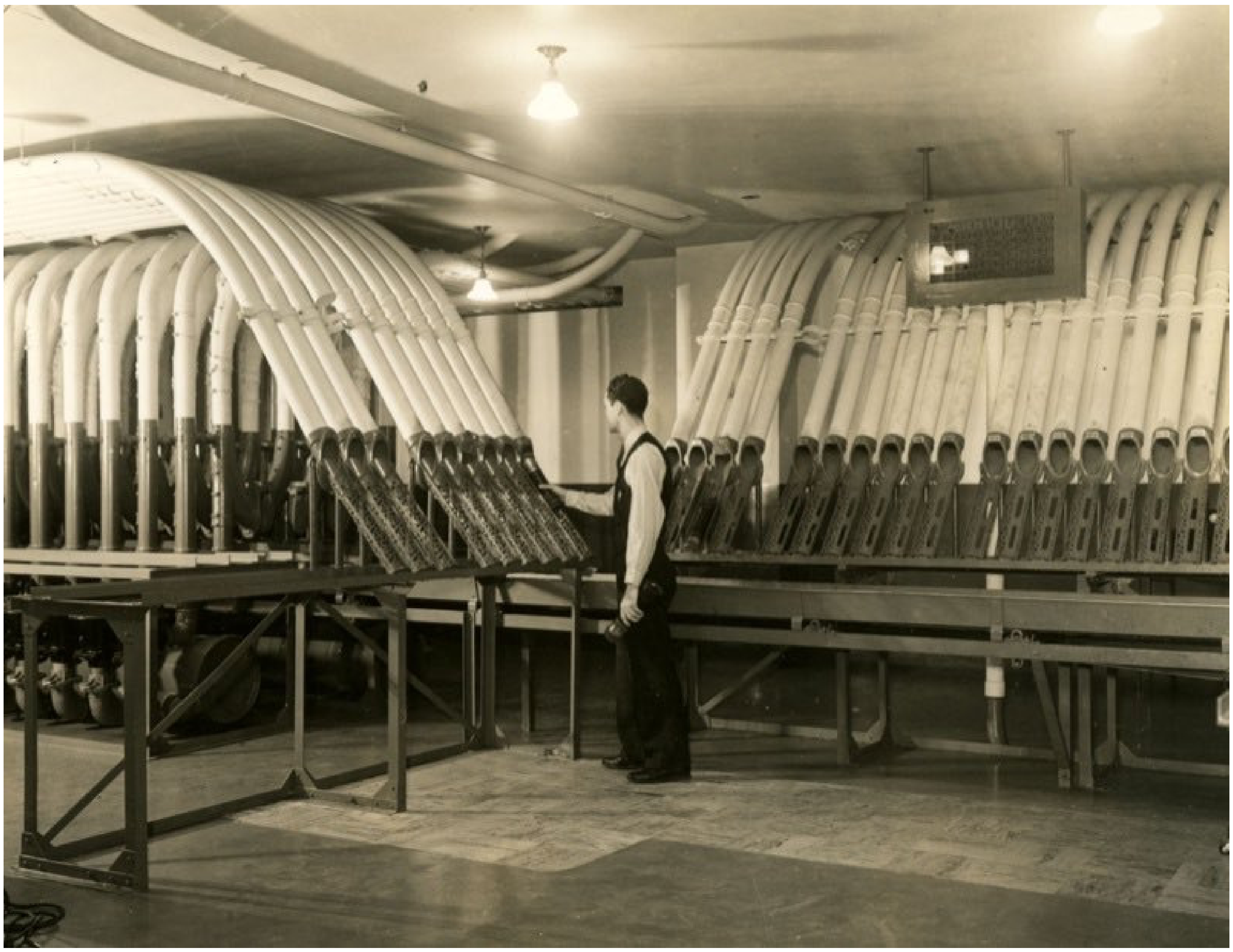
Security Now! #902 - 12-20-22
A Generic WAF Bypass

This week on Security Now!

This week we answer another collection of burning questions: Is there no honor among thieves? What was discovered during this year's Toronto Pwn2Own competition? What did we learn from last Tuesday's patchfest? Who's fault was the most recent Uber data breach? What happened when Elon tried to block all the bots? What's the first web browser to offer native support for Mastodon? What exactly is "Coordinated Inauthentic Behavior" and why is it such a problem? What will happen to GitHub submitters at the end of next year? What measure could every member of the US senate possibly agree upon? Exactly what applications are there for a zero-width space character? And finally, what larger lesson are we taught by the discovery of a serious failure to block a problem that we should never have had in the first place? The answer to all those questions and more await the listeners of today's Security Now podcast #902.

Old School "Message Routing"



Security News

A malware operation known as URSNIF, which we've noted a few times, is the 4th one this year to suffer from internal squabbles that surface in the public eye. Disagreements over Russia's invasion of Ukraine and strong pro-Russian sentiments which divided previous groups were the nominal triggers then... but it appears that simple greed was the motivation this time.

Through a Twitter account "@URSNIFleak" an ex-member of the account announced his intention to leak the real world identities of the top leaders of the group unless he received a significant payout. To prove his willingness to do this, in a succession of Tweets he leaked various pieces of internal dialog, some source code, and the names of three low-level group members.

