Security Now! #769 - 06-02-20

Zoom’s E2EE Design

This week on Security Now!

This week we look at which browsers still permit drive-by website downloads, Google's plan to blacklist notification-abusing websites, a deeper dive into local PC port scanning being performed by websites, Facebook's move to tighten up on high-impact posters, the new lawsuit against Clearview AI, some very interesting strings found embedded in Google's latest messaging app, the very worrisome return of a much more potent StrandHogg for Android, the refusal of SHA-1 to die, a more powerful new USB fuzzer, an update in some nearly finished SpinRite work... and then we take a look at Zoom's newly detailed plans to become the world's most secure teleconferencing platform.

“I’m paid to build pipes, not move rocks…”

Browser News

The state of drive-by malvertising downloads

Yesterday, a guy named Eliya Stein who's with the advertising security firm, Confiant, blogged about some research he had just finished regarding the susceptibility of our current web browsers to so-called drive-by downloading.

In a drive-by download, iframe'd script content coming from a 3rd-party domain -- like any contemporary ad server -- runs in the iframe. Thanks to the fact that JavaScript is able to query and modify the DOM -- the web page's document object model -- the malicious JavaScript is able to programmatically define a download link, then click it, to initiate a zero-click download.

Eliya's attention was drawn to this practice when BoingBoing was delivering malicious downloads to their visitors in this manner. After many complaints BoingBoing disclosed that their site had been hacked. In this case, the malicious code was placed onto their servers so that it was running in the same-origin as the page and had unfettered access to the user's page.

Eliya wrote that "Over the following weeks, we detected this attack on a multitude of sites. Usually this manifests through a CMS compromise that introduces this malicious payload. To date we have identified the presence of the IOCs associated with this attacker on 15 sites in our telemetry, but a cursory look at VirusTotal shows that activity around this particular payload has been ongoing for at least a year and that these C2 domains are leveraged by multiple variants of the malware:"

Having uncovered the mechanism that this attacker uses to drop the malware, Eliya conducted an audit of recent popular browser versions and how they handle downloads that are not initiated by user interaction. The inspiration for doing this analysis was the surprising discovery that most browsers WILL honor downloads triggered from cross-origin frames. What's more, these zero-click downloads are still often possible in Sandboxed Cross-Origin iframes. In one example this has only finally been addressed in Chrome for this last release of Chrome 83:

https://www.chromestatus.com/feature/5706745674465280

Download in Sandboxed Iframes (removed) / Security

Sandboxed iframe can initiate or instantiate downloads.

Chrome is planning on removing this capability - i.e. Chrome is going to block all downloads initiated from or instantiated in a sandboxed iframe by default. The embedder may add "allow-downloads" to the sandbox attributes list to opt in. This allows content providers to restrict malicious or abusive downloads.

Removal is expected in Chrome 83. This allows content providers to restrict malicious or abusive downloads.

Comments

Downloads can bring security vulnerabilities to a system. Even though additional security checks are done in Chrome and the operating system, we feel blocking downloads in sandboxed iframes also fits the general thought behind the sandbox.

Eliya wrote: "For our study, we will test the mechanism inside and outside of cross-origin iframes. The payload we use looks like this:

```
document.write(navigator.userAgent)
link = document.createElement('a');
// try a few different file extensions depending on platform
link.setAttribute('href', 'payload.apk');
link.setAttribute('download', 'download');
link.style.display = 'none';
document.body.appendChild(link);
link.click();
```

This simple bit of code does what I described before: It creates an new link where there was none before, sets its download HREF, makes it invisible, appends it to the existing DOM and then programmatically clicks the link to initiate a download.

So what did Eliya learn about the status of today's web browsers regarding their hosting of these ad hoc download links in iframes?

Chrome 83, does indeed block downloads initiated from sandboxed cross-origin frames. BUT even it is still willing to drop a file if the iframe's sandbox parameters are not set.

However, other desktop browsers will trigger a prompt for the download, including the security conscious Brave and Firefox. And under many circumstances, browsers will drop the download without a prompt at all. For example, the same payload in a non-sandboxed cross-origin iframe in Chrome 83 will drop the download without a prompt, and without any indication that the initiator is not the origin that's displayed in the URL bar. At least here, Firefox's prompting for action is consistent. Safari, apparently attempts to honor the download, but seems to get stuck.

And in the mobile territory behavior is inconsistent, too. For example, Android browsers will be quick to warn when the download is a file with an APK extension, but anything else often doesn't even receive a prompt.

At this point in his blog posting Eliya places a large font bold callout reading:

> **It's 2020 and we can still force downloads that are not user initiated, without any prompt from cross-origin iframes in half of the major browsers out there. Why?**

He concludes by observing: "Within the context of hacked sites as perpetrated by our site-compromising attacker, ad tech has no role to play. No amount of ad or iframe sandboxing can help to mitigate the 'last mile' of this particular attack because the download is dropped from a seemingly trusted source. So, perhaps we should think about *always* prompting users before a download takes place along with an informational modal about the *origin* of the download.

I'll note that SourceForge, for example, uses a script-driven download approach: You go to a page for downloading and a green spinner in the upper left tells you that your download should

begin shortly. And, sure enough, after a short wait you'll notice down in the lower left edge of the browser window that Chrome is downloading the file for you. This is still somewhat counterintuitive and I have sometimes obtained multiple copies of something by advancing to another page and then returning to the magical download page, which triggers another autonomous download.

