Protecting accounts from credential stuffing with password breach alerting

Kurt Thomas, Jennifer Pullman, Kevin Yeo, Ananth Raghunathan, Patrick Gage Kelley, Luca Invernizzi, Borbala Benko, Tadek Pietraszek, Sarvar Patel, Dan Boneh, Elie Bursztein
A Little about Myself

- Lead team with goal of utilizing advanced cryptographic techniques to improve user privacy
- Research Interests: Oblivious RAM, Encrypted Search, Private Information Retrieval
Outline

Problem
Credential stuffing
Password breach alerting

Solution
Protocol for password breach alerting
Privacy guarantees
Deployment at Google
Credential Stuffing
Hackers Accumulate Breached Credentials

Hackers Are Passing Around a Megaleak of 2.2 Billion

Old hacks strike again: Data from 2.2B accounts lands on the dark web

More than 600 gigabytes of hacked accounts from years ago have been compiled and are free to download.
### What do Breached Credentials look like?

<table>
<thead>
<tr>
<th>Username</th>
<th>Salt</th>
<th>Hashed Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>UserA</td>
<td>E1F53135E559C253</td>
<td>H(PasswordA + E1F53135E559C253)</td>
</tr>
<tr>
<td>UserB</td>
<td>84B03D034B409D4E</td>
<td>H(PasswordB + 84B03D034B409D4E)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Reversing Hashes

Input.

Salt: E1F53135E559C253
Hashed password: 72AE25495A7981C40622D49

Goal.

Find password P such that $H(P + E1F53135E559C253) = 72AE25495A7981C40622D49$. 
Reversing Hashes

Why is this possible?

- Utilize specialized hardware built to compute hashes efficiently
- Can try weaker passwords first
- Passwords have low entropy

Preventions

- Use “expensive-to-compute” hashes such as Scrypt or Argon2
Hijacking Accounts using Breached Credentials

Hackers account for 90% of login attempts at online retailers

By John Derishe • July 18, 2018
Defending against Breached Credentials

Protect your account with 2-Step Verification

Each time you sign in to your Google Account, you'll need your password and a verification code.

Learn more

Add an extra layer of security
Small Tail of Websites
Small Tail of Websites

Smaller websites may not employ defence in-depth.
Imbalance of Knowledge
Imbalance of Knowledge

Attackers have wide-scale access to billions of stolen usernames and passwords.
Imbalance of Knowledge

Attackers have wide-scale access to billions of stolen usernames and passwords. Users and identity providers remain in the dark about which accounts to resecure.
Password Breach Alerting

![Password breach alerting](image)
Password Breach Alerting
Password Breach Alerting

(username₁, password₁)

(username₂, password₂)

...

(usernameₙ, passwordₙ)
Password Breach Alerting

(username₁, password₁)

(username₂, password₂)

...

(usernameₙ, passwordₙ)
Password Breach Alerting

(username, password)

(username₁, password₁)
(username₂, password₂)
...
(usernameₙ, passwordₙ)
Password Breach Alerting

(username, password)

(username, password)

(\(username_1, password_1\))

(\(username_2, password_2\))

(...)

(\(username_n, password_n\))
Password Breach Alerting

(username, password) → [Breached, Not Breached] → (username_1, password_1)

(username, password) → (username_2, password_2)

...→ (username_n, password_n)
Prior Solutions

';--have i been pwned?

Check if you have an account that has been compromised in a data breach

email address

pwned?

Generate secure, unique passwords for every account

Learn more at 1Password.com

Why 1Password?
Prior Solutions

Risk of inaccurate advice due to username-only, or password-only lookup.
Prior Solutions

Risk of inaccurate advice due to username-only, or password-only lookup.

Privacy risk of what data you share with the breach alerting service.
Design Principles
Design Principles

Password Privacy
Password Privacy

(username, password)

{Breached, Not Breached}

(username₁, password₁)

(username₂, password₂)

(...)

(usernameₙ, passwordₙ)
Password Privacy

No knowledge about the queried password.
Design Principles

Password Privacy
Design Principles

Password Privacy

Mitigate Abuse Risk
Mitigate Abuse Risk

(username₁, password₁)

(username₂, password₂)

...

