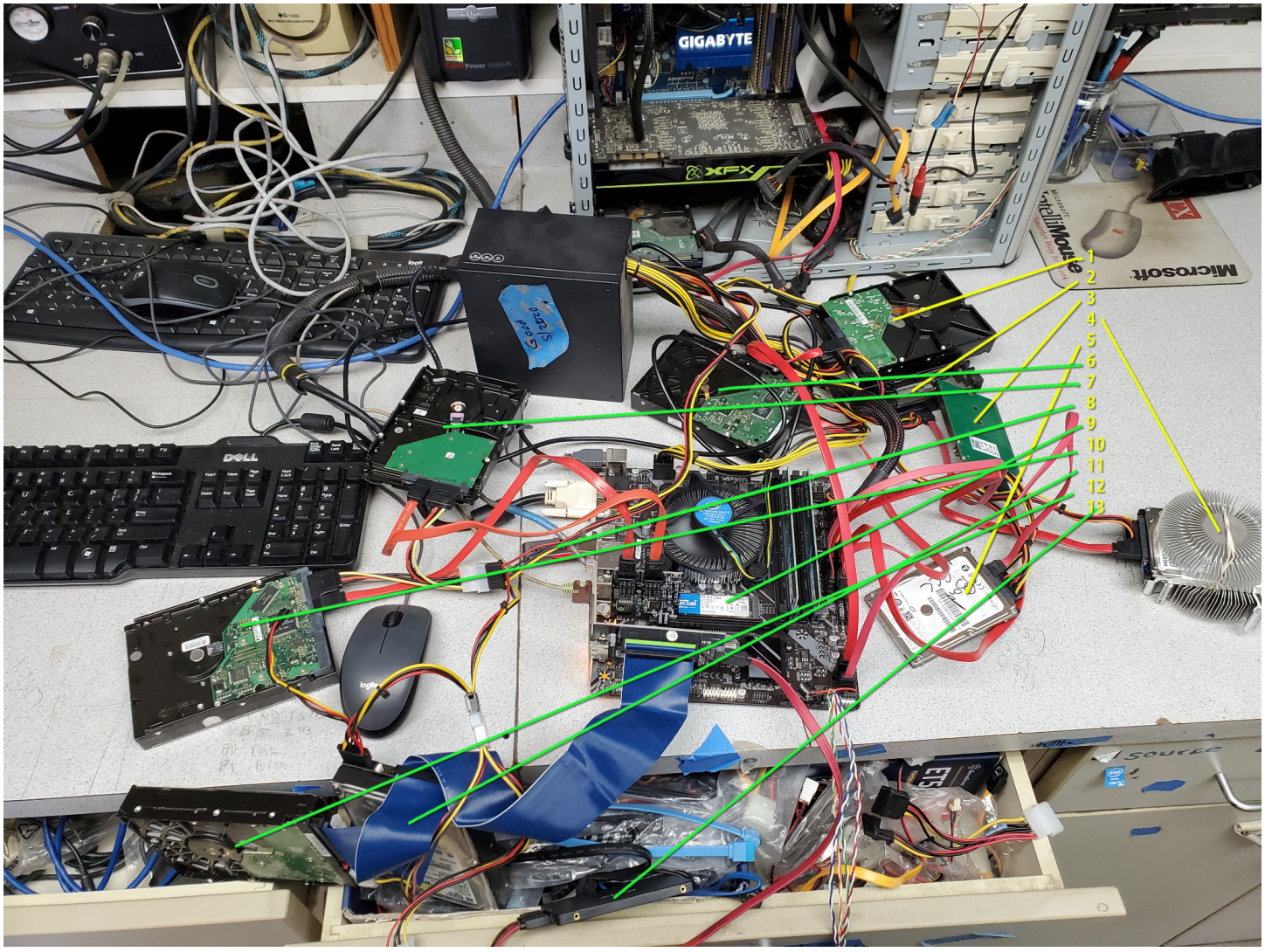
Security Now! #848 - 12-07-21

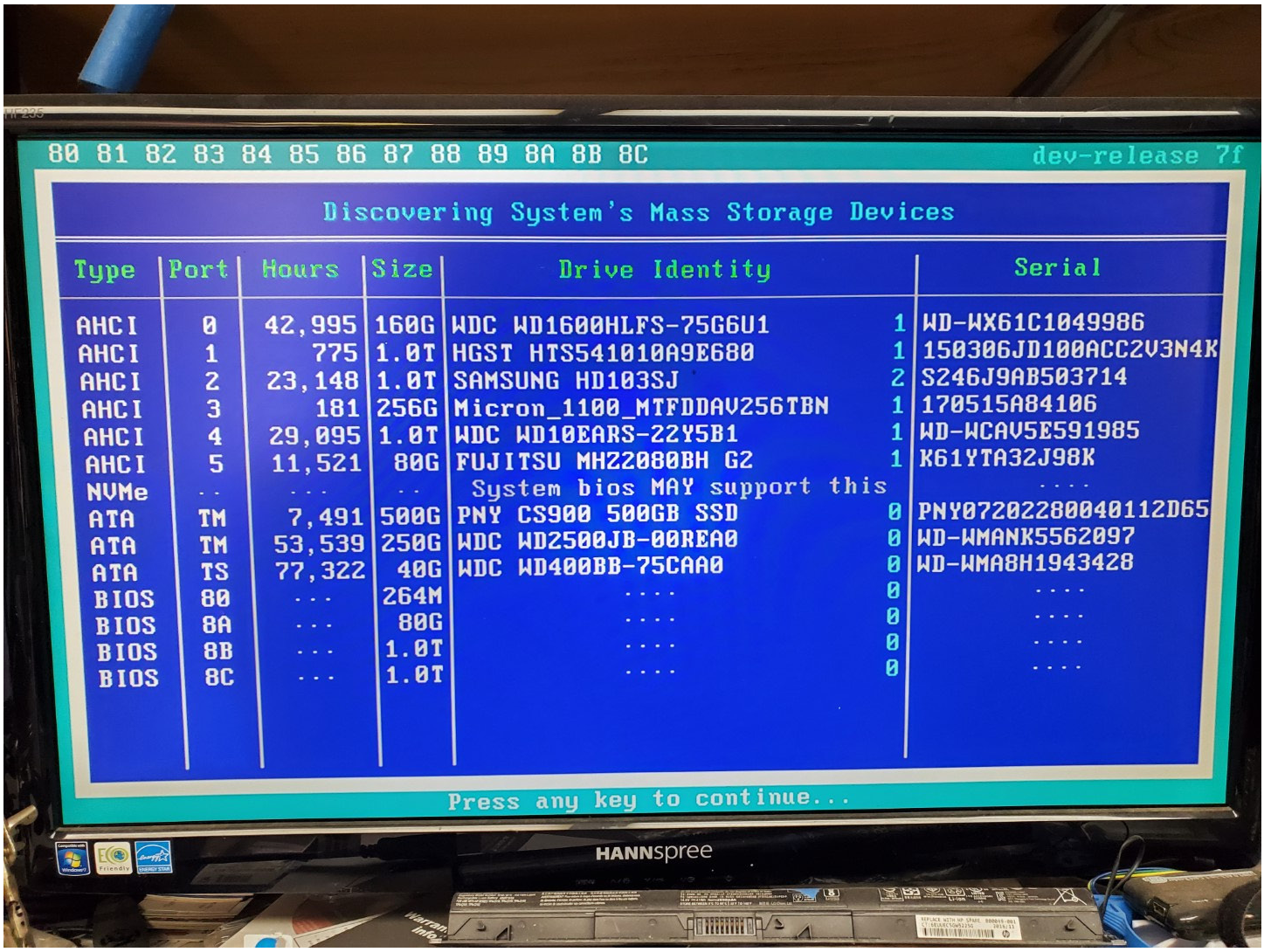
XSinator

This week on Security Now!

This week Tavis Ormandy finds a bug in Mozilla's NSS signature verification, we look at the horrifying lack of security in smartwatches for children (smartwatches for children?!?), and at the next 6 VPN services to be banned in Russia. Microsoft softens the glue between Windows 11 and Edge, bad guys find a new way of slipping malware into our machines, a Botnet uses the Bitcoin blockchain for backup communications, and HP has 150 printer models in dire need of firmware updates. We touch on Sci-Fi and SpinRite, then we look at new research into an entirely new class of cross-site privacy breaches affecting every web browser — including a test every user can run for themselves on their various browsers.

A SpinRite development tester runs it on 13 drives at once...





Security News

Tavis finds a bad bug in NSS

Google's Tavis Ormandy discovered and quietly reported an important bug in Mozilla's widely used open source NSS security suite. Last Wednesday, with the bug quickly patched, Tavis Tweeted:

> *This is a major memory corruption flaw in NSS, almost any use of NSS is affected. The Mozilla advisory is here https://t.co/AL8suyLQFF https://t.co/uTQ2gqRZ5t*
>
> *— Tavis Ormandy (@taviso) December 1, 2021*

NSS is used by Mozilla, Red Hat, SUSE, and many others in a wide variety of products, including:

- Firefox, Thunderbird, SeaMonkey, and Firefox OS.
- Open-source client applications such as Evolution, Pidgin, Apache OpenOffice, and LibreOffice.
- Server products from Red Hat: Red Hat Directory Server, Red Hat Certificate System, and the mod_nss SSL module for the Apache webserver.

- Server products from Oracle (formerly Sun Java Enterprise System), including Oracle Communications Messaging Server and Oracle Directory Server Enterprise Edition.
- SUSE Linux Enterprise Server supports NSS and the mod_nss SSL module for the Apache webserver.