So I agree with Eliya. There's a point where security really ought to trump automation convenience. Call me old fashioned, (I'm sure I am) but I would like to be "in the loop" anytime my browser is downloading a file into my machine.

Google will be blacklisting notification abusing sites
(Whoops!... I mean "bad listing".)

Google intends to shield Chrome users from two types of abusive alerts: permission request issues which attempt to mislead, trick, or force users into allowing notifications. This is exactly the behavior I talked about a few weeks ago where a website was telling me that I needed to allow notifications to "proceed" into their site. The second form of abuse are the content of the notifications themselves which are used for phishing, delivering malware, and for faking chat messages, warnings, or system dialogs

With the mid-July release of Chrome 84, the browser will also inform users that these abusive websites may be trying to trick them by phishing for private information or promoting malware.

A Google spokesperson explained that: "Abusive notification prompts are one of the top user complaints we receive about Chrome. A large percentage of notification requests and notifications come from a small number of abusive sites. Protecting users from these sites improves user safety & privacy on the web, and makes for a better browsing experience."

Although Chrome 84's abusive notification protection is designed to only trigger for new notification permission requests from abusive websites, Google is also planning to expand these protections to users who have already provided abusive sites with permissions to deliver notifications.

Google said: "Only a small fraction of websites will be affected by this change but we expect the impact on notification volumes will be significant for some users."

Google plans to give abusive websites 30 days notice before tagging them as abusive and website owners can learn whether Google has detected any abusive notification behavior on their website via the Abusive Notifications Report in Search Console.

And at least 30 calendar days prior to the start of enforcement, sites will be notified by email the first time their site fails abusive browser alert checks. This is intended to give them ample time to clean up their act, address the issues behind the status downgrade, and request a re-review.

And starting in August, Chrome will also begin blocking resource-consuming ads which tie up too much of the system's resources without the users' knowledge. These all seem like clean, sensible and responsible actions from the industry's #1 browser source.

Who else is doing the eBay-like ThreatMetrix port scanning?
Over at BleepingComputer, Lawrence Abrams was wondering, and, as he frequently does, he decided to find out. To determine which other sites might be using the same scanning script BleepingComputer reached out to DomainTools, a cybersecurity company specializing in web domain and DNS threat intelligence.

It turned out that when websites load ThreatMetrix's anti-fraud scripts, they load the script from a customer-named online-metrix.net hostname. For example, eBay loads the ThreatMetrix script from src.ebay-us.com. A query to that src.ebay-us.com domain returns a CNAME DNS record (a canonical DNS record, also known as an alias) which points to: h-ebay.online-metrix.net.

As a consequence, DomainTools provided BleepingComputer with a list of 387 unique online-metrix.net that share a similar naming scheme. Then, using this list, BleepingComputer visited the sites for many of the larger companies to verify whether they were, in fact, performing a local client-side port scanning of their visitor's computer for everyone who visited.

BleepingComputer wrote that they did, indeed, detect port scanning when visiting Citibank, TD Bank, Ameriprise, Chick-fil-A, Lendup, BeachBody, Equifax IQ connect, TIAA-CREF, Sky, GumTree, and WePay. Lawrence noted that the specific scanning behavior tended to vary from site to site:

- Citibank, Ameriprise, and TIAA-CREF, immediately port-scanned their visitors' computers when visiting the main page of the site.
- TD Bank, Chick-fil-A, Lendup, Equifax IQ connect, Sky, GumTree, and WePay only port-scanned when visitors attempted to log in.
- And BeachBody.com only port-scanned when checking out.

This detection success suggests that many of the other well-known sites on the list are also using ThreatMetrix and likely scan their visitors at other sensitive times. Those names include: Netflix, Target, Walmart, ESPN, Lloyd Bank, HSN, Telecharge, Ticketmaster, TripAdvisor, PaySafeCard, and possibly even Microsoft. For these sites, BleepingComputer was unable to trigger a scan, but it seems likely that the scans may be present on site pages not visited during their testing. Lawrence posted a Google sheet containing the entire list and I must say that it's rather daunting:

https://docs.google.com/spreadsheets/d/1Nu4lpyZ5PQUIpiLJBddXnr67t5-1y0u40dzyzSYj1gc/edit#gid=0

He concluded his report by noting that these port scans can be blocked using uBlock Origin in Firefox. Unfortunately, uBlock was unable to block the port scans in the new Microsoft Edge or Google Chrome because extensions no longer have adequate permissions to uncloak the DNS CNAME records. (This lack of access is something that uBlock Origin's quite curmudgeonly author, Gorhill, mentioned a while back.) They also tested the Brave Browser, and the port scans were allowed through. So Firefox + uBlock is the only effective blocker.