(usernameₙ, passwordₙ)
Mitigate Abuse Risk

Breached credentials are sensitive data that should not be widely spread.

$(\text{username}_1, \text{password}_1)$

$(\text{username}_2, \text{password}_2)$

$\ldots$

$(\text{username}_n, \text{password}_n)$
Mitigate Abuse Risk

(username₁, password₁)

(username₂, password₂)

...

(usernameₙ, passwordₙ)
Mitigate Abuse Risk
Mitigate Abuse Risk

(username\textsubscript{1}, password\textsubscript{1})

(username\textsubscript{2}, password\textsubscript{2})

...  

(username\textsubscript{n}, password\textsubscript{n})
Mitigate Abuse Risk

(username₁, password₁)

(username₂, password₂)

(...)

(usernameₙ, passwordₙ)
Mitigate Abuse Risk

(username₁, password₁)
(username₂, password₂)
...
(usernameₙ, passwordₙ)
Mitigate Abuse Risk
Mitigate Abuse Risk

(username_1, password_1)
(username_2, password_2)
...
(username_n, password_n)
Mitigate Abuse Risk

(username_1, password_1)
(username_2, password_2)
...
(username_n, password_n)
Mitigate Abuse Risk

\[ \text{Cost(Abusing API)} > \text{Cost(Dark Web)} \]
No knowledge about another credential (username', password').

(\text{username}, \text{password})

\{\text{Breached, Not Breached}\}

\begin{align*}
(\text{username}_1, \text{password}_1) \\
(\text{username}_2, \text{password}_2) \\
& \vdots \\
(\text{username}_n, \text{password}_n)
\end{align*}

No knowledge about the queried password.
Design Principles

- Password Privacy
- Mitigate Abuse Risk
Design Principles

- Password Privacy
- Mitigate Abuse Risk
- Simple but Effective Crypto
Protocol for Password Breach Alerting
Tools: Oblivious PRF

Input: x

Secret Key: K
Tools: Oblivious PRF

Input: $x$

Secret Key: $K$
Tools: Oblivious PRF

Input: $x$

Secret Key: $K$

Receive $PRF(K, x)$. 
Tools: Oblivious PRF

Input: $x$

Receive $\text{PRF}(K, x)$.

Secret Key: $K$

No knowledge about input $x$. 

Receive $\text{PRF}(K, x)$. 
Receive PRF(K, x).

No knowledge about secret key K.

No knowledge about input x.
Tools: Oblivious PRF Protocol

Input: x

Secret Key: K
Tools: Oblivious PRF Protocol

Input: x  
Secret Key: K

H hashes to an Elliptic Curve
Tools: Oblivious PRF Protocol

Input: x

Generate ephemeral key R.

Secret Key: K

H hashes to an Elliptic Curve
Tools: Oblivious PRF Protocol

Input: $x$

Secret Key: $K$

Generate ephemeral key $R$.

$H(x)^R$ hashes to an Elliptic Curve
Tools: Oblivious PRF Protocol

Input: x

Secret Key: K

Generate ephemeral key R.

$H(x)^R$

$H(x)^{RK}$

H hashes to an Elliptic Curve
Tools: Oblivious PRF Protocol

Input: $x$

Secret Key: $K$

Generate ephemeral key $R$.

$$\text{OPRF}(K, x) = (H(x)^{RK})^{1/R} = H(x)^K$$

$H$ hashes to an Elliptic Curve
Password Breach Protocol
Password Breach Protocol

\[ u_1, p_1 \]

\[ u_2, p_2 \]

\[ \ldots \]

\[ u_n, p_n \]
Step 1: Preprocess (u, p) pairs
Step 1: Preprocess \((u, p)\) pairs