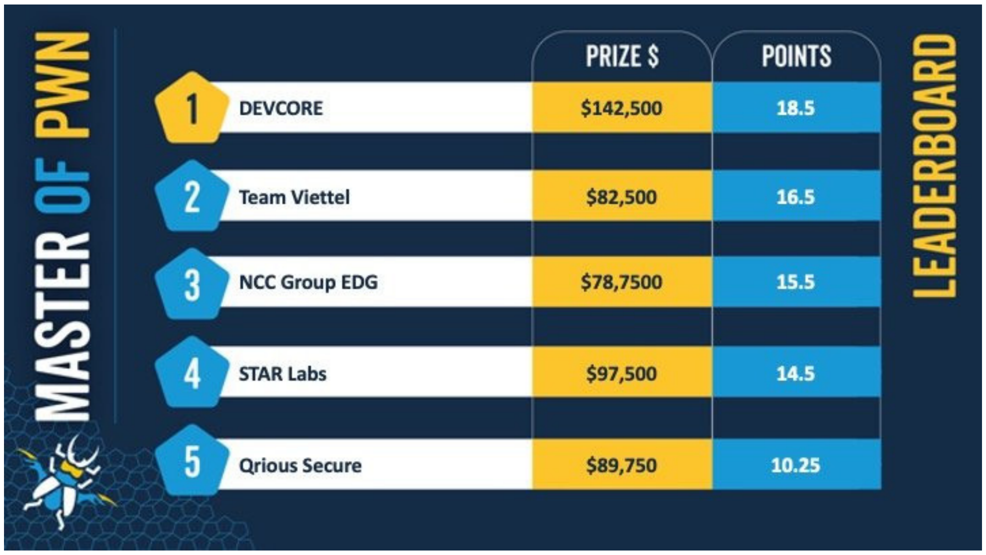
And that was enough to get this person paid. After that he Tweeted: *"I just made more money in a single week than I have made in years. Pay workers right and they won't have a reason to leak stuff."* I note that it's interesting that this person considers this to have been the act of "making money." What a different culture. Apparently, the motivation to extort was heightened by something the head of the group said in an interview with the VX-Underground project, though it's not clear what was upsetting. The URSNIFleaker Tweeted: *"The interview angered me. He has been a bad boss for a long time. I have been waiting for the right time to release."*

I don't think we can count on all of the major groups to implode, but there's probably a little extra tendency for that to happen within an organization comprised of people who must be aware that what they are going is not earning an honest day's living. At least I hope so. :-/

Pwn2Own Toronto 2022

It's always interesting to see what hackers wearing white hat are able to do to today's fully patched and up to date systems. Since vulnerabilities are blind to the shade of the hat being worn by those exploiting it, where white hats can go black hats are more than likely to follow.

The recently concluded Toronto 2022 hacking contest focused upon hacking routers, smartphones, printers, and other smart devices, and this year's 4-day contest was won by DEVCORE — the now well-known Chinese Taiwanese pen-testing group:


MASTER OF PWN
LEADERBOARD

		PRIZE $	POINTS
1	DEVCORE	$142,500	18.5
2	Team Viettel	$82,500	16.5
3	NCC Group EDG	$78,7500	15.5
4	STAR Labs	$97,500	14.5
5	Qrious Secure	$89,750	10.25

To give everyone some sense for this, I'm just going to quickly scan down a few bullet points which briefly describe the attacks:

- A stack-based Buffer Overflow attack against the Canon imageCLASS MF743Cdw printer.
- A 2 bug authentication bypass and command injection attack against the WAN interface of a TP-Link AX1800 router.
- A command injection attack which caused a Lexmark MC3224i printer which serenaded the audience with a well-known Mario Brothers tune.
- A command injection attack against the WAN interface of the Synology RT6600ax router.
- A stack-based buffer overflow attack against an HP Color LaserJet Pro M479fdw printer.
- An improper input validation attack against the Samsung Galaxy S22.
- A command injection root shell attack against the LAN interface of the Synology RT6600ax router.
- Another improper input validation attack against the Samsung Galaxy S22.
- A 2 bug attack, SQL injection and command injection, against the LAN interface of the NETGEAR RAX30 AX2400 router.
- Two different Stack-based buffer overflow attacks against a Mikrotik router and a Canon printer.
- 3 bugs (2x Missing Auth for Critical Function and an Auth Bypass) attack against the Synology DiskStation DS920+ NAS.
- 2 bugs (including a command injection) in an attack against the HP Color LaserJet Pro M479fdw printer.
- 5 different bugs leveraged in an attack against the LAN interface of the NETGEAR RAX30 AX2400 router.
- 3 different bugs against a NETGEAR router and an HP printer.

And all that was what happened during only the first day of the four-day hacking contest. It kept going like that throughout the entire event.

As we know, LAN-side attacks on routers and NAS devices are much less concerning than attacks that can be launched against the WAN interface. But this contest revealed plenty of both of those. And the number of printer vulnerabilities that still exist – well, I suppose we shouldn't be surprised. But obtaining well-hidden persistence inside a network is an overriding goal of anyone who penetrates an enterprise's perimeter. And printer protocols loudly broadcast and advertise on networks because their goal is to be found. Unfortunately, this results in highly vulnerable printers shouting their presence and creating a perfect and often unsuspected place for malicious post-intrusion malware to set up shop and wait.

