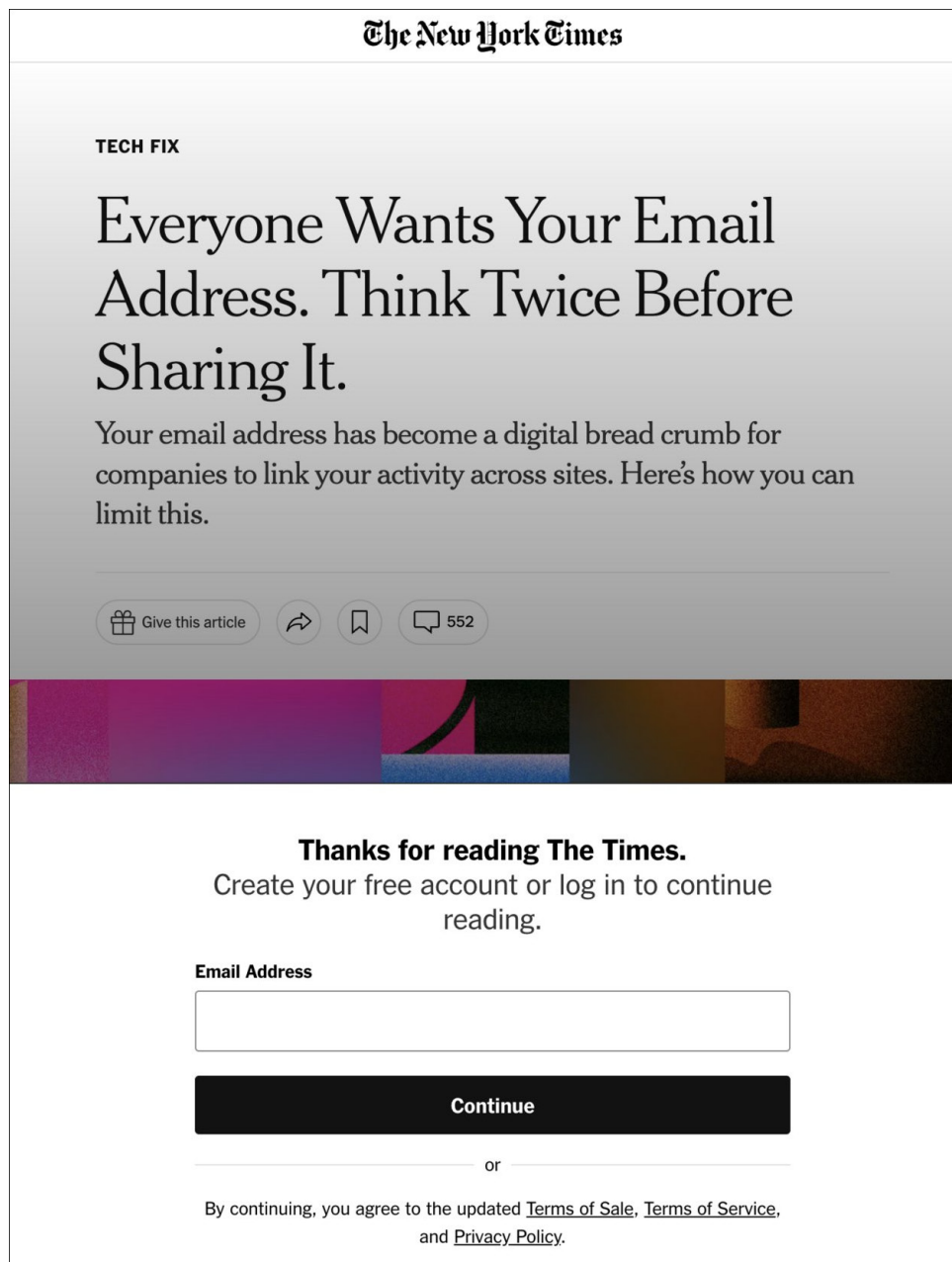


Security Now! #910 - 02-14-23

Ascon

This week on Security Now!

What more has happened with the ESXi ransomware story? Is malicious use of ChatGPT going to continue to be a problem? What exactly is Google giving away? Why is the Brave browser changing the way it handles URLs? What bad idea has Russia just had about their own hackers? Why would Amazon change its S3 bucket defaults? Now who's worried about Chinese security camera spying? And who has just breathed new life into Adobe's PDF viewer? What's on our listeners' minds, and what the heck is Ascon and why should you care? Those questions and more will be answered on today's 910th episode of Security Now!







The New York Times

TECH FIX

Everyone Wants Your Email Address. Think Twice Before Sharing It.

Your email address has become a digital bread crumb for companies to link your activity across sites. Here's how you can limit this.

 Give this article    552

Thanks for reading The Times.
Create your free account or log in to continue reading.

Email Address

Continue

or

By continuing, you agree to the updated [Terms of Sale](#), [Terms of Service](#), and [Privacy Policy](#).

Security News

ESXiArgs follow-up

We know that a story as big as last week's topic of the widespread global ransomware attacks on VMware's ESXi servers doesn't end in a few days. So we have some follow-ups to last week's reporting. First, the CISA formally released an open source ESXiArgs Ransomware Recovery Script on GitHub. Before this I had never paid much attention to CISA's extensive presence on GitHub: <https://github.com/cisagov/ESXiArgs-Recover>

Examples of projects and repositories at CISA's GitHub include:

- getgov: Building a new .gov registrar for a bright .gov future
- RedEye: a visual analytic tool supporting Red & Blue Team operations
- cset : a cybersecurity evaluation tool
- crossfeed: external monitoring for organization assets

And there are a huge number more at: <https://github.com/cisagov>

Anyway, as for CISA's ESXiArgs ransomware recovery script, CISA wrote:

CISA has released a recovery script for organizations that have fallen victim to ESXiArgs ransomware. The ESXiArgs ransomware encrypts configuration files on vulnerable ESXi servers, potentially rendering virtual machines (VMs) unusable.

