L5:
Basic Grammar Based Probabilistic Password Cracking

Sudhir Aggarwal and Shiva Houshmand
and Matt Weir
Florida State University
Department of Computer Science
E-Crime Investigative Technologies Lab
Tallahassee, Florida 32306

August 5-7, 2015

Password Cracking
University of Jyväskylä
Summer School August 2015

Our Research

- Assist Law Enforcement and Security Agencies
- Develop better ways to model how people actually create passwords
- Develop better ways to crack passwords
- Incorporate targeted attack features
- Improve attack dictionaries
- Continuously extend capabilities with new techniques
- Investigate how we can build better passwords
- Applications of our approach

Cracking Passwords

- Given a password hash or file of hashes, guess a password, compute the hash, and check against the given hashes
- There are many password hashes used: MD5, Sha1, multiple hashings such as done by TrueCrypt, etc. These last are done to increase the time to compute the hash
- Our focus is on the guessing part. Given a hash algorithm, we can always use the best implementation if possible – we have not focused on collecting a set of best implementations

Two Types of Password Cracking of Cracking of Interest

- Online
  - The system is still operational and you are allowed only a few guesses
- Offline
  - You grabbed the password hash(s) and want to crack as many as possible within a reasonable amount of time available
- Our interests
  - Would like to be good at both, but we focus on the offline case

Cracking Passwords

Generate a password guess

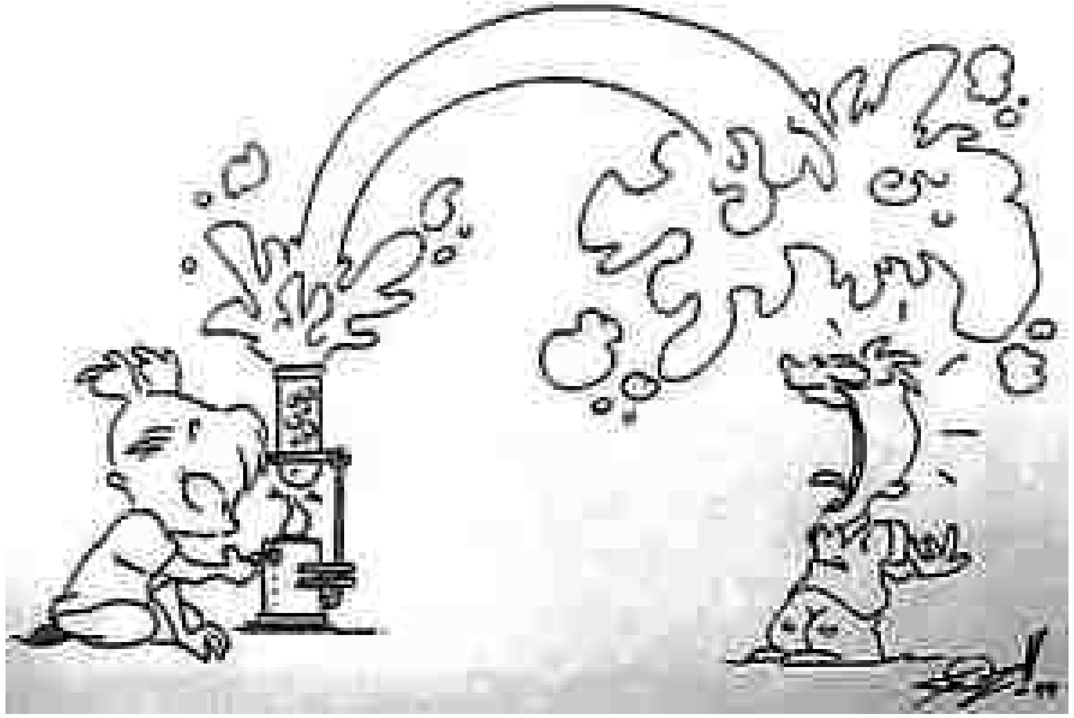
- password123

Hash the guess MD5 (128 bits), Sha1, etc.

- A5732067234F23B21

Compare the hash to the password hash you are trying to crack

Password crackers systems are proliferating



- Access Data's PRTK (commercial)
- John the Ripper (open source)
- Hashcat (open source)
- Cain & Able (old)
- L0phtcrack (old)
- Specifically for Microsoft passwords

Types

- Micro Rules
- Markov approaches
- Probabilistic Context-free grammars

Example: John the Ripper

- Open source free password cracking system
  - Runs on many different platforms
  - Runs against many different hash types
  - Can run in a number of modes
    - Single crack mode, wordlist mode, incremental mode
    - Incremental mode is the most powerful
- Most popular cracking system and the best to test against
  - Basic approach is *mangling rules* and dictionaries
  - Brute force and some Markov modeling
  - Used by law enforcement

Focus of Our Research

- Our research in this area has focused on how to make better password guesses
  - Hash neutral. Aka you would create the same guesses regardless if you are attacking a Truecrypt or a WinRAR encrypted file
- We have also explored implementing faster hashing algorithms using GPUs. This can be explored further.
  - Target program specific. Aka the hashing that Truecrypt and WinRAR uses is different
  - Prefer to use existing systems to actual compute hashes