Secret Server Key: K
Step 1: Preprocess \((u, p)\) pairs

Secret Server Key: \(K\)

\[\begin{align*}
  u_1, & \ OPRF(K, u_1 \ || \ p_1) \\
  u_2, & \ OPRF(K, u_2 \ || \ p_2) \\
  \cdots
\end{align*}\]

\[u_n, \ OPRF(K, u_n \ || \ p_n)\]
Step 1: Preprocess \((u, p)\) pairs

\(H'\) hashes to an Elliptic Curve
Step 1: Preprocess \((u, p)\) pairs

\[
\begin{align*}
  u_1, H'(u_1, p_1) \\
  u_2, H'(u_2, p_2) \\
  \vdots \\
  u_n, H'(u_n, p_n)
\end{align*}
\]

\(H'\) hashes to an Elliptic Curve
Step 1: Preprocess \((u, p)\) pairs

\[ u_1, H'(u_1, p_1) \]
\[ u_2, H'(u_2, p_2) \]
\[ \ldots \]
\[ u_n, H'(u_n, p_n) \]

Secret Server Key: \(K\)

\(H'\) hashes to an Elliptic Curve
Step 1: Preprocess \((u, p)\) pairs

\[
\begin{align*}
    &u_1, H'(u_1, p_1)^K \\
    &u_2, H'(u_2, p_2)^K \\
    &\ldots \\
    &u_n, H'(u_n, p_n)^K
\end{align*}
\]

**H’** hashes to an Elliptic Curve

Secret Server Key: \(K\)
Step 2: Preprocess $u$

<table>
<thead>
<tr>
<th>$u_1$, $H'(u_1, p_1)^K$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u_2$, $H'(u_2, p_2)^K$</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>$u_n$, $H'(u_n, p_n)^K$</td>
</tr>
</tbody>
</table>

Secret Server Key: $K$
Step 2: Preprocess $u$

$H$ hashes to a string

Secret Server Key: $K$

$u_1, H'(u_1, p_1)^K$

$u_2, H'(u_2, p_2)^K$

$\ldots$

$u_n, H'(u_n, p_n)^K$
Step 2: Preprocess $u$

Secret Server Key: $K$

$H(u_1), H'(u_1, p_1)^K$

$H(u_2), H'(u_2, p_2)^K$

...$

H(u_n), H'(u_n, p_n)^K$

$H$ hashes to a string
Step 2: Preprocess u

H(u_1)_0...B-1, H'(u_1, p_1)^K

H(u_2)_0...B-1, H'(u_2, p_2)^K

... 

H(u_n)_0...B-1, H'(u_n, p_n)^K

Secret Server Key: K
Step 2: Preprocess $u$

Secret Server Key: $K$

\[
\begin{align*}
110\ldots1, & \quad H'(u_1, p_1)^K \\
010\ldots0, & \quad H'(u_2, p_2)^K \\
\vdots & \\
110\ldots1, & \quad H'(u_n, p_n)^K
\end{align*}
\]
Step 2: Preprocess $u$

Secret Server Key: $K$

- $110...1, H'(u_1, p_1)^K$
- $010...0, H'(u_2, p_2)^K$
- $...$
- $110...1, H'(u_n, p_n)^K$
Step 2: Preprocess $u$

Bucketize all OPRF evaluations according to username hash prefix.

Secret Server Key: $K$

$110...1, H'(u_1, p_1)^K$

$010...0, H'(u_2, p_2)^K$

$\ldots$

$110...1, H'(u_n, p_n)^K$
Step 2: Preprocess $u$

Secret Server Key: $K$

- **000...0**: $H'(u_3, p_3)^K$, $H'(u_{16}, p_{16})^K$, ...
- **000...1**: $H'(u_7, p_7)^K$, $H'(u_{13}, p_{13})^K$, ...
- **...**
- **111...1**: $H'(u_4, p_4)^K$, $H'(u_{19}, p_{19})^K$, ...
Query Algorithm

Secret Server Key: K

000...0: $H'(u_3, p_3)^K, H(u_{16}, p_{16})^K, ...$

000...1: $H'(u_7, p_7)^K, H(u_{13}, p_{13})^K, ...$

... 

111...1: $H'(u_4, p_4)^K, H(u_{19}, p_{19})^K, ...$
Query Algorithm

Secret Server Key: $K$

000...0: $H'(u_3, p_3)^K$, $H(u_{16}, p_{16})^K$, ...

