SECURITY NOW!

Transcript of Episode #907

Credential Reuse

Description: 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 907th gripping episode of Security Now!, maybe before your podcast player's battery runs dry.

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-907.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-907-lq.mp3

SHOW TEASE: It's time for Security Now!. Steve Gibson is back, baby. We're going to talk about the new Apple iOS 16.3 and why hardware keys may be not exactly what they mean. To what evil purpose has ChatGPT been employed recently? You might be surprised at its capabilities. And why Meta was fined by the EU for the third time this year. Plus then we'll talk about credential stuffing, or credential reuse, as Steve prefers. That's all coming up next on a thrilling, gripping edition of Security Now!.

Leo Laporte: This is Security Now! with Steve Gibson, Episode 907, recorded Tuesday, January 24th, 2023: Credential Reuse.

It's time for Security Now!. Yay. The show that performs jiu-jitsu on the hacker community with this guy right here, our sensei, Mr. Steven Gibson. Hello, Steve.

Steve Gibson: The podcast that shows you the problems you didn't even know you had.

Leo: You didn't even know. You were blissfully coasting along.

Steve: It answers questions for you that you dare not ask in a dark room.

Leo: It's our new Q&A format. I really like it.

Steve: And 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?

Leo: Uh-oh.

Steve: 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 now 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 907th gripping episode of Security Now!, maybe before your podcast player's battery runs dry.

Leo: Well, let's hope. Let's hope.

Steve: Let's hope.

Leo: Okay, Steve. I have a camera on you now so we can see your shining face.

Steve: Oh, joy.

Leo: And we can also see the Picture of the Week.

Steve: So we have a three-frame cartoon. The first frame shows somebody sitting in front of what looks like a little laptop, and someone outside the frame, off to the right, is saying, he's advising him: "Support for Windows 8.1 has ended. You should stop using it, as you won't get any more updates." And in the middle frame the guys looks a little confused, and he says: "Wait. I don't understand." And then in the final frame he says: "You mean I should stop using it just when it became somewhat less annoying?"

Leo: I've held on long enough.

Steve: Uh-huh.

Leo: Oh, lord, it's true.

Steve: Quite nice, as a user of Windows 7, not to be bothered by these...

Leo: Wait a minute. You're still using Windows 7?

Steve: Works great. I'm sitting right in front of it right now, yeah.

Leo: I'm hearing from so many people who are so pissed off about the end of Windows 8.1. It's like, well, it's working. Everything's fine. But you also understand why Microsoft doesn't want to support it forever; right?

Steve: I do. I'm a little annoyed that Chrome and Edge are now both telling me that I have to stop using Windows 7. I mean, I don't see them that often because I've switched back to Firefox as my main driver now. It's interesting because I tried to give Edge a real run because I really do like vertical tabs. And of course I have that with an add-on in Firefox. But I thought, I'm going to give Edge a try. And it was a couple weeks ago that it wasn't, like, running some pages correctly. I mean, it just - the page wasn't working in Edge. And I just thought, you know, you would think they could at least have their browser work. But no. Worked great in Firefox. Worked fine in Chrome. Didn't work in Edge. I thought, okay, fine.

Leo: Which is weird because it's based on Chromium. That's very strange.

Steve: Well, yeah, but of course they're always trying to do weird things.

Leo: They've changed it.

Steve: I mean, there's all this other crap that you get, like what am I getting points? I'm getting points.

Leo: Oh, Bing, Bing rewards. Give me a break.

Steve: I don't want points.

Leo: Yeah, yeah.

Steve: Just give me the page I want to see. But no, we're not going to give you the page. But we're going to give you points. So that's not why I'm here using a browser.

Anyway, PayPal says that it was the victim of a credential stuffing, as is the technical term. You and I both prefer the term "credential reuse," but we'll get to that. The credential stuffing attack in early December, according to a data breach notification letter which they filed, PayPal 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, the credentials for those accounts. I've got some neat statistics that we'll be talking about later. After investigating, PayPal said that despite the breaches, they did not find any unauthorized transactions among the breached accounts.

Of course 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.

Leo: I don't know if it's related, but GitHub just stopped taking PayPal contributions for their supporter thing that they do at GitHub. I wonder if GitHub knows something we don't know.

Steve: Yeah, why would you not? One of the things that I did encounter when I was doing some research for the podcast is that many times the credentials are logged as successful, but that's it. They are compiling them for the future. And of course that's why they hope to go undetected. And in fact I make the observation at the end of the podcast that of course all we're aware of are the attacks which have tripped alarms. There may well be attacks going on that no one is aware of because they're not tripping alarms, yet they're still verifying that they have got credentials now to get into a certain person's account in a certain location. And how would we know? So anyway, we're going to be talking about credential reuse at the end of the podcast.

