Security Now! #944 - 10-17-23
Abusing HTTP/2 Rapid Reset

This week on Security Now!

How have valiDrive's first ten days of life been going and what more have we learned about the world of fraudulently fake USB thumb drives? Should passkeys be readily exportable or are they better off being kept hidden and inaccessible? Why can't a web browser be written from scratch? Can Security Now listeners have SpinRite v6.1 early?... like... now? What was that app for filling a drive with crypto noise and what's my favorite iOS OPT app? And couldn't Google Docs HTML exported links being redirected for user privacy? After we address those terrific questions posed by our listeners we're going to take a look at the surprise emergence of a potent new HTTP/2-specific DDoS attack. Is it exploiting a 0-day vulnerability as Cloudflare claims, or is that just deflection?

Slavish devotion to the specifications...

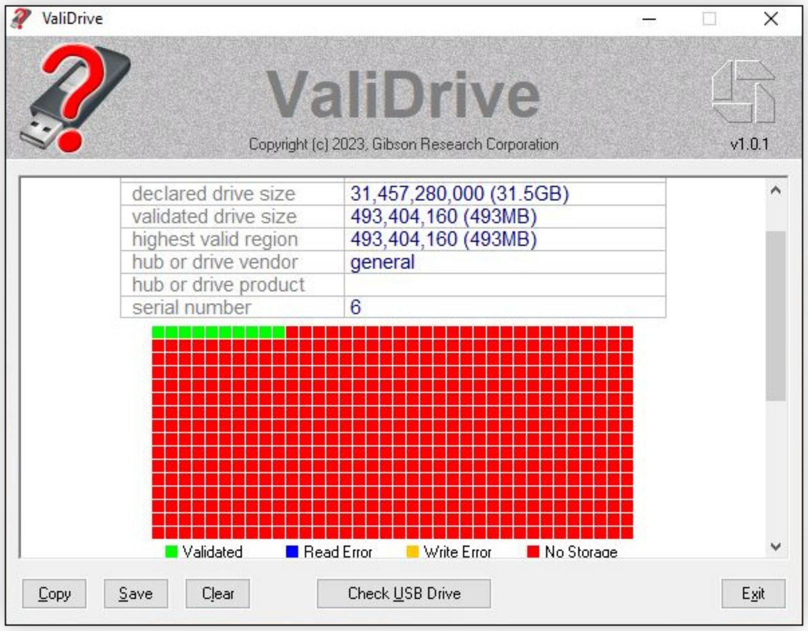
ValiDrive release follow-up

I wanted to follow-up a bit on ValiDrive. We're ten days out from its release and we're currently seeing a 7-day moving average of 2,443 downloads per day with 27,603 downloads in total. That download rate currently places it at the top among GRC's freeware goodies. It has pushed GRC's DNS Benchmark, which is currently being downloaded 1,867 times per day to the number 2 spot, though that bit of freeware has demonstrated somewhat astonishing endurance. With more than 8.5 million downloads since its release I think it's fair to say that GRC's DNS Benchmark has become a staple on the Internet. I expect that ValiDrive may have similar long- term legs, though probably at a lesser average rate once the news of its existence settles down. It'll be interesting to see.

Not surprisingly, I've received some interesting feedback relating to ValiDrive since it's announcement release 10 days ago:

qqq1 / @qqq1

I knew these would be trash. "free gifts" with stuff we order. We have a large pile of them. #ValiDrive @SGgrc



declared drive size	31,457,280,000 (31.5GB)
validated drive size	493,404,160 (493MB)
highest valid region	493,404,160 (493MB)
hub or drive vendor	general
hub or drive product	
serial number	6

This is exactly typical of what we've been seeing. ValiDrive shows a drive claiming to be 32GB. But it only contains 512MB of non-volatile storage. And it's a free gift drive. So why lie about its size? It was free! And again, to be clear, it's not just the label on the outside of the physical drive that says 32GB. It's that the drive's electronics is proactively lying about the amount of storage it contains. So as I originally noted six weeks ago, this creates a little disaster in a box, since without ValiDrive its user would have no reasonable way to detect this lie until they tried to read any file or files that were written out past the drive's first 512 megabytes. The FAT file system directory will be sitting safely at the front of the drive to enforce the illusion that everything is fine while nothing is being stored. So this cannot be a mistake. This can only be a deliberate fraud. The question is, a fraud perpetrated upon whom?

In thinking about this further, in the case of such freebie giveaway drives, it occurs to me that the **first** target of this fraud is **not** those of us who receive these freebies. Though we are victims, we're secondary victims. The primary victim is the entity from whom we received these gifts. In the instance above, for example, **they** believed that they were purchasing 32GB drives – doubtless at a bargain quantity price – to be used as appealing 32GB giveaway drives. So the fraud was perpetrated upon **them**. They believed that they were buying 32GB drives when in fact they were purchasing far less appealing 512MB drives.

So our takeaway is to be very cautious about the use of any drives you buy for a bargain at the check-out stand or may have ever received as a thoughtful thank you gift thrown in with something else. I'd say that it's safe to assume that the entity offering the bargain was likely completely unaware that they had been had, and that they were distributing these fraudulent drives in good faith.

Alain / @Alain_Gyger

Hi Steve, thank you for your work on ValiDrive, I'd really like to test a few USB sticks with it but I only have access to Linux systems. Is it possible to get this running on Ubuntu or some other distro?

Alain, I understand. The DNS Benchmark I created is deliberately WINE compatible. I spent some time making sure it would run under WINE so Linux users could take advantage of it. But ValiDrive's UI, which uses dynamic USB insert and removal events to determine which drive the user wishes to test uses features that are outside of WINE's Windows compatibility set. If it was crucial for ValiDrive to run under WINE I could probably arrange to somehow make that happen. But I've already spent more time on it than I planned to, and I'm champing at the bit to get back to finishing SpinRite. If ValiDrive turns out to be seeing strong long-term use I would be inclined to return to it and give it a significant v2.0 update. But for now it's on to finishing SpinRite v6.1.