Dictionary Based Attacks

- Password-cracking dictionaries may contain entries that are not natural language words, e.g., ‘qwerty’
- No consensus on how to use dictionaries
- Usual dictionary based attacks derive multiple password guesses from a single dictionary entry by application of fixed rules, such as ‘replace a with @’ or ‘add any two digits to the end’
  - Often could get stuck in certain types of rule such as add 6 digits to the end
- Dictionaries sometimes contain actual passwords rather than potential words that can be modified

The Original Plan

1. Try to obtain some Data-sets
2. Explore using Probabilistic Password Cracking
3. Better guess generation
4. Focus on Pass-Phrase Cracking

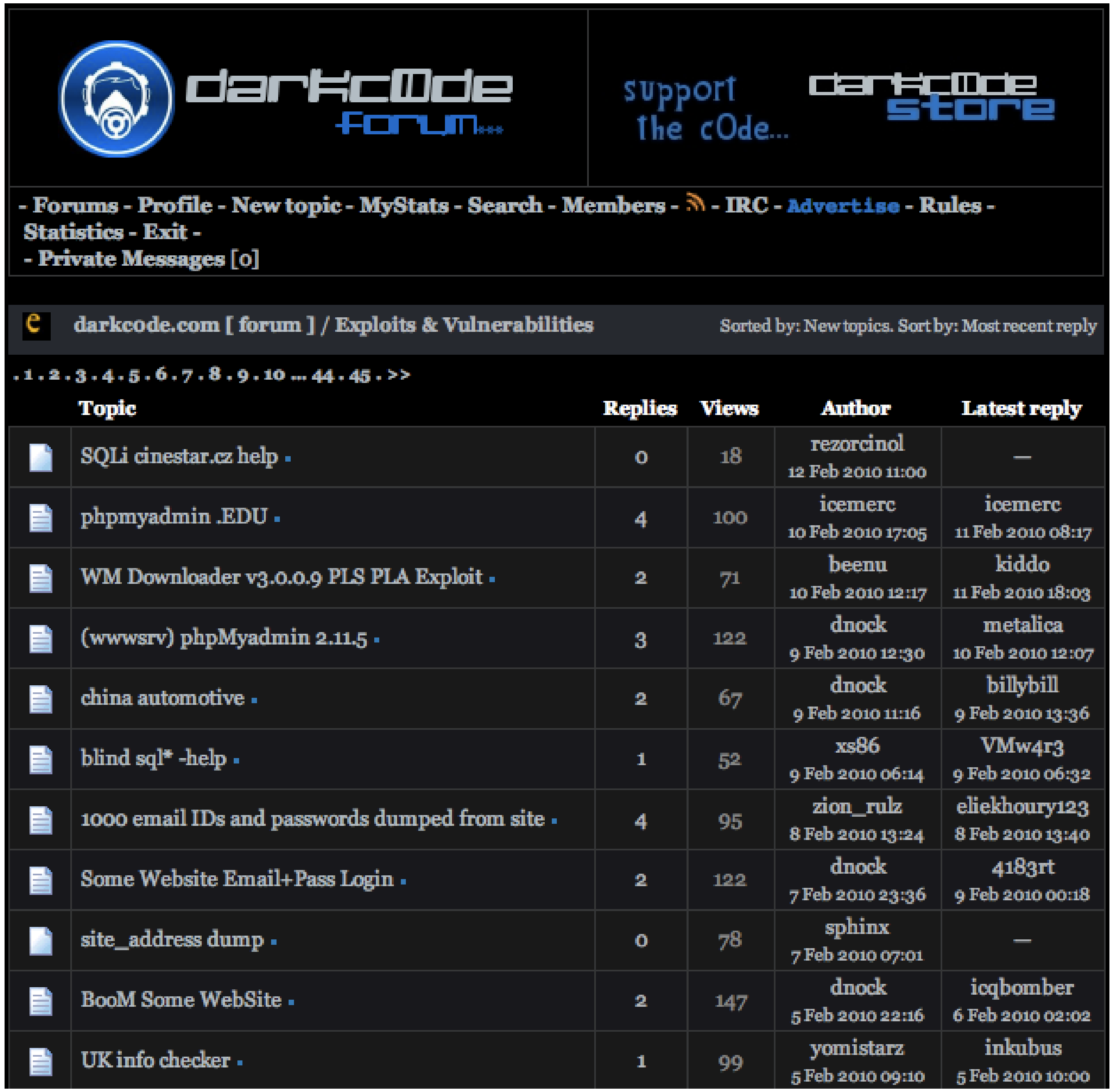
Obtaining Real Passwords

Originally we were concerned that one of the main problems with our research would be collecting valid data-sets to train/test against

Obtaining the Datasets

In reality, that hasn't been much of a problem for web-based passwords

Hacker Like to brag in Forums:



darkc0de forum...