And speaking of getting into networks...
It's not just lower-end IoT devices that are permitting bad guys to get into networks. Both Citrix and Fortinet – who are two of today's largest providers of enterprise networking equipment – recently released security updates to patch 0-day vulnerabilities that were being exploited in the wild against their devices.

In the case of the Fortinet 0-day, which created an unauthenticated remote code execution in the FortiOS that runs on the company's SSL-VPN devices, it was the way some ransomware was managing to crawl inside enterprise networks. And it was so bad that Fortinet did the right thing by also offering down-version patches for their older out-of-support devices still running their also-vulnerable 6.0 firmware. The 0-day was first spotted being used in the wild by a French security firm Olympe last week, and to Fortinet's credit they patched it over the weekend in just three days.

I mentioned two 0-days and Citrix, their's was the other. And it's also an unauthenticated RCE. This one was spotted by the NSA – yes, our own National Security Agency. In their security advisory the NSA wrote that they saw the Chinese cyber-espionage group designated APT5 leveraging that Citrix 0-day, but the NSA offered nothing further about what was being done with the obtained leverage.

Patch Tuesday
Last Tuesday was the industry's increasingly well attended final monthly patch event of the year. And those offering up incrementally more secure improvements in their code notably included Adobe, Android, Apple, Microsoft, Mozilla, SAP, and VMware.

Microsoft fixed 72 security flaws this month across their range of offerings, and that included a 0-day that was being used to circumvent Microsoft's SmartScreen and Mark-of-the-Web detection using standalone JavaScript files. We covered this trouble recently so it's great that it's fixed.

The other issue Microsoft addressed was a problem that we noted before where, somehow, malicious Windows drivers were being used by the Hive and Cuba ransomware strains and those malicious drivers were being trusted by Windows because they carried valid Microsoft signatures. What?! Yep. In this month's advisory, Microsoft wrote:

"We were notified of this activity by SentinelOne, Mandiant, and Sophos on October 19, 2022, and subsequently performed an investigation into this activity. This investigation revealed that several developer accounts for the Microsoft Partner Center were engaged in submitting malicious drivers to obtain a Microsoft signature. A new attempt at submitting a malicious driver for signing on September 29, 2022, led to the suspension of the sellers' accounts in early October."

And not to be left out, Apple also updated WebKit to fix a 0-day that was being used in targeted attacks against iOS users.

Another Uber breach?
Uber has been having a rough time recently. Recall that about four months ago the Lapsus$ gang breached Uber's security and caused trouble.

What's interesting about last week's second breach, which resulted in the leaking of the personal data of more than 77,000 Uber employees, and also source code and credentials for some of the company's internet IT network – the authenticity of which Uber has confirmed – is that this wasn't directly Uber's fault. The breach occurred in the network of an Uber-contracted IT service provider whose name suggests that all of the good names are already taken. This company chose to name itself Teqtivity – but good luck trying to find them. "I tried to find you on the Internet and you're not there, anywhere." "Oh, well your problem was, you spelled our name correctly. You don't want to do that. It's "Teqtivity" – T E Q T I V I T Y." "What?" Right.

The day after Uber outed Teqtivity as the source of their latest leak, Teqtivity themselves disclosed the breach last Thursday. Uber may have been Teqvitity's biggest customer, I have no idea, but I mention this because other notable companies may have also had their data stolen since a breach of one large service provider can potentially expose the data belonging to all of their clients.

As an industry these days, we're really sort of facing a conundrum: Do you run your own inhouse shop where you're solely responsible for your own company's security? Or do you decide that running networks and servers and points of presence and dealing with the constant need to focus upon security is not your mainline business – and that it has just become too complicated to do it right? So you farm it out to someone who promises you that it **will** be their mainline business because that's all they're going to do – it **is** their business. Today, I think that's a tough call. It can work out and be extremely cost effective so long as everything goes well. And perhaps that's the best that can be hoped for.

Elon Botches ʽBot Blockage

From the outside looking in, it's difficult to understand the mechanisms at play inside Elon Musk's Twitter reign. From the outside, anyone would get a sense of things lurching back and forth inside Twitter, presumably as Elon's self-described biological neural net fires off whimsical edicts which Twitter's remaining employees apparently quickly implement without any buffering in a desperate effort to hold onto their own paychecks in this chaotic and fragile work environment. We're done with layoffs, then we have more layoffs. No, now we're really done with layoffs, then entire departments disappear. Collections of press accounts are suspended for an interval of 7 days, until 1 day later when they're reinstated. A new policy states that anyone Tweeting a link which points to another social network will have their account suspended... until a few hours later when that policy ends. It really has been quite something to watch. And as I'm assembling the notes and details of this podcast, when I follow links to online events that would have once linked to Twitter, I'm increasingly being taken to Mastodon.

Last week, something else happened as a result of a misfiring of Elon's biological neural net: He decided that he was going to block all of the bots. This was something that had endlessly bedeviled all of the pre-Elon Twitter engineers. How to block the bots? Elon had the answer!
So he declared publicly that he had a surprise for all of the bot farms and last Monday Twitter blocked entire IP address blocks which were used by approximately 30 mobile carriers across Asia. According to Platformer, this included the primary telecom providers for all of India and Russia, as well as Indonesia's second-largest telecom. Of course, there were vastly more legitimate users in every one of those address blocks than there were bots. So three countries worth of legitimate Twitter users, who all shared the same IP address blocks as a few bots, were completely cut off from Twitter. How could anyone not anticipate that?