Elsewhere, Tavis commented: *"If you are a vendor that distributes NSS in your products, you will most likely need to update or backport the patch. Mozilla plans to produce a thorough list of affected APIs - but the summary is that any standard use of NSS is affected. The bug is simple to reproduce and affects multiple algorithms."* In other words, anyone using this should focus less on which specific APIs are vulnerable and instead just update their use of the security suite. NSS versions before 3.73 or 3.68.1 ESR are vulnerable.

All versions of NSS released since October 2012 vulnerable. Tavis said: *"We believe all versions of NSS since 3.14, which was released October 2012, are vulnerable."* However, according to Mozilla, this vulnerability doesn't impact Firefox. But all PDF viewers and email clients which use NSS for signature verification are believed to be impacted. Mozilla said: *"Applications using NSS for handling signatures encoded within CMS, S/MIME, PKCS #7, or PKCS #12 are likely to be impacted. Applications using NSS for certificate validation or other TLS, X.509, OCSP or CRL functionality may be impacted, depending on how they configure NSS."*

And just for the record, exploitation of this during signature verification can lead to a heap-based buffer overflow when handling DER-encoded DSA or RSA-PSS signatures in email clients and PDF viewers. The impact of successful heap overflow exploitation can range from program crashes and arbitrary code execution to bypassing security software if code execution is achieved.

So, if anyone is using NSS for some project, be sure to get an updated build.

Cheap Smartwatches for kids and babies? (What could possibly go wrong?)
Dr. Web A/V is a Russia-based security firm which recently examined four inexpensive smartwatches designed and marketed for children.

Before I go any further, allow me to suggest to anyone listening to this podcast that they simply resist **any** temptation whatsoever they might have to purchase a $50 smartwatch for anyone they know. Dr. Web's report is very detailed, so I'm not going to bother to describe what they found in each of the four devices. But discussing a representative example will be informative and should be sufficient to forestall any desire to put one of these little spyware beasties onto the wrist of anyone you know.

