Security Now! #918 - 04-11-23

A Dangerous Interpretation

This week on Security Now!

This week we seek answers: What did Microsoft and Fortra ask from the courts, and what did the courts say in return? When can chatting with ChatGPT leak corporate secrets? Why has Apple suddenly updated many much older of their iDevices? Why bother naming a six year old ongoing WordPress attack campaign? Which Samsung handsets just went out of security support? What two user-focused policy changes has Google just made for Android users? and do we really have additional ChatGPT hysteria? After answering those questions, and examining an example of the benefit of rewriting solid state non-volatile storage, we're going to take a rather deep dive into a tool that was meant for good, but which I fear may see more use for evil.

Why developers sometimes feel misunderstood.

Security News

Microsoft and Fortra go on the offensive

A few weeks ago we talked about how wrong it was that Sony had chosen to legally force the DNS provider 9.9.9.9 to remove DNS resolution from its service as a roundabout means of blocking access to pirated Sony intellectual property — some MP3 tunes — being hosted on the Internet. The proper action would have been to legally pursue the provider hosting the content, or illegal domain's registrar to shut down the site. But Sony's complaint said, well "they didn't return our calls."

I was reminded of this due to some news last week from Microsoft's Digital Crimes Unit, the DCU, from whom Sony could take a few lessons. Three organizations, Microsoft's DCU, the cybersecurity company FORTRA (who also happens to be sponsor here on the TWiT Network), and the Health Information Sharing and Analysis Center (Health-ISAC) have joined together to take technical and legal action to disrupt cracked, legacy copies of Cobalt Strike and other abused Microsoft software, which have been used by cybercriminals to distribute malware, including ransomware.

In a change in the way Microsoft's DCU has previously worked, they are teaming up with Fortra to remove illegal, legacy copies of Cobalt Strike so they can no longer be used by cybercriminals. We've talked about Cobalt Strike before, but it's due for a review.

Although Cobalt Strike is frequently associated with malware — which is what Microsoft and Fortra are hoping to change — Cobalt Strike is actually a legitimate Fortra software product. It's a popular and powerful post-exploitation tool used for adversary simulation. And as it happens, sometimes, older versions of the software have been altered by the bad guys who use its extreme power not for benign Red Team testing and exercises, for for their criminal purposes. "Cracked" illegal copies of Cobalt Strike have been used to launch destructive attacks, such as those against the Government of Costa Rica and the Irish Health Service Executive and countless more. Microsoft SDK's and APIs are abused as part of the coding of the malware as well as the criminal malware distribution infrastructure to target and mislead victims.

Specifically, the ransomware families associated with or deployed by cracked copies of Cobalt Strike have been linked to more than 68 ransomware attacks impacting healthcare organizations in more than 19 countries around the world. These attacks have cost hospital systems millions of dollars in recovery and repair costs, plus interruptions to critical patient care services including delayed diagnostic, imaging and laboratory results, canceled medical procedures and delays in delivery of chemotherapy treatments, just to name a few.

So, Friday before last, March 31, 2023, the U.S. District Court for the Eastern District of New York issued a court order allowing Microsoft, Fortra, and that Health-ISAC group to forcibly disrupt the infrastructure being used by criminals to facilitate their attacks. The court order allows the trio to notify relevant ISPs and computer emergency readiness teams (CERTs) who will then assist in taking the infrastructure offline which severs the connections between criminal operators and infected victim systems.

This is what Sony should have done. Rather than obtaining a court order against an innocent DNS provider, which, by the way, doesn't really solve the problem anyway, Sony should have

done whatever's necessary to take the offending pirate server off the air, just as this group is now doing with malicious Cobalt Strike instances globally.

Microsoft noted that they are also expanding a legal method used successfully to disrupt malware and nation state operations to target the abuse of security tools used by a broad spectrum of cybercriminals. Disrupting cracked legacy copies of Cobalt Strike will significantly hinder the monetization of these illegal copies and slow their use in cyberattacks, forcing criminals to re-evaluate and change their tactics.

Fortra noted that in recognition of the power of their tools, they have always taken considerable steps to prevent the misuse of their software which includes stringent customer vetting practices. However, criminals are known for stealing older versions of security software, including Cobalt Strike, creating cracked copies to gain backdoor access to machines and deploy malware. Ransomware operators have been observed using cracked copies of Cobalt Strike and Microsoft software to deploy Conti, LockBit, and other ransomware as part of the newer ransomware as a service (RaaS) business model.

Although the identities of those conducting the criminal operations are currently unknown, the group has detected malicious infrastructure across the globe, including in China, the U.S. and Russia. In addition to financially motivated cybercriminals, they've observed threat actors using cracked copies to act in the interests of foreign governments, including from Russia, China, Vietnam and Iran.

Microsoft has said that its Defender security platform has detected around 1.5 million infected computers communicating with cracked Cobalt Strike servers over the past two years. And in 2020, a Recorded Future report found that more than 1,400 malware C&C servers were using Cobalt Strike as their backend at that time. The Censys search engine currently returns ~540 Cobalt Strike servers hosted in the wild. So, yeah, it's a problem. And it's unfortunate that bad guys are using powerful software meant for training and attack simulation. But that's the world we live in and going after the bad guys' communications, since they are still needing to use the public Internet, is clearly the right solution. It sure makes a lot more sense than Sony compelling a single DNS provider to block a bad guy's DNS lookups. That's so wrong.