Elon wants to own Twitter. But Twitter is **not** technology. It is **enabled** by technology. Twitter is a community. A community can be enabled and nurtured and encouraged; the one thing it cannot be... is owned. Nobody owns Twitter's community. No one can. Not even Elon.

Vivaldi integrates Mastodon in its desktop browser.

"Vivaldi recently became the first browser to have its own Mastodon instance, Vivaldi Social. Now, the new version on the desktop is the first to integrate Mastodon into the browser itself, along with the ability to pin tab groups and other UI improvements. We believe in providing alternatives to Big Tech while putting your privacy first and launched Vivaldi Social, our Mastodon instance. And today we are integrating Vivaldi Social into the sidebar of our desktop browser becoming the first browser to offer this functionality."

https://vivaldi.com/press/vivaldi-integrates-mastodon-in-its-desktop-browser/

5,200 Dutch government warnings

On the topic of governments recognizing the growing dangers of known vulnerabilities in the networks of the enterprises within their borders, the Dutch government said that since the summer of 2021 last year, it has sent more than 5,200 warnings to Dutch companies concerning security vulnerabilities within their networks. Officials said that around 76% of these warnings were for sensitive systems being accessible via the internet (RDP, SMB, LDAP, etc.). The other 24% of the warnings regarded malware infections, leaked credentials, or unpatched systems. Since this is not the first time we've encountered this, and it seems like an entirely sane thing for governments to do to help protect their national interests, I expect we're going to be seeing more announcements of this sort in coming years.

CIB: "Coordinated Inauthentic Behavior"

A recent report from FaceBook's parent company, Meta, introduced me to a new term that's abbreviated CIB. I love the term. CIB stands for "Coordinated Inauthentic Behavior." It's such a wonderfully neutral and politically correct term to describe the behavior of organizations and countries that have figured out that they can use fraudulent postings and replies on FaceBook to influence beliefs and behavior through massive coordinated campaigns. I encounteredt this term in last Thursday's FaceBook report titled: "Recapping Our 2022 Coordinated Inauthentic Behavior Enforcements".

Facebook noted that since they began focusing upon the explicit abuse of FaceBook services for what they term covert influence operations they've disrupted 200 identifiably separate global networks. Those networks were based in 68 countries and operated in at least 42 different languages, with two thirds of the campaigns targeting their own local audiences in their home countries with only one third aimed at audiences abroad.

In terms of targets, more than 100 different countries, from Afghanistan to Zimbabwe, have been targeted by at least one CIB network — foreign or domestic — with the US being the most targeted at 34 operations, followed by Ukraine targeted by 20 CIB networks, and then the UK targeted by 16 operations. And a single covert network might often by simultaneously targeting multiple countries at once. In one case, for example, a network running from Iran was simultaneously targeting 18 countries on 4 continents.

As for the originators of these campaign networks, Russia leads the list with 34 networks identified, followed closely by Iran with 29 and then fewer than half of that with Mexico at 13 networks. Interestingly, those are the top 3 and notice that China is not among them. Russia and Iran are the biggest perps in this game.

I was surprised about Mexico, so I went looking for some more information about them. As I suspected, Most of the CIB networks originating in Mexico have focused primarily on regional or local audiences, often in the context of regional elections. These networks tended to be less tactically sophisticated, and many were linked to PR or marketing firms including instances where one network supported two rivals for the same electoral post. The report noted that this illustrates the danger of using covert "Influence Operations for hire" services that might be providing inauthentic support to not just the highest bidder, but multiple bidders at once.

GitHub to require 2FA by the end of next year

The one year clock begins. By the end of next year, all GitHub code contributors will either be using some form of second factor authentication, or they won't be logging in at all. What's going to happen is that throughout next year, users will be notified of their need to enroll in some form of second factor authentication, and at the time of notification they'll be given 45 days to do so. Given that enrollment takes about 5 minutes, 45 days should be plenty. But if for some reason those 45 days lapse they'll lose access to their GitHub account. Five minutes folks, just do it now.

Bye bye SHA-1

Bye bye SHA-1. We might say "we hardly knew ye"... but we knew ye quite well.
The NIST has formally announced what many of us have been assuming for some time: The aging original SHA-1 cryptographic hashing function is being retired. In its place is either SHA-2 or SHA-3. But I did a bit of a double take when I saw that companies have until the end of 2030, in other words until 2031, so another entire eight years from now, to make that replacement. The end of NIST's announcement said:

"Modules that still use SHA-1 after 2030 will not be permitted for purchase by the federal government. Companies have eight years to submit updated modules that no longer use SHA-1. Because there is often a backlog of submissions before a deadline, we recommend that developers submit their updated modules well in advance, so that CMVP has time to respond."

A cryptographer might have been a bit more explicit and careful in the wording of that mandate. I'd have written *"Modules that are still capable of using SHA-1 after 2030..."* The reason for the added clarity is that many cryptographic systems obtain robust interoperability by comparing acceptable protocols that both ends understand and then negotiating the best and hopefully the most secure among those. But through the years of this podcast we've examined a great many "downgrade attacks" where a malicious endpoint identifies that the other end is still offering a no longer considered safe weak cryptographic protocol. So the sneaky end pretends that it cannot use any of the stronger systems, thus tricking the agreeable end into establishing a potentially vulnerable connection. So what we want is for all systems to immediately eliminate SHA-1 from their collection of acceptable hashing functions.