ESXiArgs-Recover is a tool to allow organizations to attempt recovery of virtual machines affected by the ESXiArgs ransomware attacks.

CISA is aware that some organizations have reported success in recovering files without paying ransoms. CISA compiled this tool based on publicly available resources, including a tutorial by Enes Sonmez and Ahmet Aykac. This tool works by reconstructing virtual machine metadata from virtual disks that were not encrypted by the malware. For more information, see CISA's ESXiArgs Ransomware Virtual Machine Recovery Guidance.

Disclaimer [boilerplate]

CISA's ESXiArgs script is based on findings published by the third-party researchers mentioned above. Any organization seeking to use CISA's ESXiArgs recovery script should carefully review the script to determine if it is appropriate for their environment before deploying it. This script does not seek to delete the encrypted config files, but instead seeks to create new config files that enable access to the VMs. While CISA works to ensure that scripts like this one are safe and effective, this script is delivered without warranty, either implicit or explicit. Do not use this script without understanding how it may affect your system. CISA does not assume liability for damage caused by this script.

This script is being provided "as is" for informational purposes only. CISA does not endorse any commercial product or service, including any subjects of analysis. Any reference to specific commercial products, processes, or services by service mark, trademark, manufacturer, or otherwise, does not constitute or imply endorsement, recommendation, or favoring by CISA.

CISA recommends organizations impacted by ESXiArgs evaluate the script and guidance provided in the accompanying README file to determine if it is fit for attempting to recover access to files in their environment.

So this is quite cool. We have the cybersecurity arm of the US government being active and posting scripts on GitHub to help mitigate a massive problem.

So that's the good news. The bad news is that the next day, last Wednesday, a new version of the ESXiArgs ransomware appeared to render any such scripts useless – though it would still be potentially useful to anyone who was hit with the earlier version. I noted last week that it wasn't clear why the bad guys were not also encrypting the large system image .VMDK file. Perhaps they were just in a hurry to blast as many systems as possible in the shortest amount of time, and they figured that no one would know any better if they were to just encrypt the tiny pointer files. But now they are encrypting the main .VMDK virtual machine image file.

And also as I was expecting last week, those running OpenSLP honeypots saw the number of scanning IPs jump from 3 to over 40. And while 40 is a large number, it's certain that many of those were security firms working to assess the size of the vulnerable population once this became big news. And as a result of some of those scans we now know that some reports have pegged the total number of exposed ESXi instances as high as 83,000.

And from Rapid7's use of their telemetry data gathered through their Project Sonar, they have found nearly 19,000 currently vulnerable VMWare ESXi servers connected to the Internet without a patch for CVE-2021-21974.

I read a comment from one cloud provider who indicated that as far as they were concerned their responsibility is hardware and connectivity and no more. The maintenance of whatever software the customer is running is the customer's sole responsibility. And they feel that this is true even when they, the cloud provider, were the ones who initially established the running software, including ESXi, on the hardware. That being the case, I wonder again whether this might be a classic case of something falling through the cracks, where each party believed that the other was responsible for its maintenance, so maintenance fell through the cracks. If so, the consequences were disastrous and it's going to be interesting to see whether any behavior change occurs from anyone as a result.

ChatGPT's Malicious Use

CheckPoint Security's blog posting is titled: *"Cybercriminals Bypass ChatGPT Restrictions to Generate Malicious Content"*

Paraphrasing what CheckPoint wrote, they said: *"There have been many discussions and research on how cybercriminals are leveraging OpenAI's ChatGPT platform to generate malicious content such as phishing emails and malware. CheckPoint Research's previous blog described how ChatGPT successfully conducted a full infection flow, from creating a convincing spear-phishing email to running a reverse shell, which can accept commands in English.*

CheckPoint researchers recently found an instance of cybercriminals using ChatGPT to "improve" the code of a basic Infostealer malware from 2019. Although the code is not complicated or difficult to create, ChatGPT successfully improved the Infostealer's code."

CheckPoint noted that there are currently two ways to access and work with open AI models:

The first is the Web user interface for access to ChatGPT, DALLE-2 or the openAI playground. The second is the API which is used for building applications, processes, and so on. In this fashion it's possible to use a custom application-specific user interface which calls out to the OpenAI models and data which runs in the background.

As part of its content policy from the start, OpenAI had created barriers and restrictions to minimize the creation of malicious content with its platform. Several restrictions have been set within ChatGPT's web user interface to prevent the abuse of the models. For example, if a user requests ChatGPT to write a phishing email impersonating a bank, or asks it to write malware, ChatGPT will very politely refuse.

However, CheckPoint Research is reporting that cyber criminals are working their way around ChatGPT's restrictions and that there is active chatter in the underground forums disclosing how to use OpenAI's API to bypass ChatGPT's barriers and limitations. At the moment this is done mostly by creating Telegram bots that use the API. These bots are advertised in hacking forums to increase their exposure.

The current version of OpenAI's API is used by external applications (for example, the integration of OpenAI's GPT-3 model to Telegram channels) and has very few if any anti-abuse measures in place. As a result, it allows malicious content creation, such as phishing emails and malware code, without the limitations or barriers that ChatGPT has set on their user interface. So what CheckPoint Research is saying matches the anecdotal reports that we had heard earlier.

In an underground forum, CheckPoint found a cybercriminal advertising a newly created service: a Telegram bot using OpenAI API without any limitations and restrictions. As part of its business model, cybercriminals can access the unrestricted ChatGPT backend API for an initial 20 free queries, after which they are charged \$5.50 for every 100 queries.