Can ChatGPT keep a secret?

I stumbled upon this and knew that our listeners would get a kick out of it. It seems that some employees of Samsung Semiconductor were using ChatGPT to help them diagnose and repair some problematic code. But in order to do this they needed to upload the code and some documents to ChatGPT so that it could see what was puzzling the employees. The only problem was... the uploads contained Samsung’s sensitive proprietary information. After finding three instances of this, Samsung warned its employees against using ChatGPT in their daily work. The data uploaded to ChatGPT included internal documents and code meant to identify defective chips. Samsung now limits the length of questions submitted to ChatGPT to 1 Kbytes while the company develops its own internal ChatGPT-like AI for internal use.

This brings to light something I hadn't considered before, which is that it will likely become quite common within an organization to want to leverage the potential power of these new large

language model AIs for internal proprietary work and research. But what if the necessary details of that research cannot be allowed to leave the company's control? So this is a very different application than AI-assisted travel planning or meal design.

And this, in turn, suggests that before long we're going to begin seeing AI companies offering to sell stand-alone pre-programmed AI systems for exclusive use within and by a single organization. And such systems will likely be compartmentalized so that the published AI side can be refined and upgraded over time while keeping any proprietary information that's incrementally informed the AI separate and safe.

It's still difficult for me to believe that we're talking about this, like this. But it's suddenly very real.

Apple updates their OS's

Last Friday, Apple released security updates for iOS, iPadOS, macOS, and Safari. The updates will remove two flaws and thus terminate the use of a pair of 0-day's that were being exploited in the wild. If you check, you're looking for v16.4.1 in iOS and iPadOS and Safari, and macOS Ventura will update to v13.3.1. And sure enough, this morning I checked my iPhone and it was still lagging behind four days after the update's release. I think this demonstrates that this is not a 5-alarm fire and Apple's goal is to eventually bring all devices current without it being an emergency. These individual updates are large and there are a lot of Apple devices out in the world eager to remain current. But there's just no way — and really no need — to send the same large chunk of code out to every wandering iDevice at once. So for things like this, Apple trickles the updates out unless a user specifically checks to see whether they are current.

The point is, we're not seeing mass attacks using new vulnerabilities, because that would bring them to everyone's attention and cause them to be found, identified and quickly fixed. Instead, those who are finding ways to penetrate today's mobile devices are leveraging them for targeted infiltration. In this case, this supposition is supported by the fact that the two flaws that were fixed were reported to Apple by Google's TAG team group and Amnesty International's Security Lab. The fact that Amnesty International's Security Lab was involved further supports the notion that the devices being used against individual high-value political targets. There are headlines in the tech press urging their readers to **update immediately!** due to the danger of being a few days late. I would say "update when you get around to it if you're worried" otherwise Apple will get around to it for you and eventually you'll be asked to re-authenticate to your device after an otherwise transparent overnight auto-update.

The two vulnerabilities were a use after free issue in WebKit that lead to arbitrary code execution when processing maliciously crafted web content and an out-of-bounds write issue in IOSurfaceAccelerator that enabled apps to execute arbitrary code with kernel privileges. Apple indicated that it has addressed the first with improved memory management and the second with better input validation, adding that it's aware the bugs *"may have been actively exploited."*

Out of all of this, that's the only thing that annoys me. No one is being more proactive than Apple – and Google is clearly at parity. But it would be nice if publishers were more forthcoming with their language. I suppose that their attorneys won't let me be. Those were both 0-day code

execution flaws, obviously having been used in the wild. No one should blame Apple for this. But I suppose the point is that someone would blame them if they told the whole truth. So at least we can be thankful that our devices are being kept up to date. And it's also worth noting that further details about these two vulnerabilities have been withheld due to their active exploitation and to prevent additional bad guys from learning about and also abusing them. In other words, yeah... they're in use right now.

One last point is that we can presume that Apple is even aware that these flaws are being abused on **older** devices that would not normally receive updates... because yesterday, Apple backported patches to fix these problems to older iPhones, iPads, and Macs. Now, updates for these are also being made available for older out-of-patch cycle devices. All models of the iPhone 6s and 7, 1st gen iPhone SE, iPad Air 2, 4th gen iPad mini, and the 7th gen iPod touch. I have a 7th generation iPod touch sitting next to me because it has a headphone jack. After reading that it might be getting an update, I checked and sure enough, it's updating right now. So I'm just saying that this is not something Apple does for a "may be actively exploited" vulnerability. They know these flaws are being used against their selected users in the field. Note that even older macOS Big Sur is being updated to v11.7.6 and Monterey to v12.6.5.

(2nd Sponsor Insert??)

Wordpress under attack... again

WordPress is another constant on this podcast because around 43% of the Internet's websites are built on WordPress. It's far and away the most common CMS — Content Management System. With WordPress at around 43%, the runner up is a distant #2 held by Shopify with a mere 4.1%. In 3rd place is Wiz at 2.3% followed by SquareSpace at 2%. So, yeah, WordPress is the big target on the web.