Facebook to require identity verification for high impact posters
https://about.fb.com/news/2020/05/id-verification-high-reach-profiles/

Last Thursday's Facebook blog posting was titled:

Verifying the Identity of People Behind High-Reach Profiles

It's short and sweet, so I'll share it verbatim:

> *We want to ensure the content you see on Facebook is authentic and comes from real people, not bots or others trying to conceal their identity. In 2018, we started to verify the identity of people managing Pages with large audiences, and now we're extending ID verification to some profiles with large audiences in the US.*
>
> *Moving forward, we will verify the identity of people who have a pattern of inauthentic behavior on Facebook and whose posts start to rapidly go viral in the US. We want people to feel confident that they understand who's behind the content they're seeing on Facebook and this is particularly important when it comes to content that's reaching a lot of people.*
>
> *If someone chooses not to verify their identity or the ID provided does not match the linked Facebook account, the distribution of their viral post will remain reduced so fewer people will see it. In addition, if the person posting is a Page admin, they'll need to complete Page Publishing Authorization and will not be able to post from their Page until their account is verified through our existing Page Publisher Authorization process. IDs will be stored securely and won't be shared on the person's profile. Visit the Help Center for more information on the types of IDs we accept.*
>
> *This verification process is part of our ongoing efforts to create greater accountability on Facebook and improve people's experiences on our apps.*

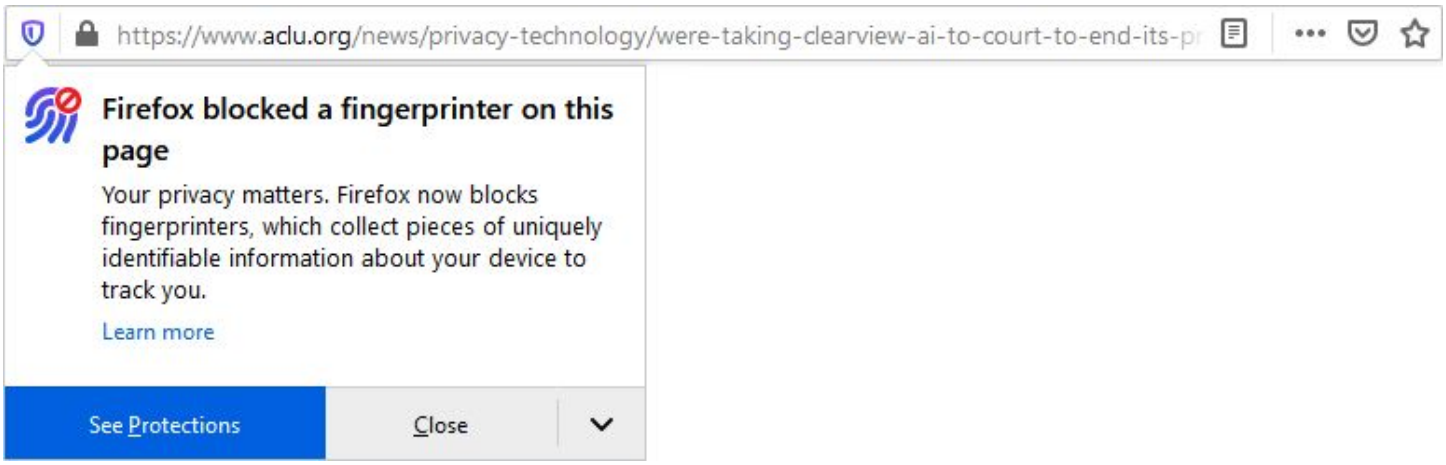
It's very clear that, as a global society, we're currently struggling to figure out how to manage this new Internet-enabled social media. Anonymity is a powerful tool for speech freedom. But that freedom can be so easily abused that some privacy-protected accountability seems like a useful first step as we learn to manage what we have created. During the weekend's energetic protest events, many of the authentic protesters were only too happy to share their identities. Their cause was righteous. But none of those who were looting the stores wanted to be known. Facebook's posting notes that: "*IDs will be stored securely and won't be shared on the person's profile.*" So I presume they are stating that this will be verified internally and not made public.

Clearview AI is in the cross-hairs again.
This time it's the ACLU that's decided that Clearview AI needs to be stopped.

The headline of the American Civil Liberties Union website is full of righteous indignation with the headline:

"We're Taking Clearview AI to Court to End its Privacy-Destroying Face Surveillance Activities"
"The company's surveillance activities are a threat to privacy, safety, and security."

But I need to stop here for just a moment to note the oh-so-delicious irony of the fact that upon visiting www.aclu.org my trust Firefox browser popped up a warning which reads: *"Firefox blocked a fingerprinter on this page."*



That's just too wonderful. You can't make this stuff up. The American Civil Liberties Union, on their website, which is about to get all wound up over the <quote>: "privacy-destroying surveillance activities" of the Clearview AI company, is using web browser fingerprinting to track their site's visitors.

I don't have any backstory for this. The ACLU is a highly-charged and often controversial organization. So perhaps they have some security or safety reason for adding non-browser-cookie tracking to their site. But it is funny to receive this warning when going to a pro-privacy anti-surveillance site.

But that aside, I agree that Clearview's arbitrary photo facial surveillance and image recognition capability represents another aspect of the Internet we have unleashed. When you combine an Internet-connected camera in every pocket with realtime fixed-camera surveillance, massive cloud storage and computing resources, you get consequences that were unintended. What the ACLU wrote and posted last Thursday contains some new information, so I want to share it:

> For several years, a little-known start-up based in New York has been amassing a database of billions of our faceprints — unique biometric identifiers akin to a fingerprint or DNA profile — drawn from personal photos on our social media accounts and elsewhere online. The company has captured these faceprints in secret, without our knowledge, much less our consent, using everything from casual selfies to photos of birthday parties, college graduations, weddings, and so much more.
>
> Unbeknownst to the public, this company has offered up this massive faceprint database to private companies, police, federal agencies, and wealthy individuals, allowing them to secretly track and target whomever they wished using face recognition technology.
>
> That company is Clearview AI, and it will end privacy as we know it if it isn't stopped. We're taking the company to court in Illinois today on behalf of organizations that represent survivors of sexual assault and domestic violence, undocumented immigrants, and other

vulnerable communities. As the groups make clear, Clearview's face surveillance activities violate the Illinois Biometric Information Privacy Act (BIPA), and represent an unprecedented threat to our security and safety.