000...1: $H'(u_7, p_7)^K$, $H(u_{13}, p_{13})^K$, ...

... 

111...1: $H'(u_4, p_4)^K$, $H(u_{19}, p_{19})^K$, ...
Query Algorithm

(u, p)

Secret Server Key: K

000...0: $H'(u_3, p_3)^K, H(u_{16}, p_{16})^K, ...$

000...1: $H'(u_7, p_7)^K, H(u_{13}, p_{13})^K, ...$

... 

111...1: $H'(u_4, p_4)^K, H(u_{19}, p_{19})^K, ...$
Query Algorithm

(u, p)

User Secret Key: R

Secret Server Key: K

000...0: $H'(u_3, p_3)^K, H(u_{16}, p_{16})^K, ...$

000...1: $H'(u_7, p_7)^K, H(u_{13}, p_{13})^K, ...$

111...1: $H'(u_4, p_4)^K, H(u_{19}, p_{19})^K, ...$
Query Algorithm

(u, p)

User Secret Key: R

H(u)_{0...B-1}

Secret Server Key: K

000...0: H'(u_3, p_3)^K, H(u_{16}, p_{16})^K, ...

000...1: H'(u_7, p_7)^K, H(u_{13}, p_{13})^K, ...

... ...

111...1: H'(u_4, p_4)^K, H(u_{19}, p_{19})^K, ...
Query Algorithm

\[
(u, p) \xrightarrow{H(u)}_{0...B-1} \xrightarrow{OPRF(u, p) \text{ Request}} \]

User Secret Key: R

Secret Server Key: K

000...0: \(H'(u_3, p_3)^K, H(u_{16}, p_{16})^K, \ldots\)

000...1: \(H'(u_7, p_7)^K, H(u_{13}, p_{13})^K, \ldots\)

111...1: \(H'(u_4, p_4)^K, H(u_{19}, p_{19})^K, \ldots\)
Query Algorithm

(u, p) \rightarrow H(u)_{0...B-1}
H'(u, p)^R

User Secret Key: R

Secret Server Key: K

000...0: H'(u_3, p_3)^K, H(u_{16}, p_{16})^K, ...
000...1: H'(u_7, p_7)^K, H(u_{13}, p_{13})^K, ...
...
111...1: H'(u_4, p_4)^K, H(u_{19}, p_{19})^K, ...
Query Algorithm

(u, p) → H(u)_{0...B-1} → H'(u, p)^R

User Secret Key: R

Secret Server Key: K

000...0: H'(u_3, p_3)^K, H'(u_{16}, p_{16})^K, ...

H(u)_{0...B-1}: H'(u_7, p_7)^K, H'(u_{13}, p_{13})^K, ...

... 

111...1: H'(u_4, p_4)^K, H'(u_{19}, p_{19})^K, ...
Query Algorithm

\[(u, p)\]

H(u)_{0...B-1}

H'(u, p)^R

\{H'(u_7, p_7)^K, H'(u_{13}, p_{13})^K, \ldots\}

User Secret Key: R

Secret Server Key: K

H(u)_{0...B-1}: H'(u_3, p_3)^K, H'(u_{16}, p_{16})^K, \ldots

000...0: H'(u_3, p_3)^K, H'(u_{16}, p_{16})^K, \ldots

H(u)_{0...B-1}: H'(u_7, p_7)^K, H'(u_{13}, p_{13})^K, \ldots

111...1: H'(u_4, p_4)^K, H'(u_{19}, p_{19})^K, \ldots
Query Algorithm

\( (u, p) \)

User Secret Key: \( R \)

\( H(u)_{0...B-1} \)

\( H'(u, p)^R \)

\{\( H'(u_7, p_7)^K, H'(u_{13}, p_{13})^K, \ldots \)\}

OPRF(\( K, u, p \)) Response

\( 000...0: H'(u_3, p_3)^K, H'(u_{16}, p_{16})^K, \ldots \)

\( H(u)_{0...B-1}: H'(u_7, p_7)^K, H'(u_{13}, p_{13})^K, \ldots \)

... \( \)

Secret Server Key: \( K \)

\( 111...1: H'(u_4, p_4)^K, H'(u_{19}, p_{19})^K, \ldots \)
Query Algorithm

(u, p)

User Secret Key: R

H(u)_{0...B-1}
H'(u, p)^R

\{H'(u_7, p_7)^K, H'(u_{13}, p_{13})^K, ...\}

H'(u, p)^{R^K}

000...0: H'(u_3, p_3)^K, H'(u_{16}, p_{16})^K, ... 

