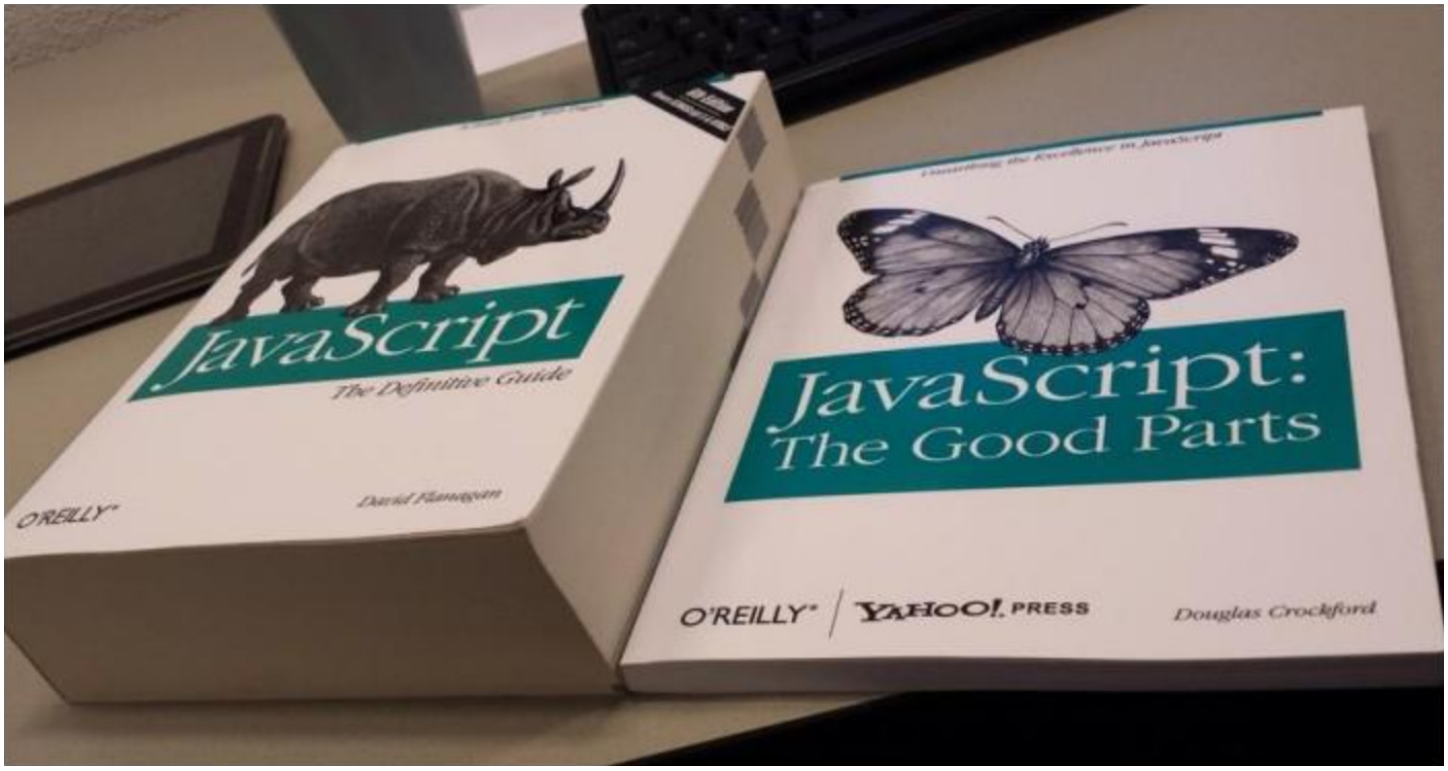
Security Now! #905 - 01-10-23

1

This week on Security Now!