Yesterday, Apple updated their various OSes for phones, pads, watches, home and all. iOS moved from 16.2 to 16.3. And with this update Apple makes its new iCloud Advanced Data Protection feature, which was the main topic of our December 13th podcast which we titled "Apple Encrypts the Cloud," now it's 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 the iCloud, offering true TNO (Trust No One) encryption where Apple no longer keeps a set of keys. Now the user's Device Backups, Message Backups, iCloud Drive, Notes, Photos, Reminders, Voice Memos, Safari Bookmarks, and Siri Shortcuts, oh, and also Wallet Passes, they're 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.

But I said 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'm unable to turn it on, and I don't know anyone who has. And I've heard you, Leo, say, you know, yeah, okay, but I've got stuff that isn't up to date, so I can't turn it on.

You know, I suppose if all you had was 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 - still miss that wonderful keyboard - will have acquired, and probably held onto, a range of iDevices over the years, many of which have stopped receiving updates, even though they continue to be useful.

As I mentioned before, I have a fully functional iPhone 6. Yup, it's right here sitting next to me. It still works great. It has its own phone number. It's replaced my beloved landline. And as we know, today's smartphones are actually little pocket computers, so it's handy to have next to me. And it's a fully participating citizen in my Apple ecosystem. It's got a headphone jack, which is great for having a lengthy hands-free conversation with headphones and a microphone which keeps Bluetooth and GSM transmitter radiation out of my ear canal. But this otherwise perfectly workable phone is stuck back on iOS 12.5.6 where it will remain forever. I also have a 7th generation iPod Touch that's a nice little iDevice. It's back on 15.7.3.

So anyway, just to be clear, this is not Apple thumbing their nose at us. This wasn't an arbitrary decision that they made in order to sort of tease people. As was clear when we discussed the technology which they've had to develop in order to pull this off, they really did need to significantly update and revise iOS in order for it to be able to operate with this new "they have no record of the keys in iCloud scheme." So the week's news is that iCloud's Advanced Data Protection is now global, and many people will be able to turn it on. But you have to have devices all able to be up at the latest and greatest.

Okay. When I saw the headline "ChatGPT Creates Polymorphic Malware," I thought, oh, great. That's just what we need. Now we have AI writing malware. The news is that cybersecurity researchers at CyberArk - and I like that name because maybe they're going to need to have an ark - interacted with OpenAI's ChatGPT to have it create a new strain of powerful polymorphic malware which did successfully manage to avoid all detection. Oh, great.

Leo: Wow.

Steve: Uh-huh. I mean, Leo. You only have to ask for it. You know, ChatGPT, please write some malware that will avoid all current detection. Oh, no problem.

Leo: Presumably it's getting it from other malware strains. But the fact that it evades detection is fascinating. It means it is original enough that it doesn't look like its predecessors.

Steve: Correct. So they recently released, the CyberArk guys, a technical write-up explaining 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 that's the phrase, "very little effort or investment by the adversary," that is pivotal here. This story is followed by one that's even more chilling we'll get to obviously in a second. But it's the leverage that this provides.

I mean, it's like how Rob, Rob said of his LastPass vault decryptor, he wouldn't have done a GUI except that ChatGPT made it so easy, he just sort of thought to - actually I think he said he wouldn't have done it at all if ChatGPT hadn't done it for him, largely. And then, hey, let's have a GUI. So, you know, give me a GUI. And out popped one. You know, there was a little bit of text overlapping problem, but boohoo, just tweak some positioning of the GUI. So anyway, this is significant. We have a couple more things we'll be talking about ChatGPT relative to here.

Anyway, and since OpenAI did not want ChatGPT to be used in this fashion, the researchers explained that the first step to creating the malware was to bypass the content filters which initially prevented ChatGPT from creating malicious tools. So, interestingly, there's some filtering going on there. But get this. Believe it or not, the CyberArk researchers managed to get past this by simply insisting, maybe used some exclamation points, and posing the same question more authoritatively.

They 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 interface version, the system does not appear to utilize its content filtering. So that was something that is not in the API. Probably they wanted to keep it pure in some sense. But for the general population, they've put something in that makes it less able to do what you can do if you talk to the API.