Okay, now this will be short-lived, of course. Can you just imagine what things must be like right now over at OpenAI? They must have been caught with their pants down by the incredible success and interest in ChatGPT. I'm sure they are aware that ChatGPT is being abused through the back end direct access API, and there can be no doubt that they're working hard to curtail this sort of ongoing abuse.

<https://blog.checkpoint.com/2023/02/07/cybercriminals-bypass-chatgpt-restrictions-to-generate-malicious-content>

Google Security Key Giveaway

Last Tuesday was "Safer Internet Day" when Google announced that they would be giving away 100,000 of their Titan hardware security keys to individuals at high risk of online attack. Google wrote:

There's no shortage of security challenges for high risk individuals — in fact, a 2022 study by Google and YouGov found that over half of journalists and professionals in politics have had their online accounts compromised. The threats are intensifying, the stakes for individuals and organizations are increasing, and we're dedicated to keeping online users safe, including helping people at higher risk of cyber attacks.

That's why on Safer Internet Day, we're announcing our new Online Safety and Security Partnership with the International Foundation of Electoral Systems (IFES) to provide free security training and tools to high-risk users. This work is designed to help shore up the defenses of democracies that work for all. We're also building on our partnership with Defending Digital Campaigns (DDC) to protect U.S. political campaigns ahead of the U.S. 2024 elections, and will be distributing 100,000 security keys globally to high-risk individuals throughout 2023, at no cost.

In addition to our partnership with IFES, we're expanding our longstanding collaboration with Defending Digital Campaigns (DDC), to equip campaigns with the security tools they need ahead of the U.S. 2024 elections. Through the Campaign Security Project, DDC will help secure campaigns in all 50 states, providing security trainings and products at no cost. Since 2020, over the course of our partnership, DDC has secured over 380 campaigns and distributed over 20,000 security keys, and we look forward to continuing to support DDC as we near a critical election cycle.

It's quite sobering to imagine that more than half of journalists and political professionals have had their accounts hacked. So moving to stronger identity authentication is a win-win.

Brave goes HTTPS-by-default

Last week we were talking the default inclusion of those 40 TLDs in the global browser HSTS list, and about the broader HSTS list in general where domains whose TLD was not among those chosen 40 would need to explicitly add themselves to the global HSTS list. And I recall further back talking about which browsers had finally started assuming that a URL entered without either an HTTP or HTTPS scheme specification would be assuming HTTPS unless that failed to function. I recall at the time noticing that the Brave browser was not among those that had decided that it was time to switch.

Well, last Thursday the Brave browser project explained their intentions in that regard and their explanation contains some interesting new information. Here's what they posted under the headline "*HTTPS by Default*" (with a bit of edited to remove obvious stuff):

Starting in version 1.50, Brave will include a new feature called "HTTPS by Default" that improves Web security and privacy by increasing HTTPS use. Brave will upgrade all sites to HTTPS, falling back to HTTP only if the site does not support HTTPS, or in the rare case a site is known to not function correctly when loaded over HTTPS. This feature is the most

protective, aggressive default HTTPS policy of any popular browser, and will be available on Android and Desktop versions of the Brave browser to begin with, with iOS coming later.

Because HTTPS is critical to privacy and security, browsers like Brave are eager to load sites over HTTPS whenever possible. And since many sites today support only HTTPS, it's simple to load these sites in a private and secure way. And encouragingly, more and more sites are being built or updated to support HTTPS only (or to use other TLS protected protocols, like secure web sockets).

Unfortunately, there are many sites on the Web that still support HTTP, and some laggards that support only insecure HTTP connections. Brave's goal is to automatically "upgrade" these sites to HTTPS whenever possible (i.e. in all cases where a site loads and functions correctly when loaded over HTTPS).

But deciding when to upgrade a site to HTTPS is tricky. In cases where a website does not support HTTPS, attempting to upgrade from HTTP to HTTPS will produce an obvious error. Other sites support both HTTPS and HTTP, but do so from different domains (e.g. <http://example.site> vs. <https://secure.example.site>), making automatic upgrades tricky. And still other sites appear to load "correctly" when fetched over HTTPS, but actually have broken functionality.

In short, ideally browsers would never load sites over HTTP, and browsers could automatically upgrade all insecure requests to HTTPS. In practice though, it is difficult to know how, when, and if a site will function correctly when upgraded from HTTP to HTTPS.

Starting in version 1.50, the Brave browser will use a new system for upgrading insecure HTTP connections to secure-and-private HTTPS connections. The new feature, called "HTTPS By Default," works as follows:

If you are about to visit a page loaded over HTTP, Brave checks to see if the destination is on a list of sites that are known to break when loaded over HTTPS. Brave maintains this list of "breaks-over-http" sites, which is open for anyone to view and use (and we'll be getting back to that in a second since it's quite interesting). If the requested site is on the list, Brave will then allow the site to load over HTTP.

Provided that it's not on the list, Brave will attempt to load the site over HTTPS by upgrading the navigation request from HTTP to HTTPS. If the server responds to the replacement HTTPS request with an error, then Brave assumes the server does not support HTTPS, and will load the site over HTTP. Otherwise, Brave's loading of the site over HTTPS will ensure a more private and secure connection.

Brave's new "HTTPS By Default" feature replaces the previous list-based approach Brave has used since our first beta versions. In that approach, Brave used the HTTPS Everywhere list (a terrific, public resource maintained by the EFF) to decide when to upgrade HTTP connections to HTTPS. But while the HTTPS Everywhere list is useful, it has two important drawbacks:

First, the HTTPS Everywhere list is no longer maintained, meaning that the list is increasingly out of date.

