Bash One-Liner to Check Your Password(s) via pwnedpasswords.com's API Using the k-Anonymity Method

Mon Follow
Feb 26, 2018 · 6 min read

. . .

TL;DR

Copy paste this to any *nix terminal:

```
echo "pass> "; read -s pass_str; sha1=$(echo -n $pass_str | tr -d
'\n' | sha1sum); echo "Hash prefix: ${sha1:0:5}"; echo "Hash suffix:
${sha1:5:35}"; result=$(curl
https://api.pwnedpasswords.com/range/${sha1:0:5} 2>/dev/null | grep
$(echo ${sha1:5:35} | tr '[:lower:]' '[:upper:]')); printf "Your
```

```
password appeared %d times in the database.\\n" "${result#*:}"
2>/dev/null
```

For Mac:

```
echo "pass> "; read -s pass_str; sha1=$(echo -n $pass_str | tr -d
'\n' | sha1sum); echo "Hash prefix: ${sha1:0:5}"; echo "Hash suffix:
${sha1:5:35}"; result=$(curl
https://api.pwnedpasswords.com/range/${sha1:0:5} 2>/dev/null | grep
$(echo ${sha1:5:35} | tr '[:lower:]' '[:upper:]')); echo "Count:
${result#*:}"
```

Sample usage:

```
$ echo "pass> "; read -s pass_str; sha1=$(echo -n $pass_str | tr -d
'\n' | sha1sum); echo "Hash prefix: ${sha1:0:5}"; echo "Hash suffix:
${sha1:5:35}"; result=$(curl
https://api.pwnedpasswords.com/range/${sha1:0:5} 2>/dev/null | grep
$(echo ${sha1:5:35} | tr '[:lower:]' '[:upper:]')); printf "Your
password appeared %d times in the database.\\n" "${result#*:}"
2>/dev/null
```

```
$ pass>
<enter your password here. In this case, we're using hello123>
```

Which gives us the following output:

```
Hash prefix: 42331
Hash suffix: 37d1c510f2e55ba5cb220b864b11033f156
Your password appeared 166621 times in the database.
```

If your password appears multiple times on the database, **CHANGE IT IMMEDIATELY**.

Now you can either close this and go your own way (😢), or you can continue reading on (👀).

. . .

Why (u do dis)

Last week, Troy Hunt released PwnedPasswords v2 as part of the Have I Been Pwned service. Containing over half a billion passwords, this database is essential in auditing the passwords we use against the list of passwords that the threat actors may be using on their attacks.

There are two ways you can go about this (at least for the old Have I Been Pwned API):

1. Send over your password to the website to check if it has been pwned

OR

2. Download the whole password dump, decompress it, and query it offline

Now, you might say that the most logical way to go about this is option 2. The password dump amounts to *approximately* 9 GB.

> *Where I'm from, that could take forever.*

So I tried option 1, with some tweaks and magic(??).

I implemented a simple(?) `bash` one-liner that checks the pwned passwords API using the k-anonymity method.

If your password appears multiple times on the database, **CHANGE IT IMMEDIATELY**.

. . .

What (k-anonywhat..?)

k-anonymity

In simple terms, the k-anonymity method aims to *anonymize* data while still making it useful.

In the context of pwned passwords, instead of querying the database for your password (we'll use hello123 as an example) as `hello123` in plaintext, you:

1. Get its SHA1 hash

```
$ echo -n "hello123" | sha1sum
4233137d1c510f2e55ba5cb220b864b11033f156
```

2. Get the first 5 characters of the SHA1 hash

`42331`

3. Perform a Range Query on the API using the first 5 characters of the SHA1 hash

```
$ curl https://api.pwnedpasswords.com/range/42331 2>/dev/null | grep
$(echo 37d1c510f2e55ba5cb220b864b11033f156 | tr '[:lower:]'
'[:upper:]')
37D1C510F2E55BA5CB220B864B11033F156:166621
```

Which means that `hello123` appeared 166,621 times on the password dump. (If you're using this password, please drop by in the comments below so I can shame you.)