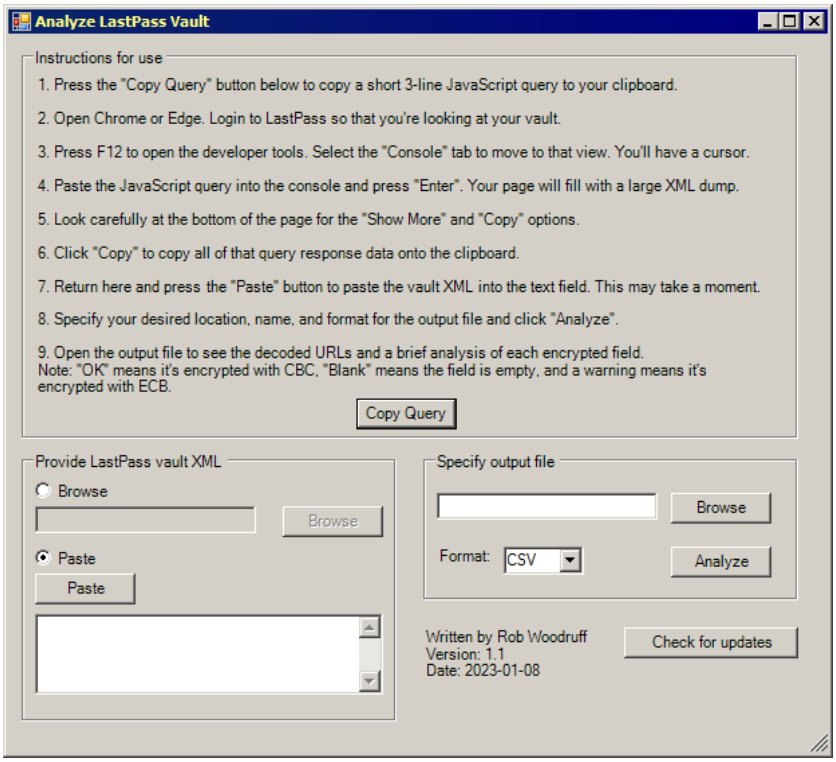
This week, in a necessary follow-up to last week's "Leaving LastPass" episode, we'll share the news of the creation of a terrific PowerShell script, complete with a friendly user interface, which quickly de-obfuscates any LastPass user's XML format vault data. What it reveals is what we expected, but seeing is believing. Then we're going to examine the conclusions drawn and consequences of the massive amount of avid (and in some cases rabid) listener feedback received since last week, and some of the truly startling things that listeners of this podcast discovered when they went looking.

Any Questions?



Security News

Okay. LastPass aftermath. There actually will be some math later in this podcast, but no test. At the top of the news for our listeners this week is that my call for the creation of a LastPass vault de-obfuscator, which would clarify exactly what information ALL of we LastPass users have had exposed, resulted in a number of great efforts. Several listeners produced JavaScript solutions. Paul Holder, whom Leo knows from their participation in the annual Advent of Code competition, wrote a portable solution in JAVA, and several others wrote solutions in C, C++ and C#. As I was sure would be the case, our audience has skills. But the solution that captured my attention was both the smallest of all and by far the most powerful. It was implemented as a Windows PowerShell script. And having seen what PowerShell's scripting language can now do, with its full access to the .Net language, it's clear that I'm going to need to make some time, someday, to take a much closer look at it. It's sort of amazing what has quietly been happening there:



So we have a solution that I'll explain, in detail, in a moment. But the way we got there is almost as interesting as what we got and I confess that it would have never occurred to me. It began with a Twitter DM from a listener named Rob Woodruff. Rob tweeted:

Rob Woodruff / @fulori

> *Alright, Steve. You asked and I delivered. I wrote a PowerShell script to parse the XML file that is your LastPass vault, identify any values encrypted with ECB rather than CBC, and decode the URLs from hex to ASCII. I chose PowerShell so that it will run on any modern Windows computer. It's not fancy but it appears to work. You'll need to specify the InFile, OutFile and Format parameters on the command line, or it will prompt you for them. InFile is the path and filename of the XML file (your LastPass vault). OutFile is the path and filename of the output file. Format is the format of the output file - either CSV or HTML. You can download it here: {Dropbox Link}*

Okay. That was just Rob's first message which is far from where we ended up. About four hours later he followed up:

Rob Woodruff / @fulori

> *This version of the script has a GUI: {Dropbox Link}*

Upon seeing this, as I said, I thought *"I'm going to have to pay more attention to PowerShell."* So I replied: *"Holy Crap, Rob! You're a PowerShell Wizard! I'll check this out tomorrow. THANK YOU so much! :)"* To which Rob replied:

Rob Woodruff / @fulori

> *My pleasure, Steve! I really enjoyed the process. First time doing a GUI in PowerShell. 😁 Not sure I can claim the title of Wizard, though. I had **ChatGPT** do most of the heavy lifting. 😬 Speaking of which, if you haven't played around with it, you must. It will blow your mind.*

Being unsure whether Rob might be pulling my leg, I wrote: *"Rob: Are you not kidding? Did ChatGPT really have a hand in that?"*

Rob Woodruff / @fulori

> *Steve, I'm not kidding! ChatGPT is supposedly fluent in every written language, including programming languages. I told it in English what I wanted to do and it spat out PowerShell code. It's not perfect, of course, and I spent a lot of time debugging the hex to ASCII conversion. Ultimately, I ended up using a code snippet for the conversion that I found using Google because ChatGPT couldn't seem to figure it out. Similarly, when I asked ChatGPT to add a GUI, it got most of it right on the first try, but it had two of the buttons being overlapped by text fields, so I had to adjust the positioning of elements manually. Overall, though, it saved me a lot of time. I probably wouldn't have even tried to tackle this project without ChatGPT.*

Wow!! We've clearly just entered a very different world. What Rob explained is consistent with everything we've heard about ChatGPT. It's not (yet) perfect, but it's VERY good. Unbelievable.