H(u)_{0...B-1}: H'(u_7, p_7)^K, H'(u_{13}, p_{13})^K, ... 

... 

111...1: H'(u_4, p_4)^K, H'(u_{19}, p_{19})^K, ...

Secret Server Key: K
Query Algorithm

(u, p)

User Secret Key: R

\{H'(u_7, p_7)^K, H'(u_{13}, p_{13})^K, \ldots\}

H'(u, p)^{RK}
Query Algorithm

(u, p)

1. Compute

\[ \text{OPRF}(K, u || p) = H'(u, p)^K. \]

\{ H'(u, p)^K_7, H'(u_{13}, p_{13})^K, \ldots \}

User Secret Key: R
Query Algorithm

1. Compute
   \[ \text{OPRF}(K, u || p) = \text{H'}(u, p)^K. \]

2. \((u, p)\) is breached iff
   \[ \text{H'}(u, p)^K \in \{ \text{H'}(u_7, p_7)^K, \ldots \}. \]

User Secret Key: \(R\)

\[ \{ \text{H'}(u_7, p_7)^K, \text{H'}(u_{13}, p_{13})^K, \ldots \} \]

\[ \text{H'}(u, p)^{RK} \]
Privacy Guarantees
Privacy Guarantees

1. Full privacy for passwords
2. K-anonymity for usernames
3. Abuse mitigations
Password Privacy

(username_1, password_1)

(username_2, password_2)

... 

(username_n, password_n)

No knowledge about the queried password.
Password Privacy

(u, p)

H(u)_{0...B-1}

H'(u, p)^R

{H'(u_7, p_7)^K, H'(u_{13}, p_{13})^K, ...}

H'(u, p)^{RK}

User Secret Key: R

Secret Server Key: K

000...0: H'(u_3, p_3)^K, H'(u_{16}, p_{16})^K, ...

H(u)_{0...B-1}: H'(u_7, p_7)^K, H'(u_{13}, p_{13})^K, ...

...}

111...1: H'(u_4, p_4)^K, H'(u_{19}, p_{19})^K, ...
Password Privacy

\[(u, p)\]

User Secret Key: \(R\)

\[H(u)_{0\ldots B-1}\]

\[H'(u, p)^R\]

\[\{H'(u_7, p_7)^K, H'(u_{13}, p_{13})^K, \ldots \}\]

\[H'(u, p)^{RK}\]

Secret Server Key: \(K\)

\[000\ldots 0: H'(u_3, p_3)^K, H'(u_{16}, p_{16})^K, \ldots \]

\[H(u)_{0\ldots B-1}: H'(u_7, p_7)^K, H'(u_{13}, p_{13})^K, \ldots \]

\[\ldots \]

\[111\ldots 1: H'(u_4, p_4)^K, H'(u_{19}, p_{19})^K, \ldots \]
Privacy Guarantees

1. Full privacy for passwords
2. K-anonymity for usernames
3. Abuse mitigations
Username Privacy

(u, p)

User Secret Key: R

H(u)

H'(u, p)

H'(u, p)^R

{H'(u_7, p_7)^K, H'(u_{13}, p_{13})^K, ...}

H'(u, p)^{RK}

Secret Server Key: K

000...0: H'(u_3, p_3)^K, H'(u_{16}, p_{16})^K, ...

H(u)_{0...B-1}: H'(u_7, p_7)^K, H'(u_{13}, p_{13})^K,

... 