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." In other words, it was by, as they said earlier, by putting multiple constraints on the task, apparently it would wander off course and not give them what they wanted, as if it was like two magnets who had the same polarity sort of repelling each other and like glancing off. ChatGPT would sort of try not to make the malicious code. So these guys would put stronger constraints in to like keep it from veering off, and that's what was necessary in order to get it to do this, as if it was sort of trying not to, so they had to give it no choice. But doing that also bogged it down. Whereas using the API it didn't have any of that trouble.

So, and then, after getting functional malware, they were able to use ChatGPT to mutate the original code to create multiple, thus polymorphic, 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 in the code makes security products' lives much more difficult." So, they said, thanks to the ability of ChatGPT to create and continually mutate these injectors, they were able to create a polymorphic program that is highly elusive and difficult to detect.

So that's the first piece of two pieces of distressing news. Second is Checkpoint Research has been looking at the same question. And I got a kick out of the title of their posting about this. They called it "OPWNAI: Cybercriminals Starting to Use ChatGPT." They published an article a couple of weeks ago titled "OPWNAI," as I said, "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 ChatGPT to create malware is already a reality. In their report, Checkpoint shared three specific case studies that resulted from their research into discussion threads ongoing now on the Dark Web. The report was illuminating and important for our future, so I'm going to share what they found.

They said: "At the end of November 2022, OpenAI released ChatGPT, the new interface for its Large Language Model, 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." And again, that's the key, less-skilled threat actors, meaning bad guys can now get in on this. You don't have to know how to code as much.

"In Checkpoint Research's previous blog," they said, "we described how ChatGPT successfully conducted a full infection flow." So they were basically doing what the CyberArk guys did, using ChatGPT, experimenting with it to see if it could create malware. Yes. "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 a hypothetical threat, or if there are already threat actors using OpenAI technologies for malicious purposes.

"Checkpoint'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," they wrote, "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 purposes."

Okay. So three case studies. The first one, Creating an Infostealer. "On December 29th, 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." In other words, you know, using what these things do as a guide, asking ChatGPT to create something that does that. Wow. "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."

Okay, now, just to pause, I'll note that that's not anything that would be difficult for any coder to code in any language that they wanted to; right? But this is presumably somebody who doesn't know how to do that. So ask ChatGPT, and out comes some Python.

So they said: "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's worth noting that the actor didn't bother encrypting or sending the files securely, so the files might end up in the hands of third parties, as well." On the other hand, you just ask ChatGPT to please use HTTPS, and it will.

"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 (Remote Access Trojan) 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."

Study number two: Creating an Encryption Tool. "On the 21st of December 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 a glance, the script seems benign, but it implements a variety of different functions. The first part of the script generates a cryptographic key. Specifically, it uses elliptic curve cryptography and the curve Ed25519 that's used for 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 and so on.

"It's 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 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 aforementioned 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 could 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." And of course now, thanks to ChatGPT, his skills have been amplified.

And finally, third case study. "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, of course, 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 prices - Monero, Bitcoin, and Ethereum - as part of the Dark Web market payment system."

So they say: "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."

And they finish: "Finally, there is no better way to learn about ChatGPT's abuse than by asking ChatGPT itself. So we asked the chat bot about the abuse options and received a pretty interesting answer. So, question, how threat actors abuse OpenAI. ChatGPT replied: 'It is not uncommon for threat actors to abuse the use of artificial intelligence and machine learning to carry out their malicious activities. One way they might abuse OpenAI specifically is by using the company's language model technology to generate convincing phishing emails or social media posts. These could be used to trick people into clicking on malicious links or giving away sensitive information.'

"'Another way that threat actors might abuse OpenAI is by using the company's machine learning algorithms to create realistic synthetic media, such as audio or video, that could be used to spread disinformation or manipulate public opinion. It is important to note,' writes the bot, 'that OpenAI itself is not responsible for any abuse of its technology by third parties. The company takes steps to prevent its technology from being used for malicious purposes, such as requiring users to agree to terms of service' - oh, that'll solve the problem - 'that prohibit the use of its technology for illegal or harmful activities.'"

Wow. As always, the real worry here, which we've seen play out for years, is that the easier it is to perpetrate a crime, the more crime will be 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.

Leo: Wow.

Steve: Leo?

Leo: I have to wonder, though, how good really, I mean, look, you could easily write that code that searches for a file, bundles it up, and sends it out. That's not...

Steve: Absolutely. Absolutely.

Leo: That's not complicated code.

Steve: But crypto, there was some serious crypto there.

Leo: That's interesting, yeah.

Steve: It was running crypto and getting a message authentication code and using public key crypto. And it was doing it all correctly.

Leo: You know, I mean, it's like Copilot, GitHub's Copilot, which writes code for you. It also uses GPT to do it. I presume it's getting that code of some kind from stuff it's scanned into its databases; right?

