Investigating the Distribution of Password Choices

David Malone and Kevin Maher, Hamilton Institute, NUI Maynooth.

19 April 2012
How to Guess a Password?

Passwords are everywhere. If you don’t know the password, can you guess it?

1. Make a list of passwords.
2. Assess the probability that each was used.
3. Guess from most likely to least likely.

A dictionary attack, but with optimal ordering.

(Appplies to computers and keys too.)
How long will that take?

If we knew probability $P_i$ of $i^{th}$ password.
Rank the passwords from 1 (most likely) to N (least likely).
Average number of guesses is:

$$G = \sum_{i=1}^{N} iP_i.$$

Note, not the same as Entropy (Massey ’94, Arikan ’96).
Does this $P_i$ really make sense?
Is there a distribution with which passwords are chosen?
Outline

• Is there password distribution?
  Is knowing it better than a crude guess?

• Are there any general features?
  Do different user groups behave in a similar way?

• Some distributions better than others.
  Can we help users make better decisions?
Want a collection of passwords to study distribution. Asked Yahoo, Google.

- ... Cracker eventually obliged.
  - 2006: flirtlife, 98930 users, 43936 passwords, 0.44.
  - 2009: hotmail, 7300 users, 6670 passwords, 0.91.
  - 2009: computerbits, 1795 users, 1656 passwords, 0.92.
  - 2009: rockyou, 32603043 users, 14344386 passwords, 0.44.

Good: cleartext!
Bad: Had to clean up data.
<table>
<thead>
<tr>
<th>Rank</th>
<th>hotmail</th>
<th>#users</th>
<th>flirtlife</th>
<th>#users</th>
<th>computerbits</th>
<th>#users</th>
<th>rockyou</th>
<th>#users</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>123456</td>
<td>48</td>
<td>123456</td>
<td>1432</td>
<td>password</td>
<td>20</td>
<td>123456</td>
<td>290729</td>
</tr>
<tr>
<td>2</td>
<td>123456789</td>
<td>15</td>
<td>ficken</td>
<td>407</td>
<td>computerbits</td>
<td>10</td>
<td>12345</td>
<td>79076</td>
</tr>
<tr>
<td>3</td>
<td>111111</td>
<td>10</td>
<td>12345</td>
<td>365</td>
<td>123456</td>
<td>7</td>
<td>123456789</td>
<td>76789</td>
</tr>
<tr>
<td>4</td>
<td>12345678</td>
<td>9</td>
<td>hallo</td>
<td>348</td>
<td>dublin</td>
<td>6</td>
<td>password</td>
<td>59462</td>
</tr>
<tr>
<td>5</td>
<td>tequiero</td>
<td>8</td>
<td>123456789</td>
<td>258</td>
<td>letmein</td>
<td>5</td>
<td>iloveyou</td>
<td>49952</td>
</tr>
<tr>
<td>6</td>
<td>000000</td>
<td>7</td>
<td>schatz</td>
<td>230</td>
<td>qwerty</td>
<td>4</td>
<td>princess</td>
<td>33291</td>
</tr>
<tr>
<td>7</td>
<td>alejandro</td>
<td>7</td>
<td>12345678</td>
<td>223</td>
<td>ireland</td>
<td>4</td>
<td>1234567</td>
<td>21725</td>
</tr>
<tr>
<td>8</td>
<td>sebastian</td>
<td>6</td>
<td>daniel</td>
<td>185</td>
<td>1234567</td>
<td>3</td>
<td>rockyou</td>
<td>20901</td>
</tr>
<tr>
<td>9</td>
<td>estrella</td>
<td>6</td>
<td>1234</td>
<td>175</td>
<td>liverpool</td>
<td>3</td>
<td>12345678</td>
<td>20553</td>
</tr>
<tr>
<td>10</td>
<td>1234567</td>
<td>6</td>
<td>askim</td>
<td>171</td>
<td>munster</td>
<td>3</td>
<td>abc123</td>
<td>16648</td>
</tr>
</tbody>
</table>

(c.f. Imperva analysis of Rockyou data, 2010)
Distribution?

**hotmail**

**flirtlife**

**computerbits**

**rockyou**
• A straight line on a log-log plot points towards heavy tail.
• Zipf?
  \[ P_r \propto \frac{1}{r^s} \]
• Slope gives \( s \).
• Can check p-values (Clauset ’09).
• \( s \) is small, less than 1.
Guesswork Predictions

**hotmail**

**flirtlife**

**computerbits**

**rockyou**
Who cares?

- Algorithm Design — exploit heavy tail?
- Can we get close to optimal dictionary attack?
- Can we make dictionary attack less effective?

2 and 3 answer questions about common behavior and helping users.
Dictionary Attack

Suppose we use one dataset as a dictionary to attack another.
Dictionary Attack — Same Story

![Graphs showing optimal guessing and guesses from different sources for hotmail, flirtlife, computerbits, and rockyou.](image)
Dictionary Attack Gawker

December 2010, Gawker, 748090 DES Hashes, well salted.

Results in paper for % passwords. Dell’Amico’10 review smart generators. This looks $\times 10$!
Helping Users

If users select passwords ‘randomly’, can we make them a better generator?

- Banned list (e.g. twitter),
- Password rules (e.g. numbers and letters).
- Act like a cracker (e.g. cracklib),
- Cap peak of password distribution (e.g. Schechter’10),
- Aim for uniform?

Metropolis-Hastings algorithm takes bad random number generator and makes it good.
Keep a frequency table $F(x)$ for requests to use password $x$.

1. Uniformly choose $x$ from all previously seen passwords.
2. Ask user for a new password $x'$.
3. Generate a uniform real number $u$ in the range $[0, F(x')]$ and then increment $F(x')$. If $u \leq F(x)$ go to step 4 (accept), otherwise return to step 2 (reject).
4. Accept use of $x'$ as password.
Rockyou-based test, 1000000 users, mean tries 1.28, variance 0.61.

Could be implemented using min-count sketch. Doesn’t store actual *use* frequencies. No parameters, aims to flatten whole distribution.
Conclusions

- Idea of distribution of password choices seems useful.
- Zipf is OK, but not perfect match.
- Different user groups have a lot in common (not peak).
- Dictionaries not great for dictionary attacks.
- Treat users as random password generators?
- Future: Generalise beyond web passwords?
- Future: Field test of Metropolis-Hastings?
- Future: What does optimal banned list look like?
From Reviews

• So much cool literature from (at least) 1979–2012.
• In security, passwords are the gift that keeps on giving.