Protocols for Checking Compromised Credentials

Lucy Li  
Cornell Tech

Bijeeta Pal  
Cornell Tech

Junade Ali  
Cloudflare Inc.

Nick Sullivan  
Cloudflare Inc.

Rahul Chatterjee  
University of Wisconsin—Madison

Thomas Ristenpart  
Cornell Tech
Password breaches

Have I Been Pwned: 406 breaches

...and many more

<table>
<thead>
<tr>
<th>Username</th>
<th>Passwords</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="mailto:lucy@email.com">lucy@email.com</a></td>
<td>myPassword123</td>
</tr>
<tr>
<td><a href="mailto:alice@yahoo.com">alice@yahoo.com</a></td>
<td>Star246, p4ssw0rd1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Breach compilation from 2018: 1.4 billion user-password pairs
Credential Stuffing

Around 40% of users reuse passwords across different websites! [Das et al. 2014, Pearman et al. 2017]

Credential stuffing is the largest source of account takeover and automated fraud [Shape Security]

<table>
<thead>
<tr>
<th>Leaked Credentials</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Username</td>
<td>Passwords</td>
</tr>
<tr>
<td><a href="mailto:lucy@email.com">lucy@email.com</a></td>
<td>myPassword123</td>
</tr>
<tr>
<td><a href="mailto:alice@yahoo.com">alice@yahoo.com</a></td>
<td>Star246, p4ssw0rd1</td>
</tr>
</tbody>
</table>

|...|...|

- Lucy@email.com
  - myPassword123
- Alice@yahoo.com
  - Star246, p4ssw0rd1
One way to prevent credential stuffing

Limitations:
- Hard for websites to keep an up-to-date copy of credential leaks
- Don’t want to have copies of leaked credentials everywhere

Leaked Credentials

<table>
<thead>
<tr>
<th>Username</th>
<th>Passwords</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="mailto:lucy@email.com">lucy@email.com</a></td>
<td>myPassword123</td>
</tr>
<tr>
<td><a href="mailto:alice@yahoo.com">alice@yahoo.com</a></td>
<td>Star246, p4ssw0rd1</td>
</tr>
</tbody>
</table>

EXISTS
lucy@email.com
myPassword123

reset password
newPassword789

lucy@email.com
myPassword123

4
Can we use a third party checker and still preserve privacy of user credentials?

Two big initial deployments:

Password Checkup extension

Offered by: google.com
Our contributions

1. Formalization of compromised credential checking (C3) protocols and threat model

2. HIBP and Google Password Checkup leak information about passwords

3. New C3 protocols that leak less
   - Frequency-smoothing bucketization
   - ID-based bucketization
Compromised Credential Checking Services

<table>
<thead>
<tr>
<th>Leaked Credentials</th>
</tr>
</thead>
<tbody>
<tr>
<td>Username</td>
</tr>
<tr>
<td><a href="mailto:lucy@email.com">lucy@email.com</a></td>
</tr>
<tr>
<td><a href="mailto:alice@yahoo.com">alice@yahoo.com</a></td>
</tr>
</tbody>
</table>

Client finds out if their credential is in the server’s database
Threat model

1. Protect leaked credential database from malicious client

Client → C3 server

Passwords
...
dog456
abc123
1password23

Passwords
...
dog456
abc123
1password23
...
1. Protect leaked credential database from malicious client

2. Protect client’s password against malicious server

Ideally, no information about password leaked

Partial information speeds up online guessing attacks
Private set membership

Need to preserve privacy for both client credentials and server database

Client
myPassword123

C3 server

Private set membership?

Passwords
...
dog456
abc123
1password23
...

Doesn’t scale to sets containing hundreds of millions of elements
Efficiency through bucketization

Combine bucketization with some private set membership protocol
Efficiency through bucketization

Have I Been Pwned and Google Password Checkup use a hash prefix as a bucket identifier.

Key security question: How much does knowing the bucket queried help an adversary guess a client’s password?

Client
- myPassword123

C3 server

Combine bucketization with some private set membership protocol.
Have I Been Pwned (HIBP)

- Existing C3 service that checks if your username OR password exists in a known data breach

Pwned Passwords

Pwned Passwords are 555,278,657 real world passwords previously exposed in data breaches. This exposure makes them unsuitable for ongoing use as they're at much greater risk of being used to take over other accounts. They're searchable online below as well as being downloadable for use in other online systems. Read more about how HIBP protects the privacy of searched passwords.
Have I Been Pwned (HIBP)

password123
Hash of password123 = 15a56bd4dd...

