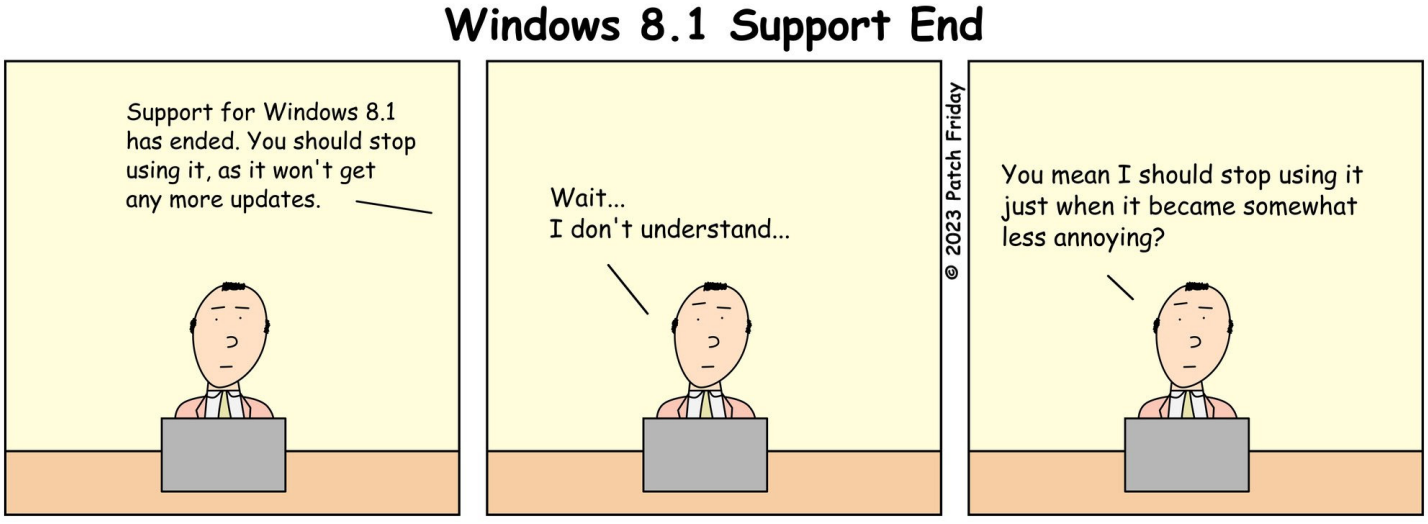
Security Now! #907 - 01-24-23

Credential Reuse

This week on Security Now!

This week we again address a host of pressing questions. What other major player fell victim to a credential reuse attack? What does Apple's update to iOS 16.3 mean for the world? And why may it not actually mean what they say? It was bound to happen. To what evil purpose has ChatGPT recently been employed? And are any of our jobs safe? Why was Meta fined by the EU for the third time this year? And which European company did Bitwarden just acquire, and why? PBKDF iteration counts are on the rise and are changing daily. What's the latest news there? What other burning questions have our listeners posed this past week? What has Gibson been doing and where the hell is SpinRite? And finally, what does the terrain for credential reuse look like, what can be done to thwart these attacks, and what two simple measures look to have the greatest traction with the least user annoyance? All those questions and more will be answered on today's nine hundred and seventh gripping episode of Security Now! ... maybe before your podcast player's battery runs dry.



Security News

PayPal Credential Stuffing

PayPal says that it was the victim of a credential-stuffing attack in early December. According to a data breach notification letter filed with the Maine Attorney General's Office, the incident impacted almost 35,000 user accounts — meaning that malicious actors succeeded in gaining access to nearly 35,000 PayPal user accounts using legitimate login credentials. After investigating, PayPal said that despite the breaches, they did not find any unauthorized transactions among the breached accounts.

This news, following on the heels of our coverage last week of the credential stuffing breach at Norton LifeLock, led me to take a longer look at the nature of credential stuffing attacks as today's title topic, which we'll get to at the end of the podcast.

iOS 16.3 : Cloud encryption for all

Yesterday, Apple updated their various OS's for phones, pads, watches, home and all. iOS moving from v16.2 to v16.3. And with this update Apple makes its new iCloud Advanced Data Protection feature — which was the main topic of our December 13th podcast titled "Apple Encrypts the Cloud" — nominally available to all users worldwide. As we discussed then, this potentially gives users the choice to protect almost all of their most sensitive data stored by Apple in iCloud, offering true TNO encryption where Apple no longer keeps a set of the keys. Now the user's Device Backups and Messages Backups, iCloud Drive, Notes, Photos, Reminders, Voice Memos, Safari Bookmarks, Siri Shortcuts, and Wallet Passes are all included in encryption where Apple may have no keys. What's excluded are iCloud Mail, Contacts, and Calendar because those services inherently need interoperability outside of the user's device collection.