It's a fine point, but for the record SHA-1 would still make a fine hash for use in a PBKDF – password based key derivation function – where a hash is iterated a great many times. But given that its presence might allow its misuse, removing it altogether is best.

WordFence's VERY useful looking WordPress add-on vulnerability database:
Last Wednesday, WordFence, the very useful 3rd-party WordPress application firewall people who have been identifying troubled WordPress add-ons launched a free and VERY useful looking vulnerability database for WordPress add-ons. I poked around it a bit and I'm impressed. So I wanted to give our WordPress users a heads-up about it:
https://www.wordfence.com/threat-intel/

Closing The Loop

Michael Lawley / @lawley

Please @SGgrc it's pronounced meddy bank

Oh, thanks! I'm glad to know that. I suppose that does sound a lot more Australian!

SKYNET / @fairlane32

Question about ADP. Once it's turned on and the keys are sent down to your device, is it stored in hardware or software? Because what happens when you get a new iPhone in the future, how do you get the keys over to your new iPhone? You can't set up the new phone and restore an iCloud backup once you log on so it would have to be by the method where you move your old iPhone close to your new phone, correct? 🧐 *Thanks.*

The primary concept is similar to the familiar pattern that LastPass and presumably other password managers use. In all of those cases, they are simply storing an encrypted blob on our behalf. They have no visibility into the blob. But by making that blob available, devices are able to share a common set of passwords or, as in the case of Apple, a common set of decryption keys.

So the process of Apple relinquishing the keys to iCloud is that Apple sends the current Keychain blob (which it is never able to decrypt) and the current iCloud keys, which until now it has held in its data centers' HSM's, to the user's device. The device uses its local private account key, which never leaves the device, to decrypt the Keychain blob and then adds the current iCloud key to the Keychain. In this way, the keys that Apples was holding are moved where Apple cannot get to them, into the user's account keychain. The device then instructs Apple to delete the iCloud keys that it just sent from all of its datacenter HSMs. Now, only the device has the old iCloud keys in its keychain. Then, wanting to be thorough, the device performs a key rotation, changing the key that encrypts the iCloud data to one that Apple has never had in its possession.

But, again, since we're all quite familiar with the notion of TNO and PIE password managers holding encrypted blobs that they cannot decrypt, I think that's likely the best analogy.

Walt Stoneburner / @waltomatic

Steve, you have warned several times that pixelation is not a safe redaction technique. Someone just wrote a beautiful GitHub project that visually brings home the point as you see unredaction being performed.

Michael Brødsted @Brodsted

Hi steve Love your show. Read this article and thought it might be interesting for you.

https://www.theverge.com/2022/12/16/23512952/anker-eufy-delete-promises-camera-privacy-encryption-authentication

This was The Verge's follow-up on their story about the Eufy cameras. Remember the cameras that promised that all storage was local and that nothing ever left the user's house and that it was all transmitted directly to their phone? This was the company or product that was owned by Anker and our conjecture was that after having launched their successful power supply product line they had perhaps purchased the Eufy cameras in order to grow their business.

Anyway, The Verge checked back and when did they find? Their follow-up story is headlined "Anker's Eufy deleted these 10 privacy promises instead of answering our questions" and the sub-head reads "Two weeks after getting caught lying to The Verge, Anker still hasn't sent us any answers about its security cameras. Instead, it's nerfed the Eufy "privacy commitment."

The Verge has a wonderful mouse-based sliding divider on their new page where you can slide the mouse left and right to show the old or the new privacy claims.

Anyway, enough said. Their original claims were a complete fabrication that did not reflect reality. So someone, somewhere, insisted that they update the Eufy camera's description so that it no longer made any false claims.

Alain / @Alain_Gyger

Thanks for another excellent episode … I do have one question about TikTok. Do you see a difference between the bans on ZTE and Huawei versus TikTok? The FCC has labeled all three an "unacceptable risk". Also, I just saw that there is a bipartisan bill that "would end all commercial operations of TikTok in the U.S. and other social media platforms that are *sufficiently controlled or influenced by America's foreign adversaries, including China, Russia, and Iran." It'll be interesting to see where this goes ...*

I wanted to include Alain's Tweet to give me the opportunity to note that last Wednesday the entire U.S. senate voted **unanimously**, passing a bill which would bar the installation of TikTok from and government-owned devices. So, whereas a handful of Republican governors and an attorney general may have been first during the previous week or two, now we have unanimous and obviously completely bipartisan agreement on this. Wow!

But to Alain's question, I do regard these selective bans, such as on ZTE and Huawei and even on TikTok as mostly ridiculous theater because we are so intimately, deeply and inexorably enmeshed with Chinese technology products. I look around and everything in my house, all of the electronics that I own and the electronics in what I drive, was fabricated in China. All of it. And I'm sure that's the case for all of the people listening to this podcast. And speaking of listening to this podcast, this podcast was literally brought to your ears thanks to networking chips and processors and transistors made in China by Chinese citizens. And much of it was designed there.