I then worked with Rob Friday evening and through the weekend to polish and perfect this little 12K gem to ready it for today's podcast. Remember that being a script, it's inherently open source and therefore readily verifiable.

Remember from last week that the key to obtaining the LastPass vault is using the developer features of any browser that's currently logged into LastPass. Though this part is a bit inconvenient there's no way around that without a huge amount of work to recreate everything that a browser does or obtaining the LastPass domain's session cookies which would be even more work.

So, first grab the PowerShell script that Rob wrote from my server. It's this week's GRC shortcut, so: https://grc.sc/905 (https://www.grc.com/miscfiles/Analyze-LastPass-Vault.zip)

That will return a tiny ZIP file containing the "Analyze-LastPassVault.ps1" PowerShell script. As

we've learned on this podcast, encapsulating the script in a ZIP prevents Windows from tagging the script inside the ZIP with the dreaded MOTW - Mark of the Web. So you won't need to see the warning about the dangers of something having been downloaded from the Internet.

As a very useful security measure, Windows will no longer run unsigned PowerShell scripts. Since I didn't want anyone to be put off by that, and since signing does provide a useful verification that nothing has been altered, I signed Rob's final script with GRC's EV code signing cert. So Windows will see that it's been signed and will run it without complaint.

So, next, launching a PowerShell prompt the Windows menu or type "PowerShell" into the Windows search box. This does not need to run with elevated Admin privileges so anyone should be able to do this. Start the script by entering .\Analyze-LastPassVault.ps1 and press Enter. The Analyze LastPass Vault app will be displayed on your desktop.

You'll find complete instructions there for proceeding so you can likely race off on your own without the rest of this. But I can provide a bit of additional background and clarification.

With Rob's app running, switch to a browser and log into LastPass so that you're looking at your vault. Press the F12 key or "Ctrl Shift I" to toggle developer mode. Select the Console tab in the developer window and wait for things to settle down if anything is scrolling. Once you have a cursor flashing you're ready.

To make this as easy as possible, we built-in the 3-lines of JavaScript code that needs to be dropped into the browser at this point. So press the "Copy Query" button in Rob's app to place those 3 lines onto the system's clipboard then bring the browser into the foreground so that it has the system focus and hit "Ctrl-V" to paste those 3 lines into the browser. Press Enter to execute the query and the page will fill with XML expressions.

As I described last week, down at the very bottom of the screen will be the options to display more or to copy the query results to the system's clipboard. So click on Copy.

Now, switch back to Rob's app and click the Paste button. That will paste the captured clipboard directly into Rob's app without needing to go through the intermediate step of saving it to a file. Note that if last week you already copied your XML-format vault out of a browser and may have terminated your account with LastPass, you can also provide Rob's app with that filename to load and process.

With the vault made available to the app, specify an output filename to receive the deobfuscated data and choose the output format either CSV or HTML. The CSV format will ideally require something in your system for viewing it, like Excel. I like the HTML format since your browser will happily display that. So set the filename, ending in either CSV or HTML as needed and finish by clicking "Analyze". The file will be written and then the resulting file will be launched for display by whatever handler the user's system has registered. If you exported HTML, the results will be displayed in a nicely formatted scrollable window with one line per login record.

Rob's app displays an "OK" on the left if the encrypted account information was encrypted using the better CBC cipher mode. If the record's encrypted information was encrypted with the

suboptimal information leaking ECB cipher mode you'll see the message "WARNING: Encrypted with ECB!".

Next on the line, and likely most interesting, will be the deobfuscated URL that's associated with the website's encrypted logon record. Taken as a whole these are all of the sites for which LastPass's vault contained your logon information.

Its notable that the vault does not appear to contain the user's unencrypted eMail address to associate who they are with the vault, nor is there any indication of the number of PBKDF2 iterations that were being used to obtain the vault's decryption key from the user's eMail address and password. That information is necessary to run the Password Based Key Derivation Function to decrypt the key which is used to decrypt the vault. Otherwise, it requires brute force. So it must be that it's provided to the user's client through another query. But LastPass' infamous December 22nd breach update said:

> *[...] the threat actor copied information from backup that contained basic customer account information and related metadata including company names, end-user names, billing addresses, email addresses, telephone numbers, and the IP addresses from which customers were accessing the LastPass service.*

So to that we can now add everything that Rob's LastPass Vault analyzer makes crystal clear.