Closing the Loop

Marko Simo / @markos@twit.social

Just visiting X to contact you 😊: Regarding passkeys and their lack of "easy exportability, QR codes" etc, have you ever thought that those exact features would make passkeys vulnerable for phishing and other social engineering attacks? They are hidden from normal users for a very good reason. Someone should now figure out what could be a phishing resistant way to move them across ecosystems. Or we all should start using SQRL 😉

I agree completely with Marko's observation. It is definitely a double-edged sword. I've actually been somewhat heartened by the lack of uptake of passkeys, not because I don't want the world to have a public key based network authentication system, but because the reason my SQRL system has gone nowhere wasn't just me. Of course, that **is** the reason it's **never** going to go anywhere. But at least passkeys hasn't yet taken the world by storm... or I would have at least expected something more from SQRL. And I used the word "yet" since I fully expect that this FIDO2 passkeys public key system will someday supplant secret passwords. But probably not in my lifetime. Seriously. All of the evidence suggests that implementing this sort of sweeping change really will be that slow. But it needs to eventually happen and I think it will... even though only this podcast's younger listeners may be around to remember these words.

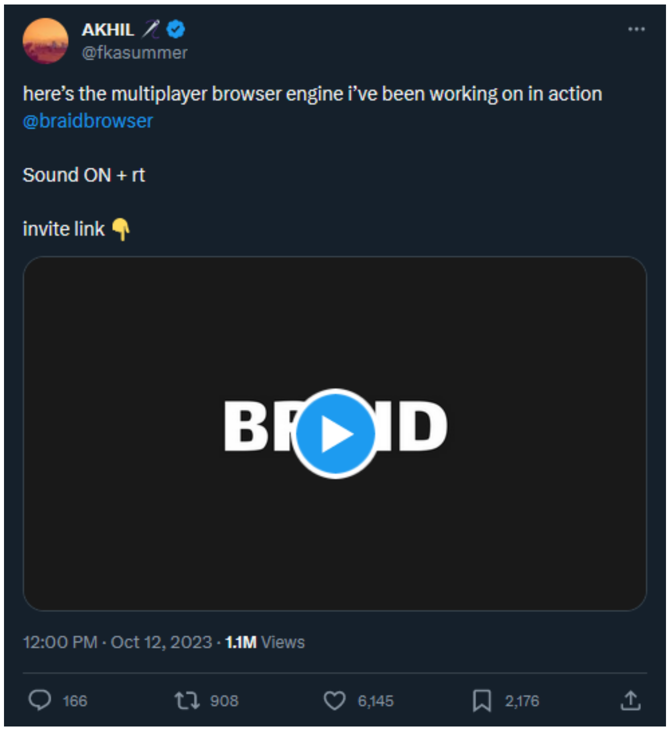
michael moloney / @fantaguy

Aren't passkeys meant to be a device verification like SSH keys? While you could copy them between devices, best security is generate them on the device you need to authorise, so if that device gets stolen or hacked, you can revoke the key for that device...also if the device is hacked logs are way more useful if it shows Paul's iPhone rather than Paul's key and then he has to redo EVERY passkey device.... Aside from backups, I hope passkeys stay non transferable. When you put your pin or fingerprint in your phone, you are basically "unlocking all the sites this passkey has access to" same as when you sign into your PC with you password that has all your SSH keys.

So that's true and I think it's a good point. This would be roughly equivalent to having multiple separate username/password pairs for logging into one's bank account with a different username/password pair stored in each device. Then, if a device was compromised the usernames and passwords that were being used by the compromised device could be independently disavowed and disabled. While that's good in theory, that really raises the level of management complexity to a new level. There's a lot of power in that approach, but also a huge amount of responsibility for keeping track of who's on first.

Brian M. Franklin / @brianmfranklin

@SGgrc: Maybe it IS still possible to write a new browser — with even better features.



AKHIL
@fkasummer

here's the multiplayer browser engine i've been working on in action @braidbrowser

Sound ON + rt

invite link 👇

12:00 PM · Oct 12, 2023 · 1.1M Views

166 908 6,145 2,176

There's browsers and then there's browsers. I love the idea of the creation of hobby browsers.

It's a modern take on the idea of a hobby operating system. A bunch of hobby operating systems have been created and, in fact, I carefully considered basing SpinRite's future upon a few of them. But lack of support for booting on UEFI-only systems ultimately led to me the embedded RTOS-32 OS.

Similarly, while I think it's possible to create a functional outline of a modern browser, the task has grown to a size like that of creating an operating system. So, yeah... there's hobby browsers and then there's production-scale Chromium, Firefox and Safari browsers.

If anyone's interested in this multi-user browser, it's at: https://braidbrowser.com/

Techgifthorse / @techgifthorse

Hi Steve, long-time listener, first time DM'er! Regarding episode 943 where you introduced that google was embedding tracking links in google docs exports, I recently noticed that google is also doing this in calendar invites. I am not sure if they were trialing it in google docs html exports, and now it is expanding, but I thought it was note-worthy.

I also wanted to say that you (and Leo) have inspired me to start my Master's in Cybersecurity, so please keep the pods coming and keep up the great work! Can't wait for your forthcoming email option so I can delete this bogus Twitter account. Sincerely, Sean

Just as soon as I have SpinRite v6.1 ready for roll-out I'll let everyone here know, of course. That's the super-easy communication path. Then I'll begin work on bringing up a new eMail facility so that I'm able to get the word out to all of v6.0's current owners.