Face recognition technology offers a surveillance capability unlike any other technology in the past. It makes it dangerously easy to identify and track us at protests, AA meetings, counseling sessions, political rallies, religious gatherings, and more. For our clients — organizations that serve survivors of domestic violence and sexual assault, undocumented immigrants, and people of color — this surveillance system is dangerous and even life-threatening. It empowers abusive ex-partners and serial harassers, exploitative companies, and ICE agents to track and target domestic violence and sexual assault survivors, undocumented immigrants, and other vulnerable communities.

By building a mass database of billions of faceprints without our knowledge or consent, Clearview has created the nightmare scenario that we've long feared, and has crossed the ethical bounds that many companies have refused to even attempt. Neither the United States government nor any American company is known to have ever compiled such a massive trove of biometrics.

Adding fuel to the fire, Clearview sells access to a smartphone app that allows its customers — and even those using the app on a trial basis — to upload a photo of an unknown person and instantaneously receive a set of matching photos.

Clearview's actions clearly violate BIPA. The law requires companies that collect, capture, or obtain an Illinois resident's biometric identifier — such as a fingerprint, faceprint, or iris scan — to first notify that individual and obtain their written consent. Clearview's practices are exactly the threat to privacy that the legislature intended to address, and demonstrate why states across the country should adopt legal protections like the ones in Illinois.

In press statements, Clearview has tried to claim its actions are somehow protected by the First Amendment. Clearview is as free to look at online photos as anyone with an internet connection. But what it can't do is capture our faceprints — uniquely identifying biometrics — from those photos without consent. That's not speech; it's conduct that the state of Illinois has a strong interest in regulating in order to protect its residents against abuse.

If allowed, Clearview will destroy our rights to anonymity and privacy — and the safety and security that both bring. People can change their names and addresses to shield their whereabouts and identities from individuals who seek to harm them, but they can't change their faces.

That's why we're teaming up with lawyers at the ACLU of Illinois and the law firm of Edelson PC, a nationally recognized leader in consumer privacy litigation, to put a stop to Clearview's egregious violations of privacy. We are asking an Illinois state court to order the company to delete faceprints gathered from Illinois residents without consent, and to stop capturing new faceprints unless it complies with the Illinois law.

There is a groundswell of opposition to face surveillance technology, and this litigation is the

latest chapter in an intensifying fight to protect our privacy rights against the dangers of this menacing technology. Across the nation, the ACLU has been advocating for bans on police use of face recognition technology, leading to strong laws in places like Oakland, San Francisco, and Berkeley, California, and Springfield and Cambridge, Massachusetts, as well as a statewide prohibition on use of the technology on police body cams in California.

We won't let companies like Clearview trample on our right to privacy.

So they, like the others who have sued previously, are using Illinois' BIPA law since it makes the strongest case. And as we know, any ruling that might arise from this will likely be challenged and appealed and may finally wind up in front of the US Supreme Court. And, as with the encryption question, that's probably for the best, since someone needs to create some laws to guide us forward... and I hope they are good laws, because this is still the early days. We're not at the beginning of the end, nor the end of the beginning. This is still the beginning of the beginning.

Google Messaging is apparently heading toward E2EE.
We've previously talked about how RCS, which stands for "Rich Communication Services" is the long awaited successor to carrier-provided SMS and MMS.

Last week the folks at APKMirror got their hands on an internal build of Google Messages version 6.2. The APK Insight team dug in to see what appears to be coming with the next version of Google Messages. The biggie that caught their attention was pretty clear evidence of end-to-end encryption being added to RCS messages.

For some years now PCS messaging has been seen as the successor to SMS and MMS messages and an open competitor to Apple's iMessage. However, one thing that iMessage and the Apple ecosystem offers that RCS could not was the ability to exchange privacy-protected messages with the benefit of end-to-end encryption. Of course, we'll need to wait for an official whitepaper before we're able to see how all of the various pieces fit together. But if nothing else Google has the significant advantage of being a late mover in this space with many examples of earlier efforts to guide their ultimate design.

But what's interesting about what the APKMirror folks have uncovered is the first evidence that such a thing is in the works at all. The reverse engineers found 12 separate strings that make very clear reference to RCS-based encryption:

1. <string name="encrypted_rcs_message">End-to-End Encrypted Rich Communication Service message</string>

2. <string name="send_encrypted_button_content_description">Send end-to-end encrypted message</string>

3. <string name="e2ee_conversation_tombstone">Chatting end-to-end encrypted with %s</string>

4. <string name=”metadata_encryption_status”>End-to-end encrypted message</string>

5. <string name=”e2ee_fail_to_send_retry_description”>Resend as chat</string>

6. <string name=”encryption_fallback_title”>Send unencrypted messages?</string>

7. <string name=”encryption_default_fallback_body”>”SMS/MMS texts aren’t end-to-end encrypted.\n\nTo send with end-to-end encryption, wait for improved data connection or send messages now as SMS/MMS.”</string>

8. <string name=”encryption_fallback_dialog_accept_button”>Send unencrypted</string>

9. <string name=”encryption_fallback_dialog_decline_button”>Wait</string>

10. <string name=”encryption_sent_fallback_body”>”SMS/MMS texts aren’t end-to-end encrypted.\n\nTo send with end-to-end encryption, wait until %1$s has data connection or send messages now as SMS/MMS.”</string>

11. <string name=”etouffee_to_telephony_setting_title”>Let other apps access end-to-end encrypted messages</string>

12. <string name=”location_attachment_picker_send_encrypted_content_description”>Send end-to-end encrypted message with selected location %1$s</string>

The return of a much more worrisome StrandHogg

More than One Billion Android phones -- all phones not running Android 10 -- are vulnerable.