Last Wednesday the team at Securi finally gave a name to a long-running WordPress exploitation campaign they've been tracking for years. They named it: **Balada Injector**. Their posting last week was titled: *"Balada Injector: Synopsis of a Massive Ongoing WordPress Malware Campaign"* and they wrote:

> *Our team at Sucuri has been tracking a massive WordPress infection campaign since 2017 — but up until recently never bothered to give it a proper name. Typically, we refer to it as an ongoing long lasting massive WordPress infection campaign that leverages all known and recently discovered theme and plugin vulnerabilities. Other organizations and blogs have described it in a similar manner, sometimes adding terms like "malvertising campaign" or naming domains that it was currently using, which amount to several hundred over the past 6 years.*
>
> *This campaign is easily identified by its preference for String.fromCharCode obfuscation, the use of freshly registered domain names hosting malicious scripts on random subdomains, and by redirects to various scam sites including fake tech support, fraudulent lottery wins, and more recently, push notification scams displaying bogus captcha pages asking users to "Please Allow to verify, that you are not a robot".*
>
> *Since 2017, we estimate that over one million WordPress websites have been infected by this*

campaign. Each year it consistently ranks in the top 3 of the infections that we detect and clean from compromised websites.

Last year, in 2022 alone, our external website scanner SiteCheck detected this malware over 141,000 times, with more than 67% of websites loading scripts from known Balada Injector domains. We currently have more than 100 signatures covering both front-end and back-end variations of the malware injected into server files and WordPress databases.

As you can imagine, referring to this massive infection campaign using generic terms has not been convenient. However, assigning a name to this malware was never at the top of our priority list. Our security researchers deal with dozens of new malware samples every day, so we typically don't dwell too much on well-known malware that is adequately covered by detection rules and only necessitates minor adjustments when we spot a new wave.

In late December last year, our colleagues at Dr.Web shared some valuable information that led us to choose the name "Balada Injector".

A post published on December 30, 2022, titled "Linux backdoor malware infects WordPress-based websites" caught our attention, and it was widely circulated in Internet security blogs with titles like "Linux Malware uses 30 plugin vulnerabilities to target WordPress sites". The article discusses two variants of the malware: Linux.BackDoor.WordPressExploit.1 and Linux.BackDoor.WordPressExploit.2 and provides a comprehensive overview, including targeted plugins and various indicators of compromise.

The interest generated by this information prompted numerous inquiries from various sources, leading us to examine the post closely on New Year's Eve to determine if immediate action was required. To our surprise, we instantly recognized the described malware as the ongoing, massive campaign we'd been tracking for years. Upon closer inspection, we found that the information provided was accurate, but the vulnerabilities, injected code, and malicious domains all dated back to 2019 and 2020.

Nevertheless, the post offered interesting details about how campaign operators searched for vulnerable websites and injected malware. We soon obtained samples of the Linux binaries written in Go language from VirusTotal, where other security researchers had been creating collections.

Most of the samples were compiled with debug information and even a simple "strings" command provided quite insightful information: names of functions, string constants, paths of files included in the project. These files consist mostly of source code for various Go libraries, providing additional functionality such as conversion functions and support for Internet protocols. However, the main malware code was located in the file C:/Users/host/Desktop/balada/client/main.go. The file path balada/client implied that the developer could refer to this software as Balada Client (we know that the malware sends data to C2 server so there could be a Balada Server part too).

Whether our assumptions were correct or not, we adopted this name internally and think that

it provides some convenience when talking about a really long lasting malware campaign.

In many languages, Balada means "Ballad". To avoid ambiguity, we added the word Injector to reflect the nature of the malware campaign that injects malicious code into WordPress sites, hence Balada Injector.

So, some interesting background about a long-lived multi-year — as in six years and counting — highly aggressive and effective focused campaign against WordPress sites. It's easy to become inured to big numbers, but one million individually infected WordPress sites is a lot of sites. So I wanted to cover this now, since I suspect this won't be the last we're hearing about this quite determined Balada Injector WordPress malware.

Mozilla's Site Breach Monitor

Mozilla has updated their Firefox Monitor data breach monitoring and alerting service giving it a dedicated website: https://monitor.firefox.com/breaches

If you go to https://monitor.firefox.com/breaches, you're greeted with the caption: *"We monitor all known data breaches to find out if your personal information was compromised. Here's a complete list of all of the breaches that have been reported since 2007."*

And as you might expect from a web page which boasts a complete listing of what amounts to every site breach, ever... you'll be scrolling for a while. Thankfully, the page is sorted from yesterday to ancient. And for those who don't know, Mozilla's Firefox Monitor site performs the same sort of checking that Troy Hunt's *"Have I been Pwned"* site offers, where registered eMail addresses are cross-referenced against the database of all previous datasets obtained from website breaches. Troy's facility offers a feature that I appreciate as the owner of GRC.COM. Once I authenticate my ownership and control over the GRC.COM domain, "Have I Been Pwned" will perform a wildcard search for any and all eMail addresses within grc.com.

But where breach notification is concerned, there's nothing wrong with having more than one such solution, and the new dedicated Firefox Monitor breach listing page is somewhat breathtaking to behold.

For example, there were two site breaches on March 31st — of "Sundry Files" and, ironically, a site named "Leaked Reality" — both from which eMail addresses, IP addresses, Passwords and Usernames were stolen. Before that, was a breach on March 24th of "TheGradCafe" which lost Email addresses, Genders, Geographic locations, IP addresses, Names, Passwords, Phone numbers, Physical addresses and Usernames. Before that, on March 11th, the site "Shopper+" lost its visitors' Dates of birth, Email addresses, Genders, Names, Phone numbers, Physical addresses and Spoken languages. And on the same day, HDB Financial Services was doubtless embarrassed to have lost its clients' Dates of birth, Email addresses, Genders, Geographic locations, Loan information, Names and Phone numbers.