So none of this posturing and saber rattling makes any sense to me. It must be that some sort of geo-political Kabuki is transpiring at a level that's far above my pay grade. I'm just a simple technologist who does understand networking and processors and transistors. So I know that if China did actually want to be evil, the West would be in deep doo-doo because in the interest of economy we've allowed ourselves to become utterly dependent upon products which we need from China. I don't want that to be a bad thing. I hope it's never going to be a bad thing. But if it is going to be a bad thing then the problem is way bigger than a couple of wayward Chinese companies.

David Ruggles / @TheRealRuggles

Reaching out RE the Zero Width Space mentioned in Security Now episode #901: I use it to fix stupid programs. For example if you want to reference an account on twitter without tagging them, enter <at sign><zero width space>account and it won't get tagged. (example: @TheRealRuggles vs @TheRealRuggles) Similarly in Excel it defaults to adding a hyperlink when you enter anything that looks like a url or email address; you can use the Zero Width Space to prevent that behavior without changing the look of the text.

That's cool. And I can see the many applications for that. So that leaves the question... how do we enter a zero-width space through the keyboard? I asked the Google and was told: "The zero width space is Unicode character U+200B. (HTML &#8203;). It's remarkably hard to type. On Windows you can type Alt-8203." Hmmm. ♂ That didn't work.

SpinRite

SpinRite is looking quite good. By the end of this past weekend with its 8th alpha release, every known weird data recovery behavior that we'd been seeing was resolved and SpinRite was now cruising through even the most damaged and troubled drives. While my focus was on getting SpinRite to properly perform its primary functions, I had also been accumulating a list of less critical but still necessary “to do” list items. And as our testers were using SpinRite a lot, some wishlist ideas were also presented.

So at the end of the day, Sunday, I told the group that I would be retrenching and disappearing while I worked my way through everything that was on the wish list. After today’s podcast, that’s what I’ll be doing. When I return with alpha release 9 it should be very close to finished. I’m sure there will be a few odds and ends, but that’s the nature of such a complex project.

I’ll note with some pride and relief that everyone who has been testing v6.1 has been very impressed by this new SpinRite’s speed and capabilities.

A Generic WAF Bypass

“ {JS-ON: Security-OFF}: Abusing JSON-Based SQL to Bypass WAF ”

As an industry, we’ve matured to the point where vulnerabilities are being discovered only in specific implementations of some solution, and generally only in specific versions of those implementations. In other words, once upon a time the entire industry would realize that an established standard could be abused in an unexpected way and everyone’s implementation would need to be changed. A perfect example was everyone’s DNS servers emitting queries from ports sequentially assigned by their underlying operating system and often giving those queries sequential identifiers. When it came to light that this would allow for successful DNS spoofing at scale, the entire industry repaired DNS overnight.

These events stand out because, thankfully, they’ve become so rare. These days, problems have generally become much more obscure and specific. For example, it might be that if you’re still using the out-of-support version 2.029.472 of JimmyCrack’s Query Reflector you need to update it to at least version 2.426.327 in order to avoid problems with query reflection back flush... and you should do so immediately.

Today’s rarity of big generic protection bypasses has made their existence extremely interesting. And a group known as Team82 recently discovered just such an industry wide mistake. They discovered an attack technique that acts as the first generic bypass of multiple web application firewalls being sold by industry-leading vendors including at least Palo Alto Networks, F5, Amazon Web Services, Cloudflare, and Imperva.

Before we proceed, we need to briefly revisit another one of those “Holy Crap!” events which hit the entire industry many years ago, and which, due to its difficulty it continues to grapple with: SQL injection.

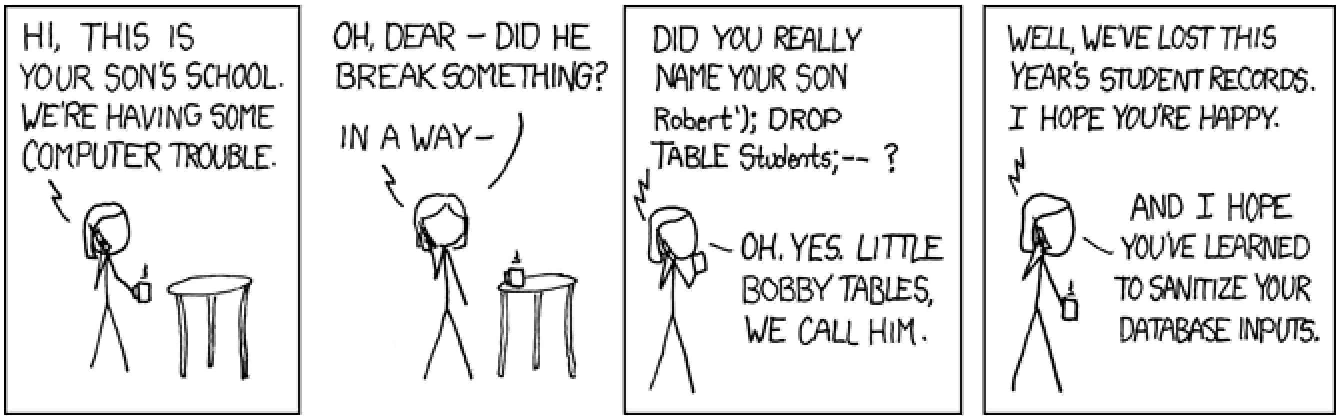
Stated succinctly, SQL injection can occur when there is some way for user-provided input to be passed to a SQL database for its interpretation. A SQL database is driven by strings of characters which express commands and queries. Simply by typing commands, new database tables can be created, they can be populated with data, queried for their data, and deleted when they are no longer needed. New users can be instantiated, passwords can be changed, privileges can be granted – all through simple text commands. And further increasing the system's power, the simplicity of this interface allows SQL databases to be queried over networks. The simplicity and power of this interface explains SQL's success.

