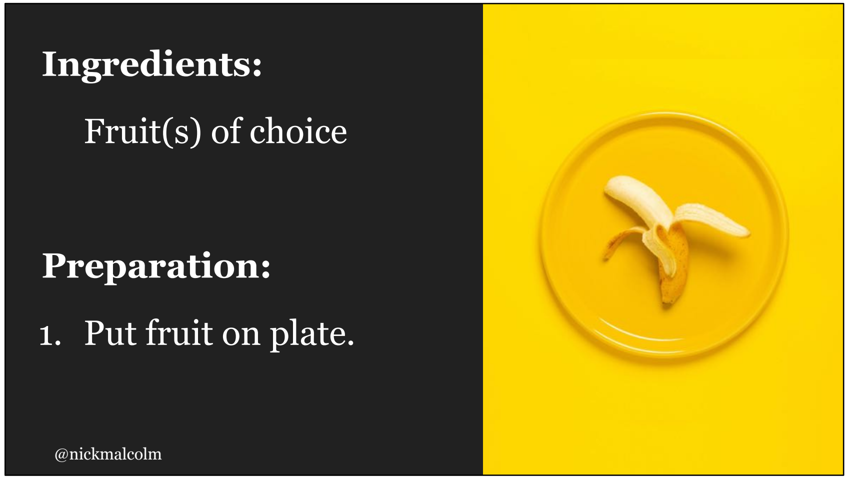
Using some cooking analogies we'll learn about keeping secrets secret.
Will our password-storing creation rise deliciously to the occasion, or will it fall flat in disappointing dispair.
Don your chef hat, and come with me.

⚠️⚠️⚠️⚠️⚠️⚠️⚠️⚠️⚠️
This talk was presented by Nick Malcolm at OWASP NZ Day 2020.
https://nick.malcolm.net.nz
Some intro slides removed.
⚠️⚠️⚠️⚠️⚠️⚠️⚠️⚠️⚠️

Thanks Pixabay and Unsplash for the photos!



As a chef, a new customer comes to you and says that their password is banana.
You take their banana, and put it on a labelled plate in the kitchen.

username	password
bluesky98	111111
Kazoo96	password
jackals4	r1&N*L&10Pmf
Sunny2323	password123

@nickmalcolm

The users table in your database is the plates of fruit. Each plate is labelled, and the fruit on the plate is just sitting there. Nothing fancy.

To validate a user's password, just look at their password

@nickmalcolm

A week later that customer comes back, gives you a piece of fruit, you take it and compare it to the one you had before. If it's a banana, they're legit.

Everyone in the kitchen can see your guest's secret ingredient(s)

@nickmalcolm

All the chefs walking through the kitchen, and the wait staff, and the cleaners, they all can see the fruit people have put on their plates.

Why is this bad?

@nickmalcolm

But no company would actually store passwords like that, right?

@nickmalcolm

People reuse passwords

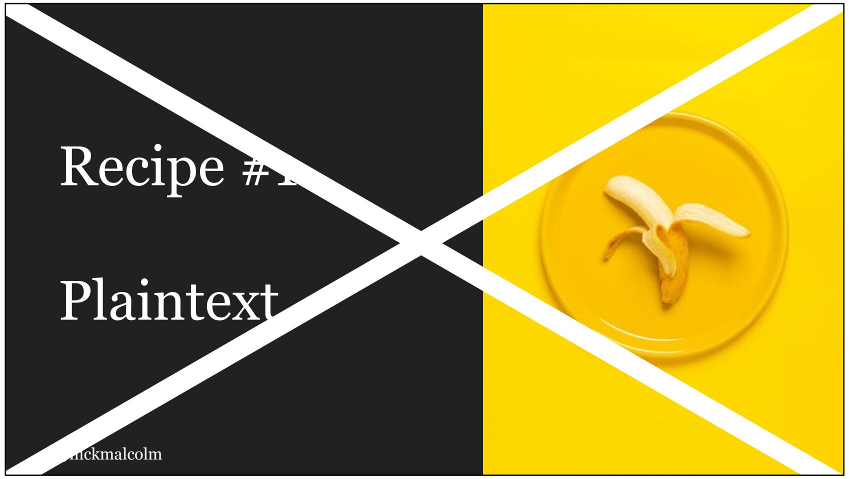
@nickmalcolm

No only that, customers probably use the same fruit at many restaurants - the same password on many websites.
Someone who knows their fruit can walk to the restaurant down the street and get a meal on them.