Back at the beginning of December last year, this was the description of Strandhogg 1.0:

> The Hacker News wrote: Cybersecurity researchers have discovered a new unpatched vulnerability in the Android operating system that dozens of malicious mobile apps are already exploiting in the wild to steal users' banking and other login credentials and spy on their activities.
>
> Dubbed Strandhogg, the vulnerability resides in the multitasking feature of Android that can be exploited by a malicious app installed on a device to masquerade as any other app on it, including any privileged system app.
>
> In other words, when a user taps the icon of a legitimate app, the malware exploiting the Strandhogg vulnerability can intercept and hijack this task to display a fake interface to the user instead of launching the legitimate application.

That was then. Last week the same team of Norwegian cybersecurity researchers who found and named Strandhogg being used in the wild unveiled details of an extremely critical successor vulnerability (CVE-2020-0096) affecting the Android operating system that could allow attackers to carry out a much more sophisticated version of Strandhogg attack. Note the low number of the 2020 CVE. This was clearly assigned during the first few hours of the new year.

Naturally, they call it "Strandhogg 2.0," and this vulnerability affects all Android devices, except those running the very latest version, Android Q / 10. Unfortunately, Q/10 is currently running on only 15-20% of Android-powered devices, which leaves the other billion+ Android smartphones vulnerable to the attackers.

https://promon.co/strandhogg-2-0/ **:**

> Promon researchers have discovered a new elevation of privilege vulnerability in Android that allows hackers to gain access to almost all apps.
>
> Classified as a 'Critical Severity' (CVE-2020-0096) by Google, the vulnerability has been named StrandHogg 2.0 by Promon due to its similarities with the infamous StrandHogg vulnerability discovered by the company in 2019.
>
> While StrandHogg 2.0 also enables hackers to hijack nearly any app, it allows for broader attacks and is much more difficult to detect, making it, in effect, its predecessor's 'evil twin'.
>
> Having learned from StrandHogg and subsequently evolved, StrandHogg 2.0 does not exploit the Android control setting 'TaskAffinity', which hijacks Android's multitasking feature and, as a result, leaves behind traceable markers.

Strandhogg 2.0 is executed through reflection, allowing malicious apps to freely assume the identity of legitimate apps while also remaining completely hidden.

Utilising StrandHogg 2.0, attackers can, once a malicious app is installed on the device, gain access to private SMS messages and photos, steal victims' login credentials, track GPS movements, make and/or record phone conversations, and spy through a phone's camera and microphone.

Whereas StrandHogg 1.0 leveraged a vulnerability in the multitasking features of Android, this next generation Strandhogg 2.0 flaw is an elevation of privilege vulnerability that allows hackers to gain access to almost all apps. When a user taps the icon of a legitimate app, the malware exploiting Strandhogg vulnerabilities can intercept and hijack this activity/task to display a fake interface to the user instead of launching the real application.

And unlike StrandHogg 1.0 that can only attack apps one at a time, the latest flaw allows attackers to dynamically attack nearly any app on a given device simultaneously at the touch of a button, all without requiring pre-configuration of each targeted app.

StrandHogg flaws are potentially dangerous and concerning because:

- it is almost impossible for targeted users to spot the attack,
- it can be used to hijack the interface for any app installed on a targeted device without requiring configuration,
- it can be used to request any device permission fraudulently,
- it can be exploited without root access,
- it works on all previous versions of Android, except Q.
- it doesn't need any special permission to work on the device.

Besides stealing login credentials through a convincing fake screen, the malware app can also escalate its capabilities significantly by tricking users into granting sensitive device permissions while posing as a legitimate app.

StrandHogg 2.0 is also much more difficult to detect because of its code-based execution.

Attackers exploiting the original StrandHogg vulnerability must explicitly and manually enter the apps they are targeting into Android Manifest, with this information then becoming visible within an XML file which contains a declaration of permissions, including what actions can be executed. This declaration of required code, which can be found within the Google Play store, is not present when exploiting StrandHogg 2.0.

As no external configuration is required to execute StrandHogg 2.0, this allows the hacker to further obfuscate the attack, as code obtained from Google Play will not initially appear suspicious to developers and security teams. Malware that exploits StrandHogg 2.0 will also be harder for anti-virus and security scanners to detect and, as such, poses a significant danger to the end-user.

Promon predicts that attackers will look to utilise both StrandHogg and StrandHogg 2.0 together because both vulnerabilities are uniquely positioned to attack devices in different ways, and doing so would ensure that the target area is as broad as possible.

Likewise, many of the mitigations that can be executed against StrandHogg do not apply to StrandHogg 2.0 and vice-versa. StrandHogg 2.0 exploits do not impact devices running Android 10. However with a significant proportion of Android users reported to still be running older versions of the OS, a large percentage of the global population is still at risk.