support the c0de... darkc0de store

- Forums - Profile - New topic - MyStats - Search - Members - - IRC - Advertise - Rules - Statistics - Exit -
- Private Messages [0]

darkcode.com [forum] / Exploits & Vulnerabilities

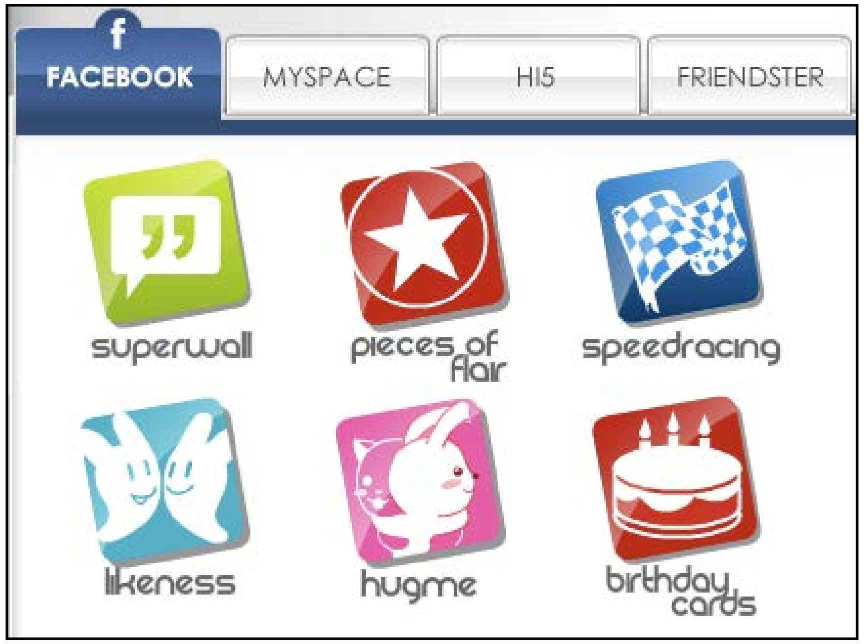
Sorted by: New topics. Sort by: Most recent reply

. 1 . 2 . 3 . 4 . 5 . 6 . 7 . 8 . 9 . 10 ... 44 . 45 . >>

Topic	Replies	Views	Author	Latest reply
SQLi cinestar.cz help	0	18	rezorcinol 12 Feb 2010 11:00	–
phpmyadmin .EDU	4	100	icemerc 10 Feb 2010 17:05	icemerc 11 Feb 2010 08:17
WM Downloader v3.0.0.9 PLS PLA Exploit	2	71	beenu 10 Feb 2010 12:17	kiddo 11 Feb 2010 18:03
(wwwsrv) phpMyadmin 2.11.5	3	122	dnock 9 Feb 2010 12:30	metalica 10 Feb 2010 12:07
china automotive	2	67	dnock 9 Feb 2010 11:16	billybill 9 Feb 2010 13:36
blind sql* -help	1	52	xs86 9 Feb 2010 06:14	VMw4r3 9 Feb 2010 06:32
1000 email IDs and passwords dumped from site	4	95	zion_rulz 8 Feb 2010 13:24	eliekhoury123 8 Feb 2010 13:40
Some Website Email+Pass Login	2	122	dnock 7 Feb 2010 23:36	4183rt 9 Feb 2010 00:18
site_address dump	0	78	sphinx 7 Feb 2010 07:01	–
BooM Some WebSite	2	147	dnock 5 Feb 2010 22:16	icqbomber 6 Feb 2010 02:02
UK info checker	1	99	yomistarz 5 Feb 2010 09:10	inkubus 5 Feb 2010 10:00

Note: The site darkc0de.com is no longer operational as it was hacked itself back in July 2010 by a group of Albanian hackers

Some of ours Lists



- LinkedIn (2012) – 6.4 million Sha1 hashes
- Yahoo (2012) – 453 K plaintext passwords
- RockYou (2009) - 32 million plaintext passwords
- MySpace – 62 K plaintext, 17 K MD5 hashes
- Etc, etc, etc.

The Soap Opera Around the Rockyou Hack

- The vulnerability originally was publicly posted on the website www.darkc0de.com
- It appears that multiple hackers used it to break into the site.
- According to the security firm Imperva, many of the webmail accounts associated with those passwords have been taken over by spammers

The Soap Opera (Continued)

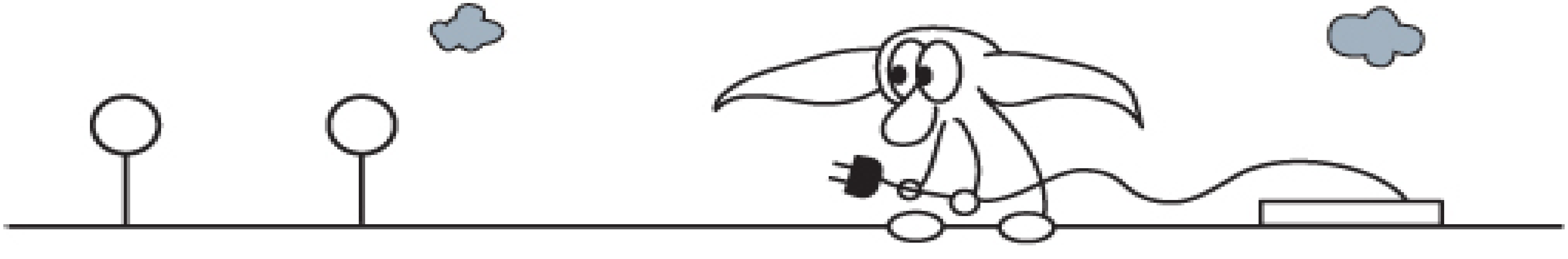
- One Slovakian hacker named Igigi claimed credit for the attack, and set up a blog detailing other website hacks
- He also started giving interviews to various news publications
- At one time he had a Facebook fan page with over 600 members...

