Security Now! #808 - 03-02-21
CNAME Collusion

This week on Security Now!

This week we discuss a welcome change coming soon to the Chrome browser, and a welcome evolution in last week's just released Firefox 86. We're going to look at questions surrounding the source of the original intrusion into SolarWinds servers, and at a new severity-10 vulnerability affecting Rockwell Automation PLC controllers. We'll touch on VMware's current trouble with exploitation of their vCenter management system, and I want to share a recent code debugging experience I think our listeners will enjoy and find interesting. Then we're going to conclude with some information about something that's been going on quietly out of sight and under the covers which **must** be made as widely public among web technologists as possible.

Not exactly confidence inspiring...

Browser News

Chrome to default to trying HTTPS first when not specified.
I'm delighted to announce that the forthcoming Chrome v90, which is slated for release in mid-April, will finally assume that any non-specific (or as Google terms it "schemeless") URL is meant to be "https://" before falling back to "http://". As our listeners know, I've mentioned often that it seems well past time for our web browsers to assume HTTPS rather than HTTP. It appears that's going to happen. This will likely influence all of the other Chromium-based browsers and we can expect Firefox and Safari to follow suit.

Last Wednesday, Google's Emily Stark tweeted: "if you're running Chrome Canary, Dev, or Beta and you want some more https in your life, go to chrome://flags and search for "#omnibox-default-typed-navigations-to-https". Chrome will then send schemeless hostnames over https:// instead of http:// by default"

The next day, Emily followed-up Tweeting: "currently the plan is to run as an experiment for a small % of users in Chrome 89, and launch fully in Chrome 90, if all goes according to plan."

The current public-channel release is #88. And I checked, 88 doesn't yet have that option, which 89 will. A test percentage of unwitting Chrome 89 users will have that turned on for them. And assuming that nothing big blows up, Google intends to turn this on for everyone in about six weeks. But once the next release, v89, shows up, we can all turn it on for ourselves.

Firefox's "Enhanced Tracking Protection" just neutered 3rd-party cookies!
It seems that this week my web browser wish list is getting some long overdue attention. Mozilla just announced that with the recently released Firefox 86, the long-running abuse of 3rd-party cookies would finally be 86'd.

As I've long lamented, the use of 3rd-party offsite cookies for tracking was never part of the plan. Netscape invented first party cookies in order to implement a simple session maintenance mechanism which, for the first time, enabled the concept of a user logging into a website and then being known as they moved about. It amounted to them being tracked as they moved about the site, but being a 1st-party cookie, it only worked for that one site. What was never intended was that 3rd-party advertisers or dark and unseen analytics providers, or Google Analytics, would insinuate themselves pervasively throughout the web and employ their own cookies for tagging and tracking the activities of individual users. But as we know, what can be done will be done and tracking is what resulted.

https://blog.mozilla.org/security/2021/02/23/total-cookie-protection/

With the release of Firefox 86, that just ended. At the top of their "total cookie protection" announcement, Mozilla wrote:

"Today we are pleased to announce Total Cookie Protection, a major privacy advance in Firefox built into ETP Strict Mode. Total Cookie Protection confines cookies to the site where they were created, which prevents tracking companies from using these cookies to track your browsing from site to site."

In other words, 3rd-party cookies are not blocked. But they are stovepiped. Assuming that 3rd-party cookies are enabled at all, any 3rd-party entity may give the user's browser its cookie. But now Firefox will associate the 3rd-party cookie it received with the website where the user was when that cookie was received. The two will be paired. If the user returns to the same site, that 3rd party cookie will be returned to the 3rd party site. But if the user visits a different site with content — an advertisement or tracking code — from that same 3rd-party site, because the user is visiting a different website, there will be no 3rd party cookie associated with the visited site. So cross-site tracking will be defeated.

If we wanted to use some highfalutin' language to describe this, we would say that Mozilla has segmented their browser cookie namespace, creating individual cookie namespaces for each website.

This is a wonderful compromise between allowing and refusing all 3rd-party cookies. 3rd-party cookies are still allowed, but they will only be returned by the same site where they were set.

We know that the pressure to track is significant even though when it really does work it is frequently reported as unnerving when some recent activity somewhere turns up in an advertisement elsewhere a few minutes later. So, big props to Mozilla for adding this welcome and long overdue feature to Firefox.