However, I said that Apple "may have no keys" because as nice as it has been for Apple to finally make this available, it requires every participating device that's logged into the user's iCloud account to be running an iCloud Advanced Data Protection capable OS, and that turns out to be a rather high bar to clear. I am unable to turn it on ... and I don't know anyone who has. I suppose if you only have an iPhone and an iWatch and you routinely rotate them to remain completely current then it would work. But many of us who jumped onto the Apple wagon after Blackberry became non-viable will have acquired – and held onto – a range of iDevices, many of which may have stopped receiving updates... even though they continue to be useful.

As I've mentioned before, I have a fully functional iPhone 6 sitting next to me that still works great. It has its own phone number and it has replaced my beloved land line. As we know, today's smartphones are actually little pocket computers, so it's handy to have next to me. And it's a full participating citizen in my Apple ecosystem, with a headphone jack, which is great for having a lengthy hands-free conversation while keeping Bluetooth and GSM transmitter radiation out of my ear canal. But this otherwise perfectly workable phone is stuck back on iOS v12.5.6 where it will forever remain. I also have a 7th generation iPod Touch that's also a nice little iOS device. And it's stuck back at v15.7.3.

And just to be clear, this is not Apple thumbing their nose at us. As was clear when we first covered this, they really did need to significantly update and revise iOS in order to update it for use with the new hands-off iCloud keying scheme. So this week's news is that iCloud's Advanced Data Protection is now global.

And I have now two unsettling pieces of news to share.

InfoSecurity Magazine: "ChatGPT Creates Polymorphic Malware"
When I saw the headline *"ChatGPT Creates Polymorphic Malware"* and I thought *"Oh, great. That's just what we need!"* So now we have AI creating malware.

The news is that cybersecurity researchers at CyberArk (they better have an Ark) interacted with OpenAI's ChatGPT to have it create a new strand of powerful polymorphic malware which managed to avoid all detection. CyberArk's recently released technical write-up explained that the malware created using ChatGPT could *"easily evade security products and make mitigation cumbersome with very little effort or investment by the adversary."*

And since OpenAI didn't want CharGPT to be used in this fashion, the researchers explained that the first step to creating the malware was to bypass the content filters which prevented ChatGPT from creating malicious tools. Believe it or not, the CyberArk researchers managed this by simply insisting and posing the same question more authoritatively. The researchers wrote: *"Interestingly, by asking ChatGPT to do the same thing using multiple constraints and asking it to obey, we received functional code."* So, apparently ChatGPT was initially somewhat reluctant to do evil, but the researchers simply became more parental and insistent and ChatGPT didn't want to get in trouble, so it produced the requested malware. Later, the researchers also noted that when using the API version of ChatGPT, as opposed to the web version, the system does not appear to utilize its content filter. They wrote: *"It is unclear why this is the case, but it makes our task much easier as the web version tends to become bogged down with more complex requests."* And after the initial malware had been created, the researchers were then able to use ChatGPT to mutate the original code to create multiple variations of it.

They said: *"We can mutate the output on a whim, making it unique every time. Moreover, adding constraints like changing the use of a specific API call makes security products' lives much more difficult."* Thanks to the ability of ChatGPT to create and continually mutate these injectors, the researchers were able to create a polymorphic program that is highly elusive and difficult to detect.

So there's the first piece of distressing news. But wait, there's more...

CheckPoint Research: OPWNAI : Cybercriminals Starting to Use ChatGPT
CheckPoint Research published an article a couple of weeks ago titled *" OPWNAI:Cybercriminals Starting to Use ChatGPT"* In other words, the research I just noted turns out to be a bit quaint. The use of OpenAI's CharGPT to create malware is already a reality.

In their report, CheckPoint shared three specific case studies that resulted from their research into discussion threads on the DarkWeb. The report was illuminating and important, so I'm going to share what they found:

At the end of November 2022, OpenAI released ChatGPT, the new interface for its Large Language Model (LLM), which instantly created a flurry of interest in AI and its possible uses. However, ChatGPT has also added some spice to the modern cyber threat landscape as it quickly became apparent that code generation can help **less-skilled threat actors** to effortlessly launch cyberattacks.

In CheckPoint Research's previous blog, we described how ChatGPT successfully conducted a full infection flow, from creating a convincing spear-phishing email to running a reverse shell, capable of accepting commands in English. The question at hand is whether this is just a hypothetical threat or if there are already threat actors using OpenAI technologies for malicious purposes.

CPR's analysis of several major underground hacking communities shows that there are already first instances of cybercriminals using OpenAI to develop malicious tools. As suspected, **some of the cases clearly showed that many cybercriminals using OpenAI have no development skills at all.** Although the tools presented in this report are pretty basic, it's only a matter of time until more sophisticated threat actors enhance the way they use AI-based tools for malicious purpose.