So, it really is quite eye opening to scroll back through this listing to get a sense for just how continuous and frequent these breaches are. We don't talk about them here every week because

it would be information overload and because no one specific site breach would be useful to most, if any, of our listeners. But I strongly recommend that everyone who's listening to this podcast take a minute or two to check-out Mozilla's page. And it certainly makes sense to get your eMail addresses registered there so that you'll be informed if or when your name pops-up in a breach. Up to five different eMail addresses may be registered per Mozilla account. That seems like a no-brainer to me. Once again: https://monitor.firefox.com/breaches

Another ChatGPT investigation:
Joining Italy's weird ban on ChatGPT we have Canada's privacy watchdog launching an official investigation into OpenAI's ChatGPT service. Canadian officials say they launched the investigation after a complaint alleging that the service was non-consensually collecting personal data. I suppose this is mostly a case of a bright light suddenly being shown on something that had been going on, unnoticed and unremarked for quite a while. And it also appears that the industry's AI ChatBots are going to need to start paying more attention to whom they're chatting with.

Some Android Goodies:
In honor of today's co-host, Jason, who is one of the hosts of TWiT's All About Android podcast, we have three quick bits of Android news:

Samsung handsets reaching EoL:
Samsung's line of 2019 smartphones has formally reached end-of-life and will therefore no longer be receiving security updates. March 2023 was the last security patch level for devices which include the Galaxy S10, Galaxy S10+, and Galaxy S10e. This doesn't mean the end of the world. But it's just a note that if any of our listeners may be using a four year old phone who are also concerned about remaining current and hooked up to the life support I.V. line of constant security patches, occasionally some which are critical, it might be time for a hand-mem-down of that device to someone who's less worried.

Less access for loan apps:
Google has moved to restrict the amount of personal data which loan apps may gather from Android users. Although this new policy took effect on April Fools day (April 1st), it's no joke. According to the new rules, loan apps can no longer access photos and user contacts. Google's policy change follows reports that some loan app makers engaged in predatory behavior, such as harassing borrowers and threatening to expose their private communications and photos unless they paid their loans or agreed to higher interest rates. Yikes!!!

The right to be forgotten:
Google also announced that in the future, all Android apps which allow users to create accounts of any kind will, by early 2024, also need to allow users to delete their accounts and any associated data. App makers must honor requests to delete accounts either directly through the app or through a web dashboard. The web dashboard requirement allows the data of an already removed app to be deleted without that app first needing to be reinstalled. The new requirement will enter into effect in early 2024.

SpinRite

Matthew Hile, the guy who's tweet I shared last week reminding me that Microsoft's Exchange Server plans would have the beneficial effect of forcing older Exchange Servers to upgrade, also shared a recent experience with SpinRite. His tweet last week noted that he was a listener from the start and had also been a development tester of SpinRite. So, Matthew wrote:

> *After hearing on SN from the user that ran level 3 on an SSD, I noticed the same slow response at the start of my 500GB Samsung SSD 860 EVO. Matthew then showed the five point benchmark performed by ReadSpeed:*
>
> *Before - 457.2 511.8 520.9 543.3 543.3*

He then ran a level 3 pass over his very nice 500GB Samsung 860 EVO drive. After doing that he re-ran the ReadSpeed benchmark and reported its findings:

> *After - 542.7 542.4 542.3 542.5 542.4*
>
> *Even better, it seems to have stopped my frequent blue screens (which still occurred after a level 2 pass). After weeks of troubleshooting, I was at the point of seriously considering blowing windows away and starting with a new copy. So glad to avoid that draconian effort.*

Now, blowing Windows away and reinstalling it, would have solved the problem. But a 60 minute level 3 pass with the alpha release of SpinRite certainly saved him a ton of time because it was the rewriting of the drive's data that was needed.

What's happening inside our SSDs is closely related to the RowHammer style DRAM problems we keep being dragged back to, which are exhibited by today's ultra high density DRAM. In both cases of DRAM and SSD it's necessary to appreciate that the density of the storage cells and the size of the feature details of these technologies are absolutely absolutely as small and tiny as they can possibly be. It's a competitive world. So if it were possible for those feature details to be any smaller, and thus for the devices to have a higher bit density while the devices still function — they would be. So just as with DRAM, where the engineers were pressured to push it perhaps a bit too far, SSD technology has a widely known problem known as "Read Disturb." If you Google "Read Disturb" you'll find out all about it. I did Google "Read Disturb" just now, and the top result was a description from the ACM — the Association for Computer Machinery. The description reads:

> *Read disturb is a circuit-level noise in solid-state drives (SSDs), which may corrupt existing data in SSD blocks and then cause high read error rate and longer read latency.*

"high read error rate and longer read latency" — which is exactly what we're seeing. Error correction is not perfect. There's a limit to how much error can be corrected and there's also a statistical probability that a particular set of bit errors won't be detected — which explains the Blue Screens that Matthew was occasionally seeing.

