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…
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.
  
  Nemo Nemosious MC 5466 1602 8888 8888 exp: 05/2017 CVV: 913
  
- 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
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
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:

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

Alice:

True password

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

Alice:

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

Honeywords (decoys)
Honeywords

Sweetwords

Alice:

\[ P_1, P_2, \ldots, P_n \]
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
- CRFRALASS$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: \[ P_1, P_2, \ldots, P_i, \ldots, P_n \]

Alice’s password index

Honeychecker

Computer System
Honeywords: Verification

Alice:

$P_1$

$P_2$

$\ldots$

$P_i$

$\ldots$

$P_n$

$P_j$

Computer System

Honeychecker

Alice’s password index

$\neq i$

Honeychecker

Alice
Honeywords: Verification Rule

- If the true password $P_i$ is submitted, the user is authenticated.
- If a password $P' \notin \{P_1, \ldots, P_n\}$ 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?

Alice:

• QrMdmkQt
• AP9LXEEna
• m7xnQVV4
• kingeloi
• y5BJKWhA
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 in our paper yields...

Alice:
- qivole
- paloma
- 123asdf
- Compaq
- asdfway
But there are problem cases…

Which is Alice’s real password?

<table>
<thead>
<tr>
<th>Alice:</th>
</tr>
</thead>
<tbody>
<tr>
<td>• hi4allaspls</td>
</tr>
<tr>
<td>• #1spongebobsymansodonttouchhim</td>
</tr>
<tr>
<td>• Travis46</td>
</tr>
<tr>
<td>• #1bruinn</td>
</tr>
<tr>
<td>• KJGS^!*ss</td>
</tr>
</tbody>
</table>
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>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>• yamahapacificer32145678987654321</td>
<td></td>
</tr>
<tr>
<td>• yamahapacificer12345678987654321</td>
<td></td>
</tr>
<tr>
<td>• yamahapacificer12345678901234567</td>
<td></td>
</tr>
<tr>
<td>• yamahapacificer62145678987654322</td>
<td></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…

<table>
<thead>
<tr>
<th>Alice:</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Blink123</td>
</tr>
<tr>
<td>• Graph128</td>
</tr>
<tr>
<td>• Froggy%71</td>
</tr>
<tr>
<td>• <strong>Blink182</strong></td>
</tr>
<tr>
<td>• Froggy!83</td>
</tr>
</tbody>
</table>
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>
Honey Encryption

Another problem with passwords

- Remember: users select weak passwords.
  - So password hashes can be easily cracked.
- Weak passwords bad not just for authentication, but for Password-Based Encryption (PBE)
- What's PBE?
Another problem with passwords

• Remember: users select weak passwords.
  – So password hashes can be easily cracked.
• Weak passwords bad not just for authentication, but for Password-Based Encryption (PBE)
• What's PBE?
Another problem with passwords

- How would you break this?
- Brute-force password guessing!
  - Like cracking hash
Another problem with passwords

• Remember: users select weak passwords.
  – So password hashes can be easily cracked.
• Weak passwords bad not just for authentication, but for Password-Based Encryption (PBE)
  – Suppose a message \( m \) is encrypted under a password \( P \) using PBE as ciphertext \( c \).
  – An attacker that guesses \( P \) can crack \( c \) and learn \( m \).
  – Given weakness of passwords, guessing \( P \) is often easy.
  – Therefore, ordinary users’ PBE ciphertexts are vulnerable to brute-force guessing!
• PBE security limited by \textit{brute-force bound}!
Another application: Password vaults

- Users perhaps choose master passwords even worse than ordinary ones.
- Vaults are also stored in the cloud... protected under the master password.
- There’s already been a breach of one vault service.
  - LastPass
  - Not once but twice!
    - Last in June 2015
- A cracked vault is very serious!

<table>
<thead>
<tr>
<th>Web site</th>
<th>Login</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.ifca.org">www.ifca.org</a></td>
<td>Alice</td>
<td>SleepyGrumpy</td>
</tr>
<tr>
<td><a href="http://www.mybank.com">www.mybank.com</a></td>
<td>Alice</td>
<td>HappyDopey</td>
</tr>
<tr>
<td><a href="http://www.iacr.org">www.iacr.org</a></td>
<td>Alice</td>
<td>DocBashful</td>
</tr>
</tbody>
</table>
How do we prevent brute-force PBE cracking?