Security News

As easy as "SolarWinds123"

As we know, post-network intrusion forensics is always difficult. But the previous SolarWinds CEO Kevin Thompson says it may have all started when an intern set an important password to "'solarwinds123." Then, adding insult to injury, the intern shared the password on GitHub.

Kevin Thompson told a joint US House of Representatives Oversight and Homeland Security Committees hearing that the password was "a mistake that an intern made. They violated our password policies and they posted that password on their own private Github account. As soon as it was identified and brought to the attention of my security team, they took that down."

However, Kevin's responsible-as-possible sounding testimony was contradicted by SolarWinds' current CEO Sudhakar Ramakrishna who confessed that the password "solarwinds123" was in use by 2017. Vinoth Kumar, the security researcher who discovered the leaked password to one of SolarWind's file servers had earlier stated that SolarWinds did not change the password until November of 2019, after he had discovered it on the Internet and reported it.

The insecure password is one of three possible avenues of attack SolarWinds has been investigating as it tries to determine how it was first compromised by the hackers. The two other theories are the brute-force guessing of company passwords, as well as the possibility the hackers could have entered via compromised third-party software.

In other words, we don't know for sure, and since post-intrusion forensics is difficult, we might never be certain. But publicly posting a private password on Github is certainly not the way to keep it a secret.

Rockwell Automation's CVE-2021-22681 is a CRITICAL 10 out of 10
The security of nearly all of Rockwell Automation's “PLC”s — Programmable Logic Controllers — are affected by the use of a single globally shared static encryption key. Yes... Every one of the hundreds of thousands of their systems in the world is “protected” by the same single key.

A Programmable Logic Controller fills an important gap within any process control system. You might be on an oil rig where the pressure of a feeder line must be maintained within a range. But the pressure might need to be taken at several places and averaged. And there might be multiple upstream sources of pressure controlled by valves with actuators. So, in a sense, it's a closed system with a handful of inputs and outputs. And once its function is defined it can and should just be left alone to work. But how do you build the control system? Once upon a time, before computers, a custom circuit would have been created and built from scratch. Today, you go over to Rockwell Automation's website and pick the PLC — the Programmable Logic Controller — that's just big enough to handle the number and types of inputs and outputs that you need. Then an engineer who's been trained up, uses Rockwell's software, called Studio 5000 Logix Designer to program the little computer that resides inside the industrial oil-rig tough little box.

So a PLC does a limited set of very specific things. Once upon a time it might have been done with discrete circuitry. A bunch of clacking relays all wired together to implement the sequencing logic required to route vials of newly synthesized vaccine though their carousel, to count them as they pass and to flip routing gates open and closed at the exact time to fill the waiting containers. But today, all of that is handled by an unseen Rockwell Automation PLC. It's programmed once and it effectively becomes a part of the overall machine.

In any industrial setting where things are moving, spinning, whirring, valves are opening and closing and stuff's happening, there are tasks that don't require a general purpose computer. And God knows you sure don't want Windows anywhere near any of that. Sure, Windows hosts Rockwell's Studio 5000 Logix Designer. But once the PLC device is programmed, it's blessedly off on its own.

So everything would be great with these work-a-day PLCs. But apparently some Bozo decided that needing to go down to the shop floor to tweak the controller of the machine that squeezes bottle caps onto coke bottles was too much to ask. So... let's put it on the network! Believe it or not, these perfect little happy worker controllers have received IP addresses — stop me if you've heard this one before. What could possibly go wrong. And yes, as I mentioned above, not only do they have IP addresses, often with public presences on the Internet, but they are all being protected by the same, now well known cryptographic key.

Last Thursday, the US CISA warned of a critical vulnerability (remember: a 10 out of 10) that allows hackers to remotely connect to Logix controllers and from there alter their configuration or application code. CISA stated that the vulnerability requires a low skill level for exploitation. They explained:

“The vulnerability, tracked as CVE-2021-22681, is the result of the Studio 5000 Logix Designer software making it possible for hackers to extract a secret encryption key. This key is hard-coded into both Logix [PLC] controllers and engineering stations and verifies communication between the two devices. A hacker who obtained the key could then mimic an engineering workstation and manipulate PLC code or configurations that directly impact a manufacturing process.”

