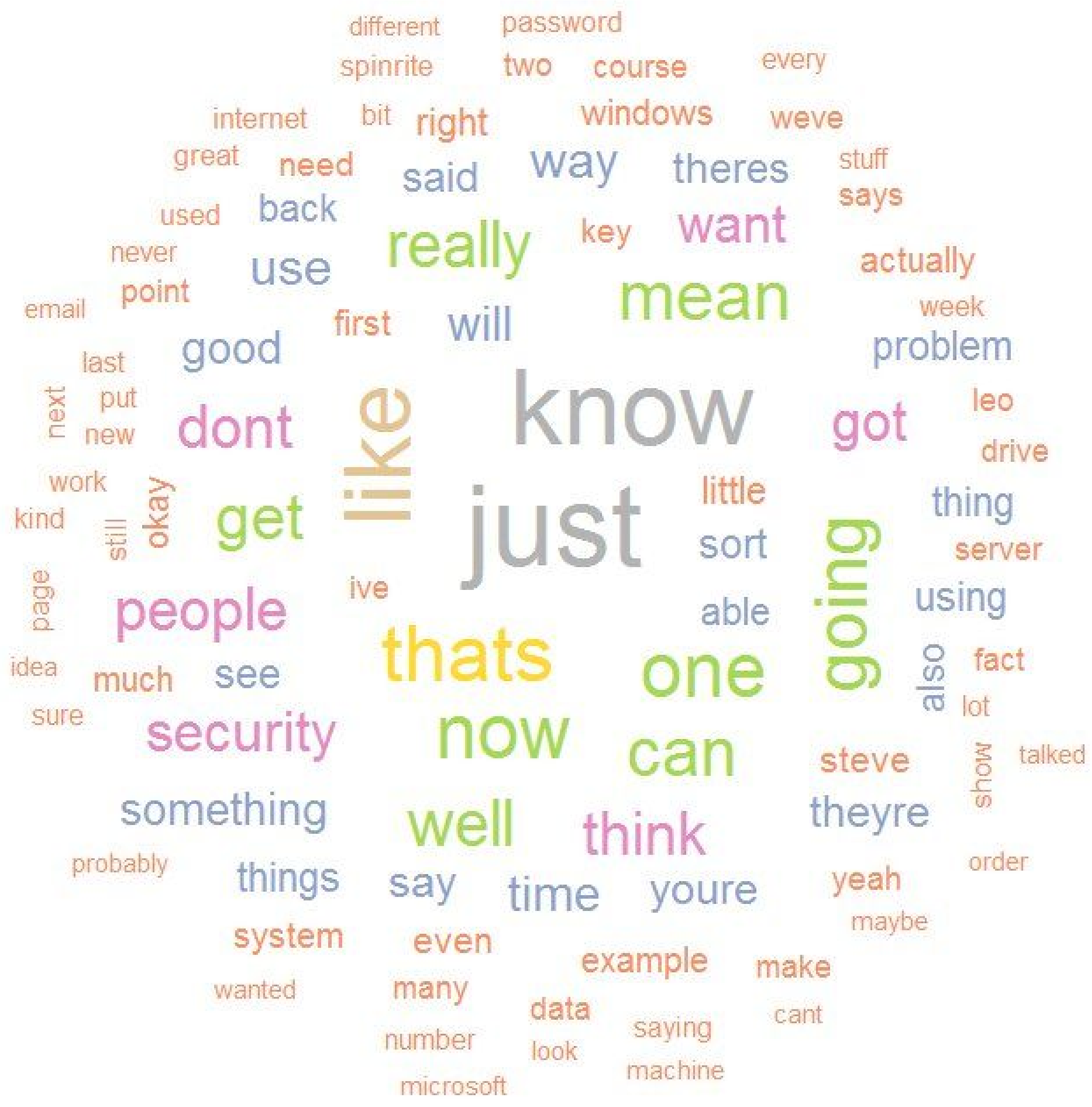
Security Now! #742 - 11-26-19
“Pushing” DoH

This week on Security Now!

This week we look at some interesting changes coming to Android and some inherent challenges presented by the nature of the Android ecosystem. We examine some newly revealed troubles with the venerable VNC clients and servers, we note a welcome change to Twitter and update on law enforcement's "forgone conclusion" strategy to force password divulgence. We then look at a surprising pre-announcement from Microsoft about DNS, then dig more deeply into the details of the emerging DoH protocol and reveal a VERY interesting and surprising and unsuspected capability.


different password
spinrite two course every
internet bit right windows weve
great need said way theres stuff says
used back key want actually
never use really mean week
email point first will problem
last good know leo
next put new dont like got drive
work kind still okay get just little thing
sort server
page ive people able going using
idea much see thats one also fact lot
sure security now can steve show talked
something well think theyre
probably things say time youre yeah order
system even maybe
wanted many example make
data saying cant
number look machine
microsoft

Security News

The future of the Linux kernel underneath the Android OS
There have been some recent rumblings about Google's possible return to the stock Linux kernel for future Android-based smartphones. There's also some very different rumblings about an entirely different new OS inside of Google, known as "Project Fuchsia"... But that appears to still be many years away.