Given the amount of effort that will generally be required – and we have a bracing update on that coming up next – to brute-force the decryption of the encrypted information in even **one** LastPass vault, coupled with the fact that tens of millions of LastPass user vaults were obtained en mass, the direct threat of decryption for a single individual, unless directly targeted, would seem very small.

But, assuming that it's possible for the bad guys to associate company names, end user names, physical addresses, eMail addresses, telephone numbers and IP addresses with specific vaults, the result – without any decryption – is a comprehensive dump of "exactly **who** logs on exactly **where**". And, if you scroll horizontally all the way to the right of Rob's HTML output, there's also a "LastTouch" field containing a time code that shows when the last logon at that domain occurred. So, even without the use of **any** brute force decryption of the vault's encrypted contents, this represents, at best, a significant privacy compromise for **every** LastPass user.

Ouch.

One last note before we look at what's been learned from a week of feedback from our listeners, which I just described as bracing: If executing Rob's PowerShell script produces errors rather than the presentation of a nice graphical interface – as it initially did for me on my Windows 7 machine – I wanted to note that it's possible to update any Windows Power Shell support, even Windows 7, to the latest version v5.1. Microsoft wants you to have it. I have a link for that at the bottom of page 4 of the show notes:

Link to update to the latest PowerShell, v5.1: (Thanks to Colby Bouma)
https://www.microsoft.com/en-us/download/details.aspx?id=54616

Okay, so what more do we know this week that we didn't know last week after our listeners had the chance to understand and peruse their LastPass settings?

We don't yet have any good sense for whether, or the degree to which, encrypted content was allowed to remain encrypted under the less desirable ECB cipher mode. I expect feedback from the use of Rob's vault analyzer to fill in a lot of that information for us by next week. So I'd appreciate knowing when and how many ECB warnings, if any, are seen by our listeners. I didn't have any ECB encryption in my vault even though I'm sure that I had login credentials dating from my first use of LastPass. Rob reported that his vault contained one ECB-encrypted entry. So I'll be interested in obtaining a much larger sampling.

But the ECB vs CBC issue is not that much of a big deal. Since AES-ECB, being based upon the AES cipher, has a cipher block size of 128 bits, sets of 16, 8-bit characters, thus forming a block of 128 bits, are encrypted from that pattern of 128 bits into a different pattern of 128 bits. That's encryption. This means that every instance of the same password will encrypt into the same pattern of 128 bits. As I noted last week, the presence of password reuse would therefore be obvious just by inspecting a user's encrypted vault without the need for any decryption.

By comparison, AES-CBC uses both the encryption key, which would not change from password to password, and also a random initialization vector – think of it like salting a hash – which is different for each password. This would completely obscure the presence of any identical passwords. So, we'll likely learn by next week the degree to which ECB ciphers were still in use in our listeners' vaults.

But the FAR more worrisome fact that was revealed when our listeners checked the settings of their LastPass vaults was the degree to which many – and I do mean many – of their password iteration settings were found to be below the 100,100 iterations mark. And in a revelation that I'm still trying to get my head around, I heard from many listeners whose PBKDF2 iteration count was set to 1. Yes, 1 iteration. And thus the shocking title of today's podcast.

I also received many reports of iterations still being set to 500 and many set to 5000, as well as many set to the newest LastPass default of 100,100. Now at first you might think that anyone whose iteration setting was 1 should be about 100,000 times more concerned than someone using the new default of 100,100. But it's actually quite a bit worse than that, because that 100,000 to 1 simple math assumes that vaults were being selected at random for attack, which would probably not be true.

We need to assume that the attackers obtained every user's account metadata including their vault's iteration counts. Those counts need to be recorded somewhere because no one's vault can be decrypted without knowledge of the count. And LastPass would have backed it up since the loss of that would be even worse than the loss of the vault backups themselves. So, assuming that the attackers obtained the iteration counts for every LastPass user, as the probably did from LastPass' backup, if opportunistic brute force decryption of user accounts was their intent it would be a reasonable strategy for the attackers to start with those LastPass users whose counts were '1'. Why would they not?

Unfortunately, there's a well-known expression to describe the situation in which all of those LastPass users who, at the time of this breach, had their LastPass password iteration counts set to 1. And that expression is "Low Hanging Fruit".

Last week, we did not look deeply into the actual performance of today's GPU-enhanced password cracking. This week we need to get a sense of scale. Current estimates of GPU hardware-enhanced password cracking places the time required to crack a 100,100 iteration PBKDF2-protected password, where that password has high entropy of 50 bits, at 200 years.