VMware's vCenter troubles
Meanwhile, hackers are mass-scanning the Internet in search of VMware servers with a newly disclosed code-execution vulnerability carrying a severity rating of 9.8 out of 10.

CVE-2021-21972, as the security flaw is tracked, is a remote code-execution vulnerability in VMware vCenter server. vCenter is an application for Windows or Linux used by admins to enable and manage virtualization of large networks. Within a day of VMware issuing a patch, proof-of-concept exploits appeared from at least six different sources. The severity of the vulnerability, combined with the availability of working exploits for both Windows and Linux machines, sent hackers scrambling to actively find vulnerable servers.

Troy Mursch, a researcher with Bad Packets wrote: “We’ve detected mass scanning activity targeting vulnerable VMware vCenter servers.” Troy said that the BinaryEdge search engine found almost 15,000 vCenter servers exposed to the Internet, while Shodan searches revealed about 6,700. The mass scanning is aiming at identifying servers that have not yet installed the patch

The flaw is just about as bad as it gets. It allows a hacker with no authorization to upload files to vulnerable vCenter servers that are publicly accessible over port 443. Successful exploits will result in hackers gaining unfettered remote code-execution privileges in the underlying operating system. The vulnerability stems from a lack of authentication in the vRealize Operations plugin, which is installed by default.

https://www.ptsecurity.com/ww-en/about/news/vmware-fixes-dangerous-vulnerabilities-that-threaten-many-large-companies/

In their blog posting, Positive Technologies, who discovered and privately reported the flaw to VMware wrote:

> *“In our opinion, the RCE vulnerability in the vCenter Server can pose no less a threat than the infamous vulnerability in Citrix (CVE-2019-19781) [which was implicated in the ransomware attacks on hospitals back in 2019] The error allows an unauthorized user to send a specially crafted request, which will later give them the opportunity to execute arbitrary commands on the server. After receiving such an opportunity, the attacker can develop this attack, successfully move through the corporate network, and gain access to the data stored in the attacked system (such as information about virtual machines and system users). If the vulnerable software can be accessed from the Internet, this will allow an external attacker to penetrate the company’s external perimeter and also gain access to sensitive data. Once again, I would like to note that this vulnerability is dangerous, as it can be used by any unauthorized user.”*

SpinRite

Last week I mentioned that one troublesome machine owned by a tester in Germany was again causing my new code some trouble. Back in the earlier ReadSpeed benchmark development days, I was worried about wearing out my welcome with this tester by producing a series of test releases in what was ultimately a futile attempt to zero-in on the trouble and fix it. Then, miraculously, his system began working. That always makes me nervous, because if you didn't really do something to fix the problem, the "miracle" might choose to reverse itself at any point. And, sure enough, when I moved the new code into SpinRite and released the first test releases of it... it no longer worked on his system.

I had purchased one of the same very old Gigabyte motherboards from eBay and have been extremely anxious to figure out exactly what's going on. That finally happened this past week. And I thought I'd give our listeners a peek into this little micro-drama, because it demonstrates a bit about the process of debugging code and serves as a beautiful example of the weird sorts of things we face in the real world, where code born in the lab actually needs to function. As it turns out, I could have never figured this out remotely, because even with the machine sitting right in front of me, what I was seeing made absolutely no sense.

The problem was occurring in a simple routine to copy the contents of a disk sector buffer from high XMS memory above one megabyte down into traditional x86 segmented memory below one megabyte. Should be a piece of cake. But the machine went into that subroutine and it never came out. So, okay, at least now I had apparently located the location of the problem that Chris and this machine was having. So I fired-up my debugger, and I followed the processor into that simple subroutine.

As we've talked about a lot, one of the reasons I find the Intel chips enjoyable to program is that they have a CISC — Complex Instruction Set Computer — architecture. The ultimate example of a CISC ISA (Instruction Set Architecture) was the DEC PDP-11 and VAX machines. They were designed at a time when a lot of code was still being written in their assembly language and when compiler design was still a nascent art. So the ships themselves presented a sort of high level language.