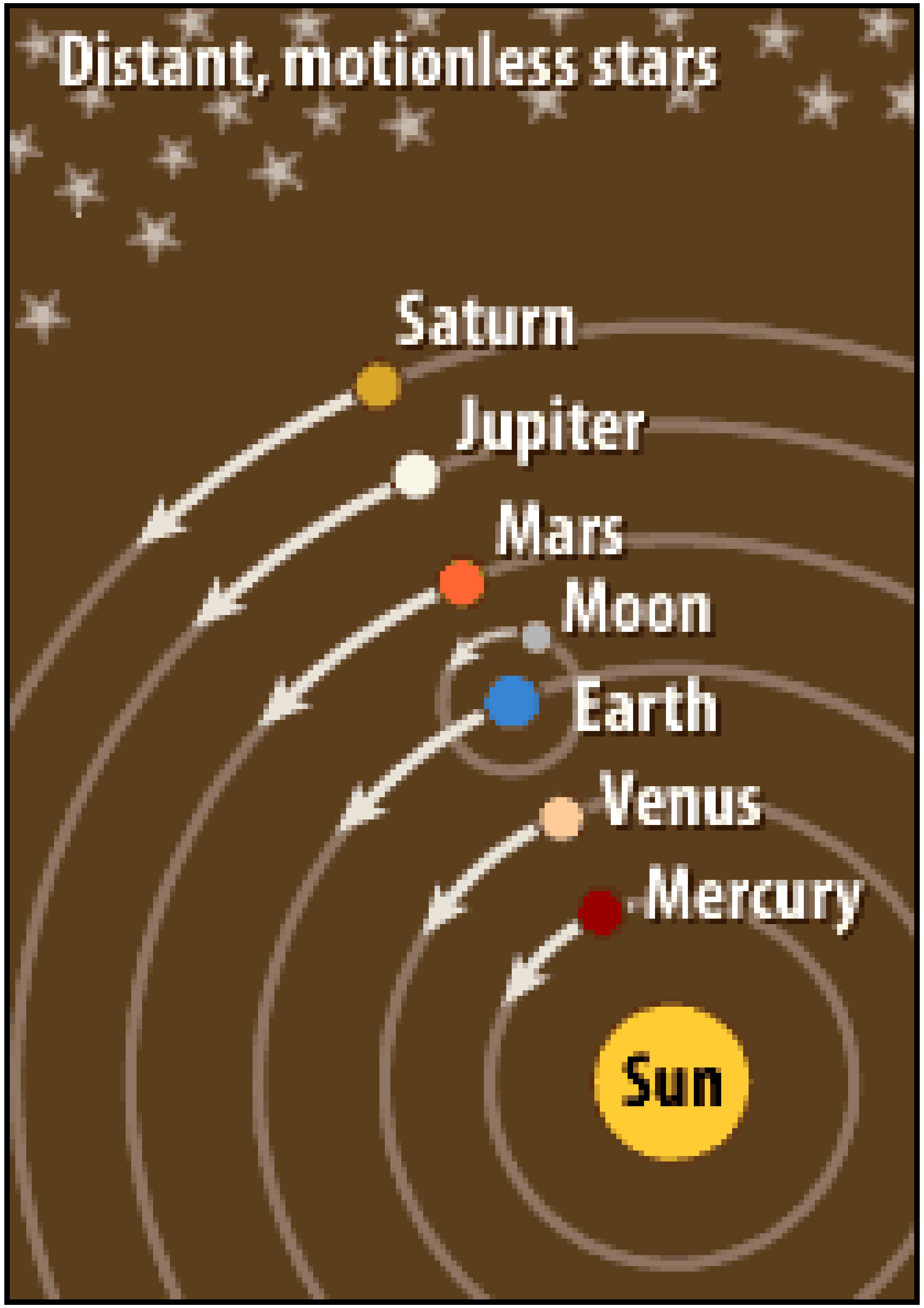
Our Idea

- Find the “correct order” in which to try the passwords
- Which should we try first?
- p@ssword1234
- password8732