The reason the front of Matthew's SSD was so much slower, was that that's where the operating system files are stored. So they are being read over and over and over and much less often, if ever, rewritten. Over time, the electrostatic charges stored in the SSD's bit cells drift away from their proper values due to the **Read Disturbance** caused by all of the adjacent read activity. It got to the point where Matthew's SSD was always needing to work much harder to read some of its data whose bits had drifted further from their proper values. They always needed to be corrected through multiple rereads, varying thresholds, and more extensive error correction. GRC's ReadSpeed benchmark, which samples the read performance of five regions with highly repeatable accuracy, saw this.

And the cure for this is simple: Simply read and rewrite the troubled data to reset the drifting bit values to their proper states. But the problem with that, as we know, is that writing fatigues SSDs. It's not something you want to be doing all the time. And you certainly don't want to do it if you don't need to.

Which brings us to the reason I've been so excited about what we discovered earlier in this work on SpinRite, which is that given the proper technology, this can be detected and fixed in a highly targeted fashion. Unfortunately, the current SpinRite doesn't have the architecture to do this. DOS and real mode doesn’t provide the sensitivity for the fine-grained read performance measurement we need. SpinRite will need to use the power of protected mode to detect the precise instant when writes are occurring in memory. This is where I'm heading with SpinRite 7.

But today's SpinRite 6.1, which is at least able to run at the maximum speed that mass storage media will go, can perform a full drive rewrite. That’s one of the several things that SpinRite’s 3rd level does. When benchmarking, and a drive's flaky behavior begins to show that it might be needed, doing that — and maybe on only the front of the drive — is still a lot better than waiting for the drive's data to degrade to the point of system failure.

In the future, SpinRite will be able to locate the exact trouble area and perform a selective rewrite of only those spots that need it. SpinRite 6.1 first, then it’s on to SpinRite 7.

A Dangerous Interpretation

Anyone who's been following this podcast for a year or two will have encountered our frequent observation about the inherent security dangers created by interpreters. It's a recurring theme here because the act of interpretation means following instructions to perform some sequence of actions. What we typically see when an interpreter's security vulnerabilities are analyzed is that the interpreter's designer inevitably assumed that valid instructions would be received.

A research paper was recently submitted for presentation during the upcoming 32nd USENIX Security Symposium. That paper described the amazing work done by a pair of researchers at the University of Texas at Austin with the help of another at Oberlin College. The interpreter in question, which these three set their sights on, is one that every one of us has multiple instances of surrounding us throughout our day and in our daily lives. And that's **video and thumbnail creation**, specifically the ubiquitous H.264 video codec.

Although the title of their paper is intended to capture attention, the content of their paper shows that the title is not overblown. The paper is titled: *"The Most Dangerous Codec in the World: Finding and Exploiting Vulnerabilities in H.264 Decoders."*

And one thing I want to point out at the outset, is that almost without exception, most of the research papers we discuss here talk about hidden and unsuspected problems that were found and then fixed. Not so this time. The problems this tool is designed to unearth are generally well known, but are too widespread and ubiquitous to have all been found. The authors offer several convincing proofs-of-concept case studies, but I fear that since the tool is now open sourced on Github for anyone to use, it won't just be the good guys who are motivated to leverage its power for finding exploitable flaws across the industry's video decoding interpreters.

Their paper leads with this Abstract:

> *Modern video encoding standards such as H.264 are a marvel of hidden complexity. But with hidden complexity comes hidden security risk. Decoding video in practice means interacting with dedicated hardware accelerators and the proprietary, privileged software components used to drive them. The video decoder ecosystem is obscure, opaque, diverse, highly privileged, largely untested, and highly exposed—a dangerous combination.*
>
> *We introduce and evaluate H26FORGE, a domain-specific infrastructure for analyzing, generating, and manipulating syntactically correct but semantically spec-non-compliant video files. Using H26FORGE, we uncover insecurity in depth across the video decoder ecosystem, including kernel memory corruption bugs in iOS, memory corruption bugs in Firefox and VLC for Windows, and video accelerator and application processor kernel memory bugs in multiple Android devices.*

Okay. So when they're talking about a *"domain-specific infrastructure for analyzing, generating, and manipulating syntactically correct but semantically spec-non-compliant video files"*, they're saying that because the bugs that might be—and turn to to be—resident within H.264 decoders

might be buried deep in the multi-layer decoding process, the much more common and much easier practice of "fuzzing" won't work here.

As we know, "fuzzing" is the common practice of throwing mostly random crap at something, a codec or an API or whatever, to see whether anything that might be sent can result in a crash. And then once the nature of that crash is understood, the next question is whether it might be exploitable to obtain a more useful outcome than a simple crash. The trouble is, H.264 decoding is so complex that random crap won't make it through the front door.
That's what they meant when they said that their H26Forge would generate *"syntactically correct but semantically spec-non-compliant video files"*. In other words, it looks like and is an entirely valid video file. It follows all of the rules and can be processed properly. But it's also a Trojan horse. It appears completely correct and valid so that it can get into the inner sanctum where the real vulnerabilities may lie.

Here's a bit more from their paper to describe and setup the situation and the environment:

Modern video encoding standards are a marvel of hidden complexity. As SwiftOnSecurity noted, the video-driven applications we take for granted would not have been possible without advances in video compression technology, notwithstanding increases in computational power, storage capacity, and network bandwidth.1 But with hidden complexity comes hidden security risk.