Now it's not just people with access to the kitchen who have your guests' passwords, it's everyone!

Yes, Neopets is still around!


Recipe #2
Symmetric Encryption
@nickmalcolm
flic.kr/p/EvurHC


Recipe #
Plaintext
nickmalcolm

So that was….

username	password
bluesky98	111111
Kazoo96	password
jackals4	r1&N*L&10Pmf
Sunny2323	password123

@nickmalcolm

This is our plaintext password storage from the first recipe

Ingredients:

Fruit(s) of choice

Preparation:

1. Put fruit in fridge.
2. Lock fridge.

@nickmalcolm

To validate a user's password,
unlock the fridge,
look at their password,
see if it matches

@nickmalcolm

This is great! Now waitstaff, cleaners, other chefs, and lost guests can't see the passwords.
They're still in plaintext inside the fridge, but you can't get in.

Recipe #2: Encrypted (a real example)

username	password	hint
blueskyg8	c57712e7eeadeb17	sixones
Kazoo96	94a012e6de4f1f0e 703c8f612c29f4a6	
jackals4	Ca97753a79cbdf8f 2a8819ef021ce73a	
Sunny2323	94a012e6de4f1f0e 6b4e6be23ec132f0	p+123

The Fridge

@nickmalcolm

This method of storing passwords was actually used, but we'll get to that in a moment.

https://gchq.github.io/CyberChef/#recipe=Triple_DES_Encrypt(%7B'option':'UTF8','string':'passwordpasswordpassword'%7D,%7B'option':'UTF8','string':''%7D,'ECB','Raw','Hex')&input=MTExMTEx

People in your organisation
could unlock the fridge
when they're not supposed to.

@nickmalcolm

Why is this bad

@nickmalcolm

You can sometimes X-Ray the
fridge to learn more
about what's inside

@nickmalcolm

The same password will look the same. You can often tell roughly how long a password is.

Breaking the lock
or finding a key
gives access to all the passwords

@nickmalcolm

But no one would actually store passwords like that, right?

@nickmalcolm

Recipe #2: Encrypted (a real example)

username	password		hint
bluesky98	C57712e7eeadeb17		sixones
Kazoo96	94a012e6de4f1f0e	703c8f612c29f4a6	password
jackals4	Ca97753a79cbdf8f	2a8819ef021ce73a	
Sunny2323	94a012e6de4f1f0e	6b4e6be23ec132f0	p+123

@nickmalcolm

https://gchq.github.io/CyberChef/#recipe=Triple_DES_Encrypt(%7B'option':'UTF8','string':'passwordpasswordpassword'%7D,%7B'option':'UTF8','string':''%7D,'ECB','Raw','Hex')&input=MTExMTEx

In this example of bad encryption, you can get information about the length of the plaintext entry.
You can also see repetitions in the source data.
Both Kazoo and Sunny have passwords which start with the same string - the word “password”.
Also the company that stored passwords like this also decided to store plaintext password hints...