Dr. Web introduces the subject with some background which I've shortened a bit:

> *Parents always strive to take care of their children. Technology innovations help them reach this goal through various wearables like smartwatches and GPS trackers. These devices are getting close to smartphones in functionality. For example, many of them can track the child's location and travel route. These devices can also make and answer phone and video calls, receive SMS and voicemail, take pictures, and listen to the surroundings. It is even possible to enable the remote control of the device. Doctor Web has analyzed the potential threats that such gadgets can pose to parents and their children.*

During their daily operation, these devices collect and transmit data to the manufacturer's servers and make it available to parents through their personal accounts. The information obtained with smartwatches is very sensitive. If malicious actors get their hands over such information, it can put children in great danger.

To understand the vulnerability and dangers of children's smartwatches, Doctor Web's specialists analyzed several popular models chosen based upon public popularity ratings and purposefully selected models from different price ranges. We purchased all smartwatches anonymously from an online store.

We carried out both static and dynamic analyses during the inspection: we searched for potential implants in the software and possible undocumented features, as well as checked network activity, transmitted data, and how it was secured.

One of the four devices they examined was the "Elari Kidphone 4G Smartwatch" ...

Several versions of the Elari Kidphone 4G watches exist. They are based on different hardware platforms but they all run an Android OS and their firmware differs slightly.

The primary threat of this model comes from its installed software. Its firmware has a built-in app for over-the-air updating, and this app has Trojan functionality.

Firstly, the application sends the child's geolocation data to its own remote server. Secondly, we found malicious code inside it, which is detected by Dr.Web as Android.DownLoader.3894. Every time the watch turns on, or network connection changes, this code launches two malicious modules, Android.DownLoader.812.origin and Android.DownLoader.1049.origin.

They connect to the C&C server to transmit various information and to receive commands. By default, the modules connect to the server every 8 hours. Because of this, a long delay exists between the first connection and the first time the watch turned on, thus reducing the chance of discovery.

The Android.DownLoader.812.origin module sends the user's mobile phone number and SIM card information, geolocation data, and information about the device itself to the C&C server. In response, it can receive commands to change the frequency of requests to the server, update the module, download, install, run and uninstall apps, and load web pages.

The Android.DownLoader.1049.origin module sends SIM card and mobile phone number information, geolocation data, a large number of data about the device and installed apps, as well as the info about the number of SMS, phone calls, and contacts in the phone book to the C&C server.

Thus, Android.DownLoader.3894 hidden in this watch can be used for cyber espionage, displaying ads, and installing unwanted or even malicious apps.

The problem, here, is that as a consumer product they're adorable and appealing little devices. Our listeners can't see what you and I are seeing here in the show notes, Leo. But these little KidPhones are adorable:

Look Billy! Just like mommy's and daddy's grown up watches!

Disarmingly, they look like toys, but they're definitely not. Anyone who purchases one of these little devils is not only exposing their children to remote anonymous attackers and trackers, but is also bringing an open mic into the lives of those childrens' parents. Who knows what might be overheard in what's presumed to be a private setting? And who would be to blame?

Now, I fully recognize that the **likelihood** of any specific child being targeted is diminishingly low. I don't mean to suggest otherwise. But knowing what we now know, who would feel comfortable strapping one of these loose cannons onto **their** child's wrist? And that's my point. The photo of these devices is utterly at odds with what's going on inside them. It's deeply unsettling, because they look so cute and harmless. Yet, inside that colorful soft plastic shell is a communicating computer running a Trojanized Android OS which autonomously connects to a remote command and control server. And Amazon sells them: https://www.amazon.com/dp/B07JZVCT15

The bullet points on Amazon's page say:

- **Security and Peace of Mind for Parents:** Monitor child's location, set safe zones and use remote audio monitoring through free parent controlled app. Get alert if the child uses the built in SOS button.

- **Easy Communication:** Stay in touch, have a 2-way voice call, use voice chat or send emojis. Limit contacts to those authorized through the app to avoid undesired communication.

- **User Friendly and Comfortable:** Kids phone watch features math game and interface with emoticons.

- **Other features:** Class time mode, alarm call, pedometer, up to 4 days standby time, ELARI SafeFamily free multilingual android ios compatible App, ability to add other ELARI smartwatch users to friends list.

The watches that the Dr. Web researchers examined had default passwords of 123456 and most of them don't bother to employ encryption in their communications. They use plaintext. What this brings to mind is the glaring gap in consumer protection that currently exists in our tech industry. There's a difference between purchasing a well-designed WiFi router that still might have an inadvertent security vulnerability and a colorful plastic child's watch that communicates without encryption with remotely located command and control servers, and is able to autonomously download any additional software at any time.

In the US, the engineers and scientists at Underwriter's Labs are able to test toasters and vacuum cleaners because they're able to carefully examine and stress test the important functional safety aspects of those consumer products. But today's tech gadgets are closed black boxes which are actively hostile to reverse engineering. I don't see any solution to this mess other than to take every possible precaution. We're able to place our IoT devices onto their own segmented network. If these wrist watches also have WiFi radios, placing them on the IoT WiFi would help. But they can still spy. If kids must receive a communicating wristwatch for Christmas the only thing the consumer can do is choose its manufacturer as wisely as possible. I'd stay far away from the "Elari" brand.

Additional VPN vendors just say no to Roskomnadzor!
Last Thursday, Russia's Internet policing agency "Roskomnadzor" announced the ban of an additional 6 VPN services, bringing the total number of banned VPN providers to 15. The most recent 6 bannings apply to Betternet, Lantern, X-VPN, Cloudflare WARP, Tachyon VPN and PrivateTunnel.

Roskomnadzor sent a request to inform the Center for Monitoring and Control of the Public Communications Network about the removal of those additional 6 services from the systems of all registered Russian companies and public organizations.

So now, the list of banned services is:

Hola! VPN, ExpressVPN, KeepSolid VPN Unlimited, Nord VPN, Speedify VPN, IPVanish VPN, VyprVPN, Opera VPN and ProtonVPN to which we now add, as I said: Betternet, Lantern, X-VPN, Cloudflare WARP, Tachyon VPN, and PrivateTunnel.