The H.264 specification is 800 pages long—despite specifying ***only*** *how to decode video,* ***not*** *how to encode it. Because decoding is complex and costly, it is usually delegated to hardware video accelerators, either on the GPU or in a dedicated block on a system-on-chip (SoC). Decoding video in practice means interacting with these privileged hardware components and the privileged software components used to drive them, usually a system media server and a kernel driver. Compared to other types of media that can be processed by self-contained, sandboxed software libraries, the attack surface for video processing is larger, more privileged, and, as we explain below, more heterogeneous.*

On the basis of a guideline they call "The Rule Of 2," the Chrome developers try to avoid writing code that does no more than 2 of the following: parses untrusted input, is written in a memory-unsafe language, and runs at high privilege. The video processing stack in Chrome violates the Rule of 2, and so do the corresponding stacks in other major browsers and in messaging apps—because the platform code for driving the video decoding hardware, on which they all depend, itself violates the Rule of 2.

Because different hardware video accelerators require different drivers, the ecosystem of privileged video processing software is highly fragmented; our analysis of Linux device trees revealed two dozen accelerator vendors. There is no one dominant open source software library for security researchers to audit. [As, for example, there was for OpenSSL.]

And the features that make modern video formats so effective also make it hard to obtain high code coverage testing of video decoding stacks by means of generic tools. Consider H.264, the most popular video format today. H.264 compresses videos by finding similarities within and

across frames; the similarities and differences are sent as entropy-encoded syntax elements. These syntax elements are encoded in a context-sensitive way: a change in the value of one syntax element completely changes the decoder's interpretation of the rest of the bitstream.

Think about that. They wrote *"a change in the value of one syntax element completely changes the decoder's interpretation of the rest of the bitstream."* This means that in working to trigger a flaw, that flaw might only present itself if the decoder is in a particular state which is determined by everything that has come before. It's clear why a sophisticated pseudo video file generator had to be built in order to find the bugs which only manifest when all of the planets are in proper alignment. Elsewhere in their paper they put their H26Forge into context by explaining how it compares with other tools that researchers have attempted to use in the past:

H26FORGE maintains the recovered H.264 syntax elements in memory and allows for the programmatic adjustment of syntax elements, while correctly entropy-encoding the adjusted values. No prior tool is suited to this task. Most software that read H.264 videos (e.g., OpenH264 and FFmpeg) focuses on producing an image as quickly as possible, so it discards recovered syntax elements once an image is generated. Tools used to debug video files (e.g., Elecard's StreamEye) do not allow the programmatic editing of syntax elements; they focus on providing feedback to tune a video encoder.

H26FORGE can be used as a standalone tool that generates random videos for input to a video decoder; it can be programmed to produce proof-of-concept videos that trigger a specific decoder bug identified by a security researcher; and it can be driven interactively by a researcher when exploring "what-if?" scenarios for a partly understood vulnerability.

At one point they begin to explain in some detail about H.264. They write:

The H.264 video codec was standardized in 2003 by the International Telecommunication Union (ITU) and the Motion Picture Experts Group (MPEG). Because of this joint effort, this codec has two names: H.264 provided by the ITU, and AVC provided by MPEG. We default to H.264 when possible.

The specification describes how to decode a video, leaving encoding strategies up to software and hardware developers. Video ***encoding*** *is the search problem of finding similarities within and between video frames, and turning these similarities into entropy-encoded instructions. The H.264 spec describes how to recover the instructions and reproduce a picture.*

Okay. They then get way into the weeds with the details of H.264 encoding. Alex Lindsay would probably love it. At least there would be lots of terms that he would have encountered through the years. But getting into that here doesn't serve our purpose of wanting to understand what they found and how they found it. Suffice to say that they explain about the YUV colorspace, 16x16 macroblocks, slices, prediction, deblocking, residues, profiles, levels, syntax elements, entropy encoding, encoded value organization, and they finish by talking about additional H.264 features and extensions.

It's quite clear that in order to write something like H26Forge, which is, by the way, posted on Github for anyone to experiment with, these guys had to really and truly deeply understand an insanely complex data encoding system that's described by an 800-page spec.

Now let's get to the crux of the matter, which is the decoding pipeline. If I had dragged everyone through the detailed description of H.264 components, what I'm going to share next would actually make some sense. But it's not going to, because we **don't** need to really understand it. But I want to share their overview of the decoding pipeline because everyone will get a good sense for just how insanely complex the world's H.264 decoders are. Here's just one paragraph that will give everyone a sense for the decoding process:

> *First, the decoder is set up by passing in an SPS and a PPS with frame and compression related properties. Then the decoder receives the first slice and parses the slice header syntax elements. The decoder then begins a macroblock-level reconstruction of the image. It then entropy decodes the syntax elements and passes them to either a residue reconstruction path or through a frame prediction path with previously decoded frames. Then the predicted frames are combined with the residue, passed through a deblocking engine, and finally stored in the DPB, where the frames can be accessed and presented.*

Sure. Piece of cake. Then it does most of that again for the next frame. By the way that "DPB" stands for the "Decoded Picture Buffer" which serves as both the output of the decoder and as an image reference for subsequently decoded frames.

So by now everyone should have a good sense for what's required to really and truly get to the bottom of vulnerabilities which may exist in any H.264 family codec. And I have to say that I wasn't super happy to read that they had open sourced all of this work and dropped it onto Github. On the one hand this will make it available to other researchers and also to vendors of these technologies. And that's all for the good. But that also means that bad guys also now have it, and they might well be more motivated to take advantage of it than anyone else.