For example, the Intel x86 architecture includes a byte range copy instruction that no self-respecting RISC chip would ever abide. The starting address of the source range is placed into the chip's SI register — SI stands for source index. And the starting address of the destination range is placed into the chip's DI register, with DI standing for destination index. The number of bytes to copy is placed into the CX register ('C' as in count). Then a single instruction is executed to cause the heavily microcoded Intel chip (or in this case AMD processor) to fetch a byte from where the SI register points, store it to where the DI register points, increment both SI and DI registers so that they will now each be pointing to the next byte in their ranges, then decrement the CX register. If the CX register has not just been decremented to 0, copy the next byte... and so on.

I'm explaining all this because as I single-stepped the processor, instruction by instruction, when I stepped into that byte range copy instruction, nothing happened. It was as if the instruction was taking forever to execute. One of the tricks we all learned back in the early days of the PC, when a system appeared to lock up, was to toggle NumLock on our keyboards by hitting that key a few times. If the keyboard's NumLock light toggles on and off, you knew that the system was kinda still alive. There was still some hope. It wasn't hard locked. Typically, if NumLock was dead, not even the famous CTRL-ALT-DEL three finger salute would work, and only the reset button would get things restarted.

In this case, NumLock was still toggling. And Chris had originally noted that the little ASCII character "spinners" on the screen kept spinning. That meant that the system was not locked up. In order for NumLock to toggle and for the spinners to spin, keyboard hardware interrupts and clock interrupts need to be serviced. So the processor was still running. But it was also apparently just sitting at that single instruction doing nothing. The Intel chips have some built-in debugging support. And this debugger works by setting a hardware breakpoint on the instruction after the one that's about to be single-stepped through. That way, when the processor comes out the other side of the instruction control returns to the debugger, the screen is updated to show the current processor state, and you can see where you are. But that breakpoint was never being tripped because the AMD Phenom II processor was never stepping out of that instruction to the next one.

So, I stared at that for a while thinking — what?!. It made no sense. This HAD to work. I think I mentioned last week that Chris had observed that everything worked just dandy if he booted from a diskette, but not when he booted from a USB thumb drive. In my subsequent experimentation before rolling up my sleeves, I learned that all was also okay when booted from any main mass storage device. And in subsequent testing I determined that it wasn't actually what booted the machine, but from where the program was run. In other words, this instruction would hang if I booted from hard drive but then ran the code from a USB thumbdrive. Yet everyone else who's been testing this code all along is also typically booting and running the code from USB. Yet, no one else is seeing this problem... which was really not surprising since this problem could not possibly be happening in the first place. But, yet, it was.

So, because what I was seeing was impossible, I decided to decompose the fancy single-instruction Intel block copy into a series of individual instructions that would accomplish the same thing. Again, I single-stepped, and again the system hung at one super-simple instruction: When the processor attempted to load the accumulator register with the contents of the location in upper memory, that instruction never completed… but also only when the code was run from USB. And DOS is not loading anything on-the-fly. It's old school. It loads everything into RAM before anything runs. After which it doesn't know or care where the code came from. So I have no idea why running from USB could possibly matter.

The only thing I can conclude is that there's a subtle bug in that old AMD Phenom II processor. The Intel x86 architecture provides six segmentation registers. I was using the default, which is DS — which stands for Data Segment. So, in a Hail Mary, I changed the code to use the FS segment register... and everything worked perfectly every time. So, henceforth, none of SpinRite code will set DS to zero and attempt to use it to access 32-bit flat memory storage. That SHOULD work, and it does work as far as we know, everywhere except on a Gigabyte motherboard with an AMD Phenom II processor when the code is loaded from USB. Welcome to my world.

When I published a test release for Chris to try — and also to check my own sanity — it did, indeed, fix his trouble, too. And someone else who had never reported in, but who had been watching, wrote to say that his similar AMD Phenom II based Gigabyte system had also never worked before... but it does now.

I thought our listeners might get a kick out of a peek inside a bit of last week's work. Most problems that I track down and resolve teach me something I didn't know. That's what makes the journey so interesting. I can't say that for this problem, but I have learned something not to do in the name of achieving total compatibility.