According to data from Google, as of April 2020, 91.8% of Android active users worldwide are on version 9.0 or earlier: Pie (2018), Oreo (2017), Nougat (2016), Marshmallow (2015), Lollipop (2014), KitKat (2013), Jellybean (2012) and Ice Cream Sandwich (2011).

The researchers responsibly reported the new vulnerability to Google in December last year, after which Google generated a patch and shared it with smartphone manufacturing companies in April. They have started rolling out software updates to their respective users this month.

Though there is no effective and reliable way to block or detect task hijacking attacks, users can still spot such attacks by keeping an eye on discrepancies such as when...

- an app you're already logged into is asking for a login,
- permission popups that do not contain an app name,
- permissions asked from an app that shouldn't require or need the permissions it asks for,
- buttons and links in the user interface do nothing when clicked on,
- the back button does not work as expected.

So... be sure to get the patch when it’s available for your device. If it is exploited, Strandhogg v2.0 will likely only be used in targeted attacks. But no one wants to have a phone that remains vulnerable to that sort of known exploitation. And, as always, any Android device that’s not updated will forever remain vulnerable.

The SHA-1 hash to finally be dropped from OpenSSH
The release notes from last Wednesday's OpenSSH update read:

> It is now possible to perform chosen-prefix attacks against the SHA-1 algorithm for less than $50,000. For this reason, we will be disabling the "ssh-rsa" public key signature algorithm by default in a near-future release.
>
> This algorithm is unfortunately still used widely despite the existence of better alternatives, being the only remaining public key signature algorithm specified by the original SSH RFCs.

The very real problem is that the world is full of aging network switches and low-cost embedded machines running ATMs and industrial control systems. They all contain older SSH servers that only speak SHA-1. This means that new systems, that will be lacking support for SHA-1, will no longer be able to connect as they always could. It's truly a mess. It effectively means that much as we may wish to stop using SHA-1, we really cannot. Old OpenSSH-based clients WILL be retained for use in connecting to old servers. But those older clients will no longer be maintainable, since any updates to them would remove their support for SHA-1. Thus, embedded servers that cannot be updated are holding back updates to clients.

What happens when you fuzz USB?
You find 26 new and previously unknown flaws, many critical, residing in our USB-supporting desktop operating systems. We've talked many times about how important fuzzing is as a semi-automated means of uncovering previously unknown flaws in software.

The researchers who built this new USB fuzzing tool used it to uncover one bug in FreeBSD, three in MacOS -- two resulting in an unplanned reboot and one freezing the system -- four flaws in Win8 and Win10 -- resulting in Blue Screens of Death. But the greatest number of bugs, and those most severe, were found in Linux: a total of 18.

16 of those 18 were memory bugs having high-security impact in various Linux subsystems (USB core, USB sound, and net-work), one bug resided in the Linux USB host controller driver, and the last in a USB camera driver.

The researchers responsibly reported these bugs to the Linux kernel team, along with proposed patches to reduce the burden on the kernel developers when fixing the reported vulnerabilities. This is one of the powerful advantages of Linux being open source. Those finding a problem can directly examine the cause of the problem in the source and propose patches to repair the problem. This is not something that can be done with Windows or MacOS since those systems are close source mysteries.

Of those 18 Linux bugs, the research team said 11 received a patch since their initial reports last year. Ten of the 11 bugs also received a CVE designation. Additional patches are expected in the near future for the seven remaining issues. The researchers said: “The remaining bugs fall into two classes: those still under embargo/being disclosed and those that were concurrently found and reported by other researchers.”

The researchers plan to present their research at the Usenix Security Symposium virtual security conference, scheduled for August 2020.

Having been preceded by other USB fuzzing tools including vUSBf, syzkaller, and usb-fuzzer, they are not the first to fuzz USB, but they feel that their "USBFuzz" tool is superior to those that came before because it provides testers more control over the test data and is portable to non-UNIX-like operating systems, unlike any of the other tools.

USBFuzz will be released on GitHub as an open source project following their Usenix talk. When that happens, the repo will be available here: https://github.com/HexHive/USBFuzz

SpinRite

Yesterday I reached the end of a surprisingly challenging bit of development for the SpinRite project. As we all know, SpinRite has always run on top of the FreeDOS clone of MS-DOS. The SpinRite v6.x series will continue to do this, since that will let get something out sooner. But although that SpinRite will no longer use the BIOS, it still uses DOS, and DOS can only boot from a BIOS. So once the SpinRite v6.x's are finished, my current plan is to immediately remove SpinRite's dependence upon DOS so that it boots natively as its own OS. And that will allow it to also boot from UEFI-based systems that will not boot any form of DOS.

One of the most frequently used words in that prior paragraph is "boot", which is where my efforts for the past three weeks have been spent. I now have a little windows command line utility that can take a removable USB drive in any possible initial state -- already formatted, unformatted, formatted with something foreign like a MacOS GPT plus FAT file system -- whatever, with or without any defined partition and whether or not bootable, and replace it with a pure and perfect new bootable system. It runs on anything from WinXP through Win10. I'm using only my own new partition and file system code so that I'll be able to evolve this codebase in the future when it needs to be prepping a more complex bootable drive that can boot under either the BIOS of UEFI. It was surprisingly tricky to get it working under every condition and circumstance… but thanks for the terrific testing of the gang who is hanging out in the grc.spinrite.dev newsgroup, it appears that we're there. Once it's finished it, that code will be moved into SpinRite's boot-prep GUI for easy use.

