Password Security

David Malone
with Kevin Maher, Sandra van Wijk and Dan Hillary

24 January 2018

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.

Want to tell you a bit about the theory of guessing, and what this game looks like.

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:

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

Applies to computers and keys too. `sci.crypt` FAQ:

> *We can measure how bad a key distribution is by calculating its entropy. This number E is the number of "real bits of information" of the key: a cryptanalyst will typically happen across the key within 2^E guesses. E is defined as the sum of $-p_K \log_2 p_K$, where p_K is the probability of key K.*

Relationship to Entropy?

Could guesswork be the same as entropy?

$$G(P) = \sum_{i=1}^{N} iP_i \overset{?}{\approx} 2^{-\sum P_i \log_2 P_i} = 2^{H(P)}$$

The answer is no (Massey '94), Entropy underestimates.

$G = H$ *Asymptoticly?*

Asymptotic Equipartition: Look at i.i.d sequences of *typical* passwords of length n with probability within ϵ of $2^{-nH(P)}$.

As $n \to \infty$ these typical words have all the mass and are roughly equally likely. Does $G(P^n) \approx H(P^n)$?

You can get asymptotic result:

$$G(P^n) \asymp \left((\sqrt{p_1} + \sqrt{p_2} + \dots)^2 \right)^n$$

Family of asymptotic results (Arikan '96, Malone and Sullivan 2004, Christiansen and Duffy 2012, ...).

Twenty Questions

- Suppose we can ask "Is the password one of X, Y, . . . ?".
- We still want to identify the exact password.
- Basically building binary tree, based on yes/no.
- Want minimise expected leaf depth.

Same problem as encoding a message into minimal number of bits.

$H(P)$ is a lower bound. Huffman Encoding (Huffman '52) optimal.

$$H(P) \leq \mathbb{E}(\text{set guesses}) \leq H(P) + 1.$$

Questions

- Does this P_i really make sense?
- Is there a distribution with which passwords are chosen?
- How would we find out?

(Mainly from Malone and Maher, WWW'12)

How are Passwords Stored?

Passwords are usually *hashed* and *salted*[1].
Storage:

1. Ask user for password, choose random unique salt.
2. Calculate hash $= h(\text{salt.password})$.
3. Store username, salt and hash.

Verify

1. Ask for username and password.
2. Look up salt and hash for username.
3. Check if hash $= h(\text{salt.password})$.

h maps strings to fixed length.
Preimages should be hard. h should slow, but not too slow[2]

[1] This has been known for ages. See *Password Security: A Case History* from 1979.

[2] 25 graphics card cracker does 350,000,000,000 NTLM/s.

Getting data

Want a collection of passwords to study distribution.
Asked Yahoo, Google.

- ...

Crackers 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.
- 2013: adobe, 129576416 users, 55855039 passwords, 0.43.

First four in clear text!
Had to clean up data.

Top Ten

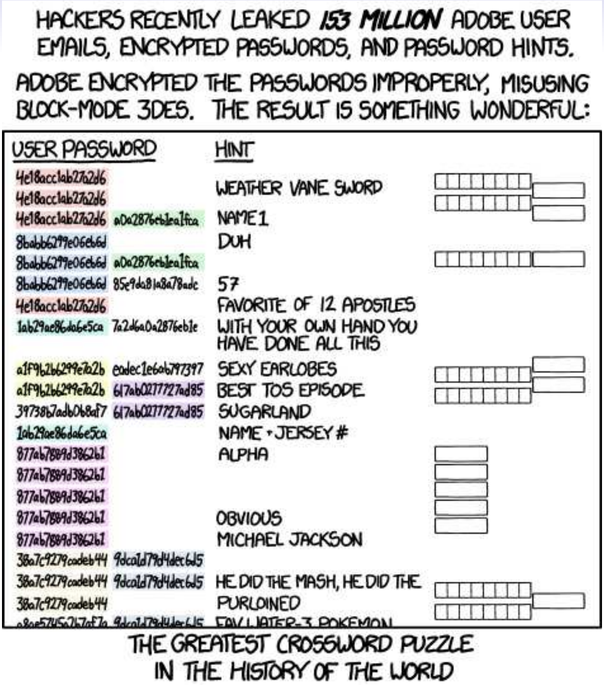
Rank	hotmail	#users	flirtlife	#users	computerbits	#users	rockyou	#users
1	123456	48	123456	1432	password	20	123456	290729
2	123456789	15	ficken	407	computerbits	10	12345	79076
3	111111	10	12345	365	123456	7	123456789	76789
4	12345678	9	hallo	348	dublin	6	password	59462
5	tequiero	8	123456789	258	letmein	5	iloveyou	49952
6	000000	7	schatz	230	qwerty	4	princess	33291
7	alejandro	7	12345678	223	ireland	4	1234567	21725
8	sebastian	6	daniel	185	1234567	3	rockyou	20901
9	estrella	6	1234	175	liverpool	3	12345678	20553
10	1234567	6	askim	171	munster	3	abc123	16648