But, since GPU use scales linearly, dividing that cracking task among 200 GPUs, which is now quite mature cracking technology, could crack the same password having 50-bits of entropy in **one** year. Right? One GPU, 200 years. 200 GPUs, 1 year. But I also noted that studies have shown that most practical passwords have an entropy of around 40 bits. It turns out it's difficult to get 50. We'll get back to what lowers true entropy in a second. Having 40 bits of entropy is approximately 1,000 times weaker than 50 bits since **bit strength scales exponentially**. In other words, random bits are worth a lot because each additional truly random bit, on average, **doubles** the time required to crack.

So, if we assume that our attackers will have the use of 200 GPUs which, in this era of GPU-laden cryptocurrency mining rigs seems entirely reasonable, cracking a **typical** password having 40 bits of entropy would require 71.338 **days** if that password was protected by 100,100 iterations of PBKDF2.

Okay, just to restate that because this is an important benchmark: A typical strength password that's been protected by 100,100 iterations of key derivation, that's attacked by a 200-GPU password cracking rig would fall against that attack in an average of 71.338 days. So if you're thinking that all of those 100,100 iterations might still not be providing you with sufficient protection, you're probably not entirely wrong. Using one million iterations would be 10 times stronger, bringing us up to 713 days, or just shy of two years. That seems much safer.

The very bad news is that for those whose LastPass iteration count was, for whatever reason, discovered to still be set to '1' – and there were many such people who reported that this past week – those same 200 GPUs could crack that same 40-bit entropy password in an average of 61.56 **seconds**, or just over one minute per single-iteration password crack. Given that, it appears to be the height of negligence, if not bordering on criminality, that for some reason – for whatever reason – many listeners of this podcast and I'm sure a great many more non-listeners, have **no effective protection** from the cracking of their LastPass vaults and the resulting disclosure of every single one of their website logon credentials, their credit cards and any other confidential documents and personal papers that were stored for them by LastPass.

Not one of these many people told me that LastPass had reached out to them to explain that due to their effectively non-existent password encryption they are at heightened risk following the data breach and that they should therefore immediately rotate all of the logon credentials being managed by LastPass and assume that any information stored in their LastPass vault has been compromised. As far as I know, that hasn't happened, and it certainly should have. LastPass knows everyone's iteration counts. And now, so do the criminals who stole them.

LastPass somehow failed to update those iteration counts for a decade after the default was raised from 1 to 500 in June of 2012. And they have not immediately and proactively assumed responsibility for that by informing their users, whose iteration counts were dangerously low, in many cases set to 1, that unfortunately, they should now assume that the unencrypted content of their LastPass vaults is now in the hands of criminals. The industry at large has been grumbling about LastPass not being forthcoming about the details of the breach. But we now have all of the information we need to assess LastPass' culpability.

A couple of weeks ago, when I was first updating myself on LastPass's client settings over the holidays, I changed my iteration count from 100,100 where, thank goodness it was still set after we talked about this five years ago, to 350,000 as is now recommended by OWASP. That change of my iteration count took, oh, perhaps five seconds. I had to provide my LastPass master password again, so that the LastPass client could rehash it under the new iteration count. And that was it. This could and should have been automated since its first increase from 1 to 500 back in 2008. No user should have been allowed to set a dangerously low iteration count and every LastPass client, which must know its user's iteration count in order to function, should have taken proactive responsibility for continually bumping it up to whatever is currently considered safe.

So, the most startling and deeply disturbing news I received throughout last week was not only that many of our listener's iteration counts were 5000 or even 500, but that many discovered that theirs was set to 1.

I am a 67 year old, lifelong entrepreneur and businessman. And you would have a difficult time finding anyone who is more deeply opposed to frivolously turning attorneys loose on each other. It's one thing to fight over an ongoing contract dispute in order to reach a resolution. That makes sense to me when the parties cannot negotiate an accord in the absence of objective judgment. But attacking an entity – after the fact – for something that was done which I'm sure they now regret, still leaves a bad taste in my mouth. On the other hand, if a lawsuit were to be brought against LastPass, not because they made a mistake that upset their customers, but over actual provable damages arising from reliance upon LastPass' assertion of the safety of vault data, then I would not consider that to be unwarranted ambulance chasing.

There is no way to paint the presence of an iteration count of 1, used for the derivation of a LastPass vault's decryption key, as anything other than a critically debilitating product defect. And for that, if actual damage results, LastPass could be, and I think should be, held wholly responsible.

> Okay. Let's take a break, then I want to talk a little bit about password strength before sharing some of the terrific feedback I received over the past week from our listeners.