• We can slow down decryption… but not enough. (At least not without inconveniencing users.)
• Can we somehow encrypt a ciphertext under a weak password to withstand *arbitrarily powerful* cracking adversary?
• Idea: Create cipher such that adversary can’t tell if message has been correctly decrypted
  – Ciphertext decrypts plausibly under any key
Example: password manager

True passwords:

<table>
<thead>
<tr>
<th>Web site</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>ifca.org</td>
<td>SleepyDopey</td>
</tr>
<tr>
<td>mybank.com</td>
<td>HappySad</td>
</tr>
<tr>
<td>iacr.org</td>
<td>Mammoth</td>
</tr>
</tbody>
</table>

Incorrect password guess 1

<table>
<thead>
<tr>
<th>Web site</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>ifca.org</td>
<td>SleepyDoc</td>
</tr>
<tr>
<td>mybank.com</td>
<td>princess</td>
</tr>
<tr>
<td>iacr.org</td>
<td>Large</td>
</tr>
</tbody>
</table>
Example: password manager

True passwords:

<table>
<thead>
<tr>
<th>Web site</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>ifca.org</td>
<td>SleepyDopey</td>
</tr>
<tr>
<td>mybank.com</td>
<td>HappySad</td>
</tr>
<tr>
<td>iacr.org</td>
<td>Mammoth</td>
</tr>
</tbody>
</table>

Incorrect password guess 2

<table>
<thead>
<tr>
<th>Web site</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>ifca.org</td>
<td>hshgdf&amp;6^</td>
</tr>
<tr>
<td>mybank.com</td>
<td>Buddy</td>
</tr>
<tr>
<td>iacr.org</td>
<td>Ninja</td>
</tr>
</tbody>
</table>

Decryption

Encrypted vault (ciphertext)
Adversarial game

Message-recovery game

1. Message \( m \) is picked from message distribution
2. Key \( k \) is picked from a key / password distribution
3. Message \( m \) is encrypted under \( k \), yielding ciphertext \( c \)
4. Adversary is given \( c \) and outputs a guess \( m' \) for the message
   - Adv knows message and key distributions
5. Adversary wins if \( m' = m \)

- With PBE:
  - Adv. wins w.h.p. after \( \mu \frac{1}{2} \) password guesses
  - Unbounded adversary wins 100% of the time
- With HE, we can do better…
Adversarial game

- E.g., tonight's movie
- In what country will the invasion happen?

Message distribution

- “SPAIN” $p = \frac{1}{4}$
- “GREECE” $p = \frac{1}{4}$
- “SICILY” $p = \frac{1}{2}$
Honey encryption: Generator $G$

Seed space

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
</tbody>
</table>

Message space

"SPAIN"

"GREECE"

"SICILY"
A **good** generator $G$ models the message distribution

<table>
<thead>
<tr>
<th>Seed space</th>
<th>Message distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>“SPAIN” $p = \frac{1}{4}$</td>
</tr>
<tr>
<td>01</td>
<td>“GREECE” $p = \frac{1}{4}$</td>
</tr>
<tr>
<td>10</td>
<td>“SICILY” $p = \frac{1}{2}$</td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
</tbody>
</table>

$p = \frac{1}{2}$
$G$ must be invertible

\[
\begin{array}{c}
\text{s} \\
00 \\
01 \\
10 \\
11 \\
\end{array} \\
\begin{array}{c}
\text{Seed} \\
\text{space} \\
\end{array}
\]

\[
\begin{array}{c}
\text{m} \\
\text{“SPAIN”} \\
\text{p} = \frac{1}{4} \\
\text{“GREECE”} \\
\text{p} = \frac{1}{4} \\
\text{“SICILY”} \\
\text{p} = \frac{1}{2} \\
\end{array} \\
\begin{array}{c}
\text{Message} \\
\text{distribution} \\
\end{array}
\]
Natural $G$ our toy example