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

Adobe

- Adobe encrypted data instead of hashing.
- 3DES ECB mode.
- Key unknown, but includes password hints.
- No hashing, lots of fun.

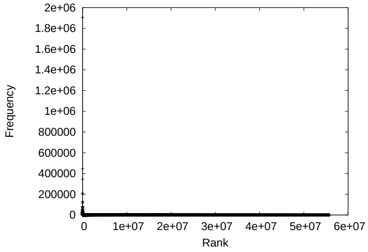
Rank	Cyphertext	indicative hint	inferred password	#users
1	EQ7fIpT7i/Q=	One to six in numeral form	123456	1905308
2	j9p+HwtWWT86aMjgZFLzYg==	1234567890 ohne 0	123456789	445971
3	L8qbAD3jl3jioxG6CatHBw==	answer is password	password	343956
4	BB4e6X+b2xLioxG6CatHBw==	adbeandonetwothree	adobe123	210932
5	j9p+HwtWWT/ioxG6CatHBw==	123456789 minus last number	12345678	201150
6	5djv7ZCI2ws=	1st 123456 letters	qwerty	130401
7	dQi0asWPYvQ=	1234567 is the password	1234567	124177
8	7LqYzKVeq8I=	6 number 1s	111111	113684
9	PMDTbP0LZxu03SwrFUvYGA==	adobe photo editing software	photoshop	83269
10	e6MPXQ5G6a8=	one two three one two three	123123	82606


HACKERS RECENTLY LEAKED 153 MILLION ADOBE USER EMAILS, ENCRYPTED PASSWORDS, AND PASSWORD HINTS.
ADOBE ENCRYPTED THE PASSWORDS IMPROPERLY, MISUSING BLOCK-MODE 3DES. THE RESULT IS SOMETHING WONDERFUL:
USER PASSWORD
HINT
4e18acc1ab27a2d6
4e18acc1ab27a2d6
WEATHER VANE SWORD
4e18acc1ab27a2d6 a0a2876eb1ea1fca
NAME1
8babb6299e06eb6d
DUH
8babb6299e06eb6d a0a2876eb1ea1fca
8babb6299e06eb6d 85e9da81a8a78adc
57
4e18acc1ab27a2d6
FAVORITE OF 12 APOSTLES
1ab29ae86da6e5ca 7a2d6a0a2876eb1e
WITH YOUR OWN HAND YOU HAVE DONE ALL THIS
a1f9b2b6299e7a2b eadec1e6ab797397
SEXY EARLOBES
a1f9b2b6299e7a2b 617ab0277727ad85
BEST TOS EPISODE
39738b7adb0b8af7 617ab0277727ad85
SUGARLAND
1ab29ae86da6e5ca
NAME + JERSEY #
877ab7889d3862b1
ALPHA
877ab7889d3862b1
877ab7889d3862b1
877ab7889d3862b1
OBVIOUS
877ab7889d3862b1
MICHAEL JACKSON
38a7c9279cadeb44 9dca1d79d4dec6d5
38a7c9279cadeb44 9dca1d79d4dec6d5
HE DID THE MASH, HE DID THE
38a7c9279cadeb44
PURLOINED
THE GREATEST CROSSWORD PUZZLE IN THE HISTORY OF THE WORLD

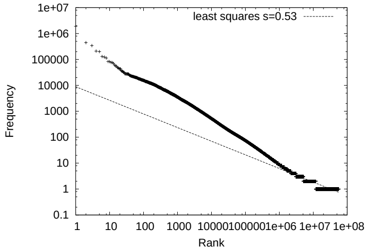
Questions for Data

- 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?

