Using Amnesia to Detect Credential Database Breaches

Ke Coby Wang

Michael K. Reiter
Credential Database Breaches

- Verizon\(^1\) confirmed **3950 database breaches globally between Nov. 2018 and Oct. 2019.**

\(^1\) Verizon, “2020 data breach investigations report”
Credential Database Breaches

- Verizon\(^1\) confirmed **3950** database breaches globally between Nov. 2018 and Oct. 2019.
  - *60% of these breaches leaked credentials.*

\(^1\) Verizon, “2020 data breach investigations report”
Credential Database Breaches

- Verizon\(^1\) confirmed 3950 database breaches globally between Nov. 2018 and Oct. 2019.
  - 60% of these breaches leaked credentials.

- Recent IBM reports\(^2, 3\) with 500+ studied organizations show:

\(^{1}\) Verizon, “2020 data breach investigations report”
\(^{2}\) IBM Security, “Cost of a data breach report 2019”
\(^{3}\) IBM Security, “Cost of a data breach report 2020”
Credential Database Breaches

- Verizon\(^1\) confirmed **3950** database breaches globally between Nov. 2018 and Oct. 2019.
  - **60%** of these breaches leaked credentials.

- Recent IBM reports\(^2, 3\) with 500+ studied organizations show:
  - *The likelihood an organization will experience a data breach in the next two years is \(\sim 30\%\)\(^2\).*

---

\(^1\) Verizon, “2020 data breach investigations report”
\(^2\) IBM Security, “Cost of a data breach report 2019”
\(^3\) IBM Security, “Cost of a data breach report 2020”
Credential Database Breaches

- Verizon\(^1\) confirmed 3950 database breaches globally between Nov. 2018 and Oct. 2019.
  - 60% of these breaches leaked credentials.

- Recent IBM reports\(^2, 3\) with 500+ studied organizations show:
  - The likelihood an organization will experience a data breach in the next two years is \(~30\%\) \(^2\).
  - *It takes on average 207 days to identify a breach*\(^3\).

\(^1\) Verizon, “2020 data breach investigations report”
\(^2\) IBM Security, “Cost of a data breach report 2019”
\(^3\) IBM Security, “Cost of a data breach report 2020”
Honeywords
(Juels & Rivest 2013)

Decoy passwords (honeywords) are generated based on the real one.

Web Server
Credential Database

UID: alice@gmail.com
Password*:
password1
password2
password3
password4
password5

Real user password

* Assuming that attacker can reverse all leaked password (salted) hashes offline, Here we ignore the possible use of hashing (and salting) for simplicity.
Honeywords
(Juels & Rivest 2013)

UID: alice@gmail.com
Password:

password1
password2
password3
password4
password5

Web Server
Credential Database
Honeywords
(Juels & Rivest 2013)

UID: alice@gmail.com
Password:
  password1
  password2
  password3
  password4
  password5

The index of the real user password

Web Server
Credential Database
Honeywords
(Juels & Rivest 2013)

UID: alice@gmail.com
Password:

password1
password2
password3
password4
password5

Web Server
Credential Database
Honeywords
(Juels & Rivest 2013)

UID: alice@gmail.com
Password:
- password1
- password2
- password3
- password4
- password5

Web Server
Credential Database

Honeychecker
UID: alice@gmail.com
Password index:
- Use a 2\textsuperscript{nd} secure component to store the index of the real passwords
Honeywords
(Juels & Rivest 2013)

UID: alice@gmail.com
Password:
password1
password2
password3
password4
password5

BREACHED Web Server
Credential Database

Honeychecker
UID: alice@gmail.com
Password index:

2
Honeywords
(Juels & Rivest 2013)

UID: alice@gmail.com
Password:
password1
password2
password3
password4
password5

BREACHED Web Server Credential Database

Honeychecker
UID: alice@gmail.com
Password index:
2

"4?"
"No. Breach alert!"