Brett Russell / @brussellZA

Hi Steven, I realize this is a request out of the blue, but my hard disk is failing on my main machine. I own a copy of Spinrite (code ##REDACTED##), but the disk is set up as GPT, which 6.0 doesn't support. Any chance you can give me a pre-release of 6.1 to run pls. There is nothing critical on the drive, but if it dies, it will take some time to bring it all back.

Of course, Brett! – and this goes for everyone: Just go to https://www.grc.com/prerelease.htm and enter your current SpinRite v6.0 serial number which is presented when SpinRite is run. I just verified that Brett's code works perfectly. You'll receive a link to download your own personalized and licensed copy of the latest pre-release of v6.1. What you'll get at this point is the DOS executable. So just place it onto your SpinRite boot drive and let'er rip! The current pre- release expires at the end of November, so that's 6 weeks from now and I'm sure we'll have updated releases well before then. And just retarding the machine's date will buy more time if needed.

At this point I'm 100% certain that SpinRite is safe. The reason is that I just checked and we have exactly 800 registered SpinRite pre-release testers who have been using and pounding on all of the SpinRite pre-releases. I've taken great pains to be very careful along the way – no one has ever experienced any data loss. This thing is ready for the world. I just need to tie up a bunch of loose ends which any three-year project of this complexity will naturally have.

Ray Franklin #PresentationCoachRay / @StageAmerica

Steve, I've looked through the current show notes for the reference: wipe drive with encryption....? What is the program to effectively wipe a drive with random data?

It's VeraCrypt: https://www.veracrypt.fr/code/VeraCrypt/

Marcus / @BadAssBG

Hi Steve. I just had a quick question. I herd you say that you do not use Authy for 2fa and I was wanting to know what you do use? Thanks.

Being an Apple ecosystem person for mobile so I use an app called **OTP Auth** and I love it. It's clean, simple and is widely praised, not just here:
https://apps.apple.com/us/app/otp-auth/id659877384

Henning / @ihbrune

Dear Steve, when listening to SN-943 I have a different explanation why Google is rewriting links in Google Drive documents published as HTML pages: Prevent the leakage of the document url through referer information. Though the drive documents are public, they are hidden by their long, obscure url. But the referer info can reveal the url in the server logs of the embedded links. The additional step through a second google service hides this information.

First of all, that's a neat thought. But these links were present in offline HTML exports. So wherever the HTML was being served from would be appearing in the Referrer header. Also, if this was a favor Google was doing for us to protect our privacy then their referring link URLs would not have been loaded down with unique tracking parameters. So, much as I love the idea that they might be protecting us, that doesn't appear to be their intent here.

Abusing HTTP/2 Rapid Reset

As I said last week, we would be continuing with the second part of our look into the Top 10 cybersecurity misconfigurations if something bigger and more important didn't bump it to the following week. Well, something did indeed.

The headline about this which first caught my attention was Cloudflare's which read: *"HTTP/2 Zero-Day vulnerability results in record-breaking DDoS attacks"*. So, yeah, Wow! The idea of their being a newly discovered 0-day vulnerability in a core Internet protocol such as HTTP/2 would be HUGE NEWS... if it were true. For the rest of this podcast we're going to explore what has happened and decide to what degree this is actually true. As always, the devil is in the details and these details are both interesting and quite important.

A few lines into their coverage of this they write: *"Cloudflare has mitigated a barrage of these attacks in recent months…"* My first reaction was to question their use of the term *"0-day."* But after wondering whether it might be an abuse or an overuse of the term, I suppose it's correct if they first learned of this unsuspected problem through an attack. We've settled upon the definition of *"0-day"* as being the exploitation of any previously unknown vulnerability. Or stated another way, the first indication we have of there being some specific vulnerability, is not when we find it through examination or when it's reported by a responsible researcher, but when we're surprised by something happening that we didn't previously believe to be possible. And so, yeah, it would probably be correct for this to be considered a 0-day vulnerability in the version 2 HTTP protocol ... if, indeed, it **was** a vulnerability in the HTTP/2 protocol, which, surprisingly, is not the case.

It is to Cloudflare's benefit to characterize this as a surprising 0-day vulnerability in HTTP/2. That deflects the blame to the protocol when it appears that the true culprit is implementation. The HTTP/2 protocol is not to blame because there's a difference between a vulnerability and the abuse of a feature that's exacerbated by Cloudflare's internally decoupled serial processing multi-proxy architecture. As I think the evidence will show, this is more about something that Cloudflare's behind the scenes request-processing architecture was ill suited to handle, than the exploitation of any previously unknown vulnerability in HTTP/2.

It's a clever and in retrospect foreseeable and probably inevitable leveraging of a feature in HTTP/2. So I don't mean to jump on Cloudflare too much about this. But characterizing this as a zero-day vulnerability in HTTP/2 is really not accurate or fair.

So let's start with a quick background review of layered network design, and then a quick history of HTTP. We refer to HTTP – the Hyper-Text Transfer protocol – as an application level protocol (also known as a layer 7 protocol) because after we've finally finished building up layer upon layer of the various underlying enabling protocols, we finally get to HTTP, which was the whole point; and it's there where something finally happens and gets done.

What do we mean by "layer upon layer"? At the bottom-most level or layer we have an agreement about wires and voltages and clock rates to signal the transmission of 1's and 0's. That's typically referred to as the physical layer. Then, with that agreed upon, we describe the groupings of those 1's and 0's into packets of data which are addressed between physical hardware endpoints on a single local network; this is the Ethernet protocol. And those Ethernet packets, in turn, contain IP (Internet Protocol) packets which carry the addresses of endpoints on the global Internet. And inside those IP packets are TCP packets where the TCP protocol is used to manage the numbered, sequential byte-oriented reliable flow of data between specific virtual ports on the devices specified by their IP addresses. And finally, the data that flows back and forth under the watchful eye of the TCP protocol is formatted as specific queries and responses in accordance with the top-level application protocol, HTTP.