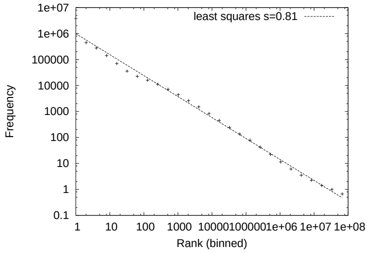
Distribution?



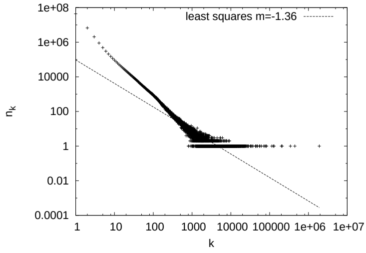
f vs. rank



$\log f$ vs. $\log$ rank

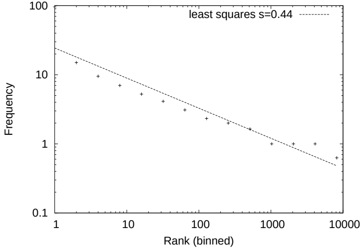


binned $\log f$ vs. $\log$ rank

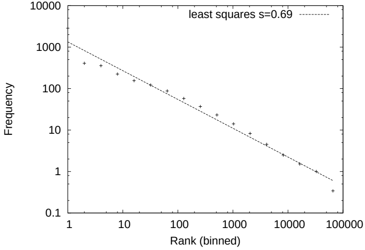


$\log n_k$ vs. $\log k$

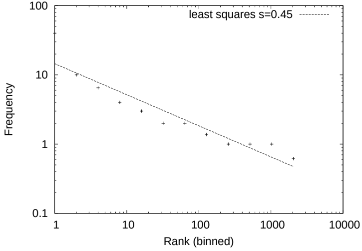
Others?



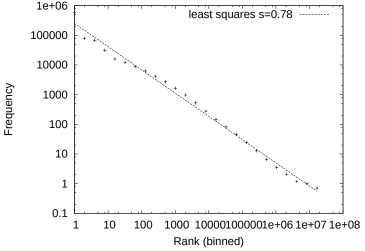
hotmail



flirtlife



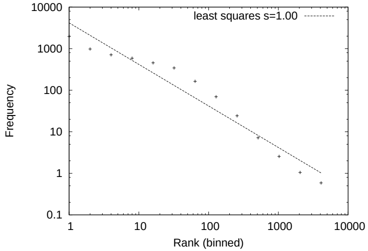
computerbits



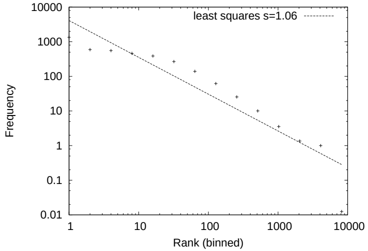
rockyou

Log-log Plots

Is everything a straight line on a log-log plot?



boys



girls

Zipf?

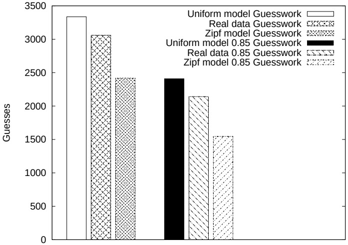
- A straight line on a log-log plot points towards heavy tail.
- Zipf?

$$P_r \propto \frac{1}{r^s}$$

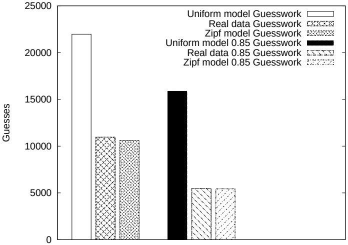
- Slope gives s.
- Can also do MLE and check p-values (Clauset '09).
- s is small, less than 1.