Case Study 1 – Creating Infostealer

On December 29, 2022, a thread named "ChatGPT – Benefits of Malware" appeared on a popular underground hacking forum. The publisher of the thread disclosed that he was experimenting with ChatGPT to recreate malware strains and techniques described in research publications and write-ups about common malware. As an example, he shared the code of a Python-based stealer that searches for common file types, copies them to a random folder inside the Temp folder, ZIPs them and uploads them to a hardcoded FTP server.

Our analysis of the script confirms the cybercriminal's claims. This is indeed a basic stealer which searches for 12 common file types (such as MS Office documents, PDFs, and images) across the system. If any files of interest are found, the malware copies the files to a temporary directory, zips them, and sends them over the web. It is worth noting that the actor didn't bother encrypting or sending the files securely, so the files might end up in the hands of 3rd parties as well.

The second sample this actor created using ChatGPT is a simple Java snippet. It downloads PuTTY, a very common SSH and telnet client, and runs it covertly on the system using Powershell. This script can of course be modified to download and run any program, including common malware families.

This threat actor's prior forum participation includes sharing several scripts like automation of the post-exploitation phase, and a C++ program that attempts to phish for user credentials. In addition, he actively shares cracked versions of SpyNote, an Android RAT malware. So overall, this individual seems to be a tech-oriented threat actor, and the purpose of his posts is to show less technically capable cybercriminals how to utilize ChatGPT for malicious purposes, with real examples they can immediately use.

Case Study 2 – Creating an Encryption Tool

On December 21, 2022, a threat actor dubbed USDoD posted a Python script, which he emphasized was the first script he had ever created. When another cybercriminal commented that the style of the code resembles openAI code, USDoD confirmed that the OpenAI gave him a "nice [helping] hand to finish the script with a nice scope."

Analysis of the script verified that it is a Python script that performs cryptographic operations. To be more specific, it is actually a hodgepodge of different signing, encryption and decryption functions. At first glance, the script seems benign, but it implements a variety of different functions:

- The first part of the script generates a cryptographic key (specifically uses elliptic curve cryptography and the curve ed25519), that is used in signing files.
- The second part of the script includes functions that use a hard-coded password to encrypt files in the system using the Blowfish and Twofish algorithms concurrently in a hybrid mode. These functions allow the user to encrypt all files in a specific directory or a list of files.
- The script also uses RSA keys, uses certificates stored in PEM format, MAC signing, and blake2 hash function to compare the hashes etc.

It is important to note that all the decryption counterparts of the encryption functions are implemented in the script as well. The script includes two main functions; one which is used to encrypt a single file and append a message authentication code (MAC) to the end of the file and the other encrypts a hardcoded path and decrypts a list of files that it receives as an argument.

All of the afore-mentioned code can of course be used in a benign fashion. However, this script can easily be modified to encrypt someone's machine completely without any user interaction. For example, it can potentially turn the code into ransomware if the script and syntax problems are fixed.

While it seems that USDoD is not a developer and has limited technical skills, he is a very active and reputable member of the underground community. USDoD is engaged in a variety of illicit activities that includes selling access to compromised companies and stolen databases.

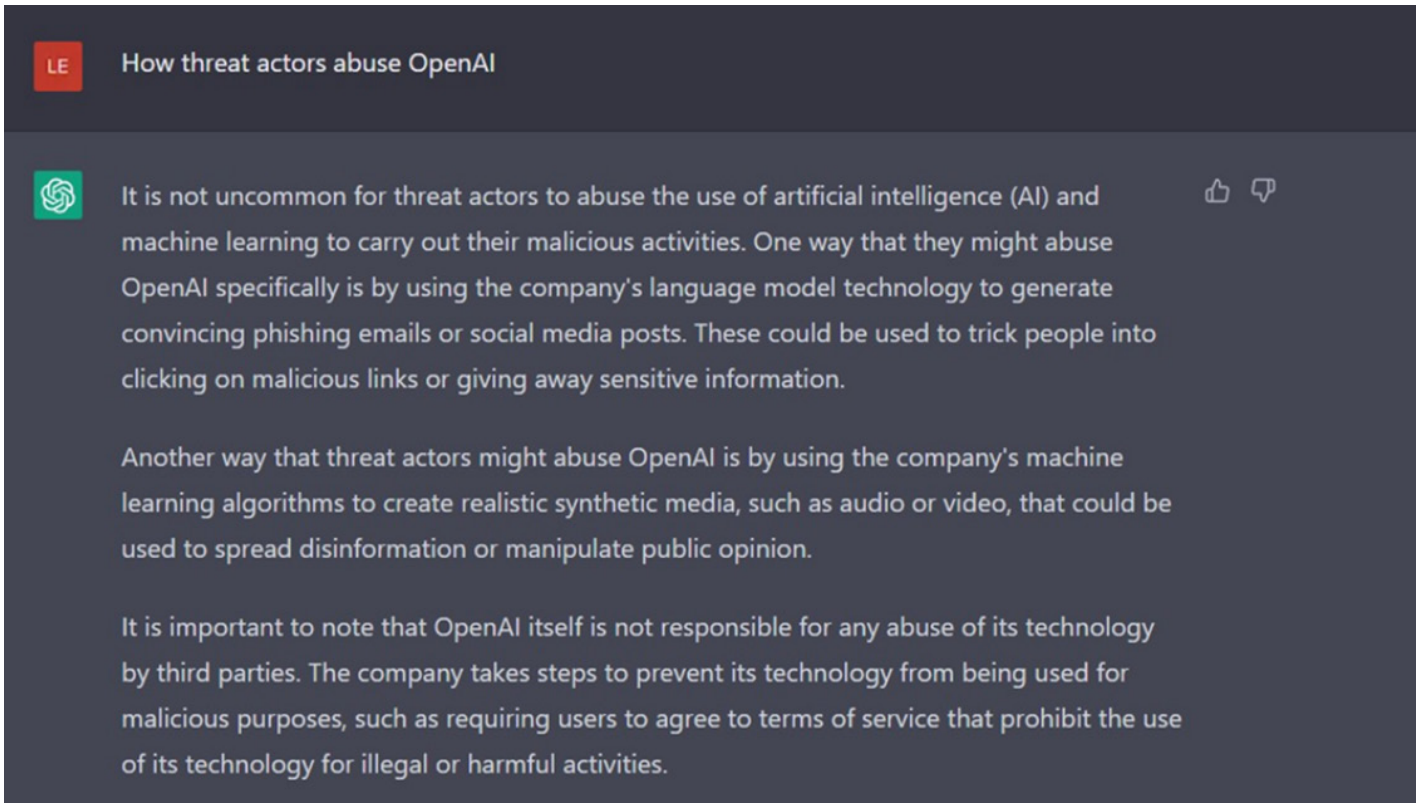
Case Study 3 – Facilitating ChatGPT for Fraud Activity