On the true strength of passwords:
I want to amplify something I touched on both last week and this week. I mentioned that an increase in iteration count provided a linear increase in strength, whereas the increase in strength provided by adding bits is exponential. I want to be certain that everyone fully appreciates the implications of that.

A few weeks ago, when I increased my LastPass client's iteration count from 100,100 to 350,000, that was an increase of 3.497, about 3 and a half times. So not quite four times. But if I had increased my password's entropy by just 2 bits, that would have been a full factor of 4 increase in cracking resistance.

True entropy is quite difficult to calculate because few of us are using a chunk of text from GRC's perfect passwords page for our master password since those are impossible to remember. So we have the situation that a single character's true entropy is difficult to calculate. If any character in a password is related to any other other character in that password in **any meaningful way** other than having been chosen purely at random – in other words, if there's **any** reason for a character to be what it is rather than something else, then that character's contribution to the true entropy of the whole is reduced. Its contribution of entropy would be significantly less than it would otherwise be.

This is why the first thing that password-guessing crackers do is use dictionary words in various ways and base their attacks upon the frequency of characters occurring in the natural language of the password's user. Those attacks model the lack of entropy that many users employ when they are choosing their passwords.

So what's the idealized potential entropy of a single character? In byte-oriented systems a single character typically occupies 8 bits. So we might be inclined to say "8 bits". But ASCII only uses the lower 7 of those 8 bits. So assuming a non-UNICODE standard ASCII character set, there are a total of 95 printable standard characters available with upper and lower case alphabetic, the 10 numeric digits and all of the other special characters. There are 95.

So here's the point I want to drive home: Increasing my iteration count from 100,100 to 350,000 yielded that three and a half times increase in password busting protection. 3.497. But just adding one single randomly chosen additional character to the end of a password increases the resulting password's anti-cracking strength by 95 times. 95! This is why, when it comes to passwords, size does matter.

You get FAR more attack protection by using even slightly longer passwords, where strength increases exponentially with length, than you do by increasing iteration counts, where strength only increases linearly.

Via DM:

I have a Corporate LastPass account and a personal pro account. The personal account was updated to 100K iterations, but the corporate account was still at just 5K. My personal account is still exposed, though, because I took advantage of the ability to share passwords between my personal and corporate accounts to reduce the number of logins. I assume that if they crack the corporate, they would have the personal anyways. Good news, my password has more than twenty 25 random characters derived from your Perfect Passwords. The bad news is it is so long and random that I used the same password for my corporate and personal LastPass accounts.

25 truly random characters chosen from the Perfect Passwords page will have been selected from an alphabet of 95 possible characters. So that's 95 x 95 x 95 and so on for a total of 25 times. That's 95^{25}. I used the Password Haystacks page to quickly do the math and show me that the resulting password has 2.80×10^{49} possible and – importantly – **all equally likely** combinations. If we take the Log base2 of that number to determine the equivalent binary bit strength, we get 164.2. So it contains a little over 164 binary bits of true entropy. Another way to look at it is that each character, when truly chosen randomly from a set of 95 possible characters, contributes 6.57 bits of entropy.

In other words, this person has absolutely nothing to worry about. His password has slightly more than 164 bits of true entropy. It will never in many lifetimes be cracked by today's or even any projected technology tomorrow. Remember, quantum computers won't help with this sort of symmetric crypto problem.

But there is something else worth noting...

Recall from last week that Mr. Grumpy Pants – what was his name? Oh yeah, Jeremi Gosney – He noted that LastPass's vault encryption key was derived from only 128 bits of entropy.

So now consider this crazy 25-character totally random password which has a bit more than 164 bits of entropy. If the attackers knew that, and there's no way that they could. But if they did, it would be far quicker to just forget about the user's insane password and attempt to directly brute force the vault's encryption key **itself**... since it "only" has 128 bits of entropy. I have "only" in air quotes because my point is that 128 bits already has so many possible combinations (3.4×10^{38}) that there's really never any reason to go above it.

Haystacks says that **20** characters chosen from that 95-character alphabet offers 3.62×10^{39}. Once again, 128 bits yields 3.4×10^{38}. 20 random characters yields 3.62×10^{39}. So, ten times what 128 bits can do.

In summary, do **not** use only 20 characters unless they are truly chosen from among all possible characters at random. But if they are, there's no benefit gained from using more.

Dave:

> *On Security Now Episode 904 Steve asked for feedback on the current value of the LastPass Password Iterations field.* ***Mine was set to 1.*** *I have no idea how/why it is one because I never changed it. Needless to say I have downloaded and installed Bitwarden and I am changing the password on every site in my vault as rapidly as I can.*

Dave has the right idea, and there's an example from among many of what our listeners discovered to their horror this past week. And sadly, it might be because he never changed it that it remained set to 1. As we know, he should not have had to change it. That should never have been his responsibility. But we're on the outside here, looking in. We have no idea of the real story behind this iteration fiasco. But there is no way to forgive this from LastPass. This is more than a mistake. This had to be someone's boneheaded decision.