John Dunn, writing for Sophos recently noted that the momentum appears to be building inside Google for a radical overhaul of what he described as Android’s tortured relationship with its precious Linux-based kernel.

He wrote: It’s a big job and has been a long time coming, arguably since the mobile operating system was unveiled in 2007.

The company hasn’t made any firm announcements on this, but journalists recently noticed a low-key video posted to YouTube of a presentation given by Android Kernel Team chief Sandeep Patil, during September’s Linux Plumbers Conference in Lisbon, Portugal.

What's the problem with things as they are today? The development model that underpins how Android uses the Linux kernel leads to a lot of complexity that slows updates, raises costs, and makes life difficult for both Google and the device makers downstream in all sorts of ways. The Linux kernel used by an Android device can be slightly different for every make and model at different moments in time.

Device makers start with the LTS (Long Term Support) kernel, before the device's so-called "out of tree" Android common kernel customisations are added to that Linux Kernel. According to Patil, there are many of these -- as of February 2018, 355 changes needed to be applied which required 32,266 insertions, and 1,546 deletions on top of LTS v4.14.0... and even that much kernel tweaking was an improvement over the past.

After that, the system-on-chip (SoC) companies, like Qualcomm, add numerous hardware customizations before, finally, manufacturers add even more vendor and device-specific software.

These multiple sequentially-applied layers of customisation require that each device use a single Linux kernel as its starting point, which then has downstream effects for all the subsequent modifications. Over time this all leads to Google and device makers supporting several forks of Linux -- simultaneously -- across numerous devices.

This serves to explain why Android devices tend to have a limited shelf life, but most problematically, it makes the application of security patches much more time-consuming, labor intensive and error prone.

So, what Google appears to be planning is to return to a base generic Linux kernel with the goal of reducing or possibly even eliminating the need for the Android kernel modifications. This would, in effect, return Android to the Linux kernel fold it once perhaps arrogantly decided to abandon in search of improvements that turned out to come at a too-high price. In this new and improved world, each generation of devices would use the same kernel and Google would supply their own modifications as well-defined modules riding on top of the unmodified kernel.

This makes for a very clean diagram with one clean kernel and many modules on top. And this effort aligns well with Google's Project Treble API which we talked about last year to speed up device patching.

At this event Tom Gall of the director of the Linaro Consumer Group showed off a Xiaomi smartphone running Android on top of the mainline Linux kernel. If nothing else this demonstrates that the goal is achievable in principle. But the effort to bring all Android devices around to this improved model won't happen overnight.

And as for Project Fuschia: Its aim appears to be eventually replacing Android entirely and presumably also the Chrome OS. If it's what it appears to be, Fuschia would integrate all of the development layers of Android while working across multiple devices, up to and including desktop computers. This suggests that Android days may be numbered, but it looks like that number will be large and that Android will remain with us for quite some time.

The Android Ecosystem Challenge

And while we're on the topic of Android, we are often talking about the special challenges faced by the world's #1 smartphone operating system. Apple has the powerful advantage of having total control over relatively few different hardware platforms. And where related platforms differ it's often in details such as screen resolution. They are not needing to deal with an essentially infinite number of hardware variations -- and there's only the one vendor: Apple.

Google's Android platform not only powers their own smartphones, but also those produced by a large number of suppliers, from the large market leaders such as Samsung, Huawei and Xiaomi, to small and obscure suppliers such as Bluboo, Leagoo, Ulefone and Walton. This creates a rich Android ecosystem, but it means that security can sometimes be left wanting. This podcast would not exist if security was easy to achieve in practice. But a recent report from Kryptowire helps to put a sharper point on the problem:

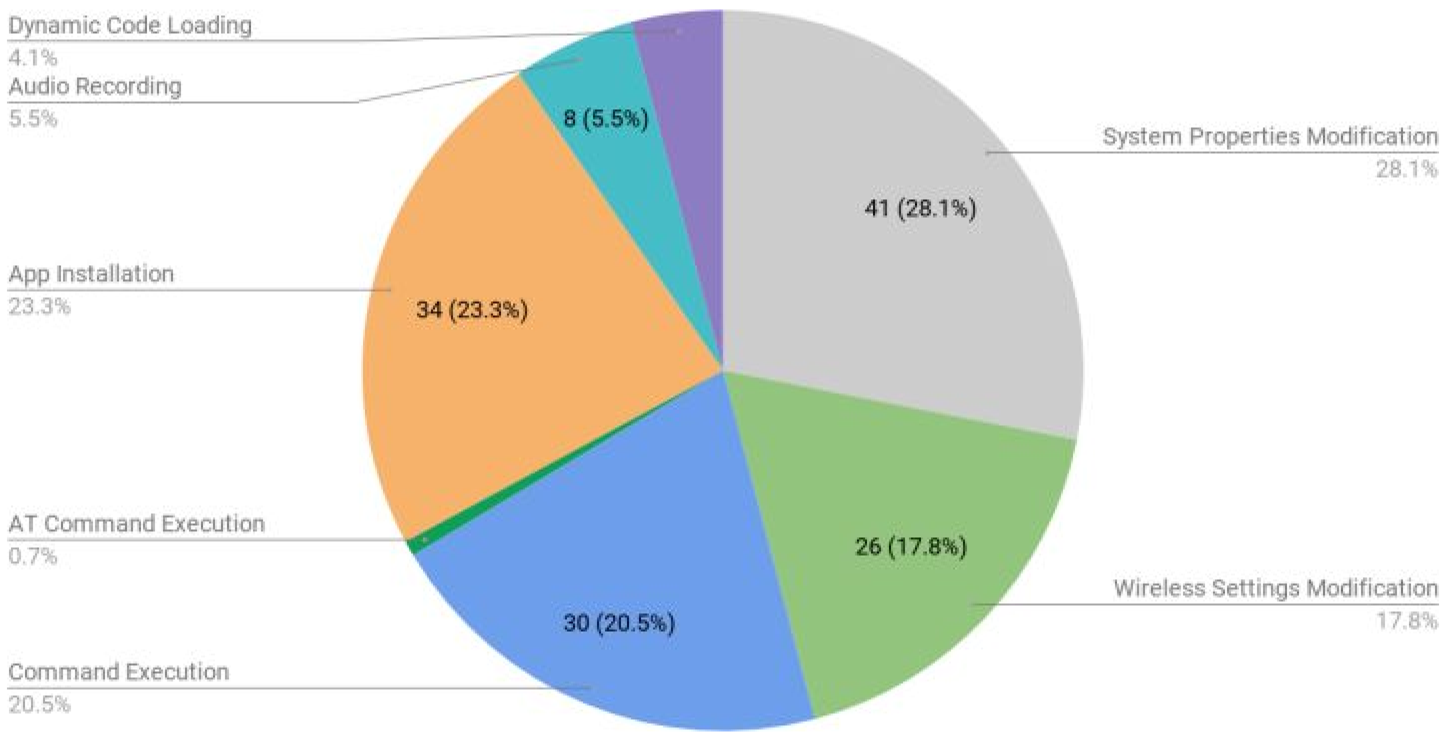
https://www.kryptowire.com/android-firmware-2019/