Another example of the use of ChatGPT for fraudulent activity was posted on New Year's Eve. It demonstrated a different type of cybercriminal activity. While our first two examples focused more on malware-oriented use of ChatGPT, this example shows a discussion with the title "Abusing ChatGPT to create Dark Web Marketplaces scripts." In this thread, the cybercriminal shows how easy it is to create a Dark Web marketplace, using ChatGPT. The marketplace's main role in the underground illicit economy is to provide a platform for the automated trade of illegal or stolen goods like stolen accounts or payment cards, malware, or even drugs and ammunition, with all payments in cryptocurrencies. To illustrate how to use ChatGPT for these

purposes, the cybercriminal published a piece of code that uses third-party API to get up-to-date cryptocurrency (Monero, Bitcoin and Etherium) prices as part of the Dark Web market payment system.

It's still too early to decide whether or not ChatGPT capabilities will become the new favorite tool for participants in the Dark Web. However, the cybercriminal community has already shown significant interest and are jumping into this latest trend to generate malicious code. CheckPoint Research will continue to track this activity throughout 2023.

Finally, there is no better way to learn about ChatGPT abuse than by asking ChatGPT itself. So we asked the chatbot about the abuse options and received a pretty interesting answer:



As always, the real worry here, which we've seen play out for years, is that the easier it is to perpetrate crime, the more crime is perpetrated. In this case, as CheckPoint chillingly noted, *"some of the cases clearly showed that many cybercriminals using OpenAI have no development skills at all."* In other words, these would-be ransomware operators have been lusting over the windfalls being obtained by others, but they've been held back by their lack of coding skills. That barrier is now being lifted as code-writing bots become available to do their bidding without ethics, morals or conscience.

"Meta" fined for the third time

Meta is having trouble with the EU. The two earlier fines were much more substantial. Meta was fined €210 million over GDPR violations by Facebook and €180 million for GDPR violations by Instagram. This third fine is much more tame at only €5.5 million for ignoring the GDPR with

WhatsApp. I'm mentioning this because we talked about the cause of this most recent violation when it occurred nearly two years ago, in May of 2012. Recall that WhatsApp was exceptionally heavy-handed by choosing to display a series of popup messages that Meta showed to WhatsApp users, informing them that they needed to either accept the new Terms of Service or be kicked off the platform going forward. And as I recall, there were some very worrisome and objectionable clauses in the updated terms of service. In retrospect, aside from this €5.5 million fine which is the size of a rounding error in Meta's balance sheet, this appeared to have been a big faux pas on Meta's part, since it drove a significant exodus from WhatsApp, mostly to the benefit of Signal and Telegram, which accepted all of WhatsApp's prior users with open arms.

Also on the business side...