Second, despite the best efforts of the EFF, any approach that uses a list of what sites should be upgraded is going to be limited, as the number of sites on the Web is enormous, and it's difficult for the list maintainers to keep up. Brave's approach, by contrast, maintains a smaller

(and more manageable) list of sites to NOT upgrade.

More broadly, the main benefit of Brave's new "HTTPS by Default" feature is that it has a better default. In its default configuration, HTTPS Everywhere would allow unknown (i.e., not on the list) sites to load over HTTP; Brave's new "HTTPS by Default" approach loads the site over HTTPS even in cases where the site is new or unknown.

So, this is a very useful improvement for Brave. And Brave's use of an excluded upgrade domain list is interesting. Nearly two years ago, on March 23, 2021, Google announced that from Chrome version 90 on, the assumed default for address bar URLs that don't specify a scheme (and, really, who's entering https:// every time?) would be switched from HTTP to HTTPS. So with Brave saying that some websites do not handle automatic connection security upgrading, I was curious about what would happen if I went to one of those "will not upgrade" sites.

The list is a GitHub under Brave, Adblock-lists, master, brave-lists, https upgrade exceptions: <https://github.com/brave/adblock-lists/blob/master/brave-lists/https-upgrade-exceptions-list.txt>

And the second and third domains on the list jumped out at me, since they were "columbia.edu" and "www.columbia.edu". But that was nuts. What their presence on the list would mean was that Columbia University could not be accessed by HTTPS. What year is this? So I tried going to <http://columbia.edu> and <http://www.columbia.edu> under both Chrome and Firefox and in every case on either browser I was promptly redirected to <https://www.columbia.edu>. Since Columbia is handling the HTTP to HTTPS redirect properly, it doesn't hurt to have them on the list, but it suggests that perhaps that list of 112 domains needs a bit of maintenance. And it would be easy to do, right? Just do what I did, try going there over https and see how that works out.

Most of the domains on the list are quite obscure. But I did try another one on the list because it caught my eye: "shakespeare.mit.edu". I don't know what's going on there, but no one is home either via HTTP or HTTPS.

In any event, I wanted to let any Brave browser users who may be listening know, that from v1.5 onward Brave would be assuming HTTPS by default for any domains that matter. So in that regard, though Brave is declaring this to be a huge innovation, it appears to me that they will be achieving parity with the rest of the industry. But that's good, too. :)

1Password makes another Passkeys move.

After we recently noted Bitwarden's acquisition of an open source Passkeys developer with the outlook for Passkeys support from Bitwarden looking good, I wanted to also note that 1Password has also just announced their support for passkeys, not as a holder of their user's private keys, but as a way for users to unlock their password vaults without needing to enter a master password. In other words, whereas biometrics are often used for convenience as a means of unlocking a previously-supplied master password, what 1Password will be doing is actually using Passkeys itself to fully replace all use of a master password, just as Passkeys should be used.

1Password had previously said that sometime in early 2023 they planned to become a Passkeys-aware password manager, meaning that they would be maintaining their user's list of

private Passkeys. This is what we're hoping for from all of the major password managers, and it's a feature that they're all going to need to support. As we know, the unfortunate adoption of Passkeys, which is based upon FIDO2, means that users will still need to manage their private Passkey keys. And this means that centralized cloud storage synchronization remains a practical requirement. Apple, Google and Microsoft will be doing this for their users within their own ecosystems, but practical use requires a platform agnostic solution.

"Russian Patriotic Hackers"

We have a new term entering our lexicon: "*Russian patriotic hackers*" The Russian government has stated that it is exploring the possibility of absolving **Russian patriotic hackers** from criminal liability for attacks carried out "in the interests of the Russian Federation." Wow. Russia is choosing to become a true, fully Western-hostile, outlaw nation. The head of the State Duma Committee on Information Policy, Alexander Khinshtein, told reporters at a press conference on Friday that an exemption would be granted to individuals located both abroad and within Russia's borders alike. Khinshtein said, as quoted by RIA and TASS: "*We will talk in more detail when it receives more of a clear wording.*"

It has always been that the creation, use, and distribution of malicious computer software was punishable in Russia with up to seven years in prison. And since there have never been any exemptions to this law, many of the current pro-Kremlin hacktivist groups are breaking Russian law and could face prosecution, especially in the aftermath of a possible regime change. But this forthcoming exemption would allow pro-Kremlin hacktivists to carry out attacks with a legal carte blanche, presumably applying to groups who attack Russia's enemies, thus defining their alliance as pro-Kremlin.

Amazon to FINALLY secure their AWS S3 instances

In wonderfully welcome news that immediately begs the question "*what the hell took them so long?*" (though in fairness that is a question that we ask on this podcast almost as often as "*what could possibly go wrong?*") AWS has announced that, believe it or not, newly created instances of S3 online storage will be secured by default. What a concept!

I haven't seen this mentioned in the press yet, but I received the notice directly from Amazon because I'm an AWS S3 subscriber. I use AWS as my master cloud cold storage archive. By that I mean that I rarely transact with it. I have a huge amount of static data sitting there. For example, many years ago I ripped and stored my lifelong collection of prized audio CDs. For safety, the uncompressed wave files are stored in multiple locations, and one of those locations is S3. Every month I receive Amazon's storage bill and I just shake my head since it's like \$2.53 because most of what Amazon charges for is bandwidth usage, and mine is zero. So S3 is a bargain for off-site glacial storage.

Anyway, as a consequence of being an AWS S3 subscriber I received some very welcome news via eMail last Tuesday. Amazon wrote:

Hello,

We are reaching out to inform you that starting in April 2023 Amazon S3 will change the default security configuration for all new S3 buckets. For new buckets created after this date, S3 Block Public Access will be enabled, and S3 access control lists (ACLs) will be disabled.