These services have been banned due to their principled refusal to abide by Roskomnadzor's demands to connect their systems to the FGIS database because doing do would defeat the

entire purpose of using a VPN connection which is to bypass access restrictions and obtain anonymity, which is a level of individual freedom which makes Russia's leaders uncomfortable.

Back in 2019, Russian authorities gave all VPN vendors operating in-country the ultimatum to comply with their rules, or else be banned. The only vendor who responded positively before the deadline was Moscow-based Kaspersky with their Secure Connection product.

The non-complying VPN vendors either established new servers just outside the Russian borders or attempted to employ traffic masking techniques which are known to work well against the "Great Firewall" of China. But the Russian authorities gradually caught up with those services which tried to bypass the new regulation and banned them in multiple action rounds. As Russian users uninstall banned products and turn to alternative options, the Russian authorities evaluate which ones emerge as the most popular and then add them to the growing block list.

Windows 11 loosens its grip on Edge

Last week's Windows Weekly was a bit depressing listening to Paul and MaryJo holding Microsoft to account for many of the choices they've made for the behavior of Windows 11. Among other things, Microsoft is clearly turning Edge from its original clean user-benefiting browser into another Microsoft profit center. And then adding insult to injury is having Windows 11 actively fighting its user's desire to switch away from Edge to any other web browser.

We've probably all seen Chrome promoting itself whenever it has the chance. It's annoying that there's no obvious way to tell Chrome that things are already just the way we want them, thank you very much. But now that Microsoft is pushing the adoption of Edge quite hard, we're seeing the same from them. BleepingComputer located the key in the Windows registry which contains the contents of the various pop-ups that Microsoft has prepared for Edge. There are currently five:

> *"Microsoft Edge runs on the same technology as Chrome, with the added trust of Microsoft. Browse securely now"*
>
> *"That browser is so 2008! Do you know what's new? Microsoft Edge. Come to the future"*
>
> *"Looking for speed and reliability? Microsoft Edge is the only browser optimized for Windows 10. Try a faster browser today"*
>
> *""I hate saving money," said no one ever. Microsoft Edge is the best browser for online shopping. Shop smarter now"*
>
> *"Microsoft Edge is fast, secure, and the browser recommended by Microsoft. Discover more benefits. Learn more about Microsoft Edge"*

In any event, I've already very clearly articulated my feelings about Windows 11 and I'm feel quite content that none of my hardware will run it. So I've been shying away from any further kicking of that dog. But in fairness I wanted to mention a bit of balancing good news that I imagine Paul will also be noting tomorrow:

The way things were originally setup in Windows 11 is that Microsoft had eliminated the simple ability of the user to change their system's default browser by removing the UI option to do so. Instead, they granularized the web browser's handler settings by file type (e.g. .HTM, .HTML, .PDF, etc.) and by protocol (e.g. HTTP, HTTPS, etc.) and they buried all of those deep in the registry. This forced any determined Windows 11 users to search through the registry, extension by extension and protocol by protocol, to manually change each one in order to unhook Edge.

But happily, with last week's release of the Windows 11 Insider build 22509, Microsoft has restored a single "Set Default" button, and has also surfaced all of those individual file and protocol types in the UI. So a non-Edge browser can be set as default and then, if desired for some reason, some other browser can override that default for specific file type and protocols.

RTF Templates being used to inject malicious content

The earliest Windows had the clean and simple Notepad app which was useful for displaying and editing unformatted and unformattable text files. I use it everyday. Notepad was implemented as a very lightweight UI around a simple Windows textbox container.

But Windows also offered Wordpad which allowed for a richer textual formatting experience. It was possible to set text color, size, treatment such as bolding, and alignment. And just as Notepad was a UI around the Windows textbox container, Wordpad's richer textual formatting experience was a UI around the Windows Rich Text Format (RTF) container.

I've long been a fan of Windows' RTF control and anyone who has ever used any of my Windows software will have seen its embedded pages with some text formatting. Those are all RTF controls.

In any event, unlike Notepad, the RTF container can also contain OLE — Object Linking and Embedding — objects. Because... why not? One of the many features of this embedding is the use of formatting templates which can be loaded on-the-fly from a file. And wouldn't you know it, malware deployers have figured out that these RTF template objects will also accept URLs. Sounds like a terrifically flexible feature! What could possibly go wrong?

Well... how about three different state-sponsored threat actors aligned with China, India, and Russia having been observed adopting RTF template injection as part of their phishing campaigns to deliver malware to targeted systems.

https://www.proofpoint.com/us/blog/threat-insight/injection-new-black-novel-rtf-template-inject-technique-poised-widespread

Researchers at ProofPoint have published the results of their research under the title: *"Injection is the New Black: Novel RTF Template Inject Technique Poised for Widespread Adoption Beyond APT Actors"*

Proofpoint threat researchers have observed the adoption of a novel and easily implemented phishing attachment technique by APT threat actors in Q2 and Q3 of 2021. This technique, referred to as RTF template injection, leverages the legitimate RTF template functionality. It subverts the plain text document formatting properties of an RTF file and allows the retrieval of

a URL resource instead of a file resource via an RTF's template control word capability. This enables a threat actor to replace a legitimate file destination with a URL from which a remote payload may be retrieved.

The sample RTF template injection files analyzed for this publication currently have a lower detection rate by public antivirus engines when compared to the well-known Office-based template injection technique. Proofpoint has identified distinct phishing campaigns utilizing the technique which have been attributed to a diverse set of APT threat actors in the wild. While this technique appears to be making the rounds among APT actors in several nations, Proofpoint assesses with moderate confidence, based on the recent rise in its usage and the triviality of its implementation, that it could soon be adopted by cybercriminals as well.