Steve: Well, and Leo, really, production coders spend a lot of time cutting and pasting; right?

Leo: Yeah. That's right.

Steve: We go find either our own previous work or somebody else's previous work and say, well, this chunk of code does what I need, so drop it in over here.

Leo: Right, right. And that's all this is probably doing. But still.

Steve: And then you, you know, you glue it together.

Leo: Yeah.

Steve: Yeah, but again, accessibility matters. That's what we're seeing.

Leo: Yeah, yeah, ease, yeah, yeah.

Steve: Ease, yes.

Leo: Point of access.

Steve: Yup. So Meta is continuing to have trouble with the EU. They got hit with a third fine of the year. The two earlier fines were, however, much more substantial. They were fined 210 million euros over GDPR violations by Facebook, and 180 million euros for GDPR violations being made by Instagram. The third fine is a lot more tame, it's only 5.5 million euros, 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, back in May of 2021. Recall that WhatsApp was exceptionally heavy-handed by choosing to display a series of pop-up messages that Meta was showing to WhatsApp users informing them that they needed to either accept the new terms of service or be kicked off the platform in the future.

And as I recall, I did a little bit of quick digging to see if I could find - I'm sure that part of the objection was there were some very worrisome and objectionable clauses in the updated terms of service, so it wasn't just the people who had to, like, say yes or else. But what it was that Meta was asking people to say yes to was objectionable. And in retrospect, aside from this 5.5 million euro fine which is the size, you know, of a rounding error in Meta's balance sheet, this appeared to have been overall, I mean, this whole crusade of theirs to be a big mistake on Meta's part since it drove a significant exodus away from WhatsApp, mostly to the benefit of Signal and Telegram, which accepted all of WhatsApp's prior users with open arms. So lesson learned, I hope.

Also on the business side, Bitwarden has acquired Passwordless.dev. And I'll admit that my heart skipped a beat when I saw the words "Bitwarden" and "acquired" near each other in the announcement text.

Leo: Oh, yeah, no.

Steve: I thought, nooooooo. No, please. But I breathed a welcome sigh 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 two-year-old EU-based startup Passwordless.dev. It's kind of cool, too. If you go to Passwordless.dev, it's nice big familiar Bitwarden blue, and they've salted Bitwarden a few times on the page to let you know that they're now a Bitwarden company. So why did they acquire the group? Because they are a company specializing in FIDO2 WebAuthn authentication solutions.

Leo: Oh.

Steve: Uh-huh.

Leo: Passkeys. Passkeys.

Steve: Yup, yup. And that's what that means. Passkeys are in Bitwarden's future. So, okay. So here's what Bitwarden had to say in their announcement of this. They said: "Today," which was 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." What is that we were talking about cutting and pasting, Leo?

"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."

So, you know, this is very cool. I mean, if nothing else, Bitwarden's acquisition moves Passwordless.dev onto everybody's radar. Suddenly it's like, what? What is this? There's an open source solution that will allow us to drop in a module and immediately get Passkeys or FIDO2 WebAuthn authentication? Great.

So they wrap up, saying: "Together, Bitwarden and Passwordless.dev provide a turnkey solution built on the FIDO2 and WebAuthn standards that are defining the future of Passwordless." Is that going to be a noun now? Oh. "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," and that's the site: Passwordless.dev. And the announcement goes on to show a development timeline with Passkeys support shown as coming this year, 2023. So yay.

In some other Bitwarden news, last week Bitwarden increased their default client-side PBKDF2 iterations to 350,000. At this point this only applies to new accounts, and it's unclear whether they plan to upgrade existing accounts automatically, though as we'll see in a second they're aware of the issue. One of the biggest lessons that LastPass's missteps taught everyone is that updating key derivation function difficulty retroactively, or retrospectively, is crucial for long-term user security.

Then this morning, when I was putting the finishing of the podcast together, their tweet over on Mastodon - do you call it a tweet? What do you call it, Leo?

Leo: It's a toot.

Steve: A toot. Great. Their toot. Like beans?

Leo: Yeah, well, it's an elephant.

Steve: Oh, of course, perfect.

Leo: Yeah.

Steve: So they tooted. They said: "In addition" - as I was saying, it was 42 minutes old when I saw it this morning. "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)." Their toot goes on: "The team is continuing to explore approaches for existing accounts." So they get that.