The majority of S3 use cases do not need public access or ACLs. For most customers, no action is required. If you have use cases for public bucket access or the use of ACLs, you can disable Block Public Access or enable ACLs after you create an S3 bucket. In these cases, you may need to update automation scripts, CloudFormation templates, or other infrastructure configuration tools to configure these settings. To learn more, read the AWS News blog [1] and What's New announcement [2] on this change or visit our user guide for S3 Block Public Access [3] and S3 Object Ownership to disable ACLs [4]. Also, see our user guide for AWS CloudFormation on these settings [5][6].

If you have any questions or concerns, please reach out to AWS Support [7].

[1] <https://aws.amazon.com/blogs/aws/heads-up-amazon-s3-security-changes-are-coming-in-april-of-2023/>

[2] <https://aws.amazon.com/about-aws/whats-new/2022/12/amazon-s3-automatically-enable-block-public-access-disable-access-control-lists-buckets-april-2023/>

[3] <https://docs.aws.amazon.com/AmazonS3/latest/userguide/access-control-block-public-access.html>

[4] <https://docs.aws.amazon.com/AmazonS3/latest/userguide/about-object-ownership.html>

[5] <https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-s3-bucket-publicaccessblockconfiguration.html>

[6] <https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-s3-bucket-ownershipcontrols.html>

[7] <https://aws.amazon.com/support>

*Sincerely,
Amazon Web Services*

When you think back over all of the hundreds of times we've talked about AWS data being exposed and compromised on the Internet, presumably without its owner's knowledge. The fact that "Block Public Access" had never been active before by default, for a cloud storage provider, just boggles the mind.

If anyone is interested in details, I've left all of the seven links that were in the eMail to me in the show notes.

More anti-Chinese camera removals

The article appearing in The Australian last Thursday was titled "*Chinese surveillance cameras in our halls of power*" and it's apparently intended to induce concern. The article begins:

Almost 1000 Chinese Community Party-linked surveillance cameras and other recording devices, some banned in the US and Britain, have been installed across Australian government buildings, leading to calls for their urgent removal amid fears data could be fed back to Beijing.

Government departments and agencies have revealed at least 913 cameras, intercoms, electronic entry systems and video recorders developed and manufactured by controversial Chinese companies Hikvision and Dahua are operating across 250 sites, including in buildings occupied by sensitive agencies such as Defence, Foreign Affairs and the Attorney-General's Department.

Australia's Five Eyes and AUKUS partners in Washington and London moved together in November to ban or restrict the installation of devices supplied by the two companies, which are both part-owned by the Chinese Communist Party.

As we've previously covered, the US and the UK did take similar steps to remove the cameras produced by those two companies from their respective government networks. And yes, as we all know, in a closed-design closed-source world, it is possible for such devices to get up to some mischief. And I suppose that in some settings a stream of encrypted communications flowing across a government network might go unnoticed. But more worrisome is that such a device might be a launching and jumping off point for malware that's waiting to spring.

As global political tensions rise and as more and more of the physical world is subsumed by the cyber world, cyber protectionism seems inevitable. And we're certainly seeing its rise. Since anything can be buried and hidden in the most innocent looking chip, there isn't any real defense against the "*but what if?*" question because "*what if?*" could be a true potential threat. If a competent technologist was to testify in front of a congressional committee on international cyber threats, and was asked if they could absolutely positively assure that some device has no ability to do something malicious, they would have to answer "No, senator."

Microsoft to embed Adobe Acrobat PDF reader into Edge

Microsoft announced that it will embed Adobe's Acrobat PDF engine into future versions of the Edge browser. The move will take place next month in March 2023 for Edge versions on both Windows 10 and Windows 11. The FAQ that Microsoft created for this announcement was unsure about when Edge for macOS would receive the same PDF reader, saying: "*Availability for MacOS is coming in the future. We will have more to share at a later date.*"

Since the current Edge PDF engine will be removed on March 31, 2024, both PDF engines will apparently be cohabitating during the change over and presumably users will have a choice.

Closing the Loop

alim s / @dutchphyscst

Hi Steve. Hope that all is going well with you. I presume that you may not want to get back to the passwords topic again. However, I am curious of an aspect with the one time passwords. They are based on a pre-shared key. You have discussed plenty about the mobile app options and etc for storing those keys on the client side. However, I have not seen much discussions around the protection of those preshared keys on the server side. Are they protected in HSMs or some other mechanisms? I also have not heard of any breaches of those preshared keys. Maybe I am being too paranoid but I would appreciate your view on this. Best. Alim

This is a great point. What one time passcodes were designed to prevent was the capture and reuse of a static password. In the recent so-called credential stuffing attacks, where the large database of previously stolen usernames and passwords has been shown to be surprisingly effective due to people still reusing “their” password at multiple websites, the success of this attempted reuse is blocked when, despite reusing a password, the site also requires the use of a time-varying 6-digit secret passcode.

The problem with these passcodes is, as Alim has noted, that this still requires that every website keeps the shared secret, secret. And this has historically been shown to be surprisingly difficult for websites to do. The inherent weakness of the one-time-passcodes system is that it uses symmetric cryptography where each end shares the same secret. Another way of saying this is that the same secret is used to both create the passcode and to verify the passcode. And that’s the weakness.

What makes SQRL and WebAuthn different is that by using asymmetric cryptography, each party in the system uses a different key, and the roles of the keys are different. The public key that’s held by the website can **only** be used to verify an identity assertion created by using the user’s private key. As I often said, SQRL gives websites no secrets to keep, and WebAuthn is the same. By the use of asymmetric public key crypto, SQRL and WebAuthn only provide websites with a user’s public key which does not need to be kept secret since the **only** thing it can be used for is to **verify** a user’s identity claim.