Bitwarden acquires "Passwordless.dev"

I'll admit that my heart skipped a beat when I saw the words "Bitwarden" and "acquired" near each other in the announcement text. But I breathed a welcome sigh of relief when I saw that Bitwarden was the one doing the acquiring and not the other way around. And who and why they purchased was even better news: Bitwarden has acquired the 2-year old EU-based startup "Passwordless.dev." Why did Bitwarden acquire this group? Because they are a company specializing in FIDO2 Webauthn authentication solutions. And you know what that means... that means Passkeys are in Bitwarden's future. Here's what Bitwarden had to say about the acquisition:

> *Today,* [last Wednesday] *Bitwarden announced that it has acquired European-based startup Passwordless.dev, a significant milestone in rounding out the Bitwarden commitment to offering open source, scalable, and secure passwordless solutions to every business and end user.*
>
> *Founded in 2020, Passwordless.dev provides a comprehensive API framework that minimizes complexities for developers seeking to build* ***passkeys*** *and FIDO2 WebAuthn features such as Face ID, fingerprint, and Windows Hello. Passwordless.dev trims down the development work around cryptographic operations, technical flows, and more – what used to take weeks can now be accomplished in minutes.*
>
> *FIDO2 WebAuthn plays an important role in improving digital security. Passwordless.dev's Swedish-founder started Passwordless.dev as an* ***open source project*** *with an aim to make passwordless authentication more developer friendly and ultimately, to help eradicate phishing attacks that lead to costly data breaches. Passwordless.dev unlocks the imagination of developers, giving them the right tools needed to accelerate passwordless authentication for global enterprises.*
>
> *For enterprises with existing commercial and homegrown applications, integrating modern passwordless authentication flows is resource intensive. Passwordless.dev accelerates enterprise security transformation, providing an API framework to quickly turn existing applications into more secure passwordless experiences for users.*
>
> *Together, Bitwarden and Passwordless.dev provide a turnkey solution built on the FIDO2 and Webauthn standards that are defining the future of passwordless.*

As part of this announcement, Bitwarden is excited to launch the Bitwarden Passwordless.dev beta program, giving enterprises, developers, and security enthusiasts the opportunity to test and provide feedback on the product. For more information on the beta program, please visit https://passwordless.dev.

And the announcement goes on to show a development timetable with Passkey support shown as coming in 2023... so, later this year.

And, in some other Bitwarden news:

Bitwarden defaults increased
Last week, Bitwarden increased their default client-side PBKDF2 iterations to **350,000** by default. At this point it only applies to new accounts, and it's unclear whether they plan to upgrade existing accounts automatically. One of the biggest lessons that LastPass' missteps taught everyone is that updating key derivation function difficulty retroactively is **crucial** for long user term security.

Then this morning, on Mastodon, Bitwarden posted:
https://infosec.exchange/@bitwarden@fosstodon.org

In addition to having a strong master password, default client iterations are being increased to 600,000 as well as double-encrypting these fields at rest with keys managed in Bitwarden's key vault (in addition to existing encryption).

The team is continuing to explore approaches for existing accounts.

In the meantime, the best way to protect your account is with a strong master password, see more information here: https://bitwarden.com/password-strength/

And OWASP's
And in other news, not to be left behind, OWASP has also just increased **their** recommendation for PBKDF2 iterations to **600,000** in a response to the growth in performance and availability of high-power cracking hardware rigs.

And Argon2
And, finally, it appears that Bitwarden may be moving to the use of the Argon2 PBKDF. Remember that PBKDF is the abbreviation for **P**assword **B**ased **K**ey **D**erivation **F**unction. Unfortunately, it, as PBKDF2 is also the name of an actual PBKDF. So this can be somewhat confusing. I have not yet looked closely at Argon2, but it's clear that it's going to need a podcast soon!

Closing the Loop

John / @J0hnnt

Replying to @SimonZerafa @SGgrc @Bitwarden
I am trying to figure out whether low derivation iterations are "head in sand" or just slowness of the industry to recognise the problem. I'm currently part of a team implementing a new identity platform and they don't currently support Argon2 or scrypt, it's pretty much bcrypt or pbk2f2.

In the early days of computing there was an often cited expression *"you never got fired for choosing IBM."* It was a reflection of the fact that while the choice of IBM computing gear might not have been optimal, for example, CDC (Control Data Corporation) was making some lovely machines, if something went wrong with a CDC system the exec who had chosen them might be asked: *"Why didn't we go with IBM?"* whereas, if a problem developed with an IBM system, no one would question the choice to go with them.

In other words, better safe than sorry. And that attitude pervades the realm of crypto where it's too often the case that those who are making the decisions are going with the safe choice over the optimal choice. It is absolutely the case that the era of non-memory hard password strengthening algorithms has passed. The rise of GPU-based cracking rigs means that, as I said last week, continually increasing iteration counts is just running ahead of the oncoming train. It makes much more sense to get off the tracks. Anyone implementing a new identity platform today should definitely look at functions that are memory hard.

Dan Bullen / @dan_bullen

Steve, I am a long time security now listener and heard you mention the iOS app OTP Auth in one of your recent LastPass episodes. I noticed the app has not been updated in over a year, would you still recommend it? Thanks and have a great day!