I spent this much time to get boot-prep exactly right, because everything in the future will be using this code, the initial SpinTest drive benchmark, which will be freeware and which everyone here will be invited to play with, all of the future SpinRite work, and also the Beyond Recall secure drive wiping utility. The utility has even managed to repair a number of its tester's thumb drives that were either dead or messed up in various ways so that they had stopped working. And since it also usually recovers a small bit of previously unavailable and unused space on USB drives, I expect I'll also release it as a freeware command-line removable USB drive formatter.

Zoom's E2EE Design

https://github.com/zoom/zoom-e2e-whitepaper/raw/master/zoom_e2e.pdf

- From Zoom Video Communications we have Josh Blum, Simon Booth, Oded Gal, Maxwell Krohn, Karan Lyons, Antonio Marcedone, Mike Maxim, Merry Ember Mou, Jack O'Connor & Miles Steele.
- From Johns Hopkins we have well-known cryptographer and professor Matthew Green.
- From Stanford University we have Alex Stamos.
- And we also have privacy expert Lea Kissner, formerly Google's global lead for privacy technology.

Their names adorn the 25-page detailed technical cryptography paper they collectively published on Github Friday before last, on May 20th. That paper is simply titled: "E2E Encryption for Zoom Meetings" and it begins by stating its purpose:

> Hundreds of millions of participants join Zoom Meetings each day. They use Zoom to learn, among classmates scattered by recent events, to connect with friends and family, to collaborate with colleagues and, in some cases, to discuss critical matters of state. Zoom users deserve excellent security guarantees, and Zoom is working to provide these protections in a transparent and peer-reviewed process. This document, mindful of practical constraints, proposes major security and privacy upgrades for Zoom.
>
> We are at the beginning of a process of consultation with multiple stakeholders, including clients, cryptography experts, and civil society. As we receive feedback, we will update this document to reflect changes in roadmap and cryptographic design.

As we've observed many times across a great many different security challenges, in the end, the weakest link in an end-to-end encryption system is the management of that system's keys. The task of actually encrypting and decrypting data at each end, given a shared key, has long ago been solved. How often are we talking about the way Apple manages their iMessage keys, and how much does our US Attorney General lust for some form of magical "Golden Key?"

So Zoom's security team has properly determined that the first problem to solve, the first place to tighten things, is with the system's key management. Once you have that, then the next weak link is identity: Verifying that meeting participant identities are not being spoofed or forged. But it makes no sense to worry about that before you know that the resulting keys are safe.

Here's how Zoom describes the operation of their current system:

> Zoom Meetings currently use encryption to protect identity, data for meeting setup, and meeting contents. Zoom provides software for desktop and mobile operating systems and embeds software in Zoom Room devices. In this document, when we refer to "Zoom clients" we include all of these various forms of packaging. Crucially, these are systems to which we can deploy cryptographic software. Zoom Meetings also supports web browsers through a combination of WebRTC and custom code. If enabled, Zoom meetings support the use of

clients not controlled by Zoom, namely phones using the Public Switched Telephone Network (PSTN) and room systems supporting SIP and H.323.

In the meeting setting (as opposed to webinars), Zoom supports up to 1,000 simultaneous users. When a Zoom client gains entry to a Zoom meeting, it gets a 256-bit per-meeting key created by Zoom's servers, which retain the key to distribute it to participants as they join. In the version of Zoom's meeting encryption protocol set for release on May 30, 2020, this per-meeting key is used to derive a per-stream key by combining the per-meeting key with a non-secret stream ID using an HMAC function. Each stream key is used to encrypt audio/video (UDP) packets using AES in GCM mode, with each client emitting one or more uniquely-identified streams. Those packets are relayed and multiplexed via one or more Multimedia Routers (MMR) in Zoom's infrastructure. The MMR servers do not decrypt these packets to route them. There is no mechanism to re-key a meeting. Videoconferencing systems in which the server relies on plaintext access to the meeting content to perform operations such as multiplexing would be exceptionally difficult to secure end-to-end. In this design, we take advantage of the fact that the Zoom servers do not require any access to meeting content, allowing end-to-end security at exceptionally large scale.

If a PSTN or SIP client is authorized to join, the MMR provides the per-meeting encryption key to specialized connector servers in Zoom's infrastructure. These servers act as a proxy: they decrypt and composite the meeting content streams in the same manner as a Zoom client and then re-encode the content in a manner appropriate for the connecting client. Zoom's optional Cloud Recording feature works similarly, recording the decrypted streams and hosting the resulting file in Zoom's cloud for the user to access. In the current design, Zoom's infrastructure brokers access to the meeting key.

This current design provides confidentiality and authenticity for all Zoom data streams, but it does not provide “true” end-to-end (E2E) encryption as understood by security experts due to the lack of end-to-end key management. In the current implementation, a passive adversary who can monitor Zoom's server infrastructure and who has access to the memory of the relevant Zoom servers may be able to defeat encryption. The adversary can observe the shared meeting key (MK), derive session keys, and decrypt all meeting data. Zoom's current setup, as well as virtually every other cloud product, relies on securing that infrastructure in order to achieve overall security; end-to-end encryption, using keys at the endpoints only, allows us to reduce reliance on the security of Zoom infrastructure.

This is what they have had, and it’s an entirely sane architecture. As we know, they were using a bad encryption mode, ECB - electronic code book - which tends to leak information by failing to chain successive blocks of encryption to the previous block. But that’s been fixed by their rapid switch to the AES-GCM cipher mode.