Once upon a time, back in 2004 when SpinRite 6.0 was released, one of the reasons it developed such a strong following was that it just always worked. I'm now in the process of wrestling this new and soon-to-emerge SpinRite v6.1 back into this state. When I am finished with it, it will always work.

CNAME Collusion

So, Criteo, a leading tracking company, sends website administrators, with whom they already have a tracking and analytics relationship, an e-mail. It asks them to make a quick change which will "only take 2 minutes" and it will "adapt their website to the evolution of browsers." Which is to say that it will work around their own website's visitors' attempts to block tracking to re-enable tracking of their site's visitors in a "more optimal way".

After presenting instructions for the site's webmaster about how to make the required change — which will, indeed, only require a couple of minutes, in the particular instance of e-mail that I saw, they conclude with: "If this is not done, you may lose 11.64% of your sales, 11.53% of your gross turnover, and 20.82% of your audience."

And this brings us to some recently published research which explores just how prevalent and pervasive this new technique has grown over the past few years. The group of five researchers will be presenting their work at the 21st Privacy Enhancing Technologies Symposium (PETS 2021) this July. But we have it now: https://arxiv.org/pdf/2102.09301.pdf

The abstract of their 21-page paper sets up the situation:

> Abstract: Online tracking is a whack-a-mole game between trackers who build and monetize behavioral user profiles through intrusive data collection, and anti-tracking mechanisms, deployed as a browser extension, built-in to the browser, or as a DNS resolver. As a response to pervasive and opaque online tracking, more and more users adopt anti-tracking tools to preserve their privacy. Consequently, as the information that trackers can gather on users is being curbed, some trackers are looking for ways to evade these tracking counter-measures. In this paper we report on a large-scale longitudinal evaluation of an anti-tracking evasion scheme that leverages CNAME records to include tracker resources in a same-site context, effectively bypassing anti-tracking measures that use fixed hostname-based block lists. Using

historical HTTP Archive data we find that this tracking scheme is rapidly gaining traction, especially among high-traffic websites. Furthermore, we report on several privacy and security issues inherent to the technical setup of CNAME-based tracking that we detected through a combination of automated and manual analyses. We find that some trackers are using the technique against the Safari browser, which is known to include strict anti-tracking configurations. Our findings show that websites using CNAME trackers must take extra precautions to avoid leaking sensitive information to third parties.

Okay... So, first of all, what are CNAME records? They are not something we've had the occasion to talk about in the past. But DNS, by comparison, is something we're pretty much always talking about. A CNAME record is simply another type of DNS record.

A DNS "A" record resolves a specific domain name to a "dotted quad" IPv4 address. A DNS "AAAA" record resolves a specific domain name to an IPv6 address. An SMTP eMail server might query a domain like grc.com for any MX records which will provide one or more IP addresses of eMail servers for that domain. And for various reasons, a domain's TXT records might be queried for information — like to provide the public key used to check a domains anti-spam signatures. So, although DNS's primary purpose is to lookup and return IP addresses, it's also a nifty general purpose distributed Internet directory capable of containing and returning all sorts of other information. And, another of those types of queries is the CNAME.

CNAME stands for Canonical Name. Whereas an "A" query returns an IPv4 address, a CNAME query returns another domain name. The domain name being queried is considered to be an alias, and what's returned is the canonical name for that alias.

CNAME records are handled specially by DNS. As Wikipedia explains:

CNAME records are handled specially in the domain name system, and have several restrictions on their use. When a DNS resolver encounters a CNAME record while looking for a regular resource record, it will restart the query using the canonical name instead of the original name. The canonical name that a CNAME record points to can be anywhere in the DNS, whether local or on a remote server in a different DNS zone.

So here's what's evil, and what that eMail above was asking website admins to do: They were asking, say, "example.com" to place a CNAME record into their site's DNS such that some arbitrary, but specified, subdomain of example.com, like say, " "dyzxrdb.example.com" would be an alias for the canonical name web-trackers-R-us.com.

So what that does, exactly, is anytime someone wants to lookup the IP address for "dyzxrdb.example.com", their assigned DNS resolver which is performing the DNS resolution for them, will query the example.com domain's nameservers for that subdomain. But because that subdomain record is a CNAME record, that's what will be returned to the querying DNS resolver. It's essentially being told: if you want "dyzxrdb.example.com", it's actually located at the following domain name... in other words, at web-trackers-R-us.com. Whereupon the user's DNS resolver asks web-trackers-R-us.com for their IP address and returns that to the original querying user.