Over the decades since their initial definitions, all of these protocols have seen some tuning, tweaking and evolution. And that's certainly been true for HTTP. With a bit of editing by me for clarification, here's how Wikipedia briefly sums up HTTP's evolution:

Development of HTTP was initiated by Tim Berners-Lee at CERN in 1989 and summarized in a simple document describing the behavior of a client and a server using the first HTTP version, named 0.9. That version was subsequently developed, eventually becoming the public 1.0.

Development of early HTTP RFCs began a few years later in a coordinated effort by the IETF (the Internet Engineering Task Force) and the W3C (World Wide Web Consortium), with work later moving exclusively to the IETF.

HTTP/1 (it's shown as /1 rather than v1 because HTTP/1 is what's sent over the wire with each query to specify the format of everything that will follow.) So version 1.0 was finalized and fully documented in 1996. It quickly evolved to version 1.1 a year later in 1997, after which its specifications were updated in 1999, 2014, and even last year in 2022. Its secure variant, HTTPS is used by more than 80% of websites.

HTTP/2, published in 2015, provides a more efficient expression of HTTP's semantics "on the wire". As of April 2023, HTTP/2 is used by 39% of websites and supported by almost all web browsers – representing over 97% of web users. HTTP/2 is also supported by major web servers over Transport Layer Security (TLS) using an Application-Layer Protocol Negotiation (ALPN) extension where TLS 1.2 or newer is required.

And, finally, dropping TCP in favor of the faster-to-setup yet less fasture-rich UDP we have HTTP/3. This successor to HTTP/2, was published last year in 2022. It's now available from over one quarter of all websites and it's at least partially supported by most web browsers. As I mentioned, unlike all of its TCP-based predecessors, HTTP/3 does not run on top of TCP. Instead it uses QUIC which runs on top of UDP. Support for HTTP/3 was added to Cloudflare and Google Chrome first, and is also enabled in Firefox. HTTP/3 has lower latency for today's web pages which are pulling information together through many separate widespread sources. If HTTP/3 is enabled on the server, such pages will load faster than with HTTP/2, which is, in turn, faster than HTTP/1.1. In some cases as much as three times faster than HTTP/1.1.

Okay. So now that we have the lay of the land, let's look at what took Cloudflare and others by surprise a few months back. There's a bit of marketing/bragging going on here, but mostly a lot of good information, and I'll insert some editorializing as we go. Cloudflare wrote:

Earlier today [this was just posted last Tuesday], Cloudflare, along with Google and Amazon AWS, disclosed the existence of a novel zero-day vulnerability dubbed the "HTTP/2 Rapid

Reset" attack. ***This attack exploits a weakness in the HTTP/2 protocol to generate enormous, hyper-volumetric Distributed Denial of Service (DDoS) attacks.*** *Cloudflare has mitigated a barrage of these attacks in recent months, including an attack three times larger than any previous attack we've ever observed, which exceeded 201 million requests per second (rps). Since the end of August 2023, Cloudflare has mitigated more than 1,100 other attacks with over 10 million rps — and 184 attacks that were greater than our previous DDoS record of 71 million rps.*

I'll pause here to note that there are several different ways to measure or characterize DDoS attacks. Traditionally, attacks leveraged bandwidth flooding and saturation. The incoming network links to a server would be so saturated with packet traffic that a site's upstream routers would be so overwhelmed with bogus packets that legitimate packet traffic stood little chance to get through to the server.

But as we know, some time ago web pages shifted from being simple static dumps of a previously written textual page (for example, as my own GRC site remains to this day), to being the public-facing aspect of a content management system (a CMS) of some sort. Under this new website paradigm, any incoming HTTP query is fielded by some sort of code running on the server. That code examines the query details and assembles the page to be returned to the client on the fly. And this, in turn, typically involves multiple queries to a back-end relational database of some kind – a SQL server.

The point of this is that under this new content management paradigm, replying to a query for a single page consumes server-side compute resources in order to pull the various components of the page together for its return to the client. The other thing is that in the earliest days of this new approach, these on-the-fly page-assembly systems were often not very efficient. So each query, especially deliberately complex queries, could be quite costly to answer. Naturally, it didn't take the bad guys long to figure this out, after which they switched their attack tactics from flooding a server's raw bandwidth pipeline with what could well just be noise, to loading the server down with actual valid queries, each of which could be quite expensive for it to answer. Thus we went from thinking of DDoS attacks in terms of bits per second to requests per second.

I take issue with their characterization of this as an attack which *"exploits a weakness in the HTTP/2 protocol"*. That's not what has been happening. As we'll see, these new attacks exploit some new features of HTTP/2 which were added to make the protocol much faster overall. But it turns out that these new features are subject to abuse, and if a server's query-processing chain is not designed to handle these new features it can be forced into collapse by attackers. This is where we'd say "it's not a bug, it's a feature" – and we'd mean it.

Cloudflare continues...

This zero-day provided threat actors with a critical new tool in their Swiss Army knife of vulnerabilities to exploit and attack their victims, at a magnitude that has never been seen before. While at times complex and challenging to combat, these attacks allowed Cloudflare the opportunity to develop purpose-built technology to mitigate the effects of the zero-day vulnerability.

Again, as we'll see and as I'll shortly provide ample evidence for back up, this is not really a vulnerability and I'm disappointed in Cloudflare's approach to this. We all know that I'm a big fan of Cloudflare, but I believe they took the wrong tack on this one.