They’ve divided the task of getting from where they are now to where they want to go into four successive phases. There’s no way for me to discuss the detailed design of the various cryptographic algorithms on an audio podcast because there is a large amount of it. They are all laid bare in this paper for anyone who is curious. And that doesn’t really matter anyway, since when it’s done correctly it works. The overall architecture is the key, and that **can** be usefully characterized.

The four phases into which the entire effort has been divided are:

1. Client Key Management
2. Identity
3. A Transparency Tree
4. Real-Time Security

Let’s look at what they have explained about each of those, briefly:

In the first phase, we will roll out public key management, where every Zoom application generates and manages its own long-lived public/private key pairs; those private keys are known only to the client. From here, we will upgrade session key negotiation so that the clients can generate and exchange session keys without needing to trust the server. In this phase, a malicious party could still inject an unwanted public key into this exchange. We offer "meeting security codes" as an advanced feature, so motivated users can verify public keys. The security to be achieved here will approximate those of Apple's FaceTime and iMessage products.

The primary improvement in Phase I is that a server adversary must now become active (rather than passive) to break the protocol. In Phase I, we will support native Zoom clients and Zoom Rooms. We will not support Web browsers, PSTN dial-in, and other legacy devices. There also will be no support for "Join Before Host", Cloud Recording, and some other Zoom features.

[So phase I takes them from the current encrypted system where the individual client encryption keys could be captured by a passive observer at the Zoom server to a system where Diffie-Hellman secure key establishment is used to blind a passive observer to the client’s session keys.]

Phase II: Identity

In the first phase, clients trust Zoom to accurately map usernames to public keys. A malicious Zoom server in theory has the ability to swap mappings on-the-fly to trick participants into entering a meeting with imposters. In Phase II, we will introduce two parallel mechanisms for users to track each other's identities without trusting Zoom's servers. For users authenticating to Zoom via Single-Sign-On (SSO), we will allow the SSO IDP (Identity Provider) to sign a binding of a Zoom public key to an SSO identity, and to plumb this identity through to the UI. Unless the SSO or the IDP has a flaw, Zoom cannot fake this identity. Second, we allow users to track contacts' keys across meetings. This way, the UI can surface warnings if a user joins a meeting with a new public key.

[So this tightens-up security in a way similar to other semi-identity-managed messaging systems such as Signal or Threema, where a contact’s public keys are remembered by the client and the user is warned if that public key should later change.]

Phase III: Transparency Tree

In the third phase, we will implement a mechanism that forces Zoom servers (and SSO providers) to sign and immutably store any keys that Zoom claims belong to a specific user, forcing Zoom to provide a consistent reply to all clients about these claims. Each client will periodically audit the keys that are being advertised for their own account and surface new additions to the user. Additionally, auditor systems can routinely verify and sound the alarm on

any inconsistencies in their purview. In this scenario, if Zoom were to lie about Alice's keys (say, in order to join a meeting which Alice is invited to), it would have to lie to everyone in a detectable way. We will obtain these guarantees by building a "transparency tree," similar to those used in Certificate Transparency and Keybase.

During this phase we will also provide the capability for meeting leaders to "upgrade" a meeting to end-to-end encrypted once it has begun, provided that all attendees are using the necessary client versions and incompatible features are not in use. Such incompatible features include PSTN dial-in, SIP/H.323 room systems and cloud recordings. Meetings that cannot be upgraded will have the option grayed-out. We also re-enable "Join Before Host" mode.

Phase IV: Real-Time Security

Consider this hypothetical attack against the Phase III design: a malicious Zoom server introduces a new "ghost" device for Bob, a user who does not have their Identity Provider vouch for their identity. The attacker, using this fake new device, starts a meeting with Alice. Alice sees a new device for Bob but does not check the key's fingerprint. After the fact, Bob can catch the server's malfeasance, but only after the attacker tricked Alice into divulging important information. The transparency tree encourages a "trust but verify" stance, where intrusions cannot be covered up. In Phase IV, we look to the future where Bob should sign new devices with existing devices, use an SSO IDP to reinforce device additions, or delegate to his local IT manager. Until one of these conditions is met, Alice will look askance at Bob's new devices.

So… across these next four phases of security architecture design, we will end up with a state-of-the-art multi-party teleconferencing system which provides robust guarantees of participant identity, takes all key-awareness away from the Zoom infrastructure, and is able to proactively alert any Zoom user when something looks wrong.

One thing I noted that I did want to point out about the crypto details was this paragraph:

"In an ideal world, all public key cryptographic operations could happen via Diffie-Hellman over Curve25519, and EdDSA over Ed25519. Relative to others, this curve and algorithm family have shown a consistent track record for resilience to common cryptographic attacks and implementation mistakes. These algorithms are currently in review for FIPS certification but, unfortunately, are not currently approved. Therefore, in some cases (like government uses that require FIPS certification) we must fall back to FIPS-approved algorithms, like ECDSA and ECDH over the P-384 curve. The protocols we use have support for both algorithm families, and for now "doubles up" all public key operations to eliminate error-prone "If/then/else" branching. Once certification succeeds, we can safely "no-op" the operations over P-384."

That's a clever solution. Since the operations are not time-sensitive and are only being used to establish keys and identities, why not use BOTH the 25519 and P-384 in series. Then, once 25519 has been shown to be sufficient, simply drop P-384 out of the chain.