Honeywords:
Making Password-Cracking Detectable

Ari Juels, Ronald Rivest
RSA Laboratories Cambridge, MIT Cambridge

Presenter:
Savvas Kastanakis
Threat Model

- Adversary has stolen file of password hashes

- Offline Attack:
  - Adversary applies brute force search to find the hashed value (we assume that he also knows salt and other hashing parameters)
Common Defenses

- One approach is to make password hashing more complex and time-consuming
  - This approach though slows down the authentication process for legitimate users, and doesn’t make successful password cracking easier to detect

- Setup fake user accounts – Honeypots
  - The attacker may crack a password of such an account and raise an alarm
  - The attacker can’t distinguish a real user from honeypots (realistic usernames, user-like generated passwords)
Honeywords

- An extension of the basic idea of honeypots is *Honeywords*
- Having multiple passwords for each account, only one of each is genuine
- An auxiliary secure server is triggered in every login by the main login server to verify the request of the user
- Honeywords can emerge as a very useful layer of distributed defense, even if it is simple to implement
An example: User Login

Sugar Word
An example: Malicious Login

<table>
<thead>
<tr>
<th>User</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>HoneywordA</td>
</tr>
<tr>
<td>Bob</td>
<td>HoneywordB</td>
</tr>
<tr>
<td>Bob</td>
<td>RealPassword</td>
</tr>
<tr>
<td>Bob</td>
<td>HoneywordC</td>
</tr>
<tr>
<td>Bob</td>
<td>HoneywordD</td>
</tr>
</tbody>
</table>

Main Server

Honeychecker

<table>
<thead>
<tr>
<th>Users</th>
<th>Check</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>1</td>
</tr>
<tr>
<td>James</td>
<td>5</td>
</tr>
<tr>
<td>Nick</td>
<td>5</td>
</tr>
<tr>
<td>Bob</td>
<td>3</td>
</tr>
<tr>
<td>Emily</td>
<td>4</td>
</tr>
</tbody>
</table>
Design Principles

- **Distributed Security**
  - Protect a system as a whole
  - Diversify resources in separate domains and maintain security even if some sub-systems are compromised

- **No Additional Risk**
  - If adversary accesses the honeychecker, cannot impersonate a user

- **Simplicity**
  - Easy to implement

- **Flexibility**
  - No need to interact with the computer system not even with the user authentication – It is configured just to sent silent alarms to administrators
Problem setting:
- make the user-chosen password indistinguishable from generated honeywords

Two classes of approaches split according to whether there is an impact on user interface (UI)
- Legacy-UI: password-change
  - UI is unchanged – user chooses real password
- Modified-UI: password-change
  - UI is modified to allow for a better honeyword generation
Honeyword Generation: Legacy–UI password changes

- “Tweak” selected character positions of the password to obtain the honeywords
  - For each selected position the character of the real password is replaced by a randomly-chosen character of the same type
  - $\text{tweak('hy558',1)} \rightarrow \text{'ny558'}$

- Alternatives
  - Chaffing–by–tail–tweaking: tweak last $t$ positions of password
  - Chaffing–by–tweaking–digits: tweak last $t$ positions containing digits

- Example
  - where $t = 3$ (in tail–tweaking)
  - $\text{BG+1a745} \rightarrow \text{BG+7a305 BG+2a177 BG+9a587 BG+0a602}$
One might also like to have some honeywords that are much **harder to crack** than the average—so much so that they would probably never be cracked by an adversary.

What the adversary do with the following list?

```plaintext
kebrton1      02123dia
a71ger        forlinux
1erapc        sbgo864959
aiwkme523     aj1aob12
9,50PEe]KV.0?RI0tc&L-:IJ"b+Wol<*!NWT/pb
xyqi3tbato    a3915
#NDYRODD_!!   venlorhan
pizzhemix01   dfdhusZ2
svenesly      ’Sb123
mobopy        WORFmgtness
```

★ Having some “tough nuts” among the honeywords might give the adversary **additional reason to pause before diving in** and trying to log in with one of the cracked ones.
Honeyword Generation: Legacy-UI password changes

- Recognizes patterns and tweaks with known words of the same pattern from public dictionaries

- Example:
  - mice3blind ->
  - gold5rings, name8honey, flat7sorts

Chaffing with a password model
Honeyword Generation: Modified-UI password changes

Take a tail

Propose a password: myPassword
Append “413” to password.
Enter new password: myPassword413

Generated honeywords:
myPassword798
myPassword982
myPassword113
myPassword056
myPassword935
myPassword664

★ This makes it harder for adversaries to distinguish honeywords from sugarword
Policy Considerations

- Password eligibility
  - Require certain pass syntax (basic8)
  - Check for/disallow:
    - dictionary words
    - common passwords

- Failover mode
  - Logins can proceed if honeychecker becomes unreachable
  - Prevent denial of-service attacks through honeychecker
What happens when a honeyword is entered?

- Honeyword entered – possible actions:
  - Setting off an **alarm** or notify system administrator
  - Letting **login** proceed as usual
  - Letting the login proceed, but on a **honeypot** system
  - **Tracing** the source of the login carefully
  - **Shutting down** that user‘s account or the computer system
Take away messages

❖ A simple and powerful new line of defence in the security of hashed passwords
❖ Makes password cracking detectable
❖ Decreases the value of the stolen password hash files

➢ Published password files (e.g., one stolen from LinkedIn [30]) provide attackers with insight into how users compose their passwords
➢ Attackers can then refine their models
➢ Thus every breach of a password server has the potential to improve future attacks
➢ Some honeyword generation strategies, particularly chaffing ones, obscure actual user password choices, and thus complicate model building for would-be hash crackers.
Thank You

Any Questions?