Password breaches → Credential Stuffing

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

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

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>

Client

Entity

C3 Server

Credential

pw

OR

(u, pw)
Compromised Credential Checking Services

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

Two big initial deployments:

Password only: `;--have i been pwned?

Username-password: Password Checkup extension

Offered by: google.com
Our contributions [Li et al. CCS 2019]

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

2. Show formally and empirically that HIBP and Google Password Checkup (GPC v1) leak information about passwords

3. New C3 protocols that leak less
   • Username-password: ID-based bucketization (now GPC v2)
   • Password only: Frequency-smoothing bucketization
Today

• Empirical results that motivate the move from GPC v1 to ID-based bucketization, and hopefully from HIBP to frequency-smoothing bucketization

• Overview of frequency-smoothing bucketization, our new password-only C3 protocol
Threat model

• Protect client’s password against malicious server

Ideally, no information about password leaked

Partial information speeds up online guessing attacks
Efficiency

• A C3 server stores hundreds of millions of passwords, or billions of username-password pairs

• Need to handle client requests efficiently
Efficiency through bucketization

Combine bucketization with some private set membership protocol
Efficiency through bucketization

HIBP: prefix of H(pw)
GPC v1: prefix of H(user || pw)

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

Combine bucketization with some private set membership protocol
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

• Measure the percentage of passwords an attacker can guess in Q queries, with access to usernames

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

![Graph showing attacker success rate (%) vs. number of queries given to the attacker. The x-axis represents the number of queries ranging from 1 to 1000, and the y-axis represents the attacker success rate ranging from 0 to 80%. The graph includes data for Baseline, Hash Prefix (20 bits), Hash Prefix (16 bits), Frequency-smoothing (q' = 100), HIBP, GPC v1, FSB (new), and GPC v2.]
ID-password C3 setting

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

• Google Password Checkup (GPC v1) **initially** implemented a protocol that uses the prefix of H(user || pw) as the bucket identifier

• As we saw in empirical evaluation, this protocol has poor security, if **username is known to attacker**

• ID-based bucketization: use prefix of H(user) as the bucket identifier
Password-only C3 setting

• Checks if a password is in breach data
  • Avoids risk of C3 server storing username-password pairs

• Have I Been Pwned leaks information about passwords that speeds up remote guessing attacks

• Frequency-smoothing bucketization leaks less
Have I Been Pwned (HIBP)

Password123
Hash of password123 = 15a56bd4dd...

20-bit hash prefix

Contains all password hashes with the same prefix
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)

Goal: Given a bucket, the probability of each password in the bucket is the same

We propose FSB as a more secure bucketization algorithm
FSB implementation details \((Q = 1)\)

Password

<table>
<thead>
<tr>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Password</td>
</tr>
</tbody>
</table>

Range for password = \([ H(\cdot) , H(\cdot) + f(Pr(\cdot)) ] \)

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

Start bucket: \(H(\cdot)\)

Proportional to probability of password

B: # buckets

Buckets: 1

\(i\)
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 relative to probability of the $Q$th most popular 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
## 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](github.com/lucy7li/compromised-credential-checking)
Conclusion

• Some 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?

cs.cornell.edu/~lucy