RTF template injection is a simple technique in which an RTF file containing decoy content can be altered to allow for the retrieval of content hosted at an external URL upon opening an RTF file. By altering an RTF file's document formatting properties, specifically the document formatting control word for "\template" structure, actors can weaponize an RTF file to retrieve remote content by specifying a URL resource instead of an accessible file resource destination. RTF files include their document formatting properties as plaintext strings within the bytes of the file. This allows the property control word syntax to be referenced even in the absence of a word processor application, providing formatting stability for the filetype across numerous platforms. However, RTF files based on the malleability of these plaintext strings within the bytes of a file are often subverted for malicious file delivery purposes in the context of a phishing campaign. While historically the use of embedded malicious RTF objects has been well documented as a method for delivering malware files using RTFs, this new technique is more simplistic and, in some ways, a more effective method for remote payload delivery than previously documented techniques.*

So, know this tune pretty well: Once again, an arguably unnecessary and seldom, if ever, used feature of RTF containers has been found to be exploitable by attackers and is now being used with growing popularity. As ProofPoint suggests, it's too easy to use and too powerful not to become much more widely adopted.

A Malicious Botnet uses the Bitcoin Blockchain

So, get a load of this one: A Botnet which Google says has infected more than 1 million Windows machines globally and continues to infect new machines at a rate of thousands more per day is using the public Bitcoin blockchain to stay in touch.

Google is suing two Russian individuals it claims are behind this sophisticated botnet operation. In a complaint filed in the US District Court for the Southern District of New York, Google names Russian nationals Dmitry Starovikov (sta-rov-a-kov) and Alexander Filippov (fill-ip-pov) as the two main operators of the Glupteba (glup-teba) botnet, citing Gmail and Google Workspace accounts they allegedly created to help them operate the criminal enterprise.

Google claims the defendants used the botnet network – which it describes as a "modern, borderless technological embodiment of organized crime" – for illicit purposes, including the theft and unauthorized use of Google users' logins and account information. Google is demanding that Dimitry and Alexander pay damages and are permanently banned from using Google services.

According to Google, the Glupteba botnet, which Google has been tracking since 2020, has so far infected approximately 1 million Windows machines worldwide and is growing at a rate of thousands of new machines each day. Once a device has been infected – typically by tricking users into downloading malware via third party "free download" sites – the botnet steals user credentials and data, secretly mines cryptocurrencies, and sets up proxies to funnel other people's internet traffic through infected machines and routers.

In its complaint, Google said: "At any moment, the power of the Glupteba botnet could be used in a powerful ransomware attack or distributed denial of service attack." Google also noted that the Glupteba botnet stands out compared to conventional botnets due to its technical sophistication and use of blockchain technology to protect itself from disruption.

As well as the launching litigation against the so-called Glupteba botnet, the company's Threat Analysis Group (TAG) – which has observed the botnet targeting victims in the U.S., India, Brazil, Vietnam and Southeast Asia – announced it has worked with internet hosting providers to disrupt the botnet's key command and control (C2) infrastructure. This means its operators no longer have control of the botnet, though Google has warned that Glupteba could return due to the fact it uses blockchain technology as a resiliency mechanism.

Google explained: "The Glupteba botnet does not rely solely on predetermined (web) domains to ensure its survival. Instead, when the botnet's C2 server is interrupted, the Glupteba malware is hard-coded to 'search' the public Bitcoin blockchain for transactions involving three specific Bitcoin addresses that are controlled by the Glupteba Enterprise. Thus, the Glupteba botnet cannot be eradicated entirely without neutralizing its blockchain-based infrastructure."

HP's has been shipping vulnerable printers for 8 years.

The guys at F-Secure carefully examined one HP printer, an M725z MFP. But once they reported their findings to HP at the end of April, HP found that their use of common firmware meant that at least 150 other printer models were also affected across the LaserJet, PageWide, and ScanJet families.

Updated firmware has been available from HP since November 1st, so anyone having any HP printer, who might be a target for either inside or outside attack will definitely wish to update all of their HP printers firmware.

The first of the two problems, tracked as CVE-2021-39238 with a CVSS rating of 9.3, describes a vulnerability can be used to create wormable exploits that can self-replicate and spread to other HP printers inside internal networks or over the internet. The trouble is particularly worrisome because it's a buffer overflow in the printer's font parser. So just causing a vunlerable printer to print a specially crafted page or PDF could take over the printer. The F-Secure researchers said that the flaw can be exploited to gain control over a printer's firmware to steal data or assemble devices into botnets, all while leaving little evidence of exploitation behind. They also indicated that attacks that abuse the vulnerability in various other ways, such as attacking internet-exposed devices or by loading the exploit code on a website or inside an ad. Users on corporate networks, or those who view the ad, will have the code reach out to ports on their internal network to exploit local HP printers. Yikes!

The second vulnerability is a local physical USB port attack tracked as CVE-2021-39237, which impacts the printer's communications board. Fortunately, this bug can only be exploited with physical access to a vulnerable device, and an attack takes up to five minutes to execute, compared to the first one, which only takes a few seconds.

So, as I said, if your enterprise uses HP printers, which are still the world's #1 printer by market share, make some time to update those device's firmware.

Sci-Fi

Just a note that my nephew is going nuts over Ryk Brown's latest novel, which presumably details the continuing adventures of Nathan, Cameron, Jessica, Vladimir, Telles, Josh, Loki and all the rest of the wonderfully articulated characters that Ryk has created. My nephew keeps texting me that the new novel is SO GOOD! (in all caps). But I know it will be there after I've seen what the Bobiverse is all about. And by then, there might even be another of Ryk's novels waiting!

SpinRite

The heuristic hardware interrupt handling solution I developed for SpinRite appears to have solved a number of current and doubtless future troubles. Additional feedback from our testers, and hours of staring at code, led to a number of additional tests and incremental improvements. As testers have swung by the development group over the weekend, one after another they've reported that this or that trouble their systems used to have is now gone and that SpinRite's latest development release is working perfectly for them. So there's been a lot of progress.