From the user's perspective, they asked for the IP of a subdomain of example.com. And they received an IP. But due to prior collusion between the website they're visiting and web-trackers-R-us.com, the IP they received was for web-trackers-R-us.com. From the standpoint of the user's web browser, this is an "in domain" same-domain query, so 3rd-party cookie restrictions do not apply and the user's web browser will treat this query as a subdomain of the website being visited.

Okay.

So what we have so far, is a horrifically sneaky means of deliberately overriding a user's wishes for anti-tracking by websites that feel that they have a superior right to track and obtain leverage from their visitors.

But, believe it or not, it gets much worse... and you're not going to believe this:

Cookies set on specific domains are accessible to, and sent to, anyone who queries their subdomains. This means that by colluding in this way to allow an untrustworthy 3rd-party tracking entity to pretend to be within a website's domain, the cookies being held by the browser of visitors to that site will be leaked to that 3rd-party entity because the web browser won't know any better. That website's visitors' logon session authentication cookies will be sent outside of that domain to untrusted and, I would argue, untrustworthy, 3rd party tracking and analytics companies. The fact that it's done over HTTPS provides no security. Anyone at any of those tracking, advertising, analytics firms — of which 13 have been identified so far by the researchers — could trivially impersonate any user of any website who didn't explicitly logoff, and who therefore has a still-valid authentication cookie. It's an unbelievable breach of trust and abuse of web technology.

I ran across a wonderful website that allows us to play with and explore our own browser's cookie and subdomain handling for understanding exactly this issue. I've made it this week's Security Now podcast shortcut of the week, so it's <https://grc.sc/808>

<https://scripts.cmbuckley.co.uk/cookies.php>

I'm running Firefox 86 with its full "Total Cookie Protection" enabled and it's **not** blocking any of this leakage because these are not 3rd-party cookies. These are sneaky subdomain cookies.

So how widespread is this? Thankfully, these researchers have gone to some effort to unearth the extent of this currently spreading industry-wide website collusion with the tracking industry. It's not difficult. You just resolve any subdomains of the primary domain that you receive from a website. You do the DNS lookup for yourself as if you were a recursive DNS resolver, and you see whether you receive a CNAME record that points to any one of the 13 current providers of this form of CNAME tracking.

The researchers found this technique currently in use on a total of 10,474 websites. And of the top 10,000 websites overall, 9.98% — 1 in 10 — of those top 10,000 are currently employing this form of CNAME tracking, cloaking, subdomain collusion.

And their research observed what they termed a “targeted treatment of Apple's Safari web browser” where the advertising technology company Criteo (who mailed the letter I opened with) switched specifically to CNAME cloaking to bypass Safari's otherwise strong privacy protections.