User Authentication
Honeywords
(Juels & Rivest 2013)

UID: alice@gmail.com
Password:
  password1
  password2
  password3
  password4
  password5

BREACHED Web Server
Credential Database

Honeychecker
UID: alice@gmail.com
Password index:

Juels & Rivest’s proposal relies on the secret (indices) persistently stored at 2nd SECURE component.
Honeywords
(Jules & Rivest 2013)

Can we still use honeywords to detect credential database breaches without assuming the security of any persistently stored secrets?

Juels & Rivest’s proposal relies on the secret (indices) persistently stored at 2nd SECURE component.
Honeywords
(Jules & Rivest 2013)

Can we still use honeywords to detect credential database breaches without assuming the security of any persistently stored secrets?

**YES!!**
Credential Stuffing

Database breaches, phishing, malware, social engineering, etc.

Valid user ID password pairs

Login

PayPal

Login

Login

Login

Amazon
Stuffing Honeywords to Avoid Detection

alice@gmail.com:
password1
password2
password3
password4

Site A

alice@gmail.com:
password2

Site B
Stuffing Honeywords to Avoid Detection

Site A

alice@gmail.com:

password1
password2
password3
password4

REAL

Site B

alice@gmail.com:

password2

Try to log in with password1/2/3/4
Two Major Contributions
Two Major Contributions

- *Detecting credential database breaches* without relying on any persistent secret state
Two Major Contributions

- Detecting credential database breaches without relying on any persistent secret state

- Detecting credential database breaches against stuffing of honeywords at other websites
Part I.

Amnesia: Detecting credential database breaches without any persistent secret state
Decoy passwords (honeywords) are generated based on the real one.
Amnesia :: Marking Process

UID: alice@gmail.com
Password:

password1
password2*
password3
password4
password5

Web Server
Credential Database

After a successful login:
Amnesia :: Marking Process

UID: alice@gmail.com
Password:

password1
password2*
password3
password4
password5

After a successful login:

1. *Mark the last submitted password*
Amnesia :: Marking Process

UID: alice@gmail.com
Password:

*password1
*password2
password3
*password4
password5

After a successful login:

1. Mark the last submitted password
2. Mark each of other passwords with a preset probability

Web Server
Credential Database
Amnesia :: Detection Process

During a login attempt:

If the submitted password is one of the marked passwords:

**Successful login & No breach alert.**

**UID:** alice@gmail.com

**Password:**

password1*
password2*
password3
password4*
password5

Web Server
Credential Database
Amnesia :: Detection Process

During a login attempt:

If the submitted password is one of the unmarked passwords:

*Breach alert!*

**UID**: alice@gmail.com

**Password:**

- password1*
- password2*
- password3
- password4*
- password5

Web Server
Credential Database
Amnesia :: User Example

User password: password2

Submitted password: password2

During login
password1
password2
password3
password4
password5

After login
password1
password2
password3
password4
password5

The real password remains marked.
Amnesia :: Attacker Example

User password: *password2*

Submitted password: 
*password4*

During login
*password1* 
*password2* 
*password3* 
*password4* 
*password5*

After login
*password1* 
*password2* 
*password3* 
*password4* 
*password5*
Amnesia :: Attacker Example

User password: password2

During login:
- password1*
- password2*
- password3
- password4*
- password5

Submitted password: password4

After login:
- password1
- password2
- password3*
- password4*
- password5*

The submitted honeyword will remain marked.
Amnesia :: Attacker Example

User password: `password2`

During login

Submitted password: *password4*

`password1*`  
`password2*`  
`password3`  
`password4*`  
`password5`

After login

`password1`  
`password2`  
`password3*`  
`password4*`  
`password5*`

It’s possible that the real user password will be *unmarked.*
Amnesia :: Attacker Example

User password: `password2`

Submitted password:
- `password4`

During login:
- `password1*`
- `password2*`
- `password3`
- `password4*`
- `password5`

After login:
- `password1`
- `password2`
- `password3*`
- `password4*`
- `password5*`

User’s next login with the real password would trigger a breach detection.

