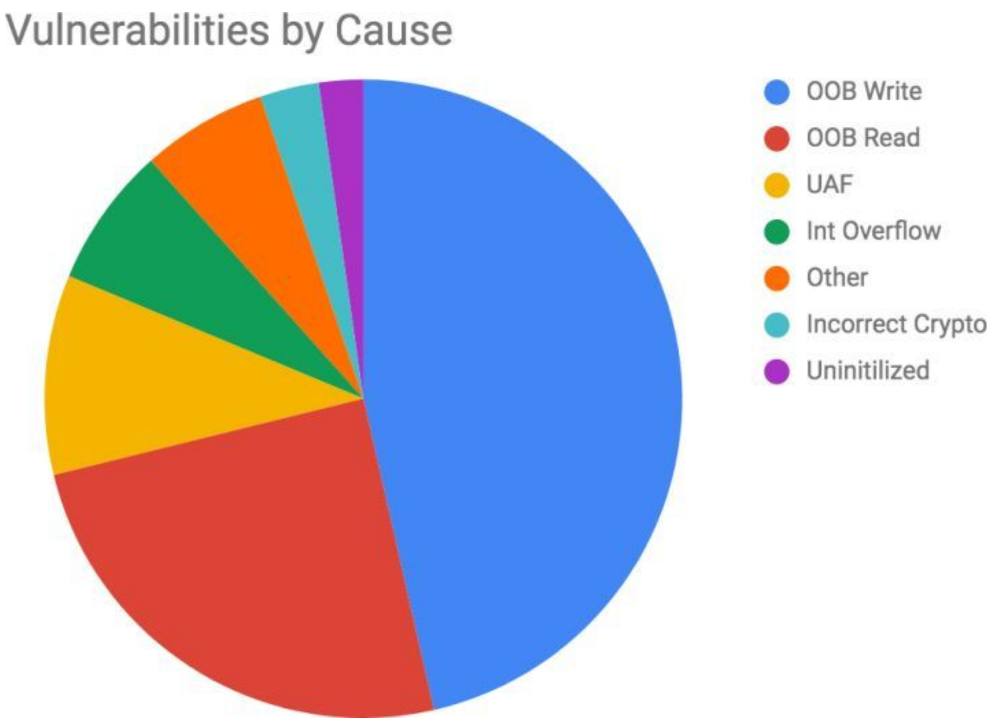
SO..... This is all great news for the most-used operating system platform in the world... But what I really want to know is what dessert "Q" is going to be???

More on "Q's" security enhancements...
https://security.googleblog.com/2019/05/queue-hardening-enhancements.html



Android Q, which is now available to developers in Beta, has continued to tighten up its code and to harden its various attack surfaces. By examining the problems found, and where they are found, rather than thinking "oh, well, that's a one-off that will never happen again, the Android engineers are successfully and iteratively hardening this very important piece of software.

If won't surprise any of our listeners to learn that the two top locations of trouble are the Media and Bluetooth management. As we know (we saw this earlier with all of the trouble with the "StageFright" exploits), processing media requires interpreting a form of real time meta language within software "codecs" (which is short for compressors / decompressors), and these interpreters are notoriously difficult to harden, since they have grown incredibly complex and there's such a strong implicit assumption that they will be fed valid media content to playback or record. But we know objectively that that's far from true.



And Bluetooth is not only a complex protocol, but wireless radio, so a very tasty target:

As Google's engineers explain...

In Android Q, we moved software codecs out of the main mediacodec service into a constrained sandbox. This is a big step forward in our effort to improve security by isolating various media components into less privileged sandboxes. As Mark Brand of Project Zero points out in his Return To Libstagefright blog post, constrained sandboxes are not where an attacker wants to end up. In 2018, approximately 80% of the critical/high severity vulnerabilities in media components occurred in software codecs, meaning further isolating them is a big improvement. Due to the increased protection provided by the new mediaswcodec sandbox, these same vulnerabilities will receive a lower severity based on Android’s severity guidelines.

I mentioned that this was an iterative process. Here's the past four states of iteration:

- Prior to N, media services are all inside one monolithic mediaserver process, and the extractors run inside the client.

- In N, we delivered a major security re-architect, where a number of lower-level media services are spun off into individual service processes with reduced privilege sandboxes. Extractors are moved into server side, and put into a constrained sandbox. Only a couple of higher-level functionalities remained in mediaserver itself.

- In O & P, the services are "treblized," and further deprivileged that is, separated into individual sandboxes and converted into HALs. The media.codec service became a HAL while still hosting both software and hardware codec implementations.

- In Q, the software codecs are extracted from the media.codec process, and moved back to system side. It becomes a system service that exposes the codec HAL interface. Selinux policy and seccomp filters are further tightened up for this process. In particular, while the previous mediacodec process had access to device drivers for hardware accelerated codecs, the software codec process has no access to device drivers.

<Google>: With this [final] move [in Android 'Q'], we now have the two primary sources for media vulnerabilities tightly sandboxed within constrained processes. Software codecs are similar to extractors in that they both have extensive code parsing bitstreams from untrusted sources. Once a vulnerability is identified in the source code, it can be triggered by sending a crafted media file to media APIs (such as MediaExtractor or MediaCodec). Sandboxing these two services allows us to reduce the severity of potential security vulnerabilities without compromising performance.

In addition to constraining riskier codecs, a lot of work has also gone into preventing common types of vulnerabilities. I read into all of the details which I won't dig into here. But suffice to say that I am very impressed by what I have read. By the maturity of the development philosophy that I see in those details. Google is VERY focused upon carefully studying the sources of problems, and applying sound software engineering solutions -- and inventing them when they don't exist -- to patiently evolve an ever-more-stable and capable software operating system platform.

Android is really shaping up to be an asset of tremendous value.

~30~