152,445,165

accounts with 3DES-encrypted passwords alongside plaintext hints

@nickmalcolm

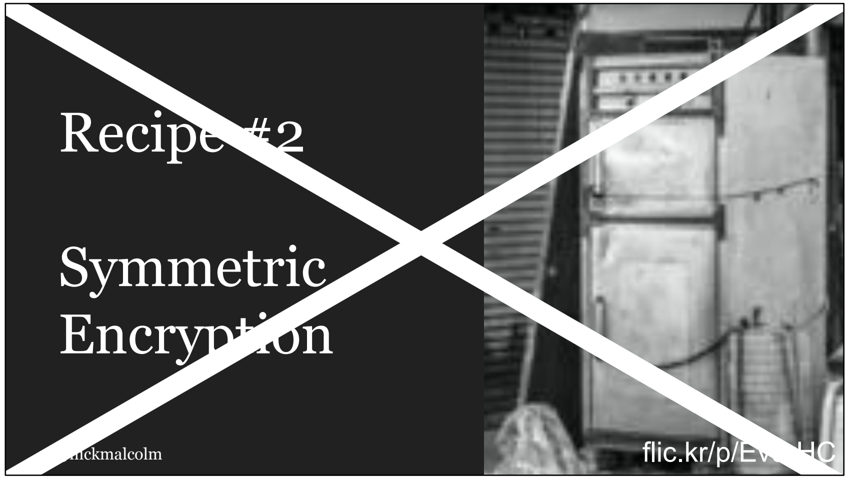
The key was never breached, but you could look at hints to guess the password.
If you figure out one password, you can see all the other users in the table who used the same password.
Or, if your own password is in there you can look and see who else has a password that starts like yours does.

“

We **deeply** regret that this incident occurred.

- Adobe, 2013

@nickmalcolm



Better, but still not very good.

The way we “blend” passwords is by using a **hashing** function.

@nickmalcolm

Hashing
Specifically: Cryptographic Hashes.

Ingredients:

Your choice of fruits, vegetables, etc.

Preparation:

1. Put ingredients in.
2. Blend.

@nickmalcolm

Very different smoothies, even with very similar ingredients. (kinda)

Very different hashes, even with very similar input.

@nickmalcolm

Add a drop of hot sauce, and the smoothie will change drastically. (Not really, but…)
Although the hashes are very different, the length is still the same. This means we're not giving away any extra information.

(Could also mention about locality-preserving hashes vs random output https://en.wikipedia.org/wiki/Locality-sensitive_hashing and https://en.wikipedia.org/wiki/Avalanche_effect)

Identical smoothie, given the exact same ingredients.

Identical hashes, given the same input.

@nickmalcolm

Deterministic

Infeasible to get back the raw ingredients.

Infeasible to recover the cleartext input.

@nickmalcolm

One-way

Fast to create a smoothie from even the chunkiest ingredients.

Fast to create a hash from even the longest or most complex input.

@nickmalcolm

Also a problem, but we’ll get to that

MD5
SHA1
SHA256
SHA384
BLAKE2…

@nickmalcolm

Hashing

Unlike a real smoothie…

the hash is always the same size, regardless of the size of the input

@nickmalcolm

Although the hashes are very different, the length is still the same. This means we're not giving away any extra information.
This is not how blenders work (more ingredients => more smoothie).

Recipe #3: Hashed with SHA1

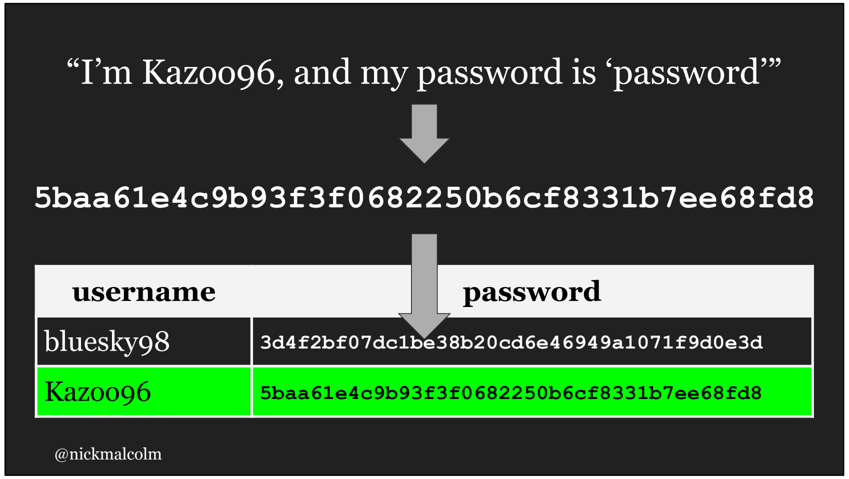
username	password
blueskу98	3d4f2bf07dc1be38b20cd6e46949a1071f9d0e3d
Kazoo96	5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8
jackals4	f8a087448bcec30f97a46094e48df6e2c76bae58
Sunny2323	cbfdac6008f9cab4083784cbd1874f76618d2a97
hackerman12	5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8

@nickmalcolm

username	password
bluesky98	111111
Kazoo96	password
jackals4	r1&N*L&10Pmf
Sunny2323	password123
hackerman12	password

@nickmalcolm

Note that they are the same length!