But the simplicity and power of this interface has also been at the heart of one of SQL's longest running vulnerabilities. Wikipedia tells us that the first known public discussion of SQL injection appeared around 1998 and cites an article in Phrack Magazine.

SQL injection has been the bugaboo of web applications from the start. The first web apps gleefully presented a form asking their user to please enter their full name to look up their record in the site's database. The designer of this form assumed that that's what anyone would do. So whatever string they provided as their name would be added into a SQL query string to access the site's database. And all was well until it occurred to some clever individual that the website had inadvertently given them direct access to that site's SQL database front-end. Rather than simply inputting their name, they could, for example, input a string which closed the open query and started another entirely separate command of their choosing. This allowed a remote visitor to directly issue SQL commands to the site's database. If the web app designer had assumed that no one else could access the database – which is of course what they assumed – the SQL account behind the website's form might have administrative rights. This would allow remote visitors to do anything they might wish.

This has been such a common and persistent problem because the fundamental architecture of this system is fragile. It is not inherently secure and resilient, it is inherently insecure. We need to take user-supplied input, like some personal details, and embed them into a database query so that we can look up their record. We have to do that. The trouble with SQL is its power. That same query channel is also SQL's command and control channel.

This has been such a longstanding and well understood problem that it found its way into one of XKCD's brilliant comics:


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

https://xkcd.com/327

The biggest problem is, through all of these years, since it was first understood near the birth of the web, no one has fixed this. Instead, we just keep patching it. We focus upon each mistake in isolation rather than recognizing that the entire architecture is wrong for this application. SQL was not created for the web. It was first designed in the early 1970's at IBM before there was an Internet or websites or web apps. Unfortunately, the web found it, and it's been a troubled marriage ever since.

The problem is, every newly created web app creates another new opportunity to make a mistake in the parsing of user-supplied input that would give a remote attacker access to the site's backend database. That's why I say that the systems we've built around this architecture are inherently brittle and fragile. That's why, still today, SQL injection attack scans are constantly sweeping the Internet looking for that newly created newly vulnerable web application, and SQL injection remains at the top of the OWASP Top 10 list of web application vulnerabilities.

So what do we do if there's no sign that we're going to fix the underlying problem?

The universal solution to protecting our networks from external hostility is to place a firewall in front of those networks and force all external traffic to be inspected and to pass through that gauntlett before it's permitted to reach our interior networks. And thus was born the idea of a web application firewall, or WAF for short.

The fundamental concept of a web application firewall is detailed traffic inspection. Whereas packet-level firewalls generally look no deeper than packet headers specifying source and destination IPs and ports for the purpose of monitoring packet flows, a web application firewall examines in detail, the content of all web application traffic transiting its boundary in order to detect and block malicious attacks.

So, in XKCD's example above, a WAF would spot and block a form's input field data that contained suspicious characters for a user's name such as close parenthesis and semi-colons. With a web application firewall positioned upstream of an organization's web application servers, that malicious data and intent would never reach any web applications that might not be adequately providing for their own protection.

So with this background, here's what Team82 had to say about their recent discovery:

Web application firewalls (WAF) are designed to safeguard web-based applications and APIs from malicious external HTTPs traffic, most notably cross-site scripting and SQL injection attacks that just don't seem to drop off the security radar.

While recognized and relatively simple to remedy, SQL injection in particular is a constant among the output of automated code scans, and a regular feature on industry lists of top vulnerabilities, including the OWASP Top 10.

The introduction of WAFs in the early 2000s was largely a counter to these coding errors. WAFs are now a key line of defense in securing organizational information stored in a database that can be reached through a web application. WAFs are also increasingly used to protect cloud-based management platforms that oversee connected embedded devices such as routers and access points.

An attacker able to bypass the traffic scanning and blocking capabilities of WAFs often has a direct line to sensitive business and customer information. Such bypasses, thankfully, have been infrequent, and one-offs targeting a particular vendor's implementation.

Today, Team82 introduces an attack technique that acts as the ***first generic bypass*** *of multiple web application firewalls sold by industry-leading vendors. Our bypass works on WAFs sold by five leading vendors: Palo Alto Networks, F5, Amazon Web Services, Cloudflare, and Imperva. All of the affected vendors acknowledged Team82's disclosure and implemented fixes to their products' SQL inspection processes.*

Our technique relies first on understanding how WAFs identify and flag SQL syntax as malicious, and then finding SQL syntax the WAF is blind to.

This turned out to be JSON – JavaScript Object Notation.

JSON is a standard file and data exchange format, and is commonly used when data is sent from a server to a web application.