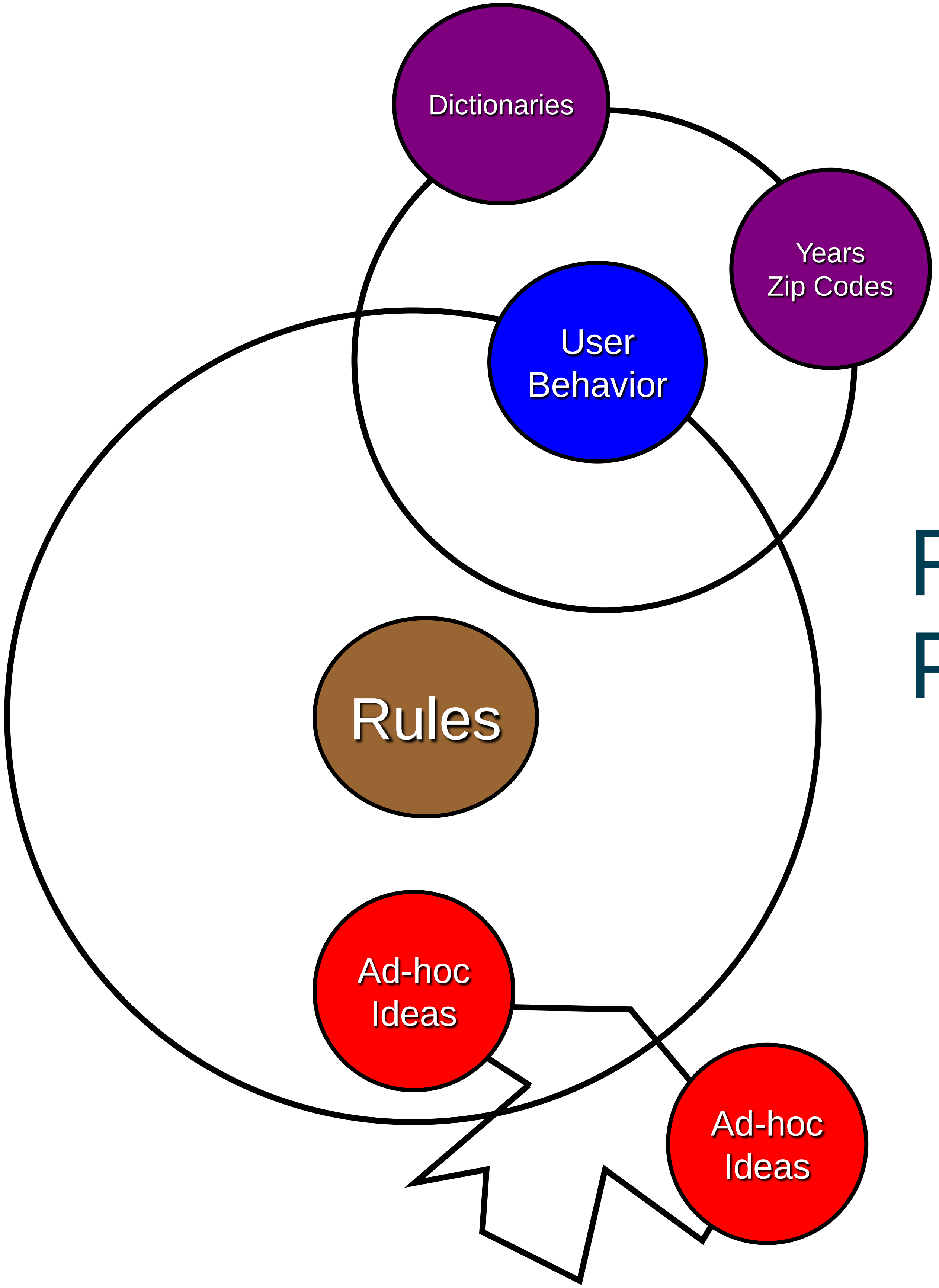
Probabilistic Cracking

- Some words are more likely than others
  - password, monkey, football
- Some mangling rules are more likely than others
  - 123, 007, $$$, Capitalize the first letter





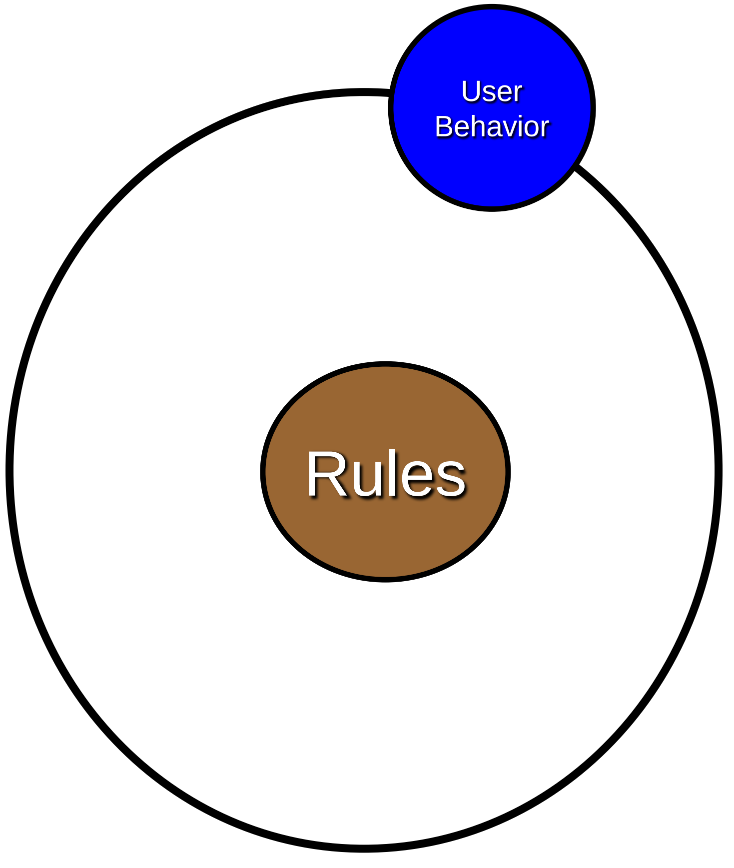
Probabilistic Password Cracking vs. Rule Based Cracking