With their acknowledgement of the importance of increasing the iteration count over time, evidenced by its default being jumped from 1 to 500 to 5000 to 100,100, someone **must** have made the decision not to bother bringing older existing iteration counts into compliance with current best practices. Someone must have decided that it would result in too much customer confusion and support calls. So just leave it wherever it is.

And the galling thing is that it could have even been 100% transparent. I'm no smarter than their crypto people. So they know this. When the user provides their eMail address and password to login to their client, at that moment the client has everything it needs to perform the upgrade transparently. Start iterating on PBKDF2. Pause at the current iteration count and take a snapshot of the current key at that point. Then keep going to the new larger iteration count and take a snapshot of that new key. Now decrypt the vault with the current key (which was sampled mid-stream) then re-encrypt the vault with the larger final iteration count key. And finally, update the stored iteration count. Done. Totally transparent. No user confusion.

David Lemire:

Hi Steve, Thanks for the excellent coverage of the LastPass breach and its consequences in SN #904. I can confirm both your smooth experience transferring from LastPass to Bitwarden, and Leo's note about Bitwarden having a lower size limit on secure notes than LastPass's; I had to delete one or two very large notes before I could successfully import my vault (thankfully they were obsolete).

I have one technical security question: Given the threat of rainbow tables, wouldn't it make sense for each individual account to have its own iteration value within a suitably secure range, rather than a common default value? (which I realize can be changed) Combining an unpredictable iteration count with salting the hashing process should raise the work factor for the creation of rainbow tables, as well as the comparison process, by a considerable factor.

I didn't mean to confuse things last week with my mention of the possibility of attacking known salt-free hashing schemes with precomputation attacks. My intention was to paint a history to remind us where we've been and how we got to where we are today. Everyone has always been protected from precomputation attacks by the inclusion of their eMail address as salt for the PBKDF2 function. Joe Siegrist was doing this from day one. Unfortunately, back in 2008, Joe was also iterating only once through PBKDF2 and as we now know, for some unlucky souls, that was never changed.

Someone is also likely to ask: What if a user deliberately set their iteration count to 1, not understanding what it was? My answer to that would be that it should absolutely never have been allowed. LastPass would certainly not allow any user to leave their password blank. A low iteration count is effectively no different. LastPass was lifting the count over time and that should have always been the minimum that any LastPass client would accept.

Via eMail:

Hello, About the LastPass Breach (Episode 904): The risk on passwords and metadata was explained very well. I wished the risk for files stored in LastPass could be explained, too. (e.g.

copies of personal ID, passports, drivers licenses) I can go through all my passwords and change them, but changing my real life documents will be more difficult. I guess, many other LastPass users will have this problem, too. Thanks, Boris

Boris makes a great point. We didn't stop to consider the many greater privacy dangers that might arise from having the contents of the LastPass vault's secure notes storage compromised. Depending upon what was in there, the damage from disclosure could be significant.

I also received a bunch of these:

Replying to @SGgrc I exported all my stuff to BitWarden earlier tonight. The process couldn't have gone more smoothly. Up and running on both my laptop and phone.

Via DM:

Steve, I listened to your podcast twice but what I don't understand is I thought you said it wouldn't matter if somebody had our blob of data because the keys only reside on our devices. So even if they had our master password, how would they crack into the blob without having the keys? Thank you for all you do.

If there was some confusion there let me clear that up: The key that's required to decrypt the LastPass vault is derived only and completely from three pieces of information: The user's eMail address, the user's password, and the iteration count. No other information is required.

The only one of these three things that LastPass and the attackers do **not** know is the user's password. They have their eMail address and iteration count. So, with an iteration count that's too low, it's quite feasible for a modern attacker to simply guess and test at ultra-high speed all possible passwords until they find the right one.

Via DM:

Hi Steve, do you think that having a non standard number of iterations lets say 168429 makes that particular password not worth the effort to try to decipher since 95% of all the passwords will have either 5000 or 100100 iterations?

This question came up often. Since each user's iteration count is known, making it non-standard will have little effect. If the attackers have adopted a "low hanging fruit first" strategy, which is what seems by far the most likely way to reap the rewards of their score, then they would sort the entire LastPass vault backup database by iteration count and prioritize attacks against all of those unlucky souls whose iteration count matches the title of this podcast. From there, sorted by iteration count, they would proceed upward.