Hash the pwd, compare it against the claimed user's hashed password.

The customer comes to you with a banana, and instead of storing the banana on a plate or in a locked fridge, you blend it.
When they come back later, they give you their fruit, you blend that too.
Then you do a taste test

Users who have the same password have identical hashes

@nickmalcolm

Identical hashes is part of the built-in feature of hashes!

Why is this bad

@nickmalcolm

We hashed passwords with SHA1, that sounds good right?

It's also really fast to pre-compute lists of passwords.

These are called **Rainbow Tables**

@nickmalcolm

Most smoothies have banana in them.
Most passwords are "password".
In anticipation of a big breach, instead of trying to brute force them at the time, I could build up a list.

Recipe #3: Hashed with SHA1

username	password
bluesky98	3d4f2bf07dc1be38b20cd6e46949a1071f9d0e3d
Kazoo96	5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8
jackals4	f8a087448bcec30f97a46094e48df6e2c76bae58
Sunny2323	cbfdac6008f9cab4083784cbd1874f76618d2a97
hackerman12	5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8

@nickmalcolm

Fast hashing functions also means fast brute-forcing

@nickmalcolm

1	123456	7c4a8d09ca3762af61e59520943dc26494f8941b
2	password	5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8
3	12345678	7c222fb2927d828af22f592134e8932480637c0d
4	qwerty	b1b3773a05c0ed0176787a4f1574ff0075f7521e
5	123456789	f7c3bc1d808e04732adf679965ccc34ca7ae3441
6	12345	8cb2237d0679ca88db6464eac60da96345513964
7	1234	7110eda4d09e062aa5e4a390b0a572ac0d2c0220
8	111111	3d4f2bf07dc1be38b20cd6e46949a1071f9d0e3d
9	1234567	20eabe5d64b0e216796e834f52d61fd0b70332fc
10	dragon	af8978b1797b72acfff9595a5a2a373ec3d9106d

@nickmalcolm

https://github.com/danielmiessler/SecLists/blob/master/Passwords/Common-Credentials/10-million-password-list-top-100.txt

“

We take the security of our users **very** seriously

- Last FM, 2012

@nickmalcolm

http://blog.last.fm/2012/06/08/an-update-on-lastfm-password-security

So no one would actually store passwords like that, right?

@nickmalcolm

"We take the **safety and security** of our members' accounts seriously.

- LinkedIn, 2016

@nickmalcolm

Not very seriously, like Last FM
https://blog.linkedin.com/2016/05/18/protecting-our-members
But they were breached in 2012, and either didn't know or didn't tell them!

37,217,682

accounts with MD5'd passwords

@nickmalcolm

MD5d passwords
Well that's MD5, what about a different hash, a different blender?

“the vast majority of passwords were quickly cracked in the days following the release of the data”
- Troy Hunt of HaveIBeenPwned.com

@nickmalcolm

https://haveibeenpwned.com/PwnedWebsites#LinkedIn

164,611,595

accounts with SHA1 passwords

@nickmalcolm

MD5d passwords

What could we do so that rainbow tables don't work, and common passwords aren't as obvious?

Zuck used "dadada" as his LinkedIn password, and reused that password on Twitter and Pinterest.

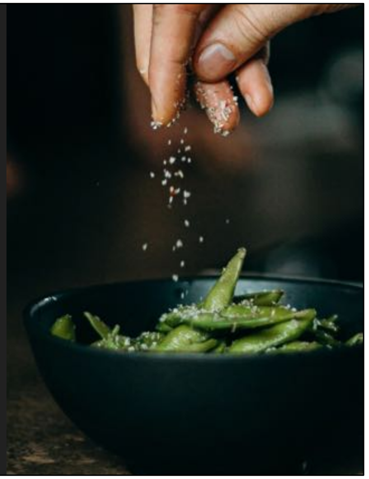
Ingredients:

Fruits, vegetables, etc.
Artisanal salt.