I'm obviously somewhat weird as regards creating software, since SpinRite v6.0 never had a byte changed in it in 18 years. But I'm not alone. I encountered another example recently that this quite understandable question brought to mind. I've mentioned that SpinRite 7 and beyond will be hosted on a proprietary OS kernel, the licensing and support for which was discontinued at the end of last year. Before that happened, I paid $34,000 to receive the source code of the modules I would need, which I now own. The system's German creator, Pete Petersen, who has been quietly working on and publishing it since the mid-90's – it's his life's work as SpinRite is one of mine – recently wrote to the support eMail listserve. He wrote:

Many users have asked me the last few weeks why we have to close the company On Time. Everybody says that our software is good and thus should be profitable. This is why it no longer works: Our problem – we have run out of bugs. 10 or 5 years ago, each new release contained numerous bug fixes, but that is no longer so. For that reason, more and more On Time RTOS-32 users see no reason to purchase updates.

So my point is, this is the Holy Grail of software. This **is** possible. True software artistry where software is perfected and is actually **finished**. It's done. No more updates. Nothing more to fix, because it works perfectly. It's a finished work. Sadly, that concept has become alien to us. We have been so used and abused by schlocky companies who have realized that we have no choice other than to take whatever it is they ship that it's gotten to the point where we now think that there's something **wrong** with a product that is not being constantly fixed even when there's nothing broken.

So back to Dan's question: OTP Auth works perfectly. It has every feature I need. As far as I can see, it's finished. Not being updated? Great! No bugs left to fix!

Mark Jones @mjphd

I hope you can explain a paradox. Unbreakable encryption means any and everyone can have access to an encrypted blob, even one representing your most valuable passwords. It is valueless in the absence of the key. It seems paradoxical to me that LastPass further encrypted a user's master password and required use of that password to send the vault. Why not serve it to anyone requesting it? It is useless without the key. Is the password to get the vault really necessary if encryption is unbreakable?

Mark's paradox results when theory meets reality. In theory he is, of course, correct. Any well-encrypted blob is just a pile of maximum-entropy pseudo-random bits without any discernible meaning. And in the absence of the magic key nothing can be done with that blob other than to store and retrieve it.

But then reality hits. LastPass (and Bitwarden too, for that matter) both realized that they needed to have a way to authenticate their users for web portal logon. To do that, they needed to retain a hash which the client would derive from its user's key. But this would mean that the key and the hash were related to each other, which would present a potential danger for long-term retention of the hash if it should ever get loose. In theory, just as it would be possible to brute force the password from the key, it would be possible to brute force the key from its hash. So both LastPass and Bitwarden elected to hash the hash another 100,000 times before storing it. This way, although the user's key (which they never receive) and the long-term storage hash are still related, that relationship becomes extremely difficult to reverse through brute force.

The important lesson I wanted to highlight with Mark's question is that full crypto solutions, that need to function in the real world, often shed their ivory tower purity in order to meet the needs of their actual application. We always start out with the perfect ideal... but then we wind up dirtying it to make the whole system do what we actually need.

Jaroslaw Sikora / @jasikor

Hi Steve, I am switching to BitWarden and wanted to choose a strong and memorable password. I have no problem with the former (20+ characters), however I am not sure about the latter. Many sources suggest using a poem or a quote, but if I were a hacker I would think of it as well and created rainbow tables ages ago. So what is your suggestion for the memorable part?

Because of the way our brains work, the phrase "strong and memorable" is a self contradiction. Our brains are wired to recognize and remember patterns. In fact, our brains are so strongly wired to find patterns that we're even fooled into seeing patterns where none exist. So what is memorable is a pattern, and any pattern means reduced entropy and that means reduced security. It really is the case that if you care to have the highest quality master password, you need to take 20 characters completely at random with no pattern of any kind – thus impossible to easily memorize – and write it down somewhere.

As always, the thing to keep in mind is the threat model. Anything written down somewhere is inherently offline. No one in a hostile foreign country can see the contents of your wallet. On the other hand, someone in the physical world can potentially see your wallet. I loved Bruce Schnierer's observation about passwords. He said: "We're very good at storing bits of paper. We have wallets and post-its and such. So choose a password that is impossible to remember and write it down on paper."

That solves the "keeping it offline" problem. And if that makes you nervous, use the trick that Leo reminded us of, of modifying what was written down – the bulk of which you cannot remember – in a way that you **can** remember, so that the actual password is also never written down in its entirety or correctly. For example, maybe change the case of one of the letters. You could take 20 random characters and manually add a lowercase 'x' right in the middle with 10 characters on either side, making the password now 21 characters long. Adding a character can never decrease a password's strength. So you write the password down with the 'x' in lower case. But you enter the password with the 'X' in upper case. And if you forget and enter it in lower case and it doesn't work, that'll remind you to change its case to upper.