In late August of this year [Cloudflare writes] *our team at Cloudflare noticed a new zero-day vulnerability, developed by an unknown threat actor, that exploits the standard HTTP/2 protocol — a fundamental protocol that is critical to how the Internet and all websites work. This novel zero-day vulnerability attack, dubbed Rapid Reset, leverages HTTP/2's stream cancellation feature by sending a request then immediately canceling it – over and over.*

By automating this trivial "request, cancel, request, cancel" pattern at scale, threat actors are able to create a denial of service and take down any server or application running the standard implementation of HTTP/2. Furthermore, one crucial thing to note about the record-breaking attack is that it involved a modestly-sized botnet, consisting of roughly 20,000 machines. Cloudflare regularly detects botnets that are orders of magnitude larger than this — comprising hundreds of thousands and even millions of machines. For a relatively small botnet to output such a large volume of requests, with the potential to incapacitate nearly any server or application supporting HTTP/2, underscores how menacing this vulnerability is for unprotected networks.

I want to pause again to make three points: First, I hope it still astonishes us when Cloudflare writes: *"Cloudflare regularly detects botnets that are orders of magnitude larger than 20,000 — comprising hundreds of thousands and even millions of machines."* This kind of power in the hands of miscreants is fundamentally destabilizing. Because the Internet is so powerful and because it mostly works so well, we've allowed ourselves to gradually grow to become utterly dependent upon its presence. But last week we noted Microsoft was observing around 1700 DDoS attacks per day. So there are websites that are occasionally denied the sort of reliability that the Internet has to offer.

The second point I wanted to highlight was that Cloudflare witnessed by far the largest attack they have ever experienced which was produced by what is unfortunately now considered to be a relatively modest-sized Botnet, consisting of only around 20,000 individual clients. Since this sort of attack rides on TCP, the attacker's source IPs cannot be spoofed since TCP requires confirmation of packet round trips before the query can be sent. So Cloudflare has the specific IP of every bot that was attacking it.

And the last point is that since this was a new form of abuse to which HTTP/2 is prone, every one of these bots needed to be updated with a module specifically designed to implement this attack against its target. So someone, somewhere, figured this out, wrote the code to do this, then either established a new Botnet for this purpose or cause an existing updatable Botnet to be updated to add this new form of attack.

Threat actors used botnets in tandem with the HTTP/2 vulnerability to amplify requests at rates we have never seen before. As a result, our team at Cloudflare experienced some intermittent edge instability.

Uh huh... *"intermittent edge instability"* Right. In other words, "we're all about DDoS protection but this one got us because we had never seen anything like it and our back-end architecture was ill-equipped to handle it.

While our systems were able to mitigate the overwhelming majority of incoming attacks, the volume overloaded some components in our network, impacting a small number of customers' performance with intermittent 4xx and 5xx errors — all of which were quickly resolved.

Once we successfully mitigated these issues and halted potential attacks for all customers, our team immediately kicked off a responsible disclosure process. We entered into conversations with industry peers to see how we could work together to help move our mission forward and safeguard the large percentage of the Internet that relies on our network prior to releasing [the news of] *this vulnerability to the general public.*

There is no such thing as a "perfect disclosure." Thwarting attacks and responding to emerging incidents requires organizations and security teams to live by an assume-breach mindset — because there will always be another zero-day, new evolving threat actor groups, and never-before-seen novel attacks and techniques.

This "assume-breach" mindset is a key foundation towards information sharing and ensuring in instances such as this that the Internet remains safe. While Cloudflare was experiencing and mitigating these attacks, we were also working with industry partners to guarantee that the industry at-large could withstand this attack.

During the process of mitigating this attack, our Cloudflare team developed and purpose-built new technology to stop these DDoS attacks and further improve our own mitigations for this and other future attacks of massive scale. These efforts have significantly increased our overall mitigation capabilities and resiliency. If you are using Cloudflare, we are confident that you are protected.

Our team also alerted web server software partners who are developing patches to ensure this vulnerability cannot be exploited — check their websites for more information.

Disclosures are never one and done. The lifeblood of Cloudflare is to ensure a better Internet, which stems from instances such as these. When we have the opportunity to work with our industry partners and governments to ensure there are no widespread impacts on the Internet, we are doing our part in increasing the cyber resiliency of every organization no matter the size or vertical.

It may seem odd that Cloudflare was one of the first companies to witness these attacks. Why would threat actors attack a company that has some of the most robust defenses against DDoS attacks in the world?

The reality is that Cloudflare often sees attacks before they are turned on more vulnerable targets. Threat actors need to develop and test their tools before they deploy them in the wild. Threat actors who possess record-shattering attack methods can have an extremely difficult time testing and understanding how large and effective they are, because they don't have the infrastructure to absorb the attacks they are launching. Because of the transparency that we share on our network performance, and the measurements of attacks they could glean from our public performance charts, this threat actor was likely targeting us to understand the capabilities of the exploit.

But that testing, and the ability to see the attack early, helps us develop mitigations for the attack that benefit both our customers and industry as a whole. Even so, whether it was Log4J, Solarwinds, EternalBlue WannaCry/NotPetya, Heartbleed, or Shellshock, all of these security incidents have a commonality. A tremendous explosion that ripples across the world and creates an opportunity to completely disrupt any Internet-dependent organization, regardless of the industry or the size.

While we wish we could say that Rapid Reset may be different this time, it is not. We're calling all CSOs — no matter if you've lived through the decades of security incidents or this is your first day on the job — this is the time to ensure you are protected and stand up your cyber incident response team.

