Honey

Thanks to Ari Juels for most of this deck!
In 1943, German intelligence made a major discovery

- Body was that of British Royal Marine Capt. (Maj.) William ‘Bill’ H.N. Martin
- Spain was neutral, but…
- A German agent in a town nearby got wind of the discovery.
- Martin was hand carrying a letter…

A Spanish fisherman discovered a body washed ashore.
In 1943, German intelligence made a major discovery

- The Germans knew the Allies’ planned a major invasion, but not where.

- Martin’s letter referred to a plan for General ‘Jumbo’ Wilson to invade Greece.

- On Hitler’s order, the Germans deployed three Panzer divisions in Greece to meet the attack.

A Spanish fisherman discovered a body washed ashore.
What happened?

• The Allies invaded **Sicily**.
• Captain Martin never existed. He was a plant.
• The British went to extraordinary lengths to fabricate Martin, e.g.,
  – Found corpse of homeless man with fluid in lungs consistent with drowning
  – Chose plausibly remote location with German agent
  – Fabricated letter from Martin’s ‘father,’ love letters (“What are those horrible dark hints… about being sent off…?”), bill for engagement ring, photo of ‘fiancée’, etc., etc.
Operation Mincemeat

- Operation Mincemeat saved an estimated 40,000 Allied lives.
- It also gave rise to a movie… *The Man Who Never Was*
“In wartime, truth is so precious that she should always be attended by a bodyguard of lies.”

—Winston Churchill
Decoys

• Decoys are fake objects designed for deceit to look real.
• Examples:
  • Inflatable tanks and fighter jets
  • Bait money
• Various objectives:
  • Guide attackers away from real objectives
  • Learn about attackers' behavior
  • Detect stealthy attacks
Decoys

• Where were first decoys deployed?

In computer security, we have “honey objects”
Honeypots

• Servers set up to lure attackers for observation
• What might you learn?
  • Detect specific attack
    • E.g., database honeypot looks for SQL injection attacks
    • (Basic firewalls don’t protect against such application-level attacks.)
  • Understand intruder tactics
    • What resources is the adversary looking for?
    • Where is the attack originating? What’s the vector of attack?
Honeypots

- Honeypots are counter-intelligence
- An adversary that detects honeypots can bypass them or show false behavior
- Counter-counter-counter-intelligence
  - So... set up some honeypots that actually look like honeypots
  - E.g., Port 365 claimed by Deception Toolkit (DTK)
  - Adversary may then think he/she has found the real honeypots when he/she hasn’t... or may just back off
Honeytokens

• Help detect breaches or other forms of compromise.
• Example: Lace a credit-card database with fakes.

If a Nemo Nemosiosis transaction turns up, you know the database has been breached.
• Not totally straightforward. Why?
Decoy documents


• Help detect *insider attacks*

• Fake documents deployed in real user settings

Fig. 3. Decoy tax document with bogus user information.
Decoy documents

- Detection via
  - Egress monitoring
  - Embedded “beacon”
  - Honeytokens
- Challenge: Non-interference / false positives
- Claim: Decoys can be created that are highly believable but have low interference (with normal activity of user)
Good news and bad about password breaches

• The good news: Whenever you want to talk about password (or PII) breaches, there are very good, recent examples.

• The bad news: This is all bad news.
Reminder: Passwords are generally protected via hashing

\[ P = \text{"CatPajamas"} \]
To verify an incoming password...

\[ H(P') = H(P) \]
Recall: Password hashing

• Hashing (plus salting) forces an attacker that learns hashes to determine passwords by brute-force (offline) guessing

• Brute-force guessing means the attacker repeatedly makes a guess $P'$ and checks if $H(P') = H(P)$

• Additionally, hashing can be hardened (slowed) in various ways (e.g. bcrypt)

• This all seems good, but…
Password hashing

• Remember: real passwords are weak and easily guessed.
  – Guessing probability (GP) in RockYou was 0.9%
  – Consistent across studies, e.g., Bonneau’s 69+ million Yahoo! password study was 1.08%