And they finished: "In the meantime, the best way to protect your account is with a strong master password. See more information here." And they have a nice-looking password strength meter. It's at Bitwarden.com/password-strength. And I poked around at it a little bit, and it does a good job. It's not impressed by my haystacks-style passwords, you know, with a lot of repetition in them, which would be difficult to brute force, because it recognizes there's not a lot of raw entropy there. But so it's a passable meter. And it did like the password that I'm using as my master password over on Bitwarden.

Leo: You probably said it, but I'll say it again, Bitwarden is a sponsor. We should disclaim that.

Steve: Oh, yes, yes. I'm glad you did.

Leo: But we do like them, too.

Steve: Everybody knows, yeah.

Leo: Yeah, yeah, yeah.

Steve: Okay. And OWASP. In other news, not to be left behind, OWASP has also just increased their recommendation for PBKDF2.

Leo: Oh.

Steve: Remember it was 350,000. Now it's also 600,000.

Leo: All right. I set it to two million. I set it to the max. I guess I'm good for a while.

Steve: That's what I would recommend.

Leo: Yeah, why not? Yeah.

Steve: Yes. So this is in response to the growth in performance and availability of high-power cracking hardware rigs. And, finally, it appears that Bitwarden may be moving to the use of Argon2 PBKDF. Okay, now, remember, PBKDF itself is an abbreviation for Password Based Key Derivation Function. Unfortunately, it as PBKDF2 is also the name of an actual PBKDF. So that can be a little confusing. I've not yet looked closely at Argon2, but it's clear that it's going to need to have a podcast of its own here coming up pretty soon because we need to know what that's all about.

Leo: And you remember one of our listeners did do an Scrypt plugin, a pull request for that. And I guess they - looks like they don't want to opt for that. Is there a reason you would choose Argon2 over Scrypt? They're both memory hard; right?

Steve: They are. And again, not having looked at them, I can't comment on either of them. I implemented Scrypt myself for SQRL. So I know all about it. But presumably Argon2 is, you know [crosstalk].

Leo: There was some competition some couple years back.

Steve: Yes, yes, yes. There was a key-strengthening competition, and Argon2 was the winner, although its initial implementation had a side-channel leakage problem because the path it took was dependent upon the password that it was being asked to strengthen. Well, that's also the case with Scrypt. As far as I recall, my decision was that because this was being done on the client only, side-channel wasn't a problem because if you already had some compromise on your client, the jig was up. They could just be logging your keystrokes and grab the password when you type it in.

Leo: Right.

Steve: So that was a deliberate decision on my part. But there are variations of Argon2 since which solve that problem. So I'm sure the one that's being used is the one that for, you know, why not get side-channel protection while you're at it.

So I have some feedback from our listeners that gives me a chance to address some other broader concerns or issues. John sent - he was replying to Simon Zerafa, me, and Bitwarden. And he said: "I'm trying to figure out whether low derivation iterations are 'head in sand' or just slowness of the industry to recognize 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 PBKDF2."

And so, you know, he's like, what's the problem? And I was reminded of the early days of computing, Leo, which you and I both have lived through.

Leo: I remember those years.

Steve: Exactly. There was the often cited expression, "You never got fired for choosing IBM." That 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 mainframe machines at the time, 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, you know, safe as in, well, no one will fire me if I choose this. Well, you know, that's PBKDF2 because they can point to everybody else who's using it. Although I would argue its day has come and gone. You know, it's absolutely the case that the era of non-memory hard password-strengthening algorithms is over. The rise of GPU-based cracking rigs means that, as I said last week, continually increasing iteration counts is just running ahead of an oncoming train. It makes much more sense to just get off the tracks. Anyone implementing a new identity platform today should definitely look at functions that are memory hard.

And, you know, I noted that one of the things that LastPass was doing, I got this when I went through that detailed crypto document of theirs for last week's podcast note, was that they did a whole bunch of PBKDF2 and then some Scrypt. It's like, why not - there's no reason not to do all. Both. More. Some. Whatever.

Leo: Do it all. Do it all.

Steve: That's right. Just throw the kitchen sink at it. Dan Bullen said: "Steve, I'm 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."

Okay, now, I'm obviously somewhat weird as regards creating software, since SpinRite 6 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 of his 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 that I would need, which I now own.

The system's German creator, Peter Petersen, who has been quietly working on and publishing this operating system kernel since the mid-'90s, you know, it's his life's work, much as SpinRite is one of mine. He recently wrote to the support email list in order to explain what happened. He said: "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. Ten or five years ago..."

Leo: That's awesome.

Steve: Yes, 10, he said, 10 or five 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." Okay. So my point is this. 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. But sadly, that concept has become alien to us. We have been so abused and used 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. Yeah, and surprising. I get it.