111...1: H'(u_4, p_4)^K, H'(u_{19}, p_{19})^K, ...
Username Privacy

\[(u, p)\]

User Secret Key: \(R\)

\[H(u)_{0\ldots B-1}\]

\[H'(u, p)^R\]

\[\{H'(u_7, p_7)^K, H'(u_{13}, p_{13})^K, \ldots\}\]

\[H'(u, p)^{RK}\]

Secret Server Key: \(K\)

\[000\ldots0: H'(u_3, p_3)^K, H'(u_{16}, p_{16})^K, \ldots\]

\[H(u)_{0\ldots B-1}: H'(u_7, p_7)^K, H'(u_{13}, p_{13})^K, \ldots\]

\[\ldots\]

\[111\ldots1: H'(u_4, p_4)^K, H'(u_{19}, p_{19})^K, \ldots\]
K-anonymity of usernames

For any username $u'$, the server can use $H(u)_{0...B-1}$ to check:

- $H(u)_{0...B-1} \neq H(u')_{0...B-1} \rightarrow u'$ was not queried
- $H(u)_{0...B-1} = H(u')_{0...B-1} \rightarrow u'$ may have been queried

The set $\{ u' \mid H(u)_{0...B-1} = H(u')_{0...B-1} \}$ is the anonymity set. All usernames in anonymity set were possible.

The parameter $K$ refers to the size of the anonymity set.
K-anonymity of usernames

How big is the anonymity set (i.e., how large is K)?

- **Local View:** Consider only known usernames (such as only those in the breached database).
- **Universal View:** Consider all possible usernames (i.e., all possible strings of some length).
How to choose hash prefix length (B)?

Smaller B implies:

- Larger anonymity sets
- Larger communication

Choose B as small as possible while satisfying efficiency requirements.
Privacy Guarantees

1. Full privacy for passwords
2. K-anonymity for usernames
3. Abuse mitigations
Mitigate Abuse Risk

No knowledge about another credential (username’, password’).

{Breached, Not Breached}

(username₁, password₁)

(username₂, password₂)

...

(usernameₙ, passwordₙ)
Mitigate Abuse Risk

(u, p)

User Secret Key: R

\( H(u)_{0...B-1} \)

\( H'(u, p)^R \)

\{\(H'(u_7, p_7)^K, H'(u_{13}, p_{13})^K, \ldots\)\}

\( H'(u, p)^{RK} \)

\( 000...0: H'(u_3, p_3)^K, H'(u_{16}, p_{16})^K, \ldots \)

\( H(u)_{0...B-1}: H'(u_7, p_7)^K, H'(u_{13}, p_{13})^K, \ldots \)

\( 111...1: H'(u_4, p_4)^K, H'(u_{19}, p_{19})^K, \ldots \)

Secret Server Key: K
Mitigate Abuse Risk

User Secret Key: R

\[(u, p)\]

\[H(u)_{0...B-1}\]

\[H'(u, p)^R\]

\[\{H'(u_7, p_7)^K, H'(u_{13}, p_{13})^K, \ldots\}\]

\[H'(u, p)^{RK}\]

Secret Server Key: K

\[000...0: H'(u_3, p_3)^K, H'(u_{16}, p_{16})^K, \ldots\]

\[H(u)_{0...B-1}: H'(u_7, p_7)^K, H'(u_{13}, p_{13})^K, \ldots\]

\[\ldots\]

\[111...1: H'(u_4, p_4)^K, H'(u_{19}, p_{19})^K, \ldots\]
Brute Force Queries

(username_1, password_1)
(username_2, password_2)
...
(username_n, password_n)
Brute Force Prevention

1. Expensive hash functions
2. Rate-limiting
Expensive Hash Functions

(u, p)

User Secret Key: R

\[ H(u_0)_{0...B-1}, H'(u, p)^R \]

\[ \{H'(u_7, p_7)^K, H'(u_{13}, p_{13})^K, \ldots\} \]

\[ H'(u, p)^{RK} \]