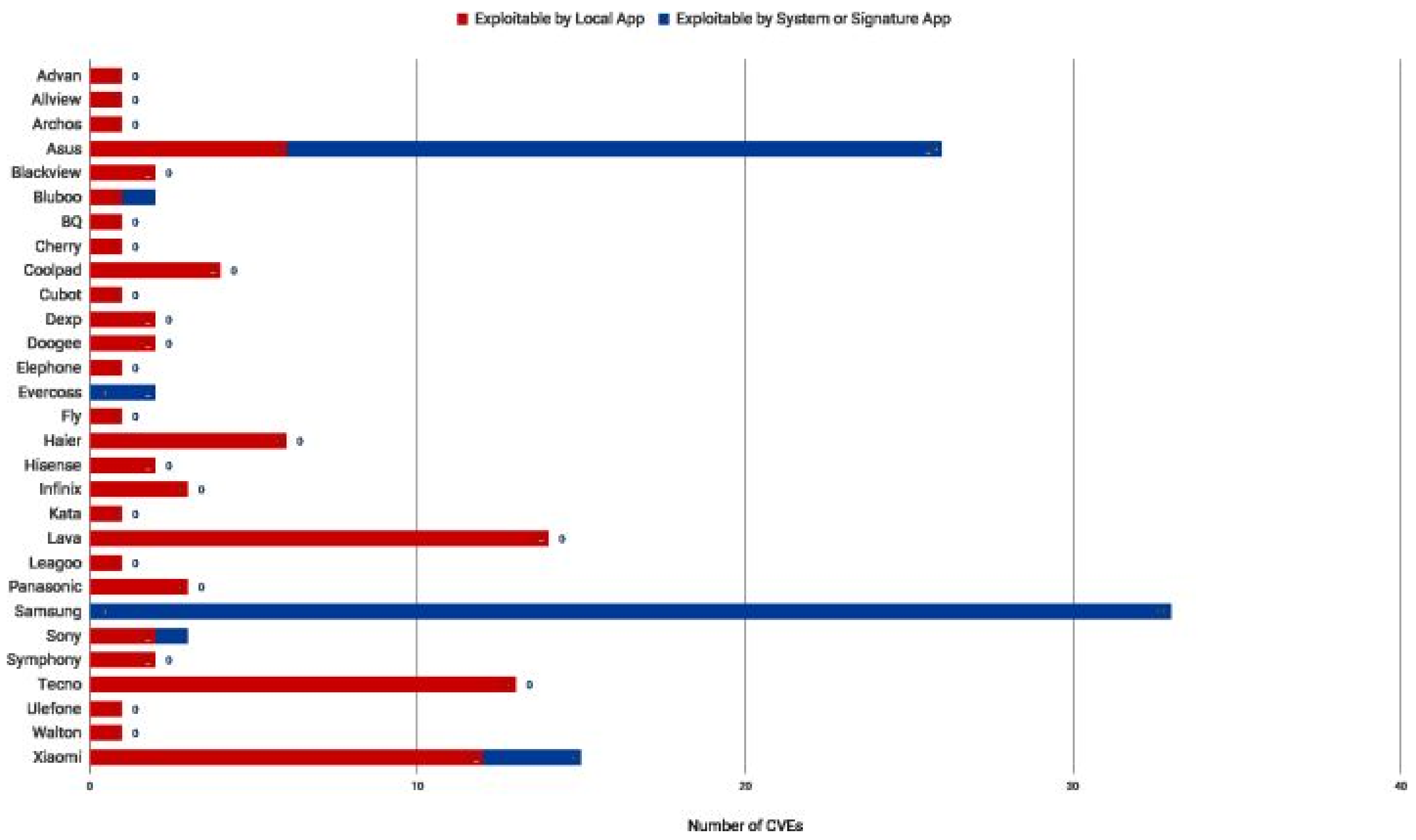
Kryptowire kicked off their just-released November 2019 report by writing:

"Pre-installed apps and firmware pose a risk due to vulnerabilities that can be pre-positioned on a device, rendering the device vulnerable on purchase. To quantify the exposure of Android end-users to vulnerabilities residing within pre-installed apps and firmware, we analyzed a wide range of Android vendors and carriers using devices spanning from low-end to flagship. Our primary focus was exposing pre-positioned threats on Android devices sold by United States (US) carriers, although our results affect devices worldwide."

The following two charts summarize Kryptowire's findings:

146 CVEs found





One thing that stood out was the fact that the most vulnerabilities were found in the devices sold by several of the most popular non-obscure suppliers: The **most** CVE vulnerabilities were found in Samsung phones, second as Asus, and third was Xiaomi. Based upon Kryptowire's research and vulnerability findings, it appears that most of the problems reside in the vendor-added "value added" smartphone add-ons. Until they mess with it, Android itself appears to be solid.

Kryptowire explains:

Devices are shipped with pre-installed software

- The apps are not present on or vetted by official app stores
- Most of the functionality is built-in and cannot be disabled
- These Apps run with privileged & system access by default

Pre-installed apps can be vulnerable and/or malicious

- There is the potential for remote and local exploitation
- "Backdoor" functionality & data exfiltration

Why is this happening?

- Vendors customize official code to all various bell & whistles features.
- Hardware suppliers provide software -- meaning that they want to sell hardware and the software is too often viewed as a necessary evil.
- This process tends to often unintentionally expose sensitive capabilities

https://www.kryptowire.com/portal/wp-content/uploads/2019/11/Mobile-Vulnerability-Analysis.pdf

During 2019, Kryptowire discovered approximately 1,000 new vulnerabilities spanning more than 30 OEM suppliers of Android smartphones.

Collectively, the vulnerabilities enabled: Command execution, log leakage, network settings modification, SMS sending/spoofing, screenshot capturing, system properties modifications, factory reset, app installation, app uninstallation, AT-command execution, audio recording, video recording, and dynamic code loading.

So our takeaways from this research is that security is **really** difficult and that the many modifications being made by 3rd-party smartphone makers -- even the big players such as Samsung -- often bring much more than new geewhiz features to the phone: All too often they open the device to both local and remote exploitation.