Mark Jones sent: "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's 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's useless without the key. Is the password to get the vault really necessary if encryption is actually unbreakable?"

Okay. So 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 pseudorandom bits without any discernible meaning. And in the absence of the magic key, nothing can be done with that blob other than to store it and retrieve it. But then reality hits.

LastPass, and as a matter of fact Bitwarden, too, 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. And of course it did. 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 them. 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 in practice.

Jaroslaw Sikora said: "Hi, Steve. I'm switching to Bitwarden and wanted to choose a strong and memorable password. I have no problem with the former, 20-plus characters," meaning the strong part. "However, I'm 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's your suggestion for the memorable part?"

Okay. And this is one tweet of many that I received, people saying, you know, need some more direction here. Okay. 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.

As always, the thing to keep in mind is the threat model. Anything written down somewhere is inherently offline. So that's a biggie; right? No one in a hostile foreign country can see the contents, for example, of your physical wallet in your back pocket. On the other hand, somebody in the physical world can potentially see your wallet.

I loved Bruce Schneier's observation about passwords. He once said something to the effect of we're very good at storing bits of paper. We have wallets and bits of paper, wallets for storing bits of paper and such. So choose a password that is impossible to remember and write it down on paper and stick it in your wallet. So that solves the keeping it offline problem. And if that makes you nervous, you can use the trick that, Leo, you reminded us of, of modifying what is written down, the bulk of which you cannot remember.

Leo: The problem with - so again, threat model. If you have in your wallet a piece of paper, I'm not worried about somebody stealing my wallet. Let's say I have it in a drawer. I've got to pull it out, put it there, and type it in every time I want to log into that password manager. That seems to me, I mean, that's something if you're in private you could do. But if I'm at work and I'm doing that, that seems to me I'd rather have something in my head.

Steve: I agree. And so the case is, if it's in your head, it is unquestionably lower entropy. I mean, because there are patterns. Even people who, like, use Diceware to choose words, you know, there are, what is it, 7,776 words, I think, in that dictionary. And so you can take that, if you use four of them to the fourth power, it's very good. But it's what the brute forcers are doing to...

Leo: It's pretty guessable, too; you know?

Steve: It's what they're going to run through in order to do...

Leo: Now, if you then added your childhood phone number interspersed with your childhood zip code, then you'd be fine; right?

Steve: Well, and so the example that I sort of liked was imagine that, if it did work for you, to take 20 random characters. Start with those. Add a lowercase "x" or something, 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 down the password with the "x" in lowercase. But you enter the password with the "X" in uppercase. And if you forget and enter it in lowercase and it doesn't work, that'll remind you to change the case to upper. I mean, you know, there are all these different solutions.

And the fact is, any of these things that we talk about on this podcast, aside from monkey123, they're all good ideas. They're way better than the typical password that the typical user chooses. We've shown lists of passwords that have been captured, like Troy Hunt's list. And it's full of 123456. And you have to really wonder what website even allows you to enter that these days. There ought to be some way, some block of code that looks, you know, JavaScript on the web page that says, what, really? No. You've got to add a seven.

Leo: Wait, c'mon, man.

Steve: C'mon, c'mon, man. Michael said: "Listening to SN-905. The LastPass app programmed by ChatGPT interested me." He said: "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." He says: "I remember working out math equations with pencil and paper at school. Do kids still do that?" He said: "My project was to light up a green LED when a connected temperature and 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."

Okay. So, you know, 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. And Leo, as we were saying, you know, a lot of coding is reusing stuff that you learned years ago. That doesn't require the edge of creativity in order to make it happen.

Leo: Right, right. I guess it depends why you're coding. If you're a code monkey, and your boss wants you to write 30 pages of code every month or day or week...

Steve: Well, Elon. Elon is...

Leo: That's what I was thinking of. You might want to, you know, enlist some support, some help.

Steve: Yeah.

Leo: I can understand that, yeah.

Steve: And you could say, ChatGPT, this is too succinct. I need you to solve the problem...

Leo: Yes, yes, verbose.

Steve: ...with a lot more - exactly. What's the hurry here? We don't need this to actually perform quickly. Just make it work.

Leo: I love it.

Steve: You know, spend some time doing other things. So someone posting as Mike Loves The Internet, he was also replying to Simon Zerafa and me. He said: "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."

Leo: It's inevitable. I mean, why even try?

Steve: So I replied: "Smart kid."

Leo: Yeah.

Steve: "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."

Leo: You're just going to spend it, and I'll have to give you more next time.

Steve: Receiving an allowance is definitely a non-random occurrence.

Leo: Very good. Steve, you'd have made a good dad.

Steve: 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.