\( H \) is Argon2

Secret Server Key: \( K \)

\[ H(u_1)_{0...B-1}, H'(u_1, p_1)^K \]

\[ H(u_2)_{0...B-1}, H'(u_2, p_2)^K \]

\[ \ldots \]

\[ H(u_n)_{0...B-1}, H'(u_n, p_n)^K \]
Brute Force Prevention

1. Expensive hash functions
2. Rate-limiting
Rate-Limiting

- Limit to X queries/hour using some criteria such as:
  - Signed-in account
  - Username Bucket
  - IP Address
  - ...
- For example, using signed-in accounts reduces problem to creating accounts.
Brute Force Prevention

1. Expensive hash functions
   a. Requires large computation on user devices

2. Rate-limiting
   a. No additional computation on user device
   b. Better for older or lower-end devices
Deployment at Google
Third-party breach corpus

4 billion
Unique usernames and passwords from breaches

Thomas et al. Data breaches, phishing, or malware? ... CCS’17.
Protecting users across Google products

1. Chrome extension (sunset)
2. Chrome browser
3. Google password manager
4. Android
Chrome Browser Extension

Password Checkup extension

Offered by: google.com

⭐⭐⭐⭐⭐ 269 | Productivity | ⬤ 1,003,965 users

By Google
Chrome Browser Extension

Password Checkup extension

None of the passwords you recently entered were detected in a data breach.

Learn more | Send feedback

Change your password

Password Checkup detected that your password for github.com is no longer safe due to a data breach.

You should change your password now.
Learn more

Ignore for this site | Close
Anonymous telemetry reported

**On login:** domain, timestamp, breach status, hashing performance

**On ignore:** domain, timestamp

**On password change:** domain, timestamp, original strength, new strength (zxcvbn)
Performance Metrics of Extension

- 667,000 users in first month
- 8.5s per query, half spent on proof of work
- 20 QPS at peak usage
- Graph showing lookup per 100 seconds from Feb 05 to Mar 04
Frequency of Warnings by Extension

- 21 million Logins scanned in first month
- 316,000 Breached credentials detected
- 1.5% Logins on the web are breached
User Response to Warnings by Extension

Change your password

Password Checkup detected that your password for github.com is no longer safe due to a data breach.

You should change your password now.
Learn more

26%
Breached credentials reset and resecured
Passwords Improve After Warning and Update

94% New passwords as strong or stronger than original
Chrome Browser

Check your passwords

A data breach on a site or app exposed your password. Chrome recommends checking your saved passwords now.
Google Password Manager

Google Account

Password Manager
See, change, or remove passwords you saved in your Google Account. Learn more

Password Checkup
Check your saved passwords to strengthen your security.

Check passwords
Google Password Manager

We analyzed your saved passwords and found the following issues:

- **No compromised passwords**
- **2 reused passwords**
  - Create unique passwords
- **2 accounts using a weak password**
  - Create strong passwords
Android Autofill

Sign up
username@google.com

⚠️ Change your password
A data breach on a site or app exposed your password. Google recommends changing your password on App now.

Check passwords  OK
Open Source

No description, website, or topics provided.

Manage topics

6 commits 1 branch 0 releases 3 contributors

Branch: master  New pull request

kwlyeo Update README.

Latest commit 375be83 17 days ago
Open Source

```plaintext
cromium / chromium / src / refs/heads/master / , / components / password_manager / core / browser /
leak_detection

    tree: 56cb379e8c52895bf7ea7892d39e71a80f06fcb9 [path history] [tag]
    BUILD.gn
    DEPS
    authenticated_leak_check.cc
    authenticated_leak_check.h
    authenticated_leak_check_unittest.cc
    encryption_utils.cc
    encryption_utils.h
    encryption_utils_unittest.cc
    fuzzer/
    leak_detection_api.proto
    leak_detection_check.h
    leak_detection_check_factory.h
    leak_detection_check_factory_impl.cc
    leak_detection_check_factory_impl.h
    leak_detection_check_factory_impl_unittest.cc
    leak_detection_delegate_interface.h
    leak_detection_request.cc
```
Conclusions

Privacy-preserving password checkup is protecting billions of users from account hijacking.
Thanks! Questions?