But there are still a few exceptions. A couple of people have reported that the size of the first drive is being confused with the size of the last BIOS drive in their system. It doesn't happen to me, nor to most people. But it reliably happens to a couple of testers. So something different about their systems. One person has a 70 terabyte Drobo directly attached and BIOS accessible to his machine. It's crashing SpinRite because a 70 terabyte drive is overflowing some registers which I didn't make big enough. I'll fix that as soon as I get back to it. And a few others are reporting that SpinRite's own division overflow pop-up is popping up.

The pop-up tells us where the division overflow occurred, but SpinRite's code doesn't have a division instruction at that location. The only conclusion is that something is causing SpinRite's own code to be overwritten. Fortunately, what's being overwritten contains either an F6 or an F7 byte, which are the x86 division opcodes.

I know where the first drive's size is stored — which is being changed when it shouldn't be — and the errant division overflow pop-up tells us where a division instruction has mysteriously appeared in SpinRite's code. So we need to track down what's going on with the drive size being changed and the code being overwritten. To do that we need to catch the overwriting action in the act, as it's happening. The problem is, I have never been able to reproduce any of that myself. It's only happening on a few distant machines.

The answer, which I will immediately implement a few hours from now after this podcast, is to build a native debugger directly into SpinRite. And that's easier than it sounds. The Intel x86 architecture has always incorporated built-in hardware debugging assistance. It has four 32-bit breakpoint registers which can be set to point to any region of the chip's memory. Each of the four breakpoint registers can be set to interrupt the processor upon the execution, the reading or the writing of that memory with a 1, 2, 4 or 8 byte span. So, the next test release of SpinRite will add a couple of command line options to load and enable those debug registers. Once a tester has seen where SpinRite reported an erroneous division instruction, they'll be able to immediately re-run it, setting a write breakpoint at that spot in SpinRite's code which will allow us to catch-in-the-act whoever it is that's overwriting SpinRite.

I'm hoping to have these remaining problems behind us by next week's podcast. :)

XSinator

From a Formal Model to the Automatic Evaluation of Cross-Site Leaks in Web Browsers

Their paper is titled: "XSinator.com: From a Formal Model to the Automatic Evaluation of Cross-Site Leaks in Web Browsers." It's the result of comprehensive work conducted by a team of five German University researchers. As a result of this work they discovered 14 new types of cross-site data leakage which are effective against a number of modern web browsers, including the Tor Browser, Firefox, Chrome, Edge, Safari, Opera and others.

They call the bugs "XS-Leaks" (as in cross-site) because they enable malicious websites to harvest personal data from their visitors as they interact with other websites in the background without their visitor's knowledge or permission.

In a recent statement about their research, which was presented during last month's 2021 ACM SIGSAC Conference on Computer and Communications Security, and which garnered a "Best Paper Award", they explained: *"XS-Leaks bypass the so-called same-origin policy, one of a browser's main defences against various types of attacks. The purpose of the same-origin policy is to prevent information from being stolen from a trusted website. In the case of XS-Leaks, attackers can nevertheless recognize individual, small details of a website. If these details are tied to personal data, those data can be leaked."*

Our listeners probably know by now that I'm a sucker for formally-proven security findings. There's certainly a place for fuzzing, which is probably the other end of the spectrum from formal proofs. And formal models won't help when modeling cannot be applied. But whenever possible, creating a mathematically formal model then using that model to reach and/or demonstrate security conclusions is, to my mind, the gold standard. Here's how this team describes what they have accomplished:

ABSTRACT

A Cross-Site Leak (XS-Leaks) describes a client-side bug that allows an attacker to collect side-channel information from a cross-origin HTTP resource. They pose a significant threat to Internet privacy, since simply visiting a web page may reveal if the victim is a drug addict or leak a sexual orientation. Numerous different attack vectors, as well as mitigation strategies, have been proposed, but a clear and systematic understanding of XS-Leak' root causes is still missing.

Recently, Sudhodanan et al. gave a first overview of XS-Leak at the Network and Distributed System Security Symposium (NDSS). We build on their work by presenting the first formal model for XS-Leaks. Our comprehensive analysis of known XS-Leaks reveals that all of them fit into this new model. With the help of this formal approach, we (1) systematically searched for new XS-Leak attack classes, (2) implemented XSinator.com, a tool to automatically evaluate if a given web browser is vulnerable to XS-Leaks, and (3) systematically evaluated mitigations for XS-Leaks. We found 14 new attack classes, evaluated the resilience of 56 different browser/OS combinations against a total of 34 XS-Leaks, and propose a completely novel methodology to mitigate XS-Leaks.