Leo: You are good.

Steve: And lastly, Simon tweeted. Five days ago he said: "@SGgrc Just a gentle reminder. Today, 19th January, is T minus 15 years and counting until the Unix Epoch bug."

Leo: Oh, 2038, yes.

Steve: Yup. 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, 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 taken to be positive or negative. The maximum positive value that can be stored in a 32-bit signed number is 2^31 minus 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 it's a Tuesday, I really don't want to still be doing the podcast on that day.

Leo: It's a Security Now! day.

Steve: 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 and be done with it.

As for 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 quite yet at zero. We're at the point where only really, really troubled drives that are really acting oddly are causing SpinRite any trouble, but they should not be causing any trouble. So after today's podcast I'm going to return to work to understand and resolve the remaining mysteries. I'm getting impatient to be done with this, but that's okay since we're also getting very close to being finished. And once there, SpinRite v6.1 will be something to again be proud of. And the truth is I can't wait to get started on 7.

Leo: And now, let's talk about stuffing your credentials.

Steve: Let's think about cross-site credential reuse. Okay. So obviously, credential reuse attacks, a.k.a. credential stuffing, is a cyberattack where attackers use lists of previously compromised user credentials to breach accounts on a system. The attacks use bots to automate and scale, and these attacks succeed because, as we know, unfortunately, none of our listeners here, no one listening to this podcast, but many other users continue to reuse usernames and passwords across multiple services. Okay. So again, not our listeners, but most of the world has not given any thought to login security. I mean, you have to sort of be on the inside to understand this idea. Most of the world still has like their password which they use everywhere.

Statistics have shown that about 0.1%, so that's one in 1,000, credentials obtained from an earlier breach, when reused on a different service, will result in a successful login. One in a thousand. So that's statistically way better than you're going to get with just brute force guessing.

So today credential reuse is an increasing threat vector for two primary reasons. First, there is broad availability of massive databases of breached credentials. There's a database, we referred to it a few years ago, known as "Collection #1-5." That made 22 billion username and password combinations widely and openly available, in plaintext, to the entire hacker community. So they have them.

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 that have too many failed logins. This is likely what we saw with the Norton LifeLock, and then more recently the PayPal attacks. Neither company had given sufficient attention to this problem, so their users became victims of their own password reuse. Technically, yeah, their fault that they reused the same password on Norton LifeLock as they did somewhere else, some site that was compromised, or also on PayPal similarly.

So the simple fact is, if the ratio of success of using a reused credential is one in a thousand, those 999 other 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, after timing out for a short period, saying, okay, well, that didn't work, this website wants to keep me on hold here, I'm just going to disconnect and make a new connection, you also need to look at the source IPs of the failed attempts, adding them to a short expiration delay reply list so that once an IP has been identified as malicious, then always delay reply.

Okay. So putting credential reuse and brute force attacks into perspective, while they're similar, there are several important differences. Brute force attacks try to guess credentials using no context, just using random strings, hopefully the most likely strings first, commonly used password patterns or dictionaries of common phrases. Brute force attacks succeed if users choose simple, guessable passwords. Hopefully nobody still is, but we know that's not the case. And brute force attacks lacking context and data from any previous breaches, just they end up with an overall much lower success rate. What that tells everybody is since we've got 22 billion usernames and passwords that are being commonly used and reused, try those first.

So what this means is that in a modern web application with basic security measures in place, brute force attacks are far more likely to fail, while credential reuse is far more likely to succeed, at least in part. Brute force attacks are just too blunt, and their very low success rate makes them stand out, makes them much more readily detectable, and thus blockable.

One aspect of the most recent credential reuse attacks that we have not mentioned is that not only will a bot fleet spread its attacks across IP addresses and across time, but they also spread them across website targets. Rather than only trying, for example, to log into Norton LifeLock, today's more sophisticated attack fleets will simultaneously be attempting to log onto, oh, I don't know, how about PayPal? And of course 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 at the same time to increase the overall rate of all of the attack, while keeping any single site's attack rate low enough to prevent tripping any alarms.

Remember, and as I said earlier in the podcast, we're only aware of the attacks which do eventually trip an alarm. No one but the bad guys is aware of all the many credential reuse attacks that remain undetected. And of course those whose accounts have been compromised may eventually become aware when bad guys actually do use them. And I did, as I said earlier, I did encounter some mention of the idea that an inventory of accounts were being accrued and collected by the bot fleets. It may very well be that those running the fleets are not interested in doing the attacking. They are building an inventory of known available logins for resale on the Dark Web.