Preparation:

1. Put fruits in blender.
2. Add salt.
3. Blend.

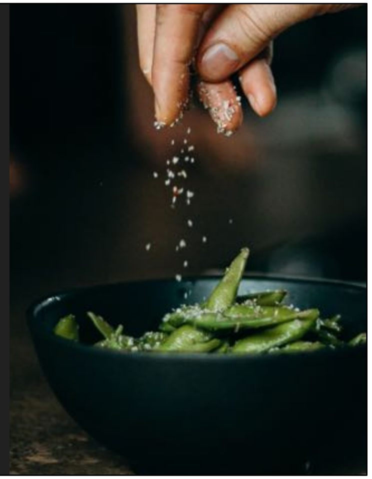
@nickmalcolm



Recipe #4

“Salted” Hash

@nickmalcolm



The type of salt is
written on a label, and
stuck to their smoothie.

@nickmalcolm

This way the chef doesn’t need to remember whether it was himalayan or rock salt.

Different salt
for each diner,
chosen by the chef.

@nickmalcolm

It’s not chosen by the user. Two users with the same requested ingredients will get different smoothies.

Plaintext isn't recoverable, &
identical input has different output

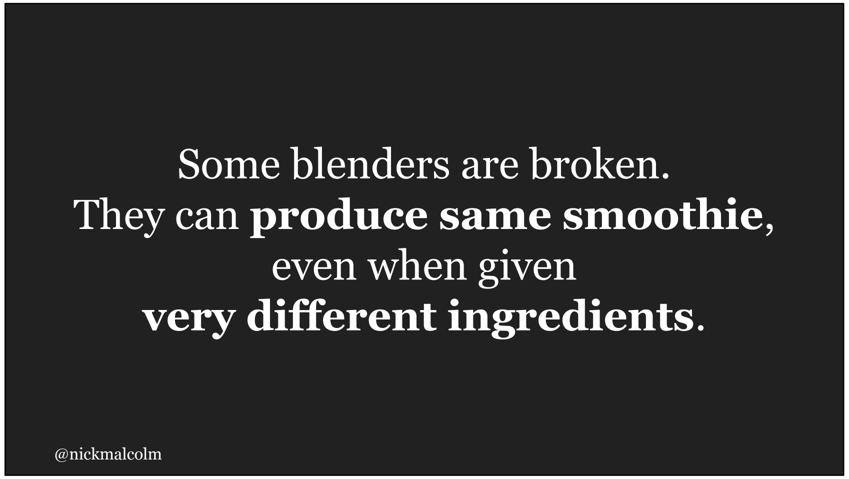
This seems fine?

@nickmalcolm

To validate a user's password,
hash the provided input
plus the salt you wrote
on the label of their last smoothie

@nickmalcolm

And check it against the hash in the DB.
Storing the salt alongside the hash allows you to recompute the hash identically, assuming you're given the same input at the start.



Salts don't do anything to stop brute forcing passwords.
An attacker can't use a pre-computed rainbow table,
and they can't see where folks have reused passwords,
but if you've used a fast blender, they can crank through passwords one by one
Especially if users use weak passwords.

"We take cyber security **very** seriously ...