It’s possible that the real user password will be **unmarked**.
Effectiveness
Effectiveness

![Effectiveness Graph](image-url)
Effectiveness

Expected fraction of time that Amnesia deprives the attacker from accessing user accounts before discovery of the breach.
Effectiveness

# of honeywords vs. Benefit

0 64 128 192 256

0 0.2 0.4 0.6 0.8 1

# of honeywords
Effectiveness

![Graph showing the relationship between the number of honeywords and benefit. The x-axis represents the number of honeywords, ranging from 0 to 256, and the y-axis represents benefit, ranging from 0 to 1. The graph shows an increasing trend as the number of honeywords increases.]
Effectiveness

![Graph showing the relationship between the number of honeywords and benefit. The graph indicates that as the number of honeywords increases, the benefit also increases, reaching a peak at 0.5 benefit.](image-url)
Effectiveness

Conservative evaluation:
Effectiveness

Conservative evaluation:

- The evaluation is for one account only: Attacker wins if not detected after accessing only one account.
Effectiveness

Conservative evaluation:

- The evaluation is for one account only: Attacker wins if not detected after accessing only one account.
- *The model checking computes benefit for the best possible attacker strategy.*
Part II.
Detecting password database breaches against stuffing of honeywords
Our Solution :: High-Level Idea

1. Obtain honeywords via a breach

2. Stuff Site A's honeywords at Site B

3. “Hey, someone submitted one of your honeywords here. Check this out.”
“Hey, someone submitted one of your honeywords here. Check this out.”
Our Solution :: Security Goals

“Hey, someone submitted one of your honeywords here. Check this out.”

- Should not leak Target’s stored passwords to Monitor
Our Solution :: Security Goals

“Hey, someone submitted one of your honeywords here. Check this out.”

- Should not leak Target’s stored passwords to Monitor
- Should not leak the submitted password at Monitor to Target if the password is not one of Target’s stored passwords
Our Solution :: Security Goals

“Hey, someone submitted one of your honeywords here. Check this out.”

- Should not leak Target’s stored passwords to Monitor
- Should not leak the submitted password at Monitor to Target if the password is not one of Target’s stored passwords
- **Should not allow the monitor to trigger a false detection if no breach has happened to Target**
Private Containment Retrieval (PCR)

Containment retrieval:
retrieve \( p \) only if \( S \) contains \( p \)

- If \( S \) contains \( p \): learn \( p \);
- Else: learns nothing

\[ \text{Target (Set: } S \text{)} \]

\[ \text{Monitor (element: } p \text{)} \]

Query message

Response message

Lears nothing
Private Containment Retrieval (PCR)

**Containment retrieval:**
retrieve *p* only if *S* contains *p*

---

Our construction (one round of interaction):
*Partially homomorphic encryption* + Cuckoo filters

---

**Target**
(Set: *S*)

**Monitor**
(element: *p*)

- Query message
- Response message

**If *S* contains *p*: learn *p***
**Else: learns nothing**

Learns nothing
Target generates a PCR query message on all passwords (real and decoy) and sends it to Monitor.
Monitoring Stuffing of Honeywords

For each password submitted in a FAILED login attempt at Monitor for alice@gmail.com, Monitor generates a PCR response ...
Monitoring Stuffing of Honeywords

**PCR query for alice@gmail.com**

Target → Monitor

**PCR response for alice@gmail.com**

... and sends it back to Target
Monitoring Stuffing of Honeywords

For each successfully retrieved password from Monitor’s responses, Target treats it as a password tried locally and runs the marking process.
PCR Performance

**Containment retrieval:**
retrieve $p$ only if $S$ contains $p$

```
<table>
<thead>
<tr>
<th>Target</th>
<th>Monitor</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Set: $S$)</td>
<td>(element: $p$)</td>
</tr>
</tbody>
</table>
```

- Query message
- Response message
PCR Performance

**Containment retrieval:**
retrieve $p$ only if $S$ contains $p$

```
Query message
Target (Set: $S$)
```

```
Response message
Monitor (element: $p$)
```
PCR Performance

**Containment retrieval:**

retrieve \( p \) only if \( S \) contains \( p \)