• Even good (& salted) hashes are often inadequate.
• Let’s just assume that hashes can be cracked and passwords are effectively in the clear.
Adversarial model

- "Smash-and-grab" attack
  - The adversary compromises the system ephemerally (usually passively).
- The adversary:
  - Steals a snapshot of password file;
  - Impersonate user(s)
Adversary always wins

Alice: “Alice”, P

P
Honeywords

Alice:

$P_1$

$P_2$

$\ldots$

$P_n$
Honeywords

Alice:

\[ P_1, P_2, \ldots, P_i = P, \ldots, P_n \]

True password
Honeywords

Alice:

\[ P_1 \]
\[ P_2 \]
\[ \ldots \]
\[ P_i = P \]
\[ \ldots \]
\[ P_n \]

Honeywords (decoys)
Alice:

\[ P_1, P_2, \ldots, P_n \]

Sweetwords

Honeywords
The adversarial game

What is $i$?

“Alice”, $P_j$

Alice:

$P_1$

$P_2$

... 

$P_n$

Given ideal honeywords, the attacker will guess correctly, $j = i$, with (small!) probability, about $1/n$. 
The adversarial game

Which is the (real) password?

Alice:

- 5512lockerno.
- tribal_3
- cshcsh.meowr.18
- 28/07/89rm
- anto_2001_jesu
- CRFRALAASS$4
- !v0nn3
- ponk.m4t
Honeyword design questions

1. Verification: How does the system check whether a submitted password $P'$ is the true password $P_i$?
   - How is index $i$ verified without storing $i$ alongside passwords?

2. Generation: How are honeywords generated?
   - How do we make bogus passwords look real?

(Many other design questions, e.g., how to respond when breach is detected using honeywords...
Honeywords: Verification

Alice: $P_1, P_2, \ldots, P_i, \ldots, P_n$

Computer System

Honeychecker

Alice’s password index $i$
Honeywords: Verification

Alice:

- $P_1$
- $P_2$
- $P_i$
- $P_n$

Computer System

Honeychecker

Alice’s password index

✔
Honeywords: Verification

Alice: $P_1, P_2, \ldots, P_i, \ldots, P_n$

Computer System

$P_j$

j

$\neq i$

Honeychecker

Alice’s password index

\[ \text{Alice: } P_1, P_2, \ldots, P_i, \ldots, P_n \]
Honeywords: Verification Rule

- If the true password $P_i$ is submitted, the user is authenticated.
- If a password $P' \neq P_1 \ldots P_n$ is submitted, it’s treated as a normal password authentication failure.
- If a honeyword $P_j \neq P_i$ is submitted, an alarm is raised by the honeychecker.
  - This is likely to happen only after a breach!
  - Honeywords (if properly chosen) will rarely be submitted otherwise.
- Note: No change in the user experience!
Some nice features of this design

• Computer system does nothing but transmit sweetword index j
  – Little modification needed

• We get the benefits of distributed security
  – Compromise of either component isn’t fatal
  – No single point of compromise
  – Compromise of both brings us back to hashed case

• Honeychecker can be minimalist, (nearly) input-only
  – Only (rare) output is alarm
Some nice features of this design

- Honeychecker can be offline
  - E.g., honeychecker sits downstream in security operations center (SOC)
  - Not active in authentication itself, but gives rapid alert in case of breach
  - If honeychecker goes down, users can still authenticate
Honeyword generation

Which is Alice’s real password?