And let's not forget that a significant amount of privacy-related information is immediately available since all of the URLs for the sites where we have stored logins is in the clear.
Zapper / @CryptoZapper35

@SGgrc Steve, being a longtime and very trusting listener I have been a LP user on your recommendation. I will now migrate to Bitwarden. May I ask you to clarify the security risk if my LP master password was 20+ characters?

Zapper's question is also quite common. So I wanted to reiterate that longer is always better, even much better, and more random is also better because it increases true entropy. But as for 20+ characters; if your password was 20 truly random characters that's 131 bits of true entropy which is absolutely secure.

Via DM:

Lastpass: I changed my 30 character master password but I still feel uneasy. I started changing all passwords but have not migrated off of lastpass yet. Any thoughts on insuring lastpass REMOVES all vault info upon cancellation of user account?

So this user is not talking about the consequences of the theft but rather the safety of remaining with LastPass. A 30-character master password will be very very very secure. Even if it's the lowercase alphabet with 1234 scattered among the 26 letters to pad it out to 30. The resulting hash conveys nothing of the password's length. So no one attacking would have any idea how long the password is. This was the key message underlying Password Haystacks. No attacker would be trying any 30-character passwords until they had exhausted all shorter passwords, which will never happen.

SKYNET / @fairlane32

Moved my vault to BitWarden and set the PBKDF2 iteration to 1 million. On my iPhone 11 Pro Max it performs fine. One million iterations. Go big or go home, Steve. 🤪

That's a good data point. For PBKDF2, I don't see any reason for using an iteration count lower than 1 million. I'd probably use 1,234,567. And if it takes too long on some platform it can always be turned back down. But really, in this era of GPU-driven password cracking, where GPUs hash at light speed, PBKDF2 is showing its age. Cranking up iterations is just running ahead of a moving train. It would be much more sane to get off the tracks.

Since SQRL's entire security model is based upon the security of a single password based key, I gave this a great deal of thought years ago. Everyone, including Bitwarden, ought to abandon the use of any non-memory-hard password key derivation which GPUs excel at.

SQRL uses "scrypt" which absolutely requires a block of dedicated memory which cannot be shared among cores. Scrypt's many parameters are tunable and I don't now recall exactly how much memory I required, but I think it was 16 megabytes. I chose that because every smartphone can spare that, and it's only needed briefly to process a user's password entry. But significantly, GPUs are unable to follow since they're unable to run an scrypt of that size **at all** when that much memory is required. So switching away from PBKDF2, whose time has passed, ought to be on every password manager's roadmap for the future.

Robert vd Breemen / @rvdbreemen

@SGgrc thanks for the honest podcast on LastPass. Your vetting years back made me use it for many years. Now moving to BitWarden. Now that I am changing my 1000+ passwords I see how broken the system of password login really is. Why is there no change password api?

That really is a good question. A uniform, standardized, cross-site password change API would make rotating all of one's passwords an automatable operation. I think that the problem is mostly per-site resistance. One of the complaints most people have is that every site is different, has different password requirements, additional bells and whistles like security questions, and different password recovery approaches. And this arises from the fact that every site wants to be different. There is no uniformity so each site gets to invent the user experience flow that they prefer. It didn't have to be this way, but it's the way it is.

Andy Olson / @AvgAndy

@SGgrc I listened to the recent Lastpass episode. Switching to Bitwarden. Just wanted to note that password changes are necessary, but I found I can also change usernames on a lot of important sites. If user login is email, change that too.

Yes, I agree with that, especially since today's standard for password recovery – handling the "I forgot my password" event – is to send the password reset link to the user's account eMail. So, while it's not imperative, if easy for you to also change your account's eMail from what it was when your LastPass vault was copied and stolen, then there's no reason not to.

Once again I've used up our time this week covering this news, which is huge for this podcast's listeners since such a large number of us chose and have been using LastPass. But even for those who had chosen another password manager, all of the information about GPU cracking strength and PBKDF2 iteration counts is universal... as is the need to urge whatever password manager you're using to move away and beyond PBKDF2.

Next week we'll be able to get some sense for the amount of the ECB cipher mode that our listeners discovered in their LastPass vaults with the aid of Rob Woodruff's (and ChatGPT's) very nice PowerShell script. But, unlike this week's bombshell that many iteration counts were 1, the presence of any lingering ECB won't represent a five alarm fire.

My work is proceeding on SpinRite. Since last week, after finally exhausting my own ToDo list, I read through all of the still-open observations that had been posted in our GitLab instance and I have about 11 smallish things to fix. Then I'll release the next alpha, which will be its 9th, and we'll work on eliminating the final bits of odd behavior which appears to surround highly damaged drives while everything else is being given a good testing.