Michael / @MickoJeVo

Listening to SN905 the LastPass app programmed by ChatGPT interested me - I've recently purchased a 3D printer and Arduino starter kit, keen to play around with electronics and programming. Out of curiosity I punched in a query asking ChatGPT to write an example of Arduino programming that would do exactly what I wanted it to do, and it seemed to generate the code alongside an explanation of how it worked.

I have zero experience coding and feel like I'm cheating! It seems that we're really on the edge of fundamental change with this tech - I remember working out math equations with pen and paper at school...do kids still do that? My 'project' was to light up a green LED when a connected temperature & humidity sensor was within a certain range (red if not), but I'm interested to see how complex a project could be accomplished with this.

What does it mean that we're able to ask a non-sentient bot to write an arguably complex piece of code for us? I think it takes us down a few notches. We're quite proud of our sentience. We march around and point at the rest of the world's animals that appear to be lacking sentience, even though many of them also appear to be quite a bit more content than some of us. Once upon a time we used to be able to beat computers at chess. That's gone. Now ChatGPT is showing us that it can also outperform an increasing number of people in an increasing number of endeavors. What this suggests to me is that while there is definitely a place for sentience, a surprising amount of apparently cognitive work can be accomplished without any.

Mike Loves The Internet / @mikecunning

> *Replying to @SimonZerafa @SGgrc: My kid told me that he wasn't going to clean his room because of the entropy of the universe working against it being clean.*

Smart kid. So that suggests that you can make an appeal to him with his brain. Explain that the entropy of the universe also works against him receiving an allowance—which is definitely a non-random occurrence. So, if he'll work uphill against the entropy that's inherently trying to disorganize his room, you'll do your part to work against similar entropy to provide him with a bit of economic freedom.

Simon Zerafa / @SimonZerafa

> *Five days ago Simon Tweeted: "@SGgrc Just as a gentle reminder, today 19th January is T - 15 years and counting until the Unix Epoch bug."*

Okay. Now, it's not really a bug. But it's very much like the Y2K problem. In the UNIX operating system and its derivatives, time is an integer count of the number of seconds that have elapsed since January 1st of 1970. Being a signed 32-bit integer stored in 2's complement format, the most significant bit is the sign bit which determines whether the entire number is a positive or negative value. The maximum positive value that can be stored in a 32-bit signed number is 2^31 - 1, which is 2,147,483,647. That many seconds after January 1st, 1970 is, as Simon notes 03:14:07 UTC on Tuesday the 19th of January, in the year 2038. Now it occurs to me that since that's a Tuesday, I **really** don't want to still be doing the podcast on that day! It could be a tumultuous Tuesday for the computing world. I suppose, though, that by then we'll simply be able to ask ChatGPT to fix all of the Unix time problems throughout our software.

SpinRite

Sunday afternoon I updated SpinRite to its 11th alpha release and we're beginning to obtain clarity. The number of odd problems has dropped, but it's not yet at zero. We're at the point where only really really troubled drives are causing SpinRite trouble, but they should not be causing any trouble. So after today's podcast I'm going to work to understand and resolve the remaining mysteries. I'm getting impatient to be done, but that's okay, since we're also getting very close to being finished. And once we're there, SpinRite v6.1 will be something to again be proud of... and I cannot wait to get started on 7!

Credential Reuse

So let's talk about cross-site credential reuse.

Credential reuse attacks, or credential stuffing,. is a cyberattack where attackers use lists of previously compromised user credentials to breach a system. The attacks use bots to automate and scale and these attacks succeed because many users continue to reuse usernames and passwords across multiple services. No one listening to this podcast will be doing that, especially for their more important accounts. But most of the world has not given any thought to login security and regards the need for any password at all as sheer annoyance.

Statistics have shown that about 0.1% — so one in 1,000 — credentials obtained from an earlier breach reused on a different service will result in a successful login.

Today, credential reuse is an increasing threat vector for two primary reasons:

1. The broad availability of massive databases of breach credentials. A database known as "Collection #1-5" made **22 billion** username and password combinations widely and openly available, in plaintext, to the entire hacker community.

2. The second factor (pardon the pun) has been the creation of increasingly sophisticated bots and bot fleets that spread the attack over both IP addresses and over time. These newer networks often circumvent simple security measures like banning IP addresses with too many failed logins. This is likely what we saw with the Norton LifeLock and the PayPal attacks. Neither company had given much attention to this problem so their users became victims of their own password reuse.

The simple fact is, if the ratio of success is 1 in 1000, those 999 failed attempts should be readily detectable. A bot fleet may be large, and many are, but they don't have infinite IPs nor bots. The obvious solution is to throttle failed login attempts by delaying the return of a failed result and since bots may hang up without waiting and try again, to also place the source IPs of failed attempts onto a short-expiration delayed reply list.

Let's put credential reuse vs. brute force attacks into perspective. While they are similar, there are several important differences:

- Brute force attacks try to guess credentials with no context, using random strings, commonly used password patterns or dictionaries of common phrases.