My results with Firefox

0	Performance API Error Leak	Detect errors with Performance API.
1	Event Handler Leak (Object)	Detect errors with onload/onerror with object.
2	Event Handler Leak (Stylesheet)	Detect errors with onload/onerror with stylesheet.
3	Event Handler Leak (Script)	Detect errors with onload/onerror with script.
4	MediaError Leak	Detect status codes with MediaError message.
5	Style Reload Error Leak	Detect errors with style reload bug.
6	Request Merging Error Leak	Detect errors with request merging.
7	CORS Error Leak	Leak redirect target URL with CORS error.
8	Redirect Start Leak	Detect cross-origin HTTP redirects by checking redirectStart time.
9	Duration Redirect Leak	Detect cross-origin redirects by checking the duration.
10	Fetch Redirect Leak	Detect HTTP redirects with Fetch API.
11	URL Max Length Leak	Detect server redirect by abusing URL max length.
12	Max Redirect Leak	Detect server redirect by abusing max redirect limit.
13	History Length Leak	Detect javascript redirects with History API.
14	CSP Violation Leak	Leak cross-origin redirect target with CSP violation event.
15	CSP Redirect Detection	Detect cross-origin redirects with CSP violation event.
16	WebSocket Leak (FF)	Detect the number of websockets on a page by exausting the socket limit.
17	WebSocket Leak (GC)	Detect the number of websockets on a page by exausting the socket limit.
18	Payment API Leak	Detect if another tab is using the Payment API.
19	Frame Count Leak	Detect the number of iframes on a page.
20	Media Dimensions Leak	Leak dimensions of images or videos.
21	Media Duration Leak	Leak duration of audio or videos.
22	Performance API Empty Page Leak	Detect empty responses with Performance API.
23	Performance API XSS Auditor Leak	Detect scripts/event handlers in a page with Performance API.
24	Cache Leak (CORS)	Detect resources loaded by page. Cache is deleted with CORS error.
25	Cache Leak (POST)	Detect resources loaded by page. Cache is deleted with a POST request.
26	Id Attribute Leak	Leak id attribute of focusable HTML elements with onblur.
27	CSS Property Leak	Leak CSS rules with getComputedStyle.
28	SRI Error Leak	Leak content length with SRI error.
29	ContentDocument X-Frame Leak	Detect X-Frame-Options with ContentDocument.
30	Performance API X-Frame Leak	Detect X-Frame-Options with Performance API.
31	Performance API CORP Leak	Detect Cross-Origin-Resource-Policy header with Performance API.
32	CORP Leak	Detect Cross-Origin-Resource-Policy header with fetch.
33	CORB Leak	Detect X-Content-Type-Options in combination with specific content type using CORB.
34	Download Detection	Detect downloads (Content-Disposition header).
35	Performance API Download Detection	Detect downloads (Content-Disposition header) with Performance API.
36	CSP Directive Leak	Detect CSP directives with CSP iframe attribute.
37	COOP Leak	Detect Cross-Origin-Opener-Policy header with popup.

My results with Chrome

0	Performance API Error Leak	Detect errors with Performance API.
1	Event Handler Leak (Object)	Detect errors with onload/onerror with object.
2	Event Handler Leak (Stylesheet)	Detect errors with onload/onerror with stylesheet.
3	Event Handler Leak (Script)	Detect errors with onload/onerror with script.
4	MediaError Leak	Detect status codes with MediaError message.
5	Style Reload Error Leak	Detect errors with style reload bug.
6	Request Merging Error Leak	Detect errors with request merging.
7	CORS Error Leak	Leak redirect target URL with CORS error.
8	Redirect Start Leak	Detect cross-origin HTTP redirects by checking redirectStart time.
9	Duration Redirect Leak	Detect cross-origin redirects by checking the duration.
10	Fetch Redirect Leak	Detect HTTP redirects with Fetch API.
11	URL Max Length Leak	Detect server redirect by abusing URL max length.
12	Max Redirect Leak	Detect server redirect by abusing max redirect limit.
13	History Length Leak	Detect javascript redirects with History API.
14	CSP Violation Leak	Leak cross-origin redirect target with CSP violation event.
15	CSP Redirect Detection	Detect cross-origin redirects with CSP violation event.
16	WebSocket Leak (FF)	Detect the number of websockets on a page by exausting the socket limit.
17	WebSocket Leak (GC)	Detect the number of websockets on a page by exausting the socket limit.
18	Payment API Leak	Detect if another tab is using the Payment API.
19	Frame Count Leak	Detect the number of iframes on a page.
20	Media Dimensions Leak	Leak dimensions of images or videos.
21	Media Duration Leak	Leak duration of audio or videos.
22	Performance API Empty Page Leak	Detect empty responses with Performance API.
23	Performance API XSS Auditor Leak	Detect scripts/event handlers in a page with Performance API.
24	Cache Leak (CORS)	Detect resources loaded by page. Cache is deleted with CORS error.
25	Cache Leak (POST)	Detect resources loaded by page. Cache is deleted with a POST request.
26	Id Attribute Leak	Leak id attribute of focusable HTML elements with onblur.
27	CSS Property Leak	Leak CSS rules with getComputedStyle.
28	SRI Error Leak	Leak content length with SRI error.
29	ContentDocument X-Frame Leak	Detect X-Frame-Options with ContentDocument.
30	Performance API X-Frame Leak	Detect X-Frame-Options with Performance API.
31	Performance API CORP Leak	Detect Cross-Origin-Resource-Policy header with Performance API.
32	CORP Leak	Detect Cross-Origin-Resource-Policy header with fetch.
33	CORB Leak	Detect X-Content-Type-Options in combination with specific content type using CORB.
34	Download Detection	Detect downloads (Content-Disposition header).
35	Performance API Download Detection	Detect downloads (Content-Disposition header) with Performance API.
36	CSP Directive Leak	Detect CSP directives with CSP iframe attribute.
37	COOP Leak	Detect Cross-Origin-Opener-Policy header with popup.

My results from running the https://xsinator.com/ test on Firefox 94 and Chrome 96.

Web Applications and User States.

In a web application, a web browser interacts with several web servers through HTTP or Web-Socket connections. The client-side logic of the web application is written in HTML, CSS, and JavaScript code, and is executed inside a tab of the browser, or inside an inline frame in another application. The execution context of a web application is defined through the concept of web origins. Web applications may call and embed other web applications to enhance functionality. For example, a hotel reservation site may embed Google Maps and public

transportation sites as an easy method to allow its customers to determine how to reach the hotel. In such situations, cross-origin HTTP requests between different web origins are necessary to retrieve data to embed and display in the web application.

When interacting with a website, a user has a well-defined state – this state typically contains the information whether the user is logged in or not. Besides the login status, the user state may contain account permissions, such as admin privileges, premium membership, or restricted accounts. The number of different user states is potentially unlimited. For example, in a webmail application, a user may or may not have received an email with the subject "top secret".

Privacy Risks of Cross-Origin Requests.
Consider the following situation: the attacker has lured a victim to a malicious web application that executes hidden cross-origin HTTP requests to different drug counseling sites. If the attacker could learn whether the victim is logged in at one of these drug counseling sites, the attacker would gain highly privacy-critical information about the victim.