Seed space

00
01
10
11

Message distribution

“SPAIN”
$p = \frac{1}{4}$

“GREECE”
$p = \frac{1}{4}$

“SICILY”
$p = \frac{1}{2}$

$p = \frac{1}{2}$
HE also uses a mapping from keys / passwords onto seeds

```
Key/password distribution
```

```
<table>
<thead>
<tr>
<th>Key/password</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;secret&quot;</td>
<td>1/3</td>
</tr>
<tr>
<td>&quot;hush!&quot;</td>
<td>1/3</td>
</tr>
<tr>
<td>&quot;Buddy&quot;</td>
<td>1/3</td>
</tr>
</tbody>
</table>
```

```
Seed space
```

```
<table>
<thead>
<tr>
<th>Seed space</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
</tr>
<tr>
<td>01</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>11</td>
</tr>
</tbody>
</table>
```

The mapping function $F$ maps these keys to the seed space.
Honey encryption (HE)  
[simplified]

- Given a message $m$ and a key $k$…
- Compute a seed $s_m = G^{-1}(m)$
  - Seed corresponding to message $m$
- Compute seed $s_k = F(k)$
  - Seed corresponding to key / password $k$
- Encrypt by composing seeds
  - $c = s_k \text{ XOR } s_m$
Example:
Encrypting “SICILY” under “hush!”

Key/password distribution

<table>
<thead>
<tr>
<th>Secret</th>
<th>P = 1/3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hush!</td>
<td>P = 1/3</td>
</tr>
<tr>
<td>Buddy</td>
<td>P = 1/3</td>
</tr>
</tbody>
</table>

Seed space

<table>
<thead>
<tr>
<th>00</th>
<th>01</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
</table>

Message distribution

<table>
<thead>
<tr>
<th>Greece</th>
<th>P = 1/4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spain</td>
<td>P = 1/4</td>
</tr>
<tr>
<td>Sicily</td>
<td>P = 1/2</td>
</tr>
</tbody>
</table>

\[ F(k) \ XOR \ G^{-1}(m) = s_k \ XOR \ s_m = 11 \ XOR \ 10 = 01 = c \]
Example: Decrypting $c = 01$ under “hush!”. 

- Key/password distribution:
  - “secret” $p = 1/3$
  - “hush!” $p = 1/3$
  - “Buddy” $p = 1/3$

- Seed space:
  - 00
  - 01
  - 10
  - 11

- Message distribution:
  - “SPAIN” $p = 1/4$
  - “GREECE” $p = 1/4$
  - “SICILY” $p = 1/2$

- XOR $c$
Example:
Deciphering \( c = 01 \) under “secret”

```
```

```
```

```
```

Intuition: Decryption under any key yields a valid message!
What can we say about HE?

• Given a perfect $G$, an adversary’s optimal strategy is (roughly) to choose the most likely password (GPP) and decrypt.

• Adversary can guess the message $m$ with probability at most $\approx GP$.
  – Recall for passwords, $GP \approx 1\%$
  – With regular PBE, adversary will eventually win 100% of the time (given bounded-length password)!
Homework #3: Rolling your own honeywords
Why study Honeywords?
They draw many concepts together…

• User password choices
Honeywords bring together much of what we’ve studied...

- Indistinguishability as adversarial model
Security game

World 0
Real password

World 1
Honeyword

World 0 or World 1?

Eve
Statistical classification

Password / Honeyword Classifier

Model Cloud9-666

Password!
Honeyword!

Eve

P
Homework #3

• Build honeyword generators:
  • You will write a program that:
    • Takes a real password $P$ as input
    • Outputs a list $P_1, P_2, \ldots, P_n$ of sweetwords
  • E.g.,

Password1 ➔ Generator ➔ sjdhf&ha&!a
Password123
Password1
ninja512
Pas$word1
Homework #3

• Three different types of data sets
  1. No password data
  2. Top 100 RockYou passwords
  3. Full RockYou data set
• No online queries
  • Why?
• (At most) half-page written description of strategy for each
Homework #4

• Break honeyword generators!
• You’ll write a program that:
  • Takes a list $P_1, P_2, \ldots, P_n$ of sweetwords as input
  • Outputs a guess $j$ of the index of the real password
• What will you try to break?
  • Classmates’ Homework #3 schemes!
    • You won’t get source code, just outputs
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…