This way, you're only revealing a small part of your password hash (the first 5 characters of the hash) in a query + you're checking the results yourself based on the results of your API query. Best of both worlds!

. . .

How

The one-liner is written below:

```
echo "pass> "; read -s pass_str; sha1=$(echo -n $pass_str | tr -d
'\n' | sha1sum); echo "Hash prefix: ${sha1:0:5}"; echo "Hash suffix:
${sha1:5:35}"; result=$(curl
https://api.pwnedpasswords.com/range/${sha1:0:5} 2>/dev/null | grep
$(echo ${sha1:5:35} | tr '[:lower:]' '[:upper:]')); printf "Your
password appeared %d times in the database.\\n" "${result#*:}"
2>/dev/null
```

Let's break it down:

```
echo "pass> "; read -s pass_str;
```

This just asks the user to input his/her password. The password that the user will supply will remain hidden though, through `read`'s `-s` flag. It will then be stored in variable that I decided to call `pass_str`.

```
sha1=$(echo -n $pass_str | tr -d '\n' | sha1sum)
```

The script above just calculates the SHA1 hash of `pass_str`.

```
echo "Hash prefix: ${sha1:0:5}"; echo "Hash suffix: ${sha1:5:35}"
```

The script above displays the first 5 characters of the hash and the remaining characters in order to counter-check if you're getting the right hash for the password you provided.

```
result=$(curl https://api.pwnedpasswords.com/range/${sha1:0:5}
2>/dev/null | grep $(echo ${sha1:5:35} | tr '[:lower:]'
'[:upper:]'))
```

The script above performs a range query on the API of pwnedpasswords.com using the first 5 characters of your password's hash via `curl`. The API will return a huge list of hashes that start with the first 5 characters you provided so we need to `grep` the list using the remaining characters in the password's SHA1 hash (converted to uppercase).

The last remaining parts of the script are only aesthetic additions but if you're wondering what `%{result#*:}` means, this technique is called `substring removal` which is a `parameter expansion` technique. If you remember, the result looks like this:

```
37D1C510F2E55BA5CB220B864B11033F156:166621
```

The `#` in `%{result#*:}` means remove, so `#*:` means "remove everything before the first occurrence of `:`" You can read more about parameter

expansion here.

This way, you're only sending the first 5 characters of your password hash to the operator of the website and checking the results yourself, in the comfort of your own terminal.

It may not look like much, but it works. 😎

. . .

Where (do we go from here)

If you think of yourself as a computer demi-god or anything near that, you can skip this 😯

I'll list some tips below regarding passwords and securing your accounts:

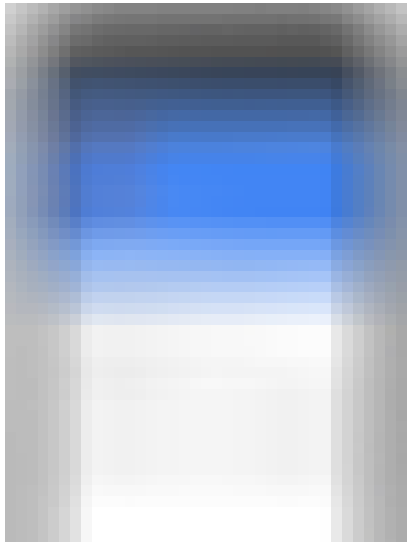
1. Use different passwords for every service
2. Use 2FA/2SV whenever possible
3. Change your passwords regularly

Different passwords for every service?

Using different passwords for every service minimizes the risk of an attacker gaining access to your other accounts in case one of your valued credentials are stolen.

2FA??

A lot of services already made it easier for us users to enable 2-Factor Authentication/2-Step Verification on our accounts. If you enable 2FA/2SV on your Google or Facebook account, for example, you will be prompted with a yes/no prompt on your (trusted, I hope) phone upon entering your email/password pair like this one:



Google Login Prompt

Other services would require you to enter a 6-digit code as your 2FA code. In such cases, you could use Authy or Google Authenticator. What this does—in case your password is already in the hands of an attacker—is that you can immediately know if someone is trying to access your account and deny them at that very same moment. Of course, please change your password once that happens.