JSON support was introduced in SQL databases going back almost 10 years. Modern database engines today support JSON syntax by default including basic searches and modifications, as well as a range of JSON functions and operators. While JSON support is the norm among database engines, the same cannot be said for WAFs. Vendors have been slow to add JSON support, which allowed us to craft new SQL injection payloads that include JSON and that completely bypassed the security WAFs provide.

Attackers using this novel technique could access a backend database and use additional vulnerabilities and exploits to exfiltrate information via either direct access to the server or over the cloud.

This is especially important for OT and IoT platforms that have moved to cloud-based management and monitoring systems. WAFs offer a promise of additional security from the cloud; an attacker able to bypass these protections has expansive access to systems.

Okay, so what happened.

History has shown that no one is able to always get SQL injection protection correct because it's so much easier for it not to be correct. So, the notion of a web application firewall is created to move the burden from individual input forms, fields and web applications to the perimeter, where a single comprehensive web application firewall will be able to protect all of an organization's applications at once.

That happened about 20 years ago in the early 2000's.

The problem, of course, is that now it's less imperative for those individual web applications, which are now safely ensconced behind their protective application barrier, to be quite so worried about their own input form field content. After all, there's a big mean web application firewall at the front gate that's going to keep little Bobby Drop Tables safely out of reach.

So all is well. But then a decade passes, and a particular syntax for describing the features and details of objects becomes popular. It outgrows its own modest origins and is adopted by other languages and applications because it does the one thing it was designed to do cleanly and efficiently. And so, the JavaScript Object Notation, JSON, grows increasingly prevalent. Perhaps it was inevitable that SQL databases would eventually choose to add their own support for JSON.

Here's what Team82 has to say about that:

JSON in SQL

In modern times, JSON has become one of the predominant forms of data storage and transfer. ***In order to support JSON syntax and allow developers to interact with data in similar ways to how they interact with it in other applications, JSON support was needed in SQL.***

Currently, all major relational database engines support native JSON syntax; this includes Microsoft SQL, PostgreSQL, SQLite, and MySQL. Furthemore, in the latest versions, all database engines enable JSON syntax by default, meaning it is prevalent in most database setups today.

Developers have chosen to use JSON features within SQL databases since it became available for a number of reasons, starting with better performance and efficiency. Since many backends already work with JSON data, performing all data manipulation and transition on the SQL engine itself reduces the number of database calls needed. Furthermore, if the database can work with the JSON data format, which the backend API most likely uses as well, less data preprocessing and postprocessing is required, allowing the application to use it immediately without the need to convert it first.

By using JSON in SQL, an application can fetch data, combine multiple sources from within the database, perform data modification and transform it to JSON format—all within the SQL API. Then, the application can receive the JSON-formatted data and work with it immediately, without processing the data as well.

While each database chose a different implementation and JSON parser, each supports a different range of JSON functions and operators. Also, they all support the JSON data type and basic JSON searches and modifications.

And here's the key underlying what Team82 discovered:

Even though all database engines added support for JSON, not all security tools added support for this comparatively new, though decade old, feature – which was added as early as 2012. This lack of support in security tools introduced a mismatch in parsing primitives between the security tool (in our case, the WAF) and the actual database engines, and caused SQL syntax misidentification.

From our understanding of how a WAF could flag requests as malicious, we concluded that we needed to find SQL syntax the WAF would not understand. If we could supply a SQL payload that the WAF would not recognize as valid SQL, but the database engine would parse, we could actually achieve the bypass.

As it turns out, JSON was exactly this mismatch between the WAF's parser and the database engine. When we passed valid SQL statements that used the less prevalent JSON syntax, the WAFs did not flag requests as malicious.

The JSON operator "@>" which checks whether the right JSON is contained in the left one, threw the WAFs into loops and allowed us to supply malicious SQL payloads and allowed us to bypass the WAFs. By simply prepending simple JSON syntax to the start of the request, we were able to exfiltrate sensitive information over the cloud!

So this forms a very interesting story.

We start with a fundamentally insecure design when a powerful database system from the 70's which was never designed to allow malicious users to access its command input stream is used as the backend database for websites, thus inadvertently giving malicious users access to its command input stream.

Rather than recognizing that using SQL in this way is fundamentally a horrific mistake, every individual website must patch their input field parsers in an attempt to prevent SQL command and query syntax from being submitted by the visitors to every site.

SQL injection becomes a meme, and XKCD captures its essence.

In an extension of the firewall concept, web application firewalls are created to centralize and concentrate the SQL syntax filtering challenge. And all seems fine for some time.

Then, SQL syntax undergoes a fundamental extension as all SQL servers implement support for the increasingly popular JavaScript Object Notation.

But despite this extension, most of the industry's web application firewalls fail to update their protection logic to incorporate an awareness that JSON can be used to encapsulate and issue SQL queries.

Fortunately, a team of white hat security researchers stumble upon this tidbit while they're working to discover just such a bypass and they quietly inform the many vendors of those vulnerable web application firewalls of their discovery.

And all is well again. Or is it?

Because SQL is still powering virtually all web applications and the fundamental problem of now an even more powerful SQL syntax existing, still remains.

If JSON could be used to slip past web application firewalls to reach the SQL database behind, how many websites that are not being protected by a big iron web application firewall might be vulnerable RIGHT NOW to exactly the same JSON bypass?

Happy New Year.