	hotmail	flirtlife	c-bits	rockyou	adobe
s MLE	0.246	0.695	0.23	0.7878	0.7927
s stderr	0.009	0.001	0.02	< 0.0001	< 0.0001
p-value	< 0.01	0.57	< 0.01	< 0.01	< 0.01

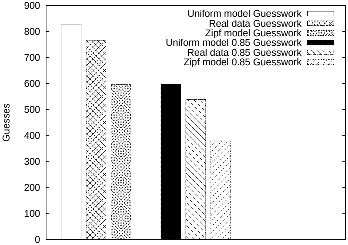
Guesswork Predictions



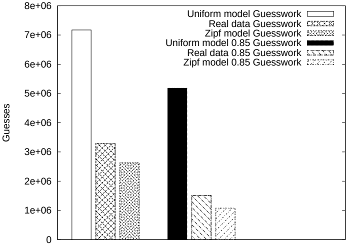
hotmail



flirtlife



computerbits



rockyou

Who cares?

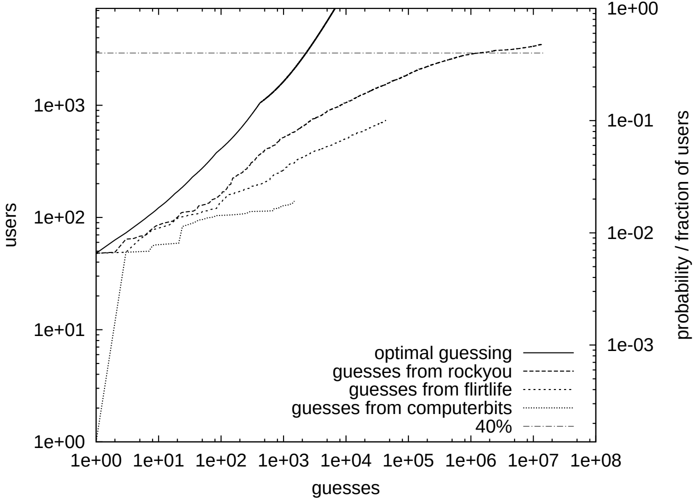
1. Algorithm Design — exploit heavy tail?
2. Can we get close to optimal dictionary attack?
3. Can we make dictionary attack less effective?

2 and 3 are questions about common behavior and helping users.

Can use the clear text data to study.

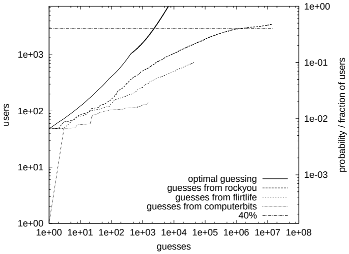
Dictionary Attack

Suppose we use one dataset as a dictionary to attack another.