VNC users: Time to update!
Kaspersky's researchers found 37 vulnerabilities in four VNC implementations:
LibVNC / UltraVNC / TightVNC 1.x / TurboVNC. The developers have fixed most, but not all, of the 37 problems.

https://www.kaspersky.com/blog/vnc-vulnerabilities/31462/

As our listeners know, being a Microsoft user I use Remote Desktop, though only with it safely ensconced behind several layers of traffic filtering firewalls. But in years past I have played with UltraVNC and TightVNC. They definitely have their appeal. Apparently they also definitely have their problems. Since I know that this podcast has a way above average technical following, my attention was captured when I saw Kaspersky's note about trouble in VNC land since I knew that some of our listeners would likely need to know. And it's perhaps unfortunate that only we're only able to reach our listeners because VNC -- which stands for Virtual Network Computing -- is a heavily used solution today. A search for public exposed VNC servers with Shodan reveals that more than 600,000 VNC servers can currently be accessed online. Kaspersky believes that the actual figure is likely to be far higher... Perhaps because many additional VNC servers are listening on non-standard ports.

So, Kaspersky's guys closely examined those four common open-source VNC implementations:

- LibVNC, as its name suggests, is a library of code on top of which developers can create fully functional customized apps. For example LibVNC is used in systems that allow remote connections to virtual machines, as well as iOS and Android mobile devices.

- TightVNC 1.X is the application recommended by vendors of industrial automation systems for connecting to a human–machine interface (HMI).

- TurboVNC specializes in enabling remote work with graphic, 3D, and video objects.

- And UltraVNC is a VNC variant built for Windows; it is also widely used in industrial settings for connecting to human-machine interfaces.

Across these four offerings vulnerabilities were found in all four systems: 1 in TurboVNC, 4 in TightVNC, 10 in LibVNC, and 22 in UltraVNC.

Being a networked client/server solution, there's a VNC component at each end of the connection. So we have a server that's listening for incoming connections and a client that's initiating the call. However, as anyone who has used VNC in the past (or is maybe using it today) knows, unlike Microsoft's RDP, the roles of which end offers the user-interface for manipulation can be switched around. It's not always the connection-receiving server that offers its desktop or user-interface.

Kaspersky found that VNC vulnerabilities were much less common on the server side, which they described as being somewhat simpler and therefore with fewer opportunities to pick up bugs. But Kaspersky did identify flaws in both sides of the applications, though attacks on the server would, in many cases, be impossible without authorization (unlike the now-infamous Windows BlueKeep RDP flaw.)

All of the bugs involve incorrect memory usage and their exploiting leads to malfunctions and denial of service or, in more serious cases, to attackers gaining unauthorized access to information on the device or the ability to release malware into the victim’s system.

Kaspersky's researchers reported all of the bugs they encountered to the developers of the respective libraries and applications. Most of them have been fixed already. But there is an exception: The creators of TightVNC no longer support the first version of their system, and they refused to patch the vulnerabilities detected in it. This is a good reason to reason moving to another VNC platform if you're currently using TightVNC.

And what's more, as is the case with many open-source projects, vulnerable code often gets used in a large number of other developments, and not all developers keep close tabs on library updates from which they borrowed snippets for their creations. There are many remote access system available today. How many of them began with a VNC codebase and grew a new UI and features from there?

I've included a link here in the show notes to the details of Kaspersky's research which covers exactly what they found item by item for each of the four VNC variants.

https://ics-cert.kaspersky.com/reports/2019/11/22/vnc-vulnerability-research/

The best advice it you're using VNS would be to check in for any updates and to be sure you're using the latest. And if you're using TightVNC it sounds like it's time to move to some other solution that's still be supported and is receptive to security reseacher's vulnerability findings.

In closing, Kaspersky had a few generic but useful checklist items to share:

- Check which devices can connect remotely, and block remote connections if not required.
- Inventory all remote access applications — not just VNC — and check that their versions are up-to-date. If you have doubts about their reliability, stop using them. If you intend to continue deploying them, be sure to upgrade to the latest version.
- Protect your VNC servers with a strong password. This will make attacking them far harder.
- Do not connect to untrusted or untested VNC servers.
- In industrial enterprise environments, use a specialized security solution for industrial automation systems, for example, Kaspersky Industrial CyberSecurity.

Twitter finally allows SMS-based 2FA to be disabled.
Twitter announced last Thursday that its users will finally be able to disable SMS-based two-factor authentication (2FA) for their accounts, and use an alternative method only, such as a mobile one-time code (OTP) authenticator app or a hardware security key.

Until last Thursday that was impossible. If users wanted to use any form of 2FA for their Twitter account, they had to first register a phone number and enable the SMS-based 2FA method, whether or not they wished to use SMS. Users wishing to use an OTP mobile authenticator app or a hardware security key had to enable the SMS-based 2FA first, and they couldn't disable it. And even if and after the user chose to use a security key, the SMS-based 2FA method was still active, and exposed the account to impersonation attacks including SIM swaps. Hackers who knew a user's password would perform a SIM swap to temporarily hijack a user's phone number, bypass SMS-based 2FA, and then take over that user's account.