<table>
<thead>
<tr>
<th>Alice:</th>
</tr>
</thead>
<tbody>
<tr>
<td>• QrMdmkQt</td>
</tr>
<tr>
<td>• AP9LXEEa</td>
</tr>
<tr>
<td>• m7xnQVV4</td>
</tr>
<tr>
<td>• kingeloi</td>
</tr>
<tr>
<td>• y5BJKWhA</td>
</tr>
</tbody>
</table>
Honeyword generation: "Chaffing with a password model"

- Password-hash crackers learn model from lexicon of breached passwords (e.g., RockYou database)
  - Make guesses from model probability distribution
- Idea: Repurpose cracker as generator!
- Simple (splicing) generator yields…

<table>
<thead>
<tr>
<th>Alice:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>•</td>
<td>qivole</td>
</tr>
<tr>
<td>•</td>
<td>paloma</td>
</tr>
<tr>
<td>•</td>
<td>123asdf</td>
</tr>
<tr>
<td>•</td>
<td>Compaq</td>
</tr>
<tr>
<td>•</td>
<td>asdfway</td>
</tr>
</tbody>
</table>
But there are problem cases…

Which is Alice’s real password?

Alice:
• hi4allaspls
• #1spongebobsmymansodonttouchhim
• Travis46
• #1bruinn
• KJGS^!*ss
Honeyword generation: Chaffing by tweaking

- [ZMR10] observed users tweak passwords during reset (e.g., HardPassword1, HardPassword2, …)
  - Proposed tweak-based cracker
- Idea: Tweak password to generate honeywords!
- E.g., tweak numbers in true password…

<table>
<thead>
<tr>
<th>Alice:</th>
</tr>
</thead>
<tbody>
<tr>
<td>• yamahapacificer32145678987654321</td>
</tr>
<tr>
<td>• yamahapacificer12345678987654321</td>
</tr>
<tr>
<td>• yamahapacificer12345678901234567</td>
</tr>
<tr>
<td>• yamahapacificer62145678987654322</td>
</tr>
</tbody>
</table>
Honeyword generation:  
A research challenge

• Blink-182 is a rock band
• This password is semantically significant
  – Tweaking would break it
  – Generation is unlikely to yield it
• Dealing with such passwords is a special challenge—relates to natural language processing
• Subject of an upcoming paper…
  – E.g., use other people’s passwords as honeywords…

Alice:

• Blink123
• Graph128
• Froggy%71
• Blink182
• Froggy!83
How good does honeyword generation have to be?

• Let $U$ be a probability distribution on user password selection
  – i.e., user chooses $P$ w.p. $U(P)$
• Let $G$ be a distribution on honeyword generation
  – i.e., honeyword $P$ generated w.p. $G(P)$
• Given list $P_1, \ldots, P_n$, adversary’s optimal strategy is to guess $P_j$ that maximizes $U(P_j) / G(P_j)$
• Thus, given chaffing-with-a-password-model, a particularly dangerous password is, e.g.:

#1spongebobsmymansodonttouchhim
How good does honeyword generation have to be?

• We might imagine practical choice of, say,
  • $n = 20$

• With a “flat” honeyword distribution, $U \approx G$, adversary hits a honeyword w.p. 95%

• Perfect flatness isn’t required

• Even if adversary can rule out all but two sweetwords, we can still detect a breach systemically with high probability
  – E.g., 50% guessing success means prob. $2^{-m}$ of compromising $m$ accounts without detection
How good does honeyword generation have to be?

- Generation strategies can be hybridized as a hedge against failure of one strategy, e.g.,

<table>
<thead>
<tr>
<th>qivole!</th>
<th>qivole#</th>
</tr>
</thead>
<tbody>
<tr>
<td>123asdf</td>
<td>111asdf</td>
</tr>
<tr>
<td>IBetNSACantCrackThisPassword89</td>
<td>IBetNSACantCrackThisPassword12</td>
</tr>
<tr>
<td>Froggy%71</td>
<td>Froggy!88</td>
</tr>
</tbody>
</table>
Takeaways

• Deception is an age-old tactic
  • Pioneered by Mother Nature

• It is very useful in computer security
  • Honeypots, honeytokens, honeywords, honey encryption

• You’ll get to play with it for the next month…