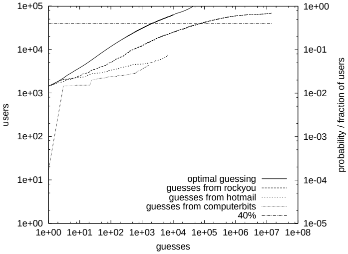


hotmail

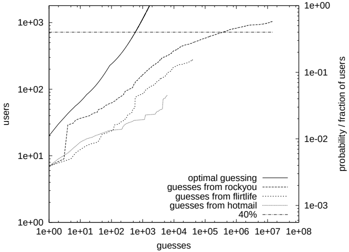
Dictionary Attack — Same Story



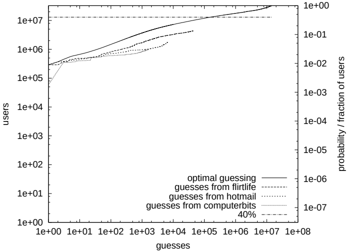
hotmail



flirtlife



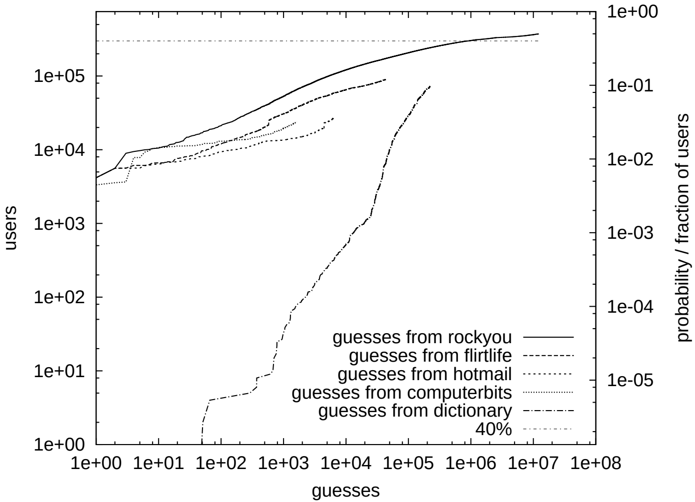
computerbits



rockyou

Dictionary Attack Gawker

December 2010, Gawker, 748090 Unix `crypt` Hashes, well salted.



Dell'Amico'10 review smart generators. This looks ×10!

Helping Users

User is biased 'random' password generator.
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?

In WWW'12 paper, looked at Metropolis-Hastings algorithm.
We'll look at rejection sampling here.

Helping with Rejection

If we know P_i, then we could use rejection sampling.

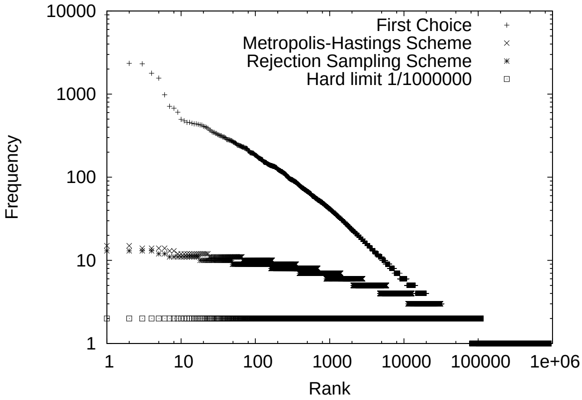
1. Have a probability r_i for each password that it is chosen.
2. Ask user for a new password x'.
3. With probability $r_{x'}$ accept the password, otherwise go to 2.

If users choices are i.i.d with distribution P_i:

- Resulting distribution is proportional to $q_i = r_i P_i$.
- $r_i = C/P_i$ results in a uniform distribution.
- Take $C = P_N$ to give least number of rejections.

How does this do?

- Generate 1000000 users.
- Rockyou-based i.i.d password choices.
- Learns $F(x)$ as it goes, uses $1/F(x)$.



How annoying? Mean tries 1.29, variance 0.63.

Limiting Mean Rejections

Suppose we want to make guesswork as large as possible, but bound $\mathbb{E}[R] \leq L$.
Note, if $a = \sum q_i$, then $\mathbb{E}[R] = 1/a - 1$.

Optimisation Problem: Maximise

$$G = \sum i \frac{q_i}{a},$$

given

$$\mathbb{E}[R] \leq L,$$

where

$$q_i = r_i P_i, 0 \leq r_i \leq 1, q_i \geq q_{i+1}.$$

Solution: Guessed to level high probability ones. Need to write it up!

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?
- Banned lists are not optimal[3].
- Current: Evaluating password advice.
- Future: Generalise beyond web passwords?
- Future: Field test Metropolis-Hastings or Rejection?

[3] Also not even unimodal!

Metropolis-Hastings for Uniform Passwords

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.

Rejection Sampling: Sketch Proof

1. Show $q_i \geq q_{i+1}$ doesn't make things worse.
2. Show the r_i^* have to be positive.
3. Show that $r_m^* = 1$ then $r_n^* = 1$ for $n \geq m$.
4. Show that if $r_m^* < 1$ then $q_1^* = q_2^* = \ldots q_m^*$.

Shows that if you can afford to, make it uniform.
Otherwise clip the probability of most likely passwords.