So exactly how vulnerable are we? They explain:

> *A wide range of software systems handle untrusted video files, providing a broad attack surface for codec bugs. An important observation is that hardware-assisted video decoding bypasses the careful sandboxing that is otherwise in place to limit the effects of media decoding bugs.*
>
> *Popular messenger apps will accept video attachments in messages and provide a thumbnail preview notification. In the default configuration of many messengers, the video is processed to produce the thumbnail without user interaction, creating a zero-click attack surface.*
>
> *There are many examples of video issues on mobile devices. Android has had historical issues in its Stagefright library [remember all of that?] for processing MP4 files. Video thumbnailing and decoding constitutes an exploitable attack surface in Apple's iMessage application despite the BlastDoor sandbox. Third-party messengers can also be affected. In September, WhatsApp disclosed a critical bug in its parsing of videos on Android and iOS.*

Web browsers have long allowed pages to incorporate video to play through the video HTML tag, leading to multiple vulnerabilities in video decoding. For example, both Chrome and Firefox were affected by a 2015 bug in VP9 parsing. Later, we describe a new vulnerability we found in Firefox's handling of H.264 files.

Despite this track record, more video processing attack surface is being exposed to the Web platform. Media Source Extensions (MSE) and Encrypted Media Extensions (EME) have been deployed in major browsers; the WebCodecs extension, currently only deployed in Chrome, will allow websites direct access to the hardware decoders, completely skipping over container format checks.

Modern browsers carefully sandbox most kinds of media processing libraries, but they call out to system facilities for video decoding. Hardware acceleration is more energy efficient; it allows playback of content that requires a hardware root of trust; and it allows browsers to benefit from the patent licensing fees paid by the hardware suppliers.

Video transcoding pipelines, such as at YouTube, and Facebook, handle user-generated content, which may contain videos that are not spec-compliant. This could lead to denial-of-service, information leakage from the execution environment or other processed videos, or even code execution on their platforms.

What about hardware video decoding? That's a big issue, too...

Video decoding in modern systems is accelerated with custom hardware. The media IP (IP as in Intellectual property) included in SoCs or GPUs is usually licensed from a third party. In one notable example, iPhone SoCs through the A11 include Imagination Technologies' D5500 media IP, as do the SoCs in several Android phones we study, with very different kernel drivers layered on top.

IP vendors build drivers for their hardware video decoders, which are then called by the OS through their own abstraction layer. The drivers will prepare the hardware to receive the encoded buffers often through shared memory.

While Stagefright is Android's Media engine, Android uses OpenMAX (OMX) to communicate with hardware drivers. OMX abstracts the hardware layer from Stagefright, allowing for easier integration of custom hardware video decoders.

Other operating systems similarly have their own abstraction layer. The Linux community has support for video decoders through the Video for Linux API version 2. Similar to OMX, it abstracts the driver so user space programs do not have to worry about the underlying hardware. Windows relies on DirectX Video Acceleration 2.0 and Apple uses VideoToolbox. Intel also has its own Linux abstraction layer called the Video Acceleration API and, similarly Nvidia has the Video Decode and Presentation API for UNIX.

We list 25 companies we found that have unique video decode IPs. Some of these may license

from other companies, or may produce their own video codec IP. The companies include providers for Single-Board Computers (SBCs), set-top boxes, tablets, phones, and video conferencing systems. Some video decode IP companies describe providing drivers and models for incorporating the IP into SoCs.

We highlight all of these companies to showcase the heterogeneity of available hardware video decoders, and thus the potential for vulnerabilities to exist within or across products.

And finally, we get to the threat model:

In this paper, we assume an adversary who (1) produces one or more malicious video files; and (2) causes one or more targets to decode the videos. Delivering videos to the user and having them be decoded—with or without user interaction—is easy to accomplish in many cases. This is the minimal set of capabilities an adversary needs to exploit a vulnerability in decoding software or hardware. For information disclosure attacks the adversary (3) must be able to read frames of decoded video. For malicious videos delivered via the web, for example, this can be accomplished via JavaScript.

So, H26FORGE was written in around 30 thousand lines of Rust code, and has a Python scripting backend for writing video modification scripts. It has three main parts: input handling, syntax manipulation, and output handling. The input handling contains the H.264 entropy decoding. Syntax manipulation has functions for modifying recovered syntax elements or generating test videos. Output handling has the H.264 entropy encoding, which outputs videos in "Annex B" format, but can also output a WebCodecs friendly AVCC file, muxed MP4 file, or JSON dump of the decoded syntax elements.

The best way to describe what they found would be to say that everywhere they looked they found problems — many of them were readily exploitable. For example:

We found two bugs in the AppleD5500.kext. The first bug enables a partly-controlled heap memory overwrite. The second bug causes an infinite loop and leads to a kernel panic. These bugs have been confirmed, patched, and assigned CVEs by Apple. We verified that they can be triggered by a web page visited in Safari.

Through reverse engineering of the H.265 decoder in the AppleD5500.kext for iOS 15.5, we discovered what appeared to be a missing bounds check potentially leading to a heap overflow in the H.265 decode object. To verify this, we modified H26FORGE with enough H.265 tooling to produce a proof-of-concept video that causes a controlled kernel heap overflow. Unlike the previously described bugs, we were able to trigger this bug only when playing a video, not through preview thumbnail generation. Apple assigned this bug CVE-2022-42850 and patched it in iOS and iPadOS version 16.2. By overwriting a pointer with the address of a fake object that itself points to a fake vtable, we can arrange to have any address of our choosing called in place of the legitimate destructor. We did not attempt to develop an end-to-end exploit chain; however, Apple's assessment was that this bug, like Bug 1, may allow an app "to execute arbitrary code with kernel privileges."