By comparison, a stolen symmetric key **can** also be used to assert a user’s identity. If a website loses their one time passcode symmetric keys, the bad guys can use those stolen keys to recreate the user’s one time passcode to spoof their identity and login as them — because the system is symmetric. But if bad guys were to steal a user’s public authentication key from a website it would be of **no** use to them since the **only** thing those keys can be used for is to **verify** a user’s identity claim. That’s a huge difference. Once this improved system is universally deployed, if it ever is, online identity authentication will be significantly improved.

Until then, we will continue seeing escalations in attack cleverness. With the increased use of one time passcodes we’ve seen bad guys circumvent its protection with the increased use of proxying attacks, where the user is visiting a spoofed intermediate page which prompts for, receives and forwards the user’s one time passcode, thus successfully accomplishing a real-time bypass of one time password security.

James Housley / @jimhousley

When I first switched from Lastpass to Bitwarden I also decided to keep using OTPAuth for my

one time codes. While listening to SN #909, I wondered if Bitwarden might be a better choice for some because it would only have a one time code for the domain it is registered to. The same way a password manager prevents putting a password into a spoofed website.

That is really a good point, which hadn't occurred to me before. Thanks for that, James. So now we have a dilemma. Giving your OTP secrets to a password manager to use, risks exposure of those secrets since they are no longer in a disconnected device. You get the benefit of cross-device synchronization, which is useful. But, again, the secrets need to be kept secret. But as James points out, the advantage of using a strict URL-matching deployment of those OTP codes is that unlike an unwitting user, the password manager would not be spoofed by a lookalike domain.

But, since we assume that the password manager's anti-spoofing protection would first apply to its not gratuitously filling out the username and password on a spoofed site, we don't really also need it to not fill-out our OTP. It already protects us from divulging the first phase of our credentials. So, on balance, I think that keeping OTP's separate still provides optimal security. But it's certainly an interesting thought experiment!

Hi Steve,

I'm a long time listener of SecurityNow, and now work at VMware on ESXi. (Naturally) I just wanted to add a couple comments on the most recent ESXiArgs discussion.

Overall I thought the discussion was good, but I was a bit taken aback by the suggestion that VMware scan the internet for exposed ESXi servers. You've never suggested such a thing for any other vendor - even Microsoft or QNAP earlier in the episode.

*It definitely seems like it **could** work on a voluntary opt-in basis, but otherwise it seems untenable to me.*

It's been talked about quite a bit in the past in various contexts, but it's rather difficult at times to figure out who to contact if a vulnerable host is found.

While in theory I agree that it would be nice if there was a quick and easy way to contact system administrators of a particular system that's directly exposed to the internet, it's quite difficult in practice (especially in a way that doesn't have far-reaching privacy implications).

*Furthermore, there is a free version of ESXi which is popular with hobbyists and those just wanting to run VMs on a single server. ESXi is not **just** used by enterprises and IT professionals.*

*On a brighter note, I'm happy to report that starting in ESXi 8.0, **all** daemons and long-running processes are sandboxed **by default**, and we've also added additional hardening to make it harder to run ELF binaries that don't come from the installed base system packages.*

It was very neat to hear from someone at VMware. And as we know, he's right that I haven't previously been suggesting that all vendors proactively scan the Internet looking for vulnerable

versions of their own software. But I truly think that this is something that needs to change. I **have** mentioned on several occasions that there is little doubt that malicious actors and likely state-level agents **are** already scanning the Internet to create quick reaction databases of what is where so that when a high profile vulnerability is found targeted attacks can be rapidly launched. Why should only the bad guys have such useful databases?

Once upon a time, Internet scanning itself was considered a hostile act. When I first created the ShieldsUP! system — which was 24 years ago, in September of 1999 — I was periodically contacted by various network administrators who were wondering why my IPs were probing their networks. I explained that it was their own users who were requesting that I check on the state of their network's security. In most cases, back then, they asked that I not honor such requests, so ShieldsUP! Has always had a "do not scan" list of IP blocks which it refuses to probe.

But if such a system were to be launched today, 24 years later, not a single peep would be heard from anyone. The network administrators of the world have all collectively given up on the entire idea of identifying all of the random crap — or really any of the random crap — that's now flying across the Internet. I mean, there are still instances of Code Red and Nimda infected Windows NT servers sending out packets. It was in acknowledgement of this that I eventually coined the term "*Internet Background Radiation.*" Oh gee! A neutrino just whizzed through my body! Where did it come from? Who knows? Who cares?

So my point is, scanning the Internet was once unusual and attention getting. It is no longer. And we've recently been talking about the moves we're seeing of multiple governments beginning to take the security of their own nation's networks into their own hands.

So, our VMware listener is correct that I haven't been suggesting that other private vendors should be doing this, but I think that needs to change, too. It would be amazing and wonderful if QNAP were to maintain a list of publicly exposed instances of their always-buggy systems.

But our listener brings up a great point: What then? How do they contact the owners of the system that's publicly exposed. Bad guys don't need to contact anyone since they want to attack the systems. But good guys DO need to contact someone since they want to remediate the trouble.

The good news is, our VMware listener said: "*It's been talked about quite a bit in the past in various contexts, but it's rather difficult at times to figure out who to contact if a vulnerable host is found.*" And he's also correct about the privacy implications which follow from any attempt to somehow make endpoints identifiable to everyone on the public Internet.

So we're left with a conundrum created by the asymmetry of that fact that bad guys want to attack exposed systems while good guys only want to inform them.

I'm really glad that the VMware guy spoke up. Thank you.