We've kept the information restricted until today to give as many security vendors as possible the opportunity to react. However, at some point, the responsible thing becomes to publicly disclose zero-day threats like this. Today is that day. That means that after today, threat actors will be largely aware of the HTTP/2 vulnerability; and it will inevitably become trivial to exploit and kickoff the race between defenders and attacks — first to patch vs. first to exploit. Organizations should assume that systems will be tested, and take proactive measures to ensure protection.

To me, this is reminiscent of a vulnerability like Log4J, due to the many variants that are emerging daily, and will continue to come to fruition in the weeks, months, and years to come. As more researchers and threat actors experiment with the vulnerability, we may find different variants with even shorter exploit cycles that contain even more advanced bypasses.

And just like Log4J, managing incidents like this isn't as simple as "run the patch, now you're done". You need to turn incident management, patching, and evolving your security protections into ongoing processes — because the patches for each variant of a vulnerability reduce your risk, but they don't eliminate it.

We don't mean to be alarmist, but we'll be direct: ***you must take this seriously****. Treat this as a full active incident to ensure nothing happens to your organization.*

While no one security event is ever identical to the next, there are lessons that can be learned. Here are our recommendations that must be implemented immediately. Not only in this instance, but for years to come:

- *Understand your external and partner network's external connectivity to remediate any Internet facing systems with the mitigations below.*
- *Understand your existing security protection and capabilities you have to protect, detect and respond to an attack and immediately remediate any issues you have in your network.*
- *Ensure your DDoS Protection resides outside of your data center because if the traffic gets to your datacenter, it will be difficult to mitigate the DDoS attack.*
- *Ensure you have DDoS protection for Applications (at network Layer 7) and ensure you have Web Application Firewalls. Additionally as a best practice, ensure you have complete DDoS protection for DNS, Network Traffic (down at Layer 3) and API Firewalls.*
- *Ensure web server and operating system patches are deployed across all Internet Facing Web Servers. Also, ensure all automation and images are fully patched so older versions of*

web servers are not deployed into production over the secure images by accident.

- *As a last resort, consider turning off HTTP/2 and HTTP/3 (which is likely also vulnerable) to mitigate the threat. This is a last resort only, because there will be a significant performance issues if you downgrade to HTTP/1.1.*

- *Consider a secondary, cloud-based DDoS L7 provider at perimeter for resilience.*

Of course, that last one is a bit self-serving since Cloudflare has just finished explaining that while they were initially caught off guard by this unanticipated and massive 0-day attack, they're now up to speed and are inviting everyone to come and hide behind their perimeter. On the other hand, they are offering it at no charge. They wrap up this overview by writing:

Cloudflare's mission is to help build a better Internet. If you are concerned with your current state of DDoS protection, we are more than happy to provide you with our DDoS capabilities and resilience ***for free*** *to mitigate any attempts of a successful DDoS attack. We know the stress that you are facing as we have fought off these attacks for the last 30 days and made our already best in class systems, even better.*

Even though this didn't give us any of the meaty technical details that this podcast never shies away from, I wanted to share it since it's what Cloudflare is telling the entire world... and sharing it here is important because I believe that it mischaracterizes HTTP/2 as having surprising and unsuspected vulnerabilities, which is not the case.

So now for some meaty details... **CVE-2023-44487**

https://www.cve.org/CVERecord?id=CVE-2023-44487

The CVE disclosure is a bit more fair, but even it contains a bit of spin and misdirection. The description of the CVE states: *"The HTTP/2 protocol allows a denial of service (server resource consumption) because request cancellation can reset many streams quickly, as exploited in the wild in August through October 2023."*

Google's summary of the problem is, I think, exactly correct. Google wrote:

Since late 2021, the majority of Layer 7 DDoS attacks we've observed across Google first-party services and Google Cloud projects protected by Cloud Armor have been based on HTTP/2, both by number of attacks and by peak request rates. A primary design goal of HTTP/2 was efficiency, and unfortunately ***the features that make HTTP/2 more efficient for legitimate clients can also be used to make DDoS attacks more efficient.***

That is precisely the truth. "The features that make HTTP/2 more efficient for legitimate clients can also be used to make DDoS attacks more efficient." That's it. Not any surprise 0-day vulnerability. Just the clever abuse of a new feature of HTTP/2 that anyone designing HTTP/2-capable servers will need to take into account. Until now, Cloudflare had not. They had been relying upon their conventional DDoS protections, which are doubtless very strong; but this new form of attack was too much even for that.

But that said, I do agree with the creation of a CVE and the raising of alerts about this since, indeed, **rapid HTTP/2 stream resets** promise to cause some havoc with any HTTP/2-capable web server that is not capable of handling these resets efficiently.

To get some sense of the scope of the problem, Cloudflare wrote:

Starting on Aug 25, 2023, we started to notice some unusually big HTTP attacks hitting many of our customers. These attacks were detected and mitigated by our automated DDoS system. It was not long however, before they started to reach record breaking sizes — and eventually peaked just above 201 million requests per second. This was nearly 3x bigger than our previous biggest attack on record.

Concerning is the fact that the attacker was able to generate such an attack with a botnet of merely 20,000 machines. There are botnets today that are made up of hundreds of thousands or millions of machines. Given that the entire web typically sees only between 1–3 billion requests per second, it's not inconceivable that using this method could focus the entire global web's worth of requests onto a small number of targets.

So, just to be clear since putting this into context is important, this attack by just 20,000 bots generated, on its own, a peak of 201 million requests per second. And the Internet across its entire expansive whole sees between 1 and 3 billion requests per second. And whereas this breathtaking attack was produced by only 20,000 bots working in concert, botnets consisting of hundreds of thousands to millions of devices are known to exist. And as Cloudflare wrote in their less technical overview, as of last Tuesday October 10th, every bad guy bot operator who may not have been in the loop until now, now knows about it – and it's a trivial attack to create. What's more, as Wikipedia told us, HTTP/2 is offered as an available protocol by around 39% of the Internet's web servers. We don't currently know what percentage of those web services poorly handle HTTP/2 stream resets. It could be that cloud providers inherently have a greater problem with this due to their more highly distributed architecture. A single stand-alone HTTP/2 server might now be much more affected by this than by an HTTP/1.1 request flood.