The possibility of observing a shared attacking context, that is, multiple sites being spread around the 'Net have no shared attacking context. But the possibility of observing a shared attacking context is an advantage provided by single large hosting providers like Cloudflare. Whereas one-off sites don't have any idea what's going on anywhere else, the front gate that Cloudflare provides, which forces all traffic through a common scrutinizing filter, affords an enhanced level of credential reuse protection for every one of the site's users who are behind that gate. So that's something sort of to keep in mind, another benefit.

Aside from urging users to invent a new password for each site's login, which is an easy task for this podcast's listeners, but is a difficult lift for most of the Internet, what else can be done to thwart the increasing risk of credential reuse? Because, I didn't mention this before, it is on the rise. By far the most powerful solution is multifactor authentication. And of course I don't mean SMS messages or email. What we really want is a time-based token. You set it up once, and then there's no transaction that occurs during the login attempt other than you providing the ever-changing six-digit code to the site.

Now, of course, I get it, doing this one-time password multifactor authentication is 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. An example is when it first became clear to me that LastPass had actually 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 have accumulated a large number of those, opting to use this time-based token even though, yes, it's more pain, whenever it's offered, and particularly for my most important sites like DigiCert and Hover, both where I use them. Of course, I was also using a very long and crazy password with an updated large iteration count so I was never actually worried. But the lesson here is that one-time passwords is a great solution.

There is another less obvious form of multifactor authentication that is increasingly being deployed. And I'm really glad to see it. It creates an unspoofable and powerful signal available to websites, and that is the presence of the proper persistent cookie for an account. As we know, cookies can have either 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, even if it's flagged as logged out. If that account's proper cookie is not received, that signal can alert the site to potential abuse.

And 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 log in. Sure, that's somewhat annoying to users, but it's less annoying that needing to go look up a time-varying one-time password for every logon, and it provides a true high degree of account protection since it, too, is a valid form of multifactor authentication that's largely unseen by a site's users.

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

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 running on the page can be used to collect information about the user's devices and to fingerprint it, what, the OS, the language, the browser, the time zone, the user agent and so forth. If the same combination of parameters are logged in several times within a short window, or attempted to be, it's likely to be a brute force or credential reuse attack. And the problem is fingerprints can be spoofed. A smart bot will make up those things and rotate them so that they appear to be different user agents each time.

What about IPs? Whereas UDP source IPs can be spoofed, the roundtrip required by packets to establish a full TCP connection completely thwarts IP 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. You know, 999 out of a thousand.

Another useful signal is attempted logins from non-residential traffic sources. For example, it's easy to identify traffic originating within 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 email addresses as the user ID. Of course this is a mixed blessing because email is inherently unique, and many sites want or need a valid email address. So making it also double purpose as the user's login username is handy. But most historically leaked credential pairs are email and password. By simply preventing users from using their email address also as their account ID or username, a dramatic reduction in the success of credential reuse will be seen and has been seen in trials.

So of all these countermeasures, the two forms of multifactor authentication basically stand out as providing the most traction. Explicit one-time password use is the first form, and the requirement for a pre-existing cookie, which the browser will send back, if it's missing, that's the thing which results in the site replying "This device is not recognized." This technology forms another very powerful and useful block against the success of cross-site credential reuse.

And when used in combination, so that the prompt for an explicit one-time password is only being issued when the presence of an implicit persistent cookie is absent, is probably the best of both worlds. Users get strong protection, the site's not going to have problems being abused, and certainly as soon as any bots realize that they are never succeeding because they're not providing a persistent cookie which they have no way of acquiring, they'll wander off and go somewhere else.

So credential reuse, a problem. And as I thought through all this, this is - aside from the need to switch password managers when the password manager you have been using has given you reason to believe that all of your account passwords may have been compromised, this is the one sane reason for changing passwords; right? I mean, in general, this whole change your password every six months, and you can't use any of the last four that you used before, that's a pain in the butt. But it is the issue of breach and reuse of those credentials that is what induces the need to change passwords. So, you know, the one piece of feedback that I really had from everybody, Leo, in the last few weeks is, boy, changing passwords everywhere is really a pain.

Leo: Ugh. Ugh. Hope we never have to do that again.

Steve: Well, with 600,000 iterations of PBKDF2 and a memory-hard algorithm coming soon, we probably won't have to.

Leo: Yeah. Actually very interested to see if we can get either Scrypt or Argon2 into Bitwarden. That would be very exciting.

Steve: Yeah.

Leo: I'm told Dashlane uses Argon2.

Steve: Yes, they do.

Leo: There are disadvantages to that, as well. Everything's a tradeoff. So nothing's perfect.