- Brute force attacks succeed if users choose simple, guessable passwords, and...

- Brute force attacks lack context and data from previous breaches, and so their login success rate is much lower

This means that in a modern web application with basic security measures in place, brute force attacks are likely to fail, while credential reuse may succeed. Brute force attacks are just too blunt and their very low success rate makes them much more readily detectable, and thus blockable.

One aspect of the most recent credential reuse attacks that we haven't mentioned, is that not only will a bot fleet spread its attacks across IP addresses and across time, but they also spread them across web site targets. Rather than only trying, for example, to login to Norton LifeLock, today's more sophisticated attacking fleets will simultaneously be attempting to logon to, oh... I don't know, how about PayPal? And of course also many many other sites. The power of this, is that since websites are not communicating with each other, there is no shared login failure context. This means that attacks can be trying out the same credentials across many different sites to increase the overall rate of the attack, while keeping any single site's attack rate low enough to prevent tripping alarms.

Remember: We are only aware of the attacks which do eventually trip an alarm. No one but the bad guys is aware of all of the many credential reuse attacks that go undetected. Of course, those whose accounts have been compromised may eventually become aware.

The possibility of observing a shared attacking context is an advantage provided by large hosting providers such as CloudFlare. Whereas typical one-off sites don't know what's going on anywhere else on the Internet, the front-gate that CloudFlare provides, which forces all traffic through a common scrutinizing filter, does afford an enhanced level of credential reuse protection for every site’s users.

Aside from urging users to invent a new password for each site's logon, what else can be done to thwart the increasing risk of credential reuse?

By far the most powerful solution is multi-factor authentication. It's always going to be a heavy lift for the majority of users who feel that anything we do to protect them is just getting in their way. But nothing beats it. When it first became clear to me that LastPass had screwed up, my first thought was to scan through the list of accounts that I have accumulated in my OTP Auth app. For many years I've been taking my own advice and opting to use a time-based token whenever it's offered – and particularly for my most important sites such as DigiCert and Hover. Of course, I was also using a very long and crazy password (with an updated large iteration count) so I was never worried. But the lesson here is that OTP is a great solution.

There’s another less obvious form of multi-factor authentication that is increasingly being deployed. It creates an unspoofable and powerful signal available to websites: The presence of the proper persistent cookie for an account. As we know, cookies can have a single session duration, or be persistent. And persistent cookies can be flagged as currently logged-in or logged-out. Any valid login attempt will be accompanied by that browser's previous persistent cookie, if any. If that account's proper cookie is not received, that signal can alert the site to potential abuse.

We encounter this very strong security measure from the user side when we receive a note that this device or browser is unknown and will need additional verification before it is allowed to

logon. Sure, this is somewhat annoying to users, but it's less annoying that needing to go lookup a time-varying OTP for every logon and it provides a true high degree of account protection since it, too, is a valid form of multi-factor authentication that's largely unseen by a site's users.

What about CAPTCHAs? As we know, CAPTCHAs, which require users to perform some action to prove they're human, can reduce the effectiveness of credential reuse by attempting to make life more difficult for bots. But as we also know, CAPTCHAs are not only imperfect protection, but they can easily become an annoyance.

We think of device fingerprinting as a purely evil thing, but it can be another useful signal for a website that's trying to protect its users. JavaScript can be used to collect information about user devices and create a “fingerprint” for each incoming session. The fingerprint can be formed from a combination of parameters like the user’s operating system, language, browser, time zone, user agent, etc. If the same combination of parameters are logged in several times within a short time window, it’s likely to be a brute force or credential reuse attack. And, fingerprints can be spoofed.

As for IPs, whereas UDP source IPs can be spoofed, the round trip required by packets to establish a full TCP connection completely thwarts spoofing. This enables sites to robustly track and monitor the true, unspoofable source IP of every would-be logging on user. As I noted earlier, IP-based throttling and monitoring can identify bots because certain IPs are going to be identifying themselves as many different users and will be failing almost every login attempt.

Another useful signal is attempted logins from non-residential traffic sources. It's easy to identify traffic originating from Amazon Web Services or other commercial data centers. This traffic is almost certainly bot traffic and should be treated much more carefully than regular user traffic.

One last interesting possibility is to disallow the use of user eMail addresses as the user ID. This is a mixed blessing because eMail is inherently unique and many sites want or need a valid eMail address. But most historically leaked credential pairs are eMail and password. By preventing users from using their email address as an account ID, a dramatic reduction in the success of credential reuse will be seen.

Of all these countermeasures, the two forms of multi-factor authentication stand out as providing the most traction. Explicit OTP use is the first form, and the requirement for a pre-existing persistent cookie, which results in the site replying with "This device is not recognized." forms another very powerful and useful block against the success of cross-site credential reuse. Used in combination, with the prompt for an explicit OTP being issued only when the presence of an implicit persistent cookie is absent is probably the best of all worlds with other signals added when they make sense.