Over the past two years many high-profile accounts have been hacked this way. But Twitter never budged on its decision to make SMS-based 2FA mandatory and an always-on option. But that all changed on August 30th when hackers used exactly this sort of SIM swap attack to gain access to the Twitter account of Twitter's CEO, Jack Dorsey.

So now, Twitter users can finally disable SMS-based 2FA, and opt for a more secure 2FA method. This also means that Twitter users finally delete the phone number associated with their account, and still be able to use 2FA. Happily, this also eliminates scenarios where SIM swappers who DON'T know a user's password can use the SMS-based password recovery feature to hijack accounts -- effectively plugging the SIM swapping attack vector for an account.

Not such a "foregone conclusion" after all...

While you were travelling, Leo, Jason and I updated our listeners to a new tact that prosecutors were taking to compel the disclosure of passwords. Before then, as we had previously discussed, the law was holding that forcing a suspect to turn over their password was "testimonial" and thus subject to the protections provided by the 5th amendment to the US constitution which gives individuals the right to "plead the 5th" to avoid self-incriminating testimony. In such a situation they cannot be held in contempt of court.

But then, what Jason and I discussed was a new prosecution tact known as the "foregone conclusion." It held that since the phone was known to belong to the suspect, and the suspect was able to access it to use it, it was therefore a foregone conclusion that they knew the password that was required to obtain access to the device, so compelling its disclosure was constitutional and was not protected under the 5th amendment.

We're talking about this once again because, once again, a higher court has overturned and reversed a lower courts ruling which depended upon the "foregone conclusion" strategy. Last week the Pennsylvania Supreme Court ruled that the 5th amendment to the US Constitution does indeed protect US citizens from being forced to turn over personal passwords to police to aid in their investigations.

In a 4-3 ruling, justices from Pennsylvania’s highest court overturned a lower-court order that required the suspect in a child-pornography case to turn over a 64-character password to his computer. The lower-court ruling had held that the compelled disclosure didn’t violate the defendant’s Fifth Amendment rights because of statements he made to police during questioning.

“It’s 64 characters and why would I give that to you,” Joseph J. Davis of Pennsylvania’s Luzerne County told investigators in response to their request for his password. “We both know what’s on there. It’s only going to hurt me. No f'***ing way I’m going to give it to you.”

Prosecutors in the case said a legal doctrine known as the “foregone conclusion exception” permitted the compelled disclosure of Davis’ password. The doctrine, which originally applied to the compelled production of paper documents, said Fifth Amendment protections against self-incrimination don’t apply when the government already knew of the existence, location, and content of the sought-after material.

In requiring Davis to turn over his password to investigators, the lower-court agreed with prosecutors that the password demand fell under the foregone conclusion exemption. The lower court said the exception applied because, under previous US Supreme Court precedent, the password was tantamount to a key or other tangible property and didn't reveal the "contents" of the defendant's mind.

The majority for the Pennsylvania Supreme Court disagreed. They wrote:

> Based upon these cases rendered by the United States Supreme Court regarding the scope of the Fifth Amendment, we conclude that compelling the disclosure of a password to a computer, that is, the act of production, is testimonial. Distilled to its essence, the revealing of a computer password is a verbal communication, not merely a physical act that would be nontestimonial in nature. There is no physical manifestation of a password, unlike a handwriting sample, blood draw, or a voice exemplar. As a passcode is necessarily memorized, one cannot reveal a passcode without revealing the contents of one's mind. Indeed, a password to a computer is, by its nature, intentionally personalized and so unique as to accomplish its intended purpose—keeping information contained therein confidential and insulated from discovery. Here, under United States Supreme Court precedent, we find that the Commonwealth is seeking the electronic equivalent to a combination to a wall safe—the passcode to unlock Appellant's computer. The Commonwealth is seeking the password, not as an end, but as a pathway to the files being withheld. As such, the compelled production of the computer's password demands the recall of the contents of Appellant's mind, and the act of production carries with it the implied factual assertions that will be used to incriminate him. Thus, we hold that compelling Appellant to reveal a password to a computer is testimonial in nature.
>
> We acknowledge that, at times, constitutional privileges are an impediment to the Commonwealth. Requiring the Commonwealth to do the heavy lifting, indeed, to shoulder the entire load, in building and bringing a criminal case without a defendant's assistance may be inconvenient and even difficult; yet, to apply the foregone conclusion rationale in these circumstances would allow the exception to swallow the constitutional privilege. Nevertheless, this constitutional right is firmly grounded in the "realization that the privilege, while sometimes 'a shelter to the guilty,' is often 'a protection to the innocent.'" Moreover, there are serious questions about applying the foregone conclusion exception to information that manifests through the usage of one's mind.