20-bit hash prefix

15a56

Contains all password hashes with the same prefix

Have I Been Pwned

15a56
Security of HIBP

An attacker with access to the bucket and $Q$ guesses has about the same success rate as an attacker with $Q \times B$ guesses! ($B$ is the total number of buckets)

For HIBP, $B = 2^{20} \approx 1$ million

With no info: 431$^{st}$ guess
With hash prefix: 1$^{st}$ guess

Compromise account with $Q=10$ guesses
Issue with HIBP

Colors in buckets correspond to probabilities of passwords given the bucket.

Easy to guess the password if you know the bucket.
Frequency-smoothing bucketization (FSB)

We propose FSB as a more secure bucketization algorithm.

Goal: Given a bucket, the probability of each password in the bucket is the same.
FSB implementation details $(Q = 1)$

To check a password with the server:
Client computes range, picks a bucket randomly

### Range for password

$[ H(\text{password}) , H(\text{password}) + f(Pr(\text{password})) ]$

### B: # buckets

Buckets: 0

Start bucket: $H(\text{password})$

Proportional to probability of password
FSB: what about $Q > 1$?

- Parameter $Q$ reflects expected online guessing budget
- Include the top $Q$ passwords in every bucket, and distribute the rest proportionally to probability
FSB: guessing budget parameter

Bucket $i$: $Q = 1$
FSB: guessing budget parameter

Bucket $i$:

Buckets: 0

Q = 2

Probability

Password
Security of FSB

• Theorem: If an attacker has \( \leq Q \) guesses, access to the FSB bucket will give no advantage over baseline guessing.

• Bounds for \( > Q \) guesses shown in our paper:
  • Higher \( Q \) \( \rightarrow \) smaller security loss
  • But also larger bucket sizes
Empirical security evaluation

- How easily can an attacker guess passwords given the bucket identifiers?

- Breach dataset of 1.4 billion username-password pairs
  - Split into test set and leaked password set

- Sample passwords randomly from test set

- Record the number of guesses needed to recover the correct password

github.com/lucy7li/compromised-credential-checking
Results

![Graph showing the attacker success rate (%)]

- **HIBP**
- **FSB (Q=100)**
- **Baseline**

**Axes:**
- **Y-axis:** Attacker success rate (%)
- **X-axis:** Number of queries given to the attacker

**Legend:**
- Baseline
- Hash Prefix (20 bits)
- Frequency-smoothing (Q=100)
Recap of password-only C3 setting

• Checks if a password is in breach data
• HIBP leaks information about passwords that speeds up remote guessing attacks
• Frequency-smoothing bucketization leaks less
• However, password-only checks may have false positives
ID-password C3 protocols

• Check for an exact username-password pair match with a C3 server

• Google Password Checkup (GPC) implemented a protocol that uses the prefix of H(user || pw) as the bucket identifier
  • Bucket contains all H(user || pw) with the same hash prefix

• Runs into same security issues as HIBP, if username is known
GPC empirical security

Attacker success rate (%) vs. Number of queries given to the attacker

Baseline
Hash Prefix (20 bits)
Hash Prefix (16 bits)
Frequency-smoothing (q' = 100)

HIBP
GPC
FSB
Baseline
ID-based bucketization (IDB)

- We propose a modification of GPC that only uses the hash prefix of the username as a bucket identifier (ID-based bucketization, IDB)
  - Google independently proposed this change in their paper, as a future modification to their initial design [Thomas et al. 2019]
- We show that knowing the IDB bucket identifier gives an attacker no advantage in guessing a user’s password (over baseline guessing)!
## Performance

<table>
<thead>
<tr>
<th>Setting</th>
<th>Protocol</th>
<th>Bandwidth (KB)</th>
<th>Total time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Password-only</td>
<td>HIBP</td>
<td>32</td>
<td>220</td>
</tr>
<tr>
<td></td>
<td>FSB</td>
<td>558</td>
<td>527</td>
</tr>
<tr>
<td>ID-password</td>
<td>GPC</td>
<td>1,066</td>
<td>489</td>
</tr>
<tr>
<td></td>
<td>IDB</td>
<td>1,066</td>
<td>517</td>
</tr>
</tbody>
</table>

*Total time* includes client-server communication and client- and server-side computations.

[github.com/lucy7li/compromised-credential-checking](https://github.com/lucy7li/compromised-credential-checking)
Conclusion

• Existing deployed C3 protocols leak a lot of information about a user’s password to the C3 server

• To leak less information, we recommend using:
  • Password-only: Frequency-smoothing bucketization
  • Username-password: ID-based bucketization

• Questions?

github.com/lucy7li/compromised-credential-checking