there's no need to **panic**

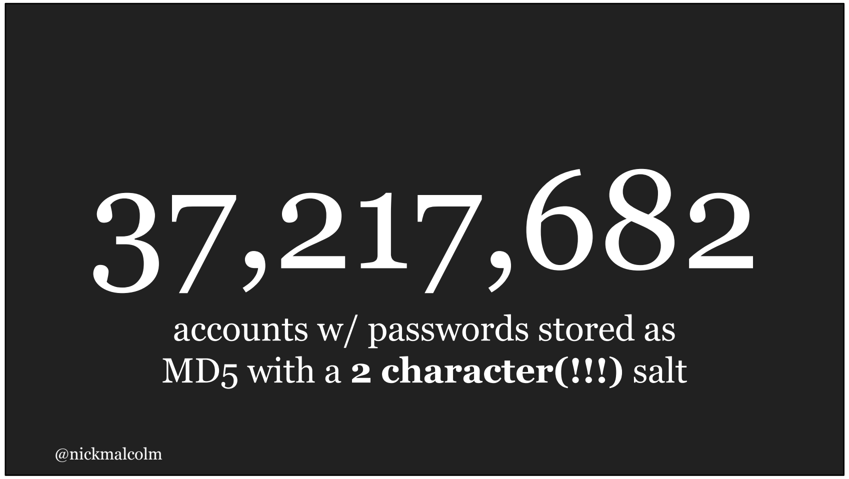
- Zomato, 2017

@nickmalcolm

https://www.zomato.com/blog/security-notice
https://www.zomato.com/blog/security-notice-update
https://haveibeenpwned.com/PwnedWebsites

But no one would actually store passwords like that, right?

@nickmalcolm



Because it was MD5 it was incredibly easy for researchers to brute force the passwords.
https://www.vice.com/en_us/article/z4j5g4/restaurant-app-zomato-says-your-stolen-password-is-fine-but-is-it

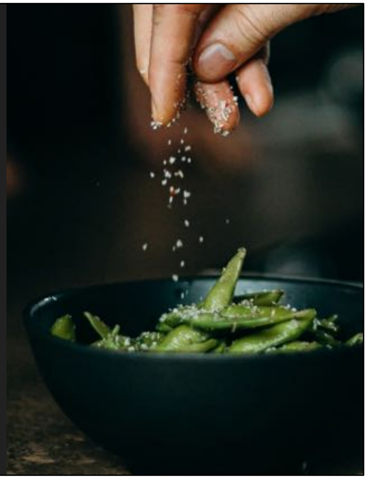
Ingredients:

Fruits, vegetables, etc.
Artisanal salt.

Preparation:

1. Put fruits and salt in a **good** blender
2. Blend.

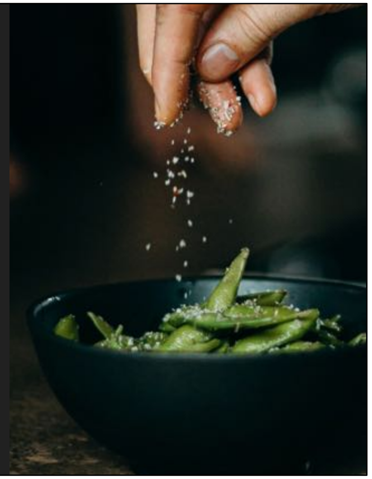
@nickmalcolm



Recipe #4.5

Salted Hash using a **good** blender.

@nickmalcolm



Blend as slowly as your users will tolerate.

Slow down your blender every now and then.

@nickmalcolm

This is called a “work factor”, or “rounds”. Different algorithms let you tune it in different ways.

Fast to create a smoothie,
but ***not too fast***.

@nickmalcolm

argon2id
is the algorithm of choice today

@nickmalcolm

“Winner of a several year project to identify a successor to bcrypt/PBKDF/scrypt methods of securely storing passwords”

Like bcrypt, Argon2id will choose the salt for you, let you customise and change how slowly it blends, and also has something extra we’ll get to soon.

PBKDF2.
Bcrypt.
Scrypt.
Argon2.

@nickmalcolm

These four options are safe choices, give you a configurable work factor, and handle the salt for you.
Don’t make your own blender.

PBKDF2 will be familiar to .NET and Django developers. It’s getting a little old. One popular cryptographer called it “the worst of the acceptable options”.
Bcrypt has been around since 1999. It’s pretty good and really easy to use, but doesn’t stand up against parallel or GPU - powered brute forcing as well as scrypt and argon2. It’s commonly found in rails and php apps.
Scrypt addresses some of bcrypt’s issues, and is a solid choice, but not super common.
Argon2 is the new kid on the block, and recommended as the best choice today.

https://haveibeenpwned.com/PwnedWebsites#AshleyMadison

There are a few strategies for this that I don’t have time to go in to,
But the OWASP password cheat sheet explores them.
Just realise that some ways of changing are better than others