Cross-Site Leaks on User States.
To distinguish between two user states, the attacker's JavaScript code must be able to identify differences in its own execution environment resulting from different responses to cross-origin HTTP requests. These different responses must correspond to different user states at the target web application. If this differentiation is possible, we call this vulnerability an XS-Leak. The attacker can then craft a malicious website, which triggers the XS-Leak once the victim visits it:

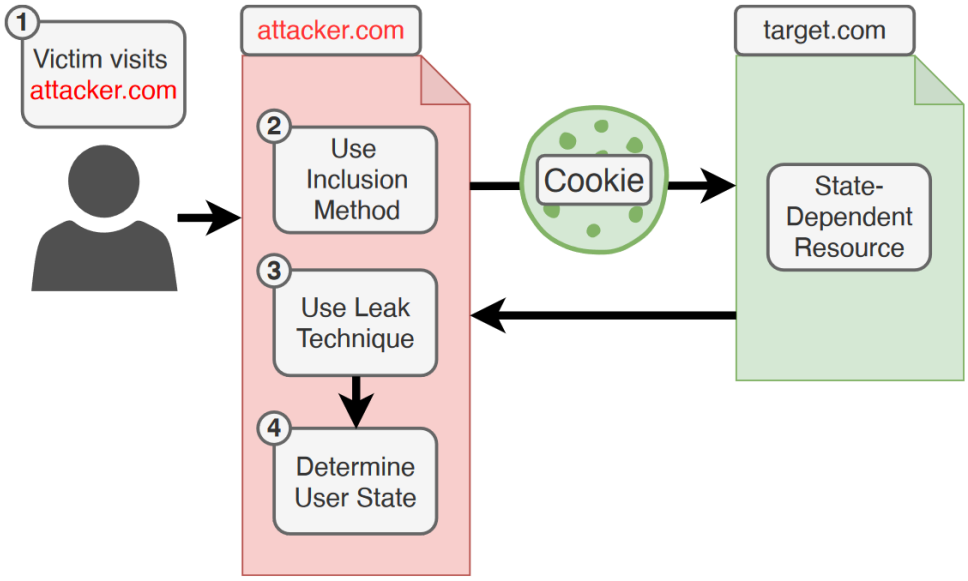


Figure 1: XS-Leak attack flow example. The victim (1) visits the attacker-controlled website, which (2) uses an inclusion method to request a state-dependent resource from a target website. The attacker then uses (3) a leak technique to (4) determine the victim's user state.

In the following two real-world examples, we try to exemplify the scope of the problem.

XS-Leak on Gitlab.
Gitlab is a popular web application for collaborative software development hosted by many companies. Gitlab provides a profile URL https://git.company.com/profile: if the user is not logged in, this URL redirects the user to https://git.company.com/users/sign_in; if the user is logged in, the current user's profile information is shown. However, since the attacker embeds Gitlab cross-origin into the attacker's own web page, the attacker cannot directly read the URL.

Listing 1: XS-Leak on Gitlab.

```
let url = 'https://git.company.com/profile'
let ref = window.open(url, '_blank')
// wait until pop-up is loaded
let counted_frames = ref.window.length;
if (counted_frames === 0) {
    // User is logged in
} else if (counted_frames === 3) {
    // User is NOT logged in
}
```

In Listing 1 (above), we use the window.length property, which is readable cross-origin, to determine the user state; the profile page does not contain any iframes, but the login page includes three frames. If this property has the value 3, the user is not logged in. If it has the value the 0, the user is logged in. By scanning different company websites hosting Gitlab, the attacker may collect information on a programmer's affiliation.

XS-Leak on Google Mail.
Google Mail is one of the most popular webmail applications. In 2019, Terjanq reported a XS-Leak which could determine whether an email with a certain subject (e.g., "drug counseling") or content was present in the user's inbox cross-origin. The XS-Leak abused the common cache that web applications share. By using the advanced search option, which can be called cross-origin, Google Mail marks search results (if any exist) with a dedicated image. To perform an XS-Leak attack, the attacker first empties the web cache, then calls Google Mail advanced search, and finally checks if the dedicated image is available in the cache. If true, the search was successful, and the attacker learned that an email containing the used search term exists in the victim's inbox.

So let's make sure that everyone understands where we are: Any gMail user visits some random nosey website. Without any indication of any kind, that nosey website running JavaScript on the user's web browser is able to use subtle and clever tricks to perform as many go/no-go searches as they wish against the user's gMail inbox entirely behind their backs.

We've talked about this general class of web browser hacks a number of times in the past. Broadly speaking, it is incredibly difficult to robustly prevent any cross-domain leakage. We've seen that initially benign seeming convenience features, such as changing the color of previously visited links, can compromise privacy when website A uses the user's browser to display website B off screen and then probes the website B DOM — its document object model — for the colors of various links. In this way, information that is none of website A's business leaks from the

user’s previous use of website B.

What this team has done is further explore and map brand new forms of subtle side-channel cross-origin information leakage. And they’ve come away with a surprisingly large number of new ways for clever attackers to leverage deliberate features of JavaScript — like using the window length of some other website’s page — to infer information about the browser’s owner.

We’ve seen that previous cross-domain cookie and cache attacks have led browser developers to segment all cookie storage, by domain, and to similarly segment all browser cache by domain. Since the performance impact of this is negligible, the great impact is upon browser complexity and maintenance, which have skyrocketed. Have you looked at the number of processes today’s web browsers are launching in our systems? Like today’s operating systems, web browsers are no longer something that can be casually assembled.

But the nature of the leakage these guys have uncovered and demonstrated is different, because it leverages non “grey area” features of our browsers. They are employing features that are there by design. This suggests that in the future, our web browsers may choose to deliberately strip these side-channel features and limit JavaScript’s interaction with cross-origin domains. That would probably be a good thing.