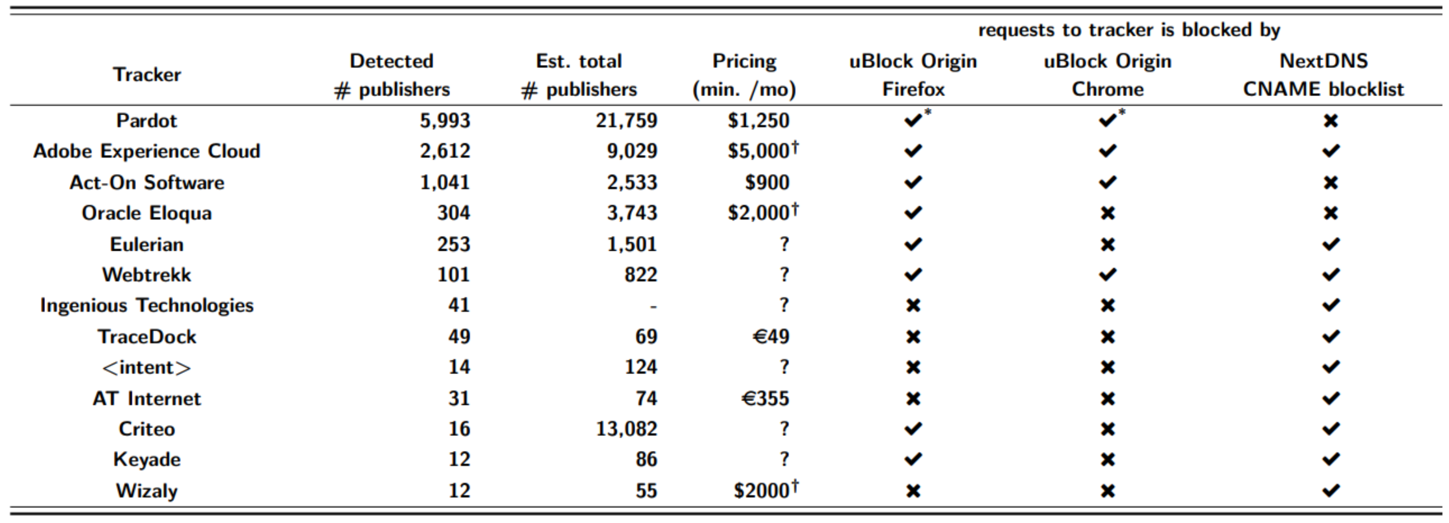
And as for data leaks? Significant cookie data leaks were found on 95% of the sites that used CNAME tracking, all of which sent cookies containing private information such as full names, locations, email addresses, and even session authentication cookies to trackers of other domains without the user have any knowledge or control.

The entire presumption of cookies is that, bad and abuse-prone as they may be, at least they stay within the domain that set them. At least their content, whatever it might be — even if it’s a user’s actual name and real world identity, bad practice as that would be — at least it remains between those two parties. So while cookies can be used for tracking, the only data that is ever returned to a domain is something that that domain earlier sent. Thus, by definition it’s not secret to that domain.

But now, thanks to the horrendous abuse of CNAMEs being used to deliberately confuse cookie domains, data is being sent with queries by user’s web browsers to entities who never set that data in the first place. As the researchers noted, that data which should never be exposed to any third party, often contains information that tracking firms would die to have, and leverage. But now they don’t have to. They just need to get websites to collude with them by adding a CNAME record to that domain’s DNS.

The only good news here is that good old Gorhill’s **uBlock Origin** add-on is at least partially effective at spotting and blocking accesses to these despicable subdomains:

Table 1. Overview of the analyzed CNAME-based trackers, based on the HTTP Archive dataset from October 2020.



				requests to tracker is blocked by		
Tracker	Detected # publishers	Est. total # publishers	Pricing (min. /mo)	uBlock Origin Firefox	uBlock Origin Chrome	NextDNS CNAME blocklist
Pardot	5,993	21,759	$1,250	✔*	✔*	✖
Adobe Experience Cloud	2,612	9,029	$5,000†	✔	✔	✔
Act-On Software	1,041	2,533	$900	✔	✔	✖
Oracle Eloqua	304	3,743	$2,000†	✔	✖	✖
Eulerian	253	1,501	?	✔	✖	✔
Webtrekk	101	822	?	✔	✔	✔
Ingenious Technologies	41	-	?	✖	✖	✔
TraceDock	49	69	€49	✖	✖	✔
<intent>	14	124	?	✖	✖	✔
AT Internet	31	74	€355	✖	✖	✔
Criteo	16	13,082	?	✔	✖	✔
Keyade	12	86	?	✔	✖	✔
Wizaly	12	55	$2000†	✖	✖	✔

†: Pricing information does not originate from original source, but as reported in reviews of the product.
*: Requests made to the CNAME subdomain triggered by a third-party analytics script hosted on `pardot.com`; the blocklist prevents the analytics script from loading. If this script was loaded from the CNAME domain, it would not be blocked.

The #1 worst offender by far, which is infecting 5,993 detected websites is “Pardot”, a SalesForce company which bills itself as “Powerful B2B Marketing Automation” stating that “Pardot offers powerful marketing automation to help marketing and sales teams find and nurture the best leads, close more deals, and maximize ROI.”

Number two on the hit list is Adobe Experience Cloud, and the good news is, uBlock Origin blocks them both. But in a note on this table the researchers observe that Pardot is being blocked because the 3rd-party script being sourced from pardot.com is blocked, and that if that script was not blocked then CNAME abuse would succeed.