Change your passwords regularly??

Yes, I know, I know. Maintaining a lot of passwords is too much of a hassle. Heck, you'll end up writing your passwords on notebooks or notepads, etc. right? No.

You can use Password Managers such as LastPass or KeePass to help store your passwords. One benefit in using Password Managers is that you only have to

remember **ONE password** to rule them all (your passwords). For the love of everything that's good, you only have to remember **one** password in this. Make it as complicated but remember-able(??) as possible or you'll mess this whole thing up. Password Managers also generate passwords, which makes it easier to create unique passwords for different services. It will take time to get used to, but trust me—it will pay off eventually.

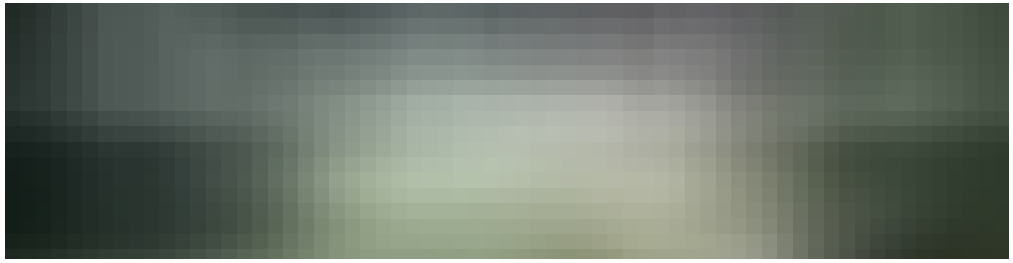
. . .

Final Thoughts

That may be a lengthy read, but I hope you learned something new, reader!

As always, thank you for reading!

Obligatory Star Wars meme:



Security

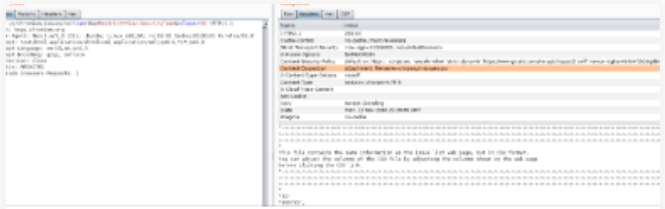
18 claps

2

Mon

Internet noob.

Follow

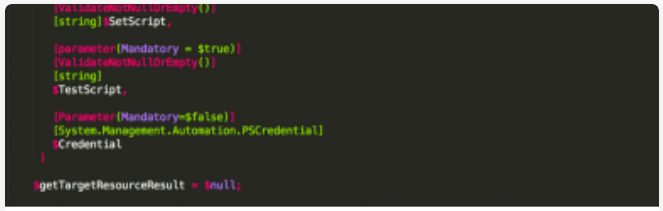


Related reads

XS-Searching Google's bug tracker to find out vulnerable source code

Luan Herrera
Nov 19, 2018 · 6 min read

915

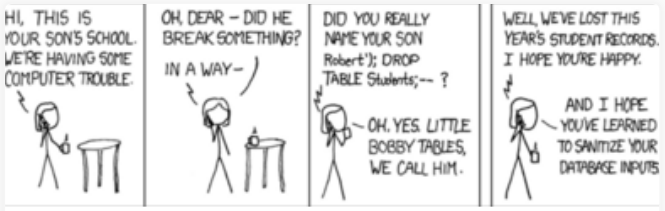


Related reads

CVE-2018–8212: Device Guard/CLM bypass using MSFT_ScriptResource

Matt Nelson
Oct 10, 2018 · 4 min read

45



Also tagged Security

How to avoid ruining lives (front-end security matters)

Luke Mayhew
Apr 30 · 6 min read ★

Responses

Write a response...

Show all responses