Miscellany

- The Mandalorian: Okay... I'm hooked for the next five weeks... then I'm out.

“Pushing” DoH

Microsoft has not only just jumped into the DNS-over-HTTPS world, but has in one fell swoop instantly legitimized the various web browsers’ efforts to protect their user’s DNS queries from the prying eyes of their ISPs. Sorry, CONCAST!

https://techcommunity.microsoft.com/t5/Networking-Blog/Windows-will-improve-user-privacy-with-DNS-over-HTTPS/ba-p/1014229

The news reduces to “Windows 10 will be getting native support for DoH”. Here’s what Microsoft recently shared with the world:

> Here in Windows Core Networking, we’re interested in keeping your traffic as private as possible, as well as fast and reliable. While there are many ways we can and do approach user privacy on the wire, today we’d like to talk about encrypted DNS. Why? Basically, because supporting encrypted DNS queries in Windows will close one of the last remaining plain-text domain name transmissions in common web traffic.
>
> Providing encrypted DNS support without breaking existing Windows device admin configuration won't be easy. However, at Microsoft we believe that "we have to treat privacy as a human right. We have to have end-to-end cybersecurity built into technology."
>
> We also believe Windows adoption of encrypted DNS will help make the overall Internet ecosystem healthier. There is an assumption by many that DNS encryption requires DNS centralization. This is only true if encrypted DNS adoption isn’t universal. To keep the DNS decentralized, it will be important for client operating systems (such as Windows) and Internet service providers alike to widely adopt encrypted DNS.
>
> With the decision made to build support for encrypted DNS, the next step is to figure out what kind of DNS encryption Windows will support and how it will be configured. Here are our team's guiding principles on making those decisions:
>
> - **Windows DNS needs to be as private and functional as possible by default without the need for user or admin configuration because Windows DNS traffic represents a snapshot of the user’s browsing history.** To Windows users, this means their experience will be made as private as possible by Windows out of the box. For Microsoft, this means we will look for opportunities to encrypt Windows DNS traffic without changing the configured DNS resolvers set by users and system administrators.
>
> - **Privacy-minded Windows users and administrators need to be guided to DNS settings even if they don't know what DNS is yet.** Many users are interested in controlling their privacy and go looking for privacy-centric settings such as app permissions to camera and location but may not be aware of or know about DNS settings or understand why they matter and may not look for them in the device settings.

- **Windows users and administrators need to be able to improve their DNS configuration with as few simple actions as possible.** We must ensure we don't require specialized knowledge or effort on the part of Windows users to benefit from encrypted DNS. Enterprise policies and UI actions alike should be something you only have to do once rather than need to maintain.

- **Windows users and administrators need to explicitly allow fallback from encrypted DNS once configured.** Once Windows has been configured to use encrypted DNS, if it gets no other instructions from Windows users or administrators, it should assume falling back to unencrypted DNS is forbidden.

Based on these principles, we are making plans to adopt DNS over HTTPS (or DoH) in the Windows DNS client. As a platform, Windows Core Networking seeks to enable users to use whatever protocols they need, so we're open to having other options such as DNS over TLS (DoT) in the future. For now, we're prioritizing DoH support as the most likely to provide immediate value to everyone. For example, DoH allows us to reuse our existing HTTPS infrastructure.

For our first milestone, we'll start with a simple change: use DoH for DNS servers Windows is already configured to use. There are now several public DNS servers that support DoH, and if a Windows user or device admin configures one of them today, Windows will just use classic DNS (without encryption) to that server. However, since these servers and their DoH configurations are well known, Windows can automatically upgrade to DoH while using the same server. We feel this milestone has the following benefits:

- **We will not be making any changes to which DNS server Windows was configured to use by the user or network.** Today, users and admins decide what DNS server to use by picking the network they join or specifying the server directly; this milestone won't change anything about that. Many people use ISP or public DNS content filtering to do things like block offensive websites. Silently changing the DNS servers trusted to do Windows resolutions could inadvertently bypass these controls and frustrate our users. We believe device administrators have the right to control where their DNS traffic goes.
- **Many users and applications that want privacy will start getting the benefits without having to know about DNS.** In line with principle 1, the DNS queries become more private with no action from either apps or users. When both endpoints support encryption, there's no reason to wait around for permission to use encryption!
- **We can start seeing the challenges in enforcing the line on preferring resolution failure to unencrypted fallback.** In line with principle 4, this DoH use will be enforced so that a server confirmed by Windows to support DoH will not be consulted via classic DNS. If this preference for privacy over functionality causes any disruption in common web scenarios, we'll find out early.

In future milestones, we'll need to create more privacy-friendly ways for our users to discover their DNS settings in Windows as well as make those settings DoH-aware. This will give users, device admins, and enterprise admins the ability to configure DoH servers explicitly.