"A blogger who went after the the bcrypt hashes recovered only ***4000 passwords*** *in a week.*

In contrast, CynoSure Prime *recovered the passwords for over* ***11 million*** *of the MD5 hashes in about 10 days."*

- Paul Ducklin of "Naked Security"

@nickmalcolm

https://nakedsecurity.sophos.com/2015/09/10/11-million-ashley-madison-passwords-cracked-in-10-days/

In July 2017, Avid Life Media (renamed Ruby Corporation) agreed to settle two dozen lawsuits stemming from the breach for $11.2 million.

https://en.wikipedia.org/wiki/Ashley_Madison_data_breach

https://arstechnica.com/information-technology/2015/09/once-seen-as-bulletproof-11-million-ashley-madison-passwords-already-cracked/

30,811,934

accounts with bcrypt'd passwords
but still kept 15M unsalted MD5 hashes around

@nickmalcolm

Side note: Canva's response was actually pretty good. Better than others. (Still not great: https://www.smartcompany.com.au/startupsmart/analysis/canva-data-breach-response/)

But the database is out there, and there will be people who are targeting specific accounts and trying to brute force those passwords.

“

This is **regrettable.**

- Canva, 2019

@nickmalcolm

https://support.canva.com/contact/customer-support/may-24-security-incident-faqs/

So it’s fine to store passwords
when hashed with a good aglorithm?

@nickmalcolm

So it’s fine to store passwords like that?

Yes, but…

@nickmalcolm

137,272,116

accounts with bcrypt’d passwords

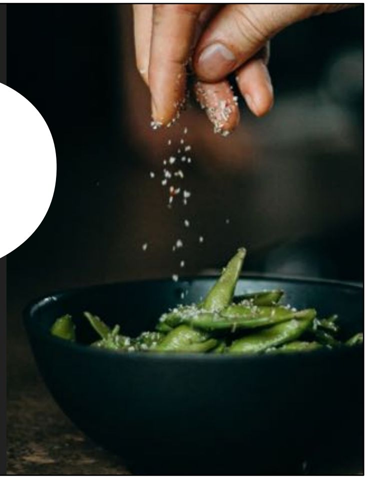
@nickmalcolm

Side note: Canva’s response was actually pretty good. Better than others. (Still not great: https://www.smartcompany.com.au/startupsmart/analysis/canva-data-breach-response/)

But the database is out there, and there will be people who are targeting specific accounts and trying to brute force those passwords.

Recipe #4.5?

Salted Hash using a good blender.



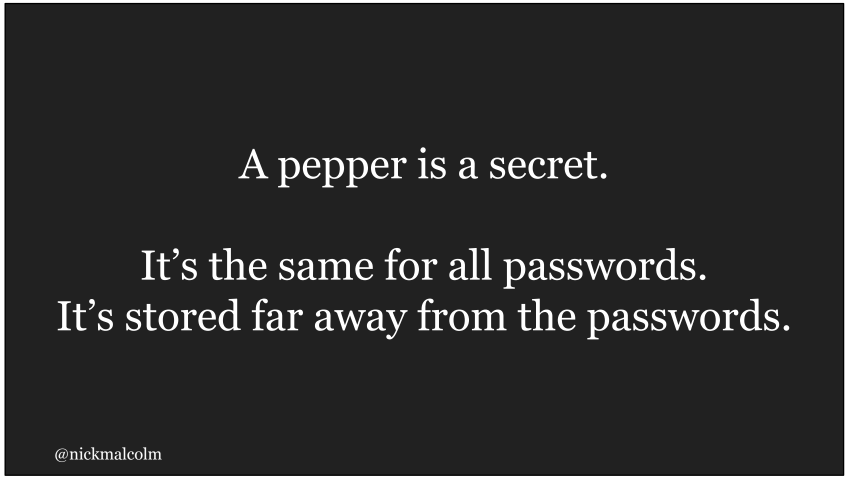
@nickmalcolm

If not that, then what?

An attacker can still (slowly) brute-force passwords, one at a time.