The researchers had the following to say about CNAME Countermeasures:

Countermeasures

In response to a report that a tracker was using CNAMEs to circumvent privacy blocklists, uBlock Origin released an update for its Firefox version that thwarts CNAME cloaking. The extension blocks requests to CNAME trackers by resolving the domain names using the [browser's own] browser.dns.resolve API method to obtain the last CNAME record (if any) before each request is sent. Subsequently, the extension checks whether the domain name matches any of the rules in its blocklists, and blocks requests with matching domains while adding the outcome to a local cache.

Although uBlock Origin also has a version for Chromium-based browsers, the same defense cannot be applied because Chromium-based browser extensions do not have access to an API to perform DNS queries. As such, at the time of this writing, it is technically impossible for these extensions to block requests to trackers that leverage CNAME records to avoid detection. uBlock Origin for Chrome, which does not have an explicit defense for CNAME-based tracking, still manages to block several trackers. This is because the requests to the trackers matched an entry of the blocklist with a URL pattern that did not consider the hostname. Unfortunately, it is fairly straightforward for the tracker to circumvent such a fixed rule-based measure, e.g. by randomizing the path of the tracking script and analytics endpoint, as is evidenced by the various trackers that could only be blocked by the uBlock Origin version on Firefox.

And the researchers wrap up their research with the following conclusion:

Our research sheds light on the emerging ecosystem of CNAME-based tracking, a tracking scheme that takes advantage of a DNS-based cloaking technique to evade tracking countermeasures. Using HTTP Archive data
and a novel method, we performed a longitudinal analysis of the CNAME-based tracking ecosystem using crawl data of 5.6M web pages. Our findings show that unlike other trackers with similar scale, CNAME-based trackers are becoming increasingly popular, and are mostly used to supplement “typical” third-party tracking services.

We evaluated the privacy and security threats that are caused by including CNAME trackers in a same-site context. Through manual analysis we found that sensitive information such as email addresses and authentication cookies leak to CNAME trackers on sites where users can create accounts. Furthermore, we performed an automated analysis of cookie leaks to CNAME trackers and found that cookies set by other parties leak to CNAME trackers on 95% of the websites that we studied.

Finally we identified two major web security vulnerabilities that CNAME trackers caused. We disclosed the vulnerabilities to the respective parties and have worked with them to mitigate the issues. We hope that our research helps with addressing the security and privacy issues that we highlighted, and inform development of countermeasures and policy making with regard to online privacy and tracking.

So what we have here is a real mess. No form of explicit tracking was EVER designed into our use of the web. It happened as an unintended consequence of single advertising services having an appearance on multiple hosting websites. And those providers were allowed to set cookies in our browsers just like their originally-intended first party cousins. We should have stopped it then. We should have just said no. But the trouble was, this tracking was effectively invisible. It went completely unseen by the public. But it wasn't the public's just to stop it. It was technologists' job to say no and stop it. But that didn't happen.

Then, when an awareness began to emerge, and 3rd-party cookies were being threatened and sometimes disabled or deleted, browser fingerprinting emerged as a means for allowing what had grown into a tracking industry to retain its grip on our browsers and on us.

Since fingerprinting was more difficult to defend than cookies, it received a stronger pushback from browser vendors who didn't like the idea that cookies were being bypassed as a means of tracking their users.

And now we have what is perhaps the ultimate abuse in tracking technology: Thanks to explicit collusion among a growing number of websites, third parties are being allowed to receive a website's cookies — apparently without the website knowing or caring. Our logged-in session authentication cookies are being received by 3rd party tracking entities with whom users have no relationship; and with whom they would surely refuse to share their logon session and various other possibly personal details if they were made aware of what the technology they are using was doing behind their backs.

As with the original abuse of third party cookies, where all of this began, it cannot be the responsibility of those who do not understand this, to say no. People just use the technology we give them. It MUST BE the responsibility of those who DO understand this to push back in every way possible.

I am SO glad that this research has shined a bright light on this next-generation tracking practice. It needs to be shut down immediately. But it takes technologists knowing about it for that to happen. Today, more do. And still more will, soon.