Why announce our intentions in advance of DoH being available to Windows Insiders? With encrypted DNS gaining more attention, we felt it was important to make our intentions clear as early as possible. We don’t want our customers wondering if their trusted platform will adopt modern privacy standards or not.

This strikes me as an odd move for Microsoft and it feels as though there might be some heavy duty political lobbying going on behind the scenes with the browser vendors. After all, Microsoft is now on the Chromium bandwagon.

It turns out, there’s more to DoH than might at first be obvious.

However, after doing some reading into DoH in depth I learned that there may be more to DoH than might at first be obvious. And I wanted to share what I learned, and what the future might look like, with our listeners.

The first thing to observe is that if our goal is encryption for privacy and certificate verification for authentication we have both DoH and DoT at our disposal. Since DoH is DNS over HTTPS, and HTTPS is HTTP over TLS, my immediate instinct was to prefer the simpler of the two solutions, which would be simply DNS over TLS with no HTTP(S) involvement at all.

But it turns out that DNS could very nicely leverage some of the advances we've previously discussed in the HTTP protocol, as we moved from HTTP/1.1 to HTTP/2

Rather than create a new URI label, such as dns://, the designers of DoH chose reuse all of the external HTTP protocol and to work within the HTTP query headers. So the path of the query woudl be “/dns-query”, the accept types header would be “application/dns-message” and the content-type header would also be “application/dns-message”. These headers clearly identify and flag the query as DNS and both HTTP GET and POST queries are supported.

Both query types carry the standard binary DNS-on-the-wire data. That binary is the body of the POST query with the query's content-length header indicating its exact binary length. And for a GET query the binary data is Base64url encoded as the object being queried.

The replies to these queries are exactly the same binary DNS data as would have been returned over-the-wire via a traditional UDP query.

Although the IETF's work which is in progress to standardize this uses the DNS on-the-wire format for the payload of DNS queries and responses, there are also implementations of DNS over HTTPS where the payload is formatted using JSON encoding, as defined in RFC 8427. In that case, when this is supported, the query's "accept:" header specifies "application/dns+json". In all other respects, the approaches are identical and the transformation between the on-the-wire format and a JSON format is straightforward.

So the point is that in every way DoH is a DNS query that fits entirely within HTTP and is completely compatible with HTTP. Any system or transport that could handle and process HTTP queries could do the same when those queries are DNS rather than any of the other traditional HTTP content such as text, jpgs, gifs, mp3's and so forth.

We have often talked about the crazy mess that the typical web page has become where the browser pulls down and parses the base page, then turns around and fires off hundreds of DNS queries to obtain the IP service addresses of each of the separate individual domains which are supplying content to the page, then, upon receiving them it establishes connections to each of them and requests the assets they are serving.

If we have seen anything over the past decade of work on the web it has been browsers and protocols being revised and updated and massaged to provide the fastest possible experience for their visitors. We talked about how HTTP/2 is inherently a multiplex protocol where multiple queries are allowed to be outstanding at once and the server sends back what it can as fast as it can. With individual channel packets being tagged, the channel can even be sending multiple replies, overlapped, at once to be reassembled by the receiver. And, as we have discussed, HTTP/2 even allows for the webserver to PUSH the content that it KNOWS the web browsing client will eventually be needing... even before it asks for it. The browser will simply find the asset already in its cache.

Okay... with that background, now imagine a future where the HTTPS website being connected to also offers DNS resolution services via HTTPS. Since DoH is an HTTP protocol that's inherently compatible with everything else flowing through that channel to the browser.

In this future, the website knows all of the large number of domains being referenced by the page the browser has requested. And DNS is inherently a caching protocol and system. So the DNS resolving web server can have previously looked up and cached the most recent IP service addresses of all of the domains it will be asking the browser to go fetch from, and using HTTP/2's built-in send-ahead PUSH technology, that webserver could pre-populate the client's local DNS cache with fresh and valid DNS lookup addresses, thus short circuiting the entire time consuming separate DNS lookup loop.

For this to be safe, we would probably want the website's DNS replies to be DNSSEC so that they could be trusted without concern by their recipient, but that's on the way too.

When you think about it, this change of model is also so much more efficient.

In the original distributed DNS model, every client of an active website needed to perform its own DNS lookups, which are inherently (though admittedly not absolutely) redundant. In this newer model, the website which will be asking its clients to obtain resources from other sites can supply not only the domain names of those resources, but the current IP addresses from which those assets may be retrieved.

This is the potential for DoH and it's the reason why DNS-over-TLS, while it appears to be the lower common denominator, lacks the same powerful potential as DNS-over-HTTP.

And now we know why this podcast’s title is “Pushing DoH.”