However, unless and until any vulnerable web servers are updated and/or moved behind some sort of perimeter protection, they're going to be in danger of exploitation from this novel server-side resource depletion attack. And what's significant is that this is true even from the great many more lesser bot operators who run much smaller botnets. Once the code to implement this attack has circulated through the underground – and as I noted above, it's trivial to do – and given how things are likely to proceed with web server upgrades that won't appear to be necessary until it's too late, I expect we'll be reporting on additional DDoS-based service outages shortly.

So, also dated last Tuesday, October 10th was Cloudflare's posting titled: *"HTTP/2 Rapid Reset: deconstructing the record-breaking attack"*

Their description begins:

This attack was made possible by abusing some features of the HTTP/2 protocol and server *implementation details (see CVE-2023-44487 for details). Because the attack abuses an underlying weakness in the HTTP/2 protocol, we believe any vendor that has implemented HTTP/2 will be subject to the attack. This includes every modern web server. We, along with Google and AWS, have disclosed the attack method to web server vendors who we expect will implement patches. In the meantime, the best defense is using a DDoS mitigation service like Cloudflare's in front of any web-facing web or API server.*

Again, characterizing this as the abuse of an underlying weakness is misguided because it is actually the abuse of HTTP/2's performance optimizing strength. And I would also argue that the easiest defense might be what Cloudflare reluctantly added in their overview piece, which was to simply disable HTTP/2 support until server updates are available and installed. It might be that a website depends upon the added speed of HTTP/2. But it might also be that a website is running HTTP/2 because it was enabled by default and that the site might be just as happy running HTTP/1.1. The point is, it may be that the only way to avoid this Armageddon is to hide behind a DDoS protection service. It might be that simply turning off a site's advertisement of its HTTP/2 support until the server has been upgraded would be sufficient.

So, what exactly is an HTTP/2 Stream Reset? I could write this up myself, but in the interest of saving some time so that I have time to assemble other interesting things this week, I'm going to switch to Google's explanation. I'm switching to Google because Cloudflare is unable to stop blaming this on HTTP/2 and calling it a 0-day vulnerability. Google explains:

HTTP/2 uses "streams", bidirectional abstractions used to transmit various messages, or "frames", between the endpoints. "Stream multiplexing" is the core HTTP/2 feature which allows higher utilization of each TCP connection. Streams are multiplexed in a way that can be tracked by both sides of the connection while only using one Layer 4 connection. Stream multiplexing enables clients to have multiple in-flight requests without managing multiple individual connections.

We talked about this a million years ago. With HTTP/1.1, a single TCP connection can be created to a server, and the client is able to send multiple queries to the server sequentially in a sort of pipeline. But the server must receive them in-order and reply to each query in-order so that the sequence of replies matches the sequence of queries. The big trouble with this is that many smaller and faster-to-answer replies could get held up behind an earlier query that takes the server more time. Perhaps it needs to query something else to obtain an answer. Meanwhile, some queries for simple images, for example, would be held up waiting for the big slow query to be returned.

When Google says that HTTP/2 uses "streams" they mean that ID tags are added to individual queries and ID tags are returned with replies. This creates a sort of parallel abstraction where even though there's still only a single connection, a highly capable HTTP/2 server could accept every incoming query the moment it arrives, assign an independent task or thread to begin assembling the reply, and then send back that query's reply, with its associated ID tag as soon as the reply is ready. So, Google continues...

One of the main constraints when mounting a Layer 7 DoS attack is the number of concurrent *transport connections. Each connection carries a cost, including operating system memory for socket records and buffers, CPU time for the TLS handshake, as well as each connection needing a unique four-tuple, the IP address and port pair for each side of the connection, constraining the number of concurrent connections between two IP addresses.*

In HTTP/1.1, each request is processed serially. The server will read a request, process it, write a response, and only then read and process the next request. In practice, this means that the rate of requests that can be sent over a single connection is one request per round trip, where a round trip includes the network latency, proxy processing time and backend request processing time. While HTTP/1.1 pipelining is available in some clients and servers to increase a connection's throughput, it is not prevalent amongst legitimate clients.

That was interesting. What Google just said was that while HTTP/1.1 **does** support allowing clients to "send ahead" queries (as I mentioned), in the process known as pipelining, in practice that has not been widely adopted so that most clients will wait until they've received the reply to their outstanding query before submitting the next query.

With HTTP/2, the client can open multiple concurrent ***streams*** *on a single TCP connection, each stream corresponding to one HTTP request. The maximum number of concurrent open streams is, in theory, controllable by the server, but in practice clients may open 100 streams per connection and the servers process these requests in parallel.*

For example, the client can open 100 streams and send a request on each of them in a single round trip; the proxy will read and process each stream serially, but the requests to the backend servers can again be parallelized. The client can then open new streams as it receives responses to the previous ones. This gives an effective throughput for a single connection of 100 requests per round trip, with similar round trip timing constants to HTTP/1.1 requests. This will typically lead to almost 100 times higher utilization of each connection.

So this is where, why and how HTTP/2 offers potentially far greater performance over HTTP/1.1. HTTP/2 solicits clients (typically our web browsers) to dump all of their queries into a single connection at once. (Since it's just a single TCP connection they cannot actually move across the wire simultaneously, but they can be queued up and sent so that they are packed very tightly and so that few smaller and less efficient packets are sent.) Then, assuming that the backend has sufficient parallel serving capability, all of the then outstanding replies can be assembled and, in any order, the moment any reply is ready it can be queued up for return to the client with its ID tag identifying which query it's the reply for. It's really a beautiful and elegant system; and it's clearly the future.