Rule Centric View of Password Cracking

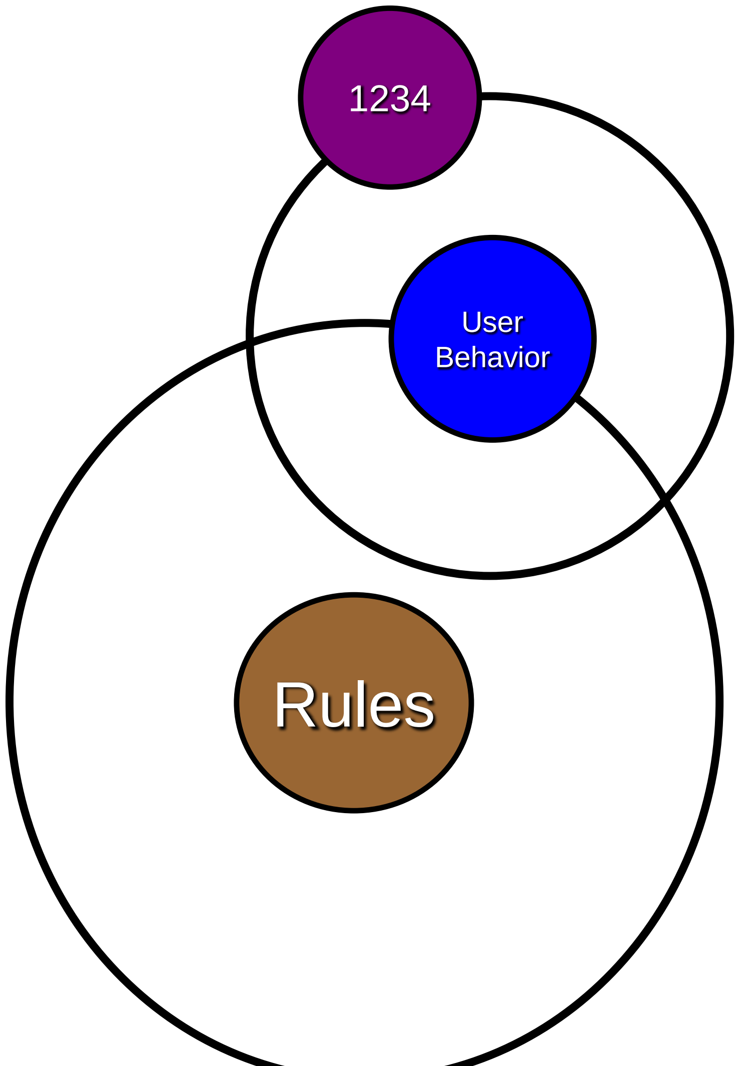
Rule Based Optimizations

1. Append 4 Digits



Rule Based Optimizations

1. Append 1234