- **Target** (Set: \( S \))
- **Monitor** (element: \( p \))
PCR Performance

**Containment retrieval:**
retrieve \( p \) only if \( S \) contains \( p \)

<table>
<thead>
<tr>
<th>Target</th>
<th>Monitor</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Set: ( S ))</td>
<td>(element: ( p ))</td>
</tr>
</tbody>
</table>

- Query message
- Response message

**Response gen. time**
constant in \( |S| \)

\[ |S| = \# \text{ of honeywords} + 1 \]
PCR Performance

**Containment retrieval:**
retrieve \( p \) only if \( S \) contains \( p \)

\[ \text{Target (Set: } S \text{)} \quad \text{Monitor (element: } p \text{)} \]

- **Query message**
- **Response message**

*Response gen. time constant in \( |S| \)*

*\( |S| = \# \text{ of honeywords} + 1 \)*

*E.g., < 9ms, single core CPU (128-bit symmetric security)*
PCR Performance

**Containment retrieval:**
retrieve $p$ only if $S$ contains $p$

<table>
<thead>
<tr>
<th>Target (Set: $S$)</th>
<th>Monitor (element: $p$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query message</td>
<td>Response message</td>
</tr>
</tbody>
</table>

**Response revealing**
time constant in $|S|$,  
If $p$ is not in $S$

$|S| = \# \text{of honeywords} + 1$
PCR Performance

**Containment retrieval:**
retrieve \( p \) only if \( S \) contains \( p \)

<table>
<thead>
<tr>
<th>Target</th>
<th>Monitor</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Set: ( S ))</td>
<td>(element: ( p ))</td>
</tr>
</tbody>
</table>

Query message

Response message

Response revealing
time constant in \( |S| \),
If \( p \) is not in \( S \)

\( |S| = \# \text{ of honeywords} + 1 \)

**E.g., \(~3.5\text{ms, single core CPU}\)**
PCR Performance

**Containment retrieval:**
retrieve \( p \) only if \( S \) contains \( p \)

\[
|S| = \# \text{ of honeywords} + 1
\]
PCR Performance

Containment retrieval:
retrieves $p$ only if $S$ contains $p$

Target
(Set: $S$)

Monitor
(element: $p$)

Query message

Response message

Response message size
constant in $|S|$

$|S| = \# \text{ of honeywords} + 1$

E.g., ~1KB per response
(128-bit symmetric security)
Summary
Summary

- Amnesia can detect password database breaches without any persistent secret state
Summary

- Amnesia can detect password database breaches without any persistent secret state
- *Amnesia is extended to detect password database breaches against stuffing of honeywords*
Summary

- Amnesia can detect password database breaches without any persistent secret state
- Amnesia is extended to detect password database breaches against stuffing of honeywords
  - A new two-party private containment retrieval protocol
Summary

- Amnesia can detect password database breaches without any persistent secret state
- Amnesia is extended to detect password database breaches against stuffing of honeywords
  - A new two-party private containment retrieval protocol
  - *Converting remote stuffing to local online guessing*
Summary

- Amnesia can detect password database breaches without any persistent secret state
- Amnesia is extended to detect password database breaches against stuffing of honeywords
  - A new two-party private containment retrieval protocol
  - Converting remote stuffing to local online guessing
- Evaluation results:
  - *Amnesia would substantially shorten the time between credential database breaches and their discovery*
Summary

- Amnesia can detect password database breaches without any persistent secret state
- Amnesia is extended to detect password database breaches against stuffing of honeywords
  - A new two-party private containment retrieval protocol
  - Converting remote stuffing to local online guessing
- Evaluation results:
  - Amnesia would substantially shorten the time between credential database breaches and their discovery
  - *Our PCR is practical and well suited for Amnesia*
Summary

- Amnesia can detect password database breaches without any persistent secret state
- Amnesia is extended to detect password database breaches against stuffing of honeywords
  - A new two-party private containment retrieval protocol
  - Converting remote stuffing to local online guessing
- Evaluation results:
  - Amnesia would substantially shorten the time between credential database breaches and their discovery
  - Our PCR is practical and well suited for Amnesia

Thank you!