Note that the most modern pre-existing high request rate attacks are already leveraging this parallel streaming capability of HTTP/2 tosubject back-end servers to much higher request rates than were possible using HTTP/1 or HTTP/1.1. But now, here comes the problem that's created by the new abuse of this fancy parallel architecture. Google writes:

The HTTP/2 protocol allows clients to indicate to the server that ***a previous stream should be canceled by sending a RST_STREAM frame****. The protocol does not require the client and server to coordinate the cancellation in any way, the client may do it unilaterally. The* *client may also assume that the cancellation will take effect immediately when the server receives the RST_STREAM frame, before any other data from that TCP connection is processed.*

*This attack is called **Rapid Reset** because it relies on the ability for an endpoint to send a RST_STREAM frame immediately after sending a request frame, which makes the other endpoint start working and then rapidly resets the request. The request is canceled, but leaves the HTTP/2 connection open for additional requests and cancellations. Therefore, the HTTP/2 Rapid Reset attack which is enabled by this capability is simple: The client opens a large number of streams at once as in the standard HTTP/2 DDoS attack, but rather than waiting for a response to each request stream from the server or proxy, the client cancels each request immediately.*

And since HTTP/2 specifies that the client may assume that any canceled stream is immediately available for another request, the HTTP/2 abusing attacker may immediately follow a stream reset with another new request using the same stream.

The ability to reset streams immediately allows each connection to have an indefinite number of requests in flight. By explicitly canceling the requests, the attacker never exceeds the limit on the number of concurrent open streams – which is typically 100. The number of in-flight requests is no longer dependent on the round-trip time (RTT), but only on the available network bandwidth.

In a typical HTTP/2 server implementation, the server will still have to do significant amounts of work for canceled requests, such as allocating new stream data structures, parsing the query and doing header decompression, and mapping the URL to a resource. Moreover, in reverse proxy implementations, the request may be proxied to the backend server before the RST_STREAM frame is processed. This requires the proxy to forward the stream reset to the appropriate back-end server. By comparison, the attacking client has almost no cost for sending the requests. This creates an exploitable cost asymmetry between the server and the client.

Another advantage the attacker gains is that the explicit cancellation of requests immediately after creation means that a reverse proxy server won't send a response to any of the requests. Canceling the requests before a response is returned thus reduces the returning bandwidth to the attacker. Normal high request rate attacks may be throttled by the attacker's incoming bandwidth since the replies must be received. Here, all of the work is loaded onto the servers with very little traffic returning.

Although it's diabolical it's not a flaw in HTTP/2, it's just an abuse of a deliberate design feature.

So what does Google recommend? They begin by explaining...

We don't expect that simply blocking individual stream requests is a viable mitigation against this class of attacks — instead the entire TCP connection needs to be closed when abuse is detected. HTTP/2 provides built-in support for closing connections, using the GOAWAY frame type.

Now, ya gotta love that! The designers of the HTTP/2 protocol defined a "GOAWAY" frame that could be sent back up the link to the client telling it to, well, go away. Google says:

The RFC defines a process for gracefully closing a connection that involves first sending an informational GOAWAY that does not set a limit on opening new streams, and one round trip later sending another that forbids opening additional streams.

However, this graceful GOAWAY process is usually not implemented in a way which is robust against malicious clients. This form of mitigation leaves the connection vulnerable to Rapid Reset attacks for too long, and should not be used for building mitigations as it does not stop the inbound requests. Instead, the GOAWAY should be set up to limit stream creation immediately.

This leaves the question of deciding which connections are abusive. A client which cancels requests is not inherently abusive, the feature exists in the HTTP/2 protocol to help better manage request processing. Typical situations are when a browser no longer needs a resource it had requested due to the user navigating away from the page, or applications using a long polling approach with a client-side timeout.

Mitigations for this attack vector can take multiple forms, but mostly center around tracking connection statistics and using various signals and business logic to determine how useful each connection is. For example, if a connection has more than 100 requests with more than 50% of the given requests canceled, it could be a candidate for a mitigation response. The magnitude and type of response depends on the risk to each platform, but responses can range from forceful GOAWAY frames as discussed before to closing the TCP connection immediately.

To mitigate against the non-cancelling variant of this attack, we recommend that HTTP/2 servers should close connections that exceed the concurrent stream limit. This can be either immediately or after some small number of repeat offenses.

So that's the story. Exactly as Google initially characterized the problem, the deliberate design of HTTP/2, which can be used to significantly increase its efficiency, can also be used to significantly increase the potency of DDoS attacks. Attackers figured out that rather than using HTTP/2 simply to simultaneously ask for tons of resources and then be flooded by their return, they could, instead, immediately cancel their request and reissue another. Against this threat, today's servers, especially those in the cloud which have distributed their request handling among multiple back-end components, which might make canceling those issued requests much more tricky and time consuming, would be seriously taxed by that strategy.

It will be interesting to see whether anything can be done to change HTTP/2 to prevent or limit this abuse. At the moment the various servers are testing themselves and modestly tweaking their request handling to do a "less bad" job of dealing with this abuse of this HTTP/2 feature.

Since, as I mentioned above, it's not possible to spoof the IP addresses of anything that's riding on top of TCP, the best solution might be to dynamically black-list or at least significantly throttle any IP that is found to be abusing HTTP/2 Rapid Reset. In that way, the bots would be recognized and quickly ignored at the perimeter of a large hosting provider like Cloudflare.