2. Append 4 Digits

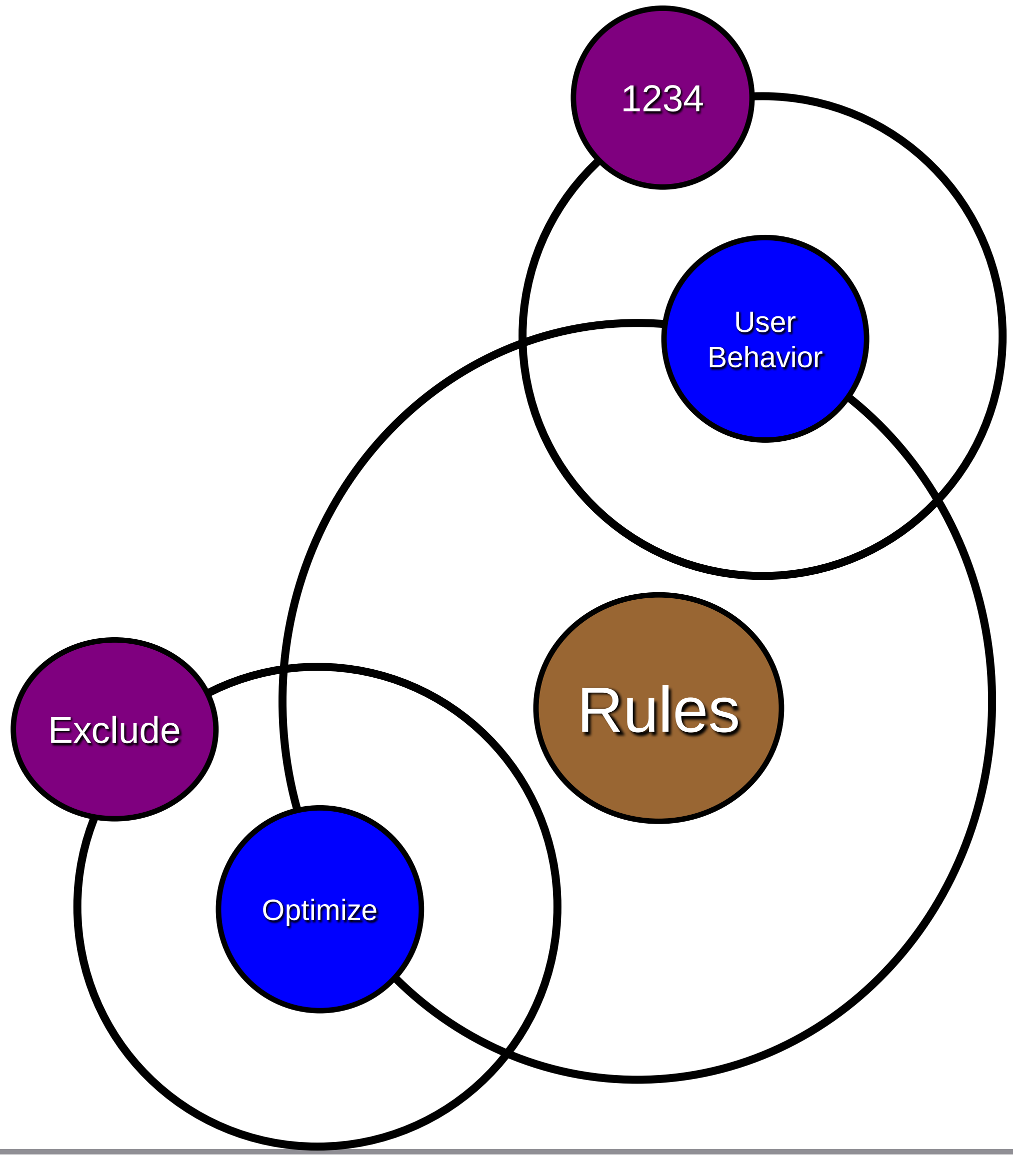


Rule Based Optimizations

1. Append 1234

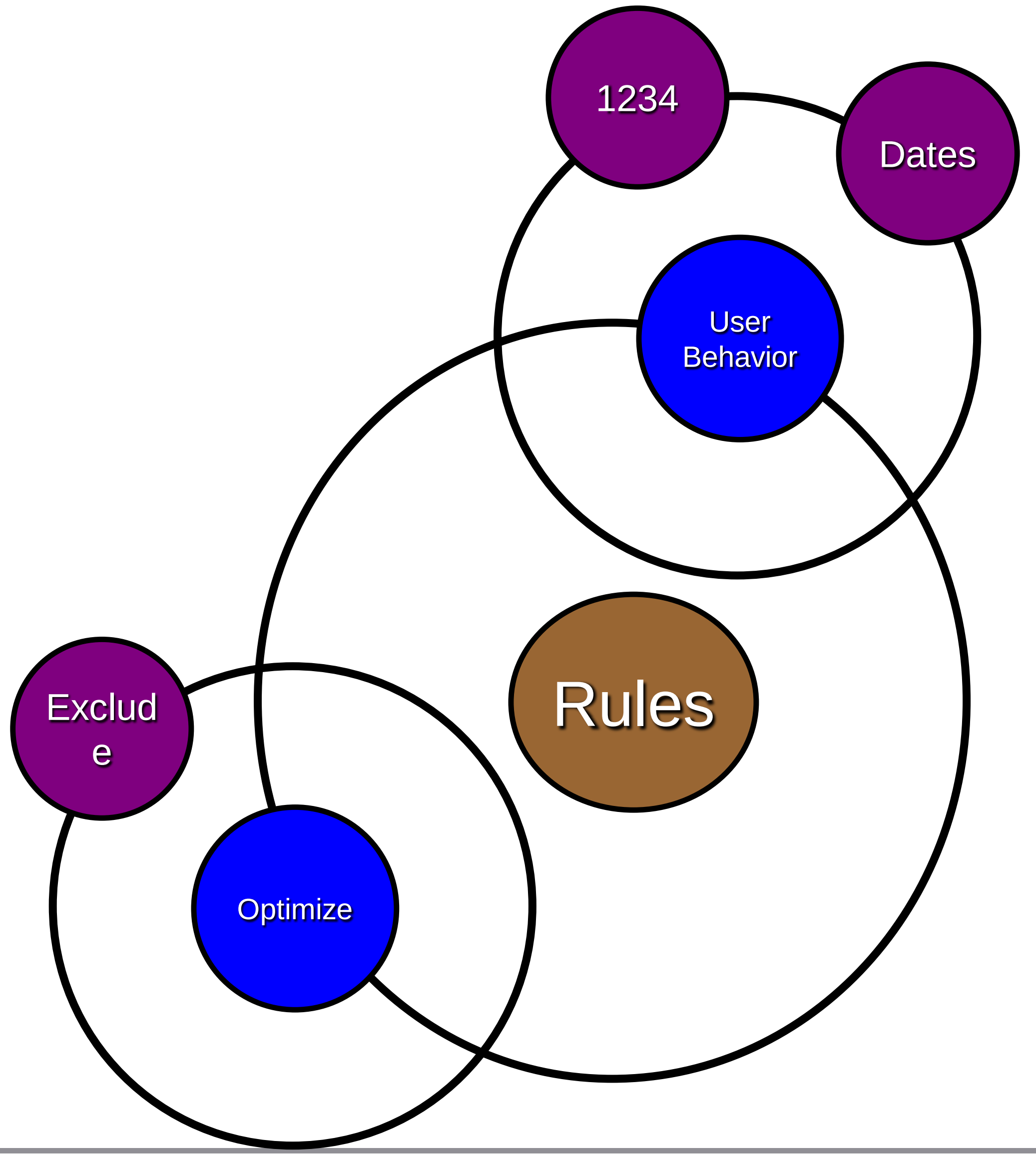
2. Append 0000-1233

3. Append 1235-9999



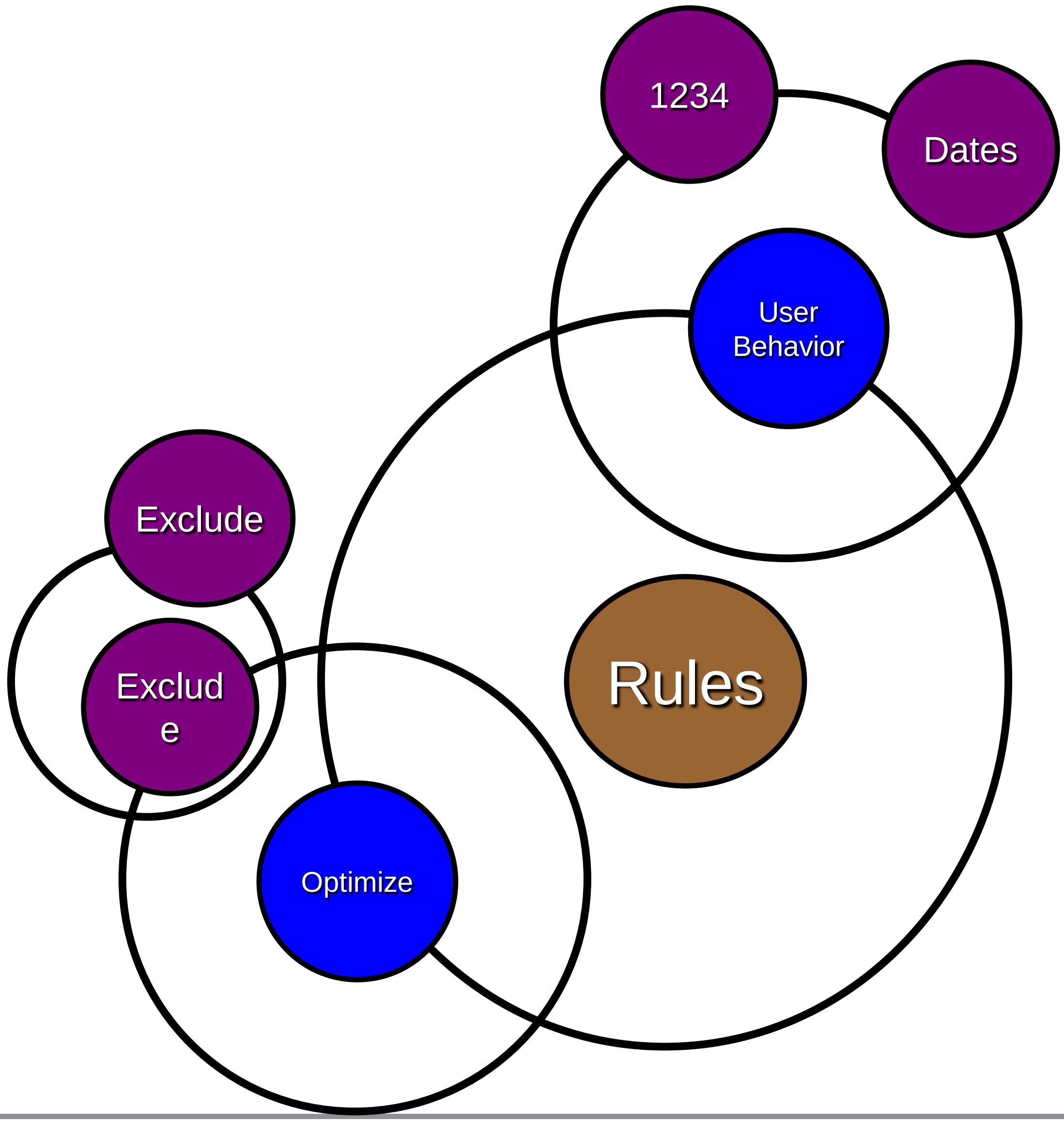
Rule Based Optimizations

1. Append 1234
2. Append 1950-2010
3. Append 0000-1233
4. Append 1235-9999