@nickmalcolm

If the database server is misconfigured, if there’s SQL injection, if a backup goes missing, an attacker has access to a table of salted hashes.



Remember that salts were like a label stuck to a smoothie. They weren’t a secret, they just made hashes unique.
A pepper is not stuck to the smoothie. It’s stored elsewhere, like KFC’s 11 Secret Herbs and Spices.
You might store it in your app config, or in a hardware security module.

All passwords are encrypted and decrypted by the application, using this secret.

An attacker needs to
know the secret pepper
before they can even start
brute-forcing salted hashes

@nickmalcolm

If you get SQLi, or find a misplaced backup, or steal the database's harddisk, you're stuck.
You are missing the key.

If you compromise a web app, maybe you can get the salted hashes from there.

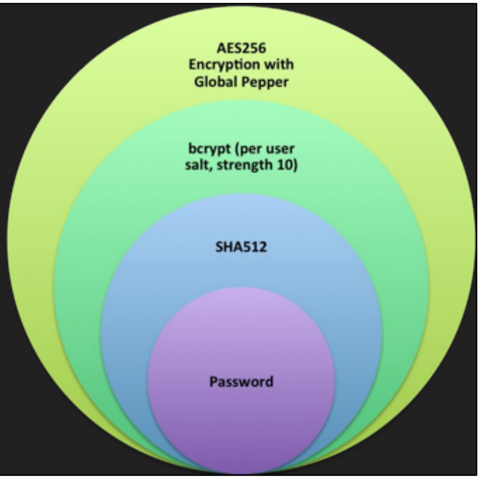
Again, not foolproof, but it'd stop many of the breaches we've discussed from being possible.

When a user logs in you
take their password.
You fetch the pepper from the vault.
You "unlock" their hash,
hash their input + salt,
and compare.
Phew!

@nickmalcolm

Like when you set a password on zip file, or your WiFi, or your password manager app.
The same password lets you unlock the secrets each time.

Dropbox,
after a breach of
68M passwords



@nickmalcolm

https://blogs.dropbox.com/tech/2016/09/how-dropbox-securely-stores-your-passwords/
They adopted this model sometime after their breach of 68,648,009 passwords, half of which were SHA1 and half were bcrypt.
Dropbox was breached in 2012, because an employee who had access to the password records was reusing a password exposed in the LinkedIn dump!

Hash & Salt the passwords.

Then symmetrically encrypt the hash using the pepper as the secret.

@nickmalcolm

Like when you set a password on zip file, or your WiFi, or your password manager app.
The same password lets you unlock the secrets each time.

Argon2id has native support for peppers.

Password storage is only

a small

part of the problem.

@nickmalcolm

With or without a pepper,
salted hashes are the way to go.

@nickmalcolm

You’re basically buying your users time to change their passwords if there’s a breach.

A determined attacker can still try to brute force a pepper,

What we *have* talked about is how to store passwords properly, if that's what we need to do

We've focused on storage, but there are lots of other factors when it comes to good authentication.
All these could be their own topics

OWASP Password Cheat Sheet

Wikipedia

google: “paragonie passwords 2016”

@nickmalcolm

Cheat sheet has examples and references
For the blender you’re using, how to set it to the right blend-time.

Wikipedia can give you insight into the mechanics of the blenders, how they work, the actual values they store in the database, etc

The paragonie blog has legit code snippets for argon2, bcrypt, and scrypt, for PHP, Node, .Net, Ruby, Java, Python

Never store the password itself.

Use a slow blender with built-in salt.
Add pepper if you can.

Then store the hash.

@nickmalcolm

In summary

Adobe Breach
https://nakedsecurity.sophos.com/2013/11/04/anatomy-of-a-password-disaster-adobes-giant-sized-cryptographic-blunder/
https://filippo.io/analyzing-the-adobe-leaked-passwords/

Argon2
https://medium.com/@mpreziuso/password-hashing-pbkdf2-scrypt-bcrypt-and-argon2-e25aaf41598e

Code snippets for password storage for PHP, .Net, Java, and Ruby
https://paragonie.com/blog/2016/02/how-safely-store-password-in-2016