Brad Jones / @bradjonesxr

Steve, I know we are all tired of talking about the security of our passwords but I am

interested in your thoughts on the following. Selecting a known BitWarden master password that is already considered somewhat secure, then running it through something like a Base64 calculator to generate an actual password used to unlock the vault. As you would likely never remember the password, anytime you need to unlock you would run the known password through the calculator to generate the vault password. For example, although clearly you would never use this, Password12345678 would generate the password you would use as your Master Password to be UGFzc3dvcmQxMjMONTY3OA== . There are many Desktop and Mobile applications that can complete this calculation without running the known password through an online encoder/decoder.

Algorithmic password generators are an interesting idea. The concept was to use an HMAC function which is essentially a keyed hash function. So the user would generate a single secret permanent key which would key the hash function. Then they would enter the domain they're visiting into the hash function and it would output an absolutely maximum entropy password. Since every domain would produce a different unique password, there would be no password reuse, and there would also be nothing to remember per domain since any domain's password could be recreated on demand.

This idea intrigued me so much that I wondered whether it might be possible to create a truly secure paper-based encryption system. Our long time listeners may recall that the idea I hit upon and developed was the idea of traversing a per-user customized Latin Square. I called the system "Off The Grid" since the passwords came from a grid and the system was offline and used no electricity.

So, anyway, Brad's right that yet another means of generating unique per-site passwords would be to employ some form of algorithmic system. But using a secretly keys hash function would be a good place to start. :)

Ascon

Last Wednesday, the U.S. National Institute of Standards and Technology (NIST) announced that a family of authenticated encryption and hashing algorithms known collectively as Ascon will be standardized for applications in lightweight cryptography. I'll have a bit more to say about that in a second.

This final selection was the result of a four-year competition, from 2019 to 2023, among competing proposals. This was the same process that led us to the selection of the Rijndael cipher to become the AES standard. So that's (if you'll pardon me) our "military strength" cryptography. Ascon has now become the standard for providing high 128-bit security for low end devices. For example, there's an entire suite of symmetric cryptographic Ascon functions for the Arduino.

NIST summarized the competitive selection process by writing...

"NIST has initiated a process to solicit, evaluate, and standardize lightweight cryptographic algorithms that are suitable for use in constrained environments where the performance of current NIST cryptographic standards is not acceptable. In August 2018, NIST published a call for algorithms (test vector generation code) to be considered for lightweight cryptographic standards with authenticated encryption with associated data (AEAD) and optional hashing functionalities. The deadline for submitting algorithms has passed. NIST received 57 submissions to be considered for standardization. After the initial review of the submissions, 56 were selected as Round 1 candidates. Of the 56 Round 1 candidates, 32 were selected to advance to Round 2."

So, what exactly does "lightweight crypto" mean? Rather than aiming for the highest conceivably needed security which would adequately protect data for the next several decades at least, NIST's Lightweight Cryptography competition was set up to select the contemporary crypto system that would be best suited for deployment on today's much more limited IoT systems. This means that, for example, while sharing the same 128-bit block length as the Rijndael AES cipher, because a cipher's block length really cannot be reduced much below 128 bits, the key lengths can safely be reduced by half to 128 bits. This allowed the use of a family of faster, more efficient and easier to implement algorithms which can run efficiently on much lighter weight hardware.

Some have wondered whether 128 bits is sufficient for contemporary security. The fact is, it's insanely plenty of security. We only went to 256-bit keys because why not if we have desktop and server machines and there's virtually no cost for doubling the key length.

So let's examine the implications of 128-bits for a minute. 2^{128} is 3.4×10^{38} . So that's a 34 followed by 37 0's. Now, say that we want to brute force the key's value and for the sake of argument say that it would be possible to completely test a candidate key at the full clock rate, say 3.4 GHz, of a GPU. It's not possible to actually do that since a GPU, fast as they are, still requires many clock cycles to get work done. But for the sake of this thought experiment, say

that it was possible to fully test one 128-bit key every clock cycle at 3.4 giga-tests per second. So that's 3.4×10^9 of these brute force tests per second.

To test all possible 128-bit symmetric keys, where there are 3.4×10^{38} possible keys, we divide 3.4×10^{38} by 3.4×10^9 , so we just subtract 9 from 38 yielding 29. This tells us that testing candidate keys at the rate of 3.4 billion tests per second — which no actual system is even capable of coming close to doing — we would need 10^{29} seconds to brute force all combinations.

Now, there are only 31.5 million seconds per year. If we triple that it brings us to 94.5 million which is close to 100 million, so we'll round up, which is 10^8 . So, 8 from 29 leaves 21. Thus, brute forcing a 128-bit secret key, at 3.4 billion guesses per second, would require roughly 3×10^{21} years. Given that we all generally live less than 10^2 years, we're left with plenty of security margin.

In other words, 128-bit security is an insanely ample amount of security. And now, thanks to this device-constrained cryptographic competition, we have this super-strong security available to lightweight embedded processors. NIST's original call for submissions explained their need as follows:

"The deployment of small computing devices such as RFID tags, industrial controllers, sensor nodes and smart cards is becoming much more common. The shift from desktop computers to small devices brings a wide range of new security and privacy concerns. In many conventional cryptographic standards, the tradeoff between security, performance and resource requirements was optimized for desktop and server environments, and this makes them difficult or impossible to implement in resource-constrained devices. When they can be implemented, their performance may not be acceptable.

Lightweight cryptography is a subfield of cryptography that aims to provide solutions tailored for resource-constrained devices. There has been a significant amount of work done by the academic community related to lightweight cryptography; this includes efficient implementations of conventional cryptography standards, and the design and analysis of new lightweight primitives and protocols."

The winner was Ascon, a family of authenticated encryption and hashing algorithms which are not only designed to be lightweight and easy to implement but also with added countermeasures against side-channel attacks. And Ascon was not only selected as NIST's new lightweight crypto standard, but it also previously won the same spot in the CAESAR competition which ran for five years from 2014 through 2019. CAESAR is the acronym for "*Competition for Authenticated Encryption: Security, Applicability, and Robustness.*" (Clearly one of those reverse engineered acronyms that's always a bit awkward.) So, Ascon is also CAESAR's primary choice for lightweight authenticated encryption.

I've mentioned authenticated encryption here. We've talked about the need for both encrypting the data for privacy and also authenticating the data to prevent its manipulation at any time. And assuming that the two steps of encryption and authentication are separate, we've examined

the question of in which order the two operations should be accomplished. The universal agreement is that you first encrypt and then apply authentication to the encrypted result. Then, on the receiving/decrypting end, the message is first authenticated to assure that it has not been modified in any way and only if it authenticates should the message then be decrypted. This prevents the use of some very subtle attacks that we discussed years ago.

But, the use of so-called "authenticated encryption" offers a better solution. As its name implies, authenticated encryption algorithms are able to perform both operations at the same time. And there's one very valuable extension to this known as Authenticated Encryption with Associated Data or AEAD for short. AEAD algorithms allow the so-called "associated data" to remain unencrypted and in the clear while still being encapsulated within the authentication envelope.

I have some experience using these algorithms, since SQRL's local on-disk identity storage format needed to have some local storage which would always be kept encrypted on disk, but would also make available user settings which needed to be available all of the time even without the user's key. And the whole thing needed to be tamper proof. So I chose to use AES-GCM which is an AEAD cipher. This perfectly resolved the need for having both secret and non-secret data that could not be altered without breaking the authentication of the whole.

In any event, AEAD ciphers are extremely useful. It's what NIST wanted and it's the contest winning technology that Ascon provides to IoT devices. Ascon also brings and provides similar features to hashing. Ascon was designed by a team of cryptographers from Graz University of Technology, Infineon Technologies, Lamarr Security Research, and Radboud University. The bullet points highlighting its benefits include:

- Authenticated encryption and hashing (fixed or variable output length) with a single lightweight permutation
- Provably secure mode with keyed finalization for additional robustness
- Easy to implement in software and hardware
- Lightweight for constrained devices: small state, simple permutation, robust mode
- Fast in hardware
- Fast in software: Pipelinable, bit-sliced 5-bit S-box for 64-bit architectures
- Scalable for more conservative security or higher throughput
- Timing resistance: No table look-ups or additions
- Side-channel resistance: S-box optimized for countermeasures
- Key size = tag size = security level (128 bits recommended)
- Minimal overhead (ciphertext length = plaintext length)
- Single-pass, online (encryption and decryption), nonce-based

The NIST requirements were for a key size no shorter than 128 and there could be a family of variants having differing parameter lengths. The design targets were interesting. NIST wrote:

"Submitted AEAD algorithms and optional hash function algorithms should perform significantly better in constrained environments (hardware and embedded software platforms) compared to current NIST standards. They should be optimized to be efficient for short messages (e.g., as short as 8 bytes). Compact hardware implementations and embedded software

implementations with low RAM and ROM usage should be possible. The performance on ASIC and FPGA should consider a wide range of standard cell libraries. The algorithms should be flexible to support various implementation strategies (low energy, low power, low latency). The performance on microcontrollers should consider a wide range of 8-bit, 16-bit and 32-bit microcontroller architectures. For algorithms that have a key, the preprocessing of a key (in terms of computation time and memory footprint) should be efficient.

The implementations of the AEAD algorithms and the optional hash function algorithms should lend themselves to countermeasures against various side-channel attacks, including timing attacks, simple and differential power analysis (SPA/DPA), and simple and differential electromagnetic analysis (SEMA/DEMA).

Designs may make tradeoffs between various performance requirements. A submission is allowed to prioritize certain performance requirements over others. To satisfy the stringent limitations of some constrained environments, it may not be possible to meet all performance requirements stated in the previous paragraph. The submission document should, however, explain the bottlenecks that were identified and the tradeoffs that were made.”

So, we now have a new family of power, memory and general resource efficient algorithms that have been studied and pounded on by many academics and cryptographers. GitHub offers implementations C, C with assembly, Python, JAVA and RUST. And there are RISC-V implementations as well. I have links to everything in the show notes:

GitHub also has an Ascon Suite of cryptographic functions all based on what's known as the Ascon permutation which is the core cryptographic function at the heart of Ascon. That GitHub page provides the open source code for the following functions:

<https://github.com/rweather/ascon-suite>

- Authenticated Encryption with Associated Data (AEAD)
- Hashing
- Pseudorandom Function (PRF)
- Message Authentication Code (MAC)
- HMAC-based Key Derivation Function (HKDF)
- Hashed Message Authentication Code (HMAC)
- ISAP AEAD Mode with Side Channel Protections
- Keyed Message Authentication Code (KMAC)
- Password-Based Key Derivation Function (PBKDF2)
- Pseudorandom Number Generation (PRNG)
- Synthetic Initialization Vector (SIV)
- Extensible Output Functions (XOF)
- Direct Access to the ASCON Permutation

And, in a related page, all of the same functions for the Arduino:

<https://github.com/rweather/ascon-arduino>

So, the industry now has a robust and seriously secure set of lightweight functions, which have

already been ported to many languages, platforms and hardware, which are tuned for and suited to the needs of embedded resource-constrained IoT-style devices.

<https://ascon.iaik.tugraz.at/index.html>

<https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/final-lwc-submission-requirements-august2018.pdf>

<https://csrc.nist.gov/projects/lightweight-cryptography>

<https://competitions.cr.yp.to/index.html>

https://github.com/ascon/ascon_collection