We tested generated videos on Firefox 100, and discovered an out-of-bounds read that causes a crash of the Firefox GPU utility process and a user-visible information leak. The issue arises from conflicting frame sizes provided in the MP4 container as well as multiple SPSes across video playback. Note that both the crash and information leak are caused by a single video. To exploit this vulnerability an attacker has to get the victim to visit a website on a vulnerable Firefox browser. We reported this finding to Mozilla, and it has been assigned CVE-2022-3266 and patched in version 105.

On VLC for Windows version 3.0.17, we discovered a use-after-free vulnerability in FFmpeg's libavcodec that arises when interacting with Microsoft Direct3D11 Video APIs. We found this by testing generated videos in VLC. The bug is triggered when an SPS change in the middle of the video forces a hardware re-initialization in libavcodec. If exploited, an attacker could gain arbitrary code execution with VLC privileges. We reported this issue to VLC and FFmpeg, and they have fixed it in VLC version 3.0.18 and an FFmpeg commit.

We tested the videos produced by H26FORGE on a variety of Android devices with varying hardware video decoders. In doing so, we found issues that span different hardware manufacturers, and more serious vulnerabilities in hardware decoders and their associated kernel drivers. To target a breadth of video decode IP, we went with older, cheaper SoCs, but note that some of our findings impact newer MediaTek devices as well, and the videos produced by H26FORGE can be used to test new and future devices.

In reporting the problems they discovered, most mainstream publishers like Apple, Google, Samsung and so on were quite responsive. But in some cases when they needed to get in touch with a vendor who only OEMs chips to other major customers, they received no answer to repeated attempts to make those companies aware of the vulnerabilities in their decoders.

They summarized their work by writing:

We have described H26FORGE, domain-specific infrastructure for analyzing, generating, and manipulating syntactically correct but semantically spec-non-compliant video files. Using H26FORGE, we have discovered (and responsibly disclosed) multiple memory corruption vulnerabilities in video decoding stacks from multiple vendors.

We draw two conclusions from our experience with H26FORGE.

First, domain-specific tools are useful and necessary for improving video decoder security. Generic fuzzing tools have been used with great success to improve the quality of other kinds of media-parsing libraries, but that success has evidently not translated to video decoding.

The bugs we found and described have been present in iOS for a long time. We have tested that our proof-of-concept videos induce kernel panics on devices running iOS 13.3 (released December 2019) and iOS 15.6 (released July 2022). Binary analysis suggests that the first bug we identified was present in the kernel as far back as iOS 10, the first release whose kernel binary was distributed unencrypted.

We make H26FORGE available at https://github.com/h26forge/h26forge under an open source license. We hope that it will facilitate followup work, both by academic researchers and by the vendors themselves, to improve the software quality of video decoders.

Second, the video decoder ecosystem is more insecure than previously realized. Platform vendors should urgently consider designs that deprivilege software and hardware components that process untrusted video input.

Browser vendors have worked to sandbox media decoding libraries as have messaging app vendors, with the iMessage BlastDoor process being a notable example. Mobile OS vendors have also worked to sandbox system media servers. These efforts are undermined by parsing video formats in kernel drivers.

Our reading of reverse-engineered kernel drivers suggests that current hardware relies on software to parse parameter sets and populate a context structure used by the hardware in macroblock decoding. It is not clear that it is safe to invoke hardware decoding with a maliciously constructed context structure, which suggests that whatever software component is charged with parsing parameter sets and populating the hardware context will need to be trusted, whether it is in the kernel or not. It may be worthwhile to rewrite this software component in a memory-safe language, or to apply formal verification techniques to it.

An orthogonal direction for progress, albeit one that will require the support of media IP vendors, would redesign the software–hardware interface to simplify it. The Linux push for stateless hardware video decoders is a step in this direction. Similarly, encoders that produce outputs that are software-decoder friendly, such as some AV1 encoders, help reduce the expected complexity of video decoders.

So the industry has just received the gift of an extremely impressive, powerful and flexible new tool for generating correctly formatted yet subtly defective test videos for the purpose of finding and perfecting exploitable flaws in pretty much all current H.264 video decoders. I remain more than a little bit worried that bad guys will jump on this and use it to locate powerful new vulnerabilities that can be turned into zero-click exploits which they will obviously not report to the device's manufacturer.

The problem, as always, is one of motivation and economics. Not much imagination is required to picture the NSO Group pouncing on this to enhance their next-generation of Pegasys smartphone penetration spyware. And we learned last week that the NSO Group has a couple dozen also-ran wannabe competitors. Everyone is going to be on this. And the problem is that they're highly motivated to use it since there's a pot of gold at the end of the development of a new successful exploit.

So I wonder who's going to be that motivated on the good guy's side? Perhaps, since the discovery of one of these would be of tremendous value, a substantial bug bounty would be awarded for a power 0-click remote device takeover exploit.

At the end of today's show notes I have the link to the Github page: https://github.com/h26forge/h26forge

And to their entire 18-page research paper: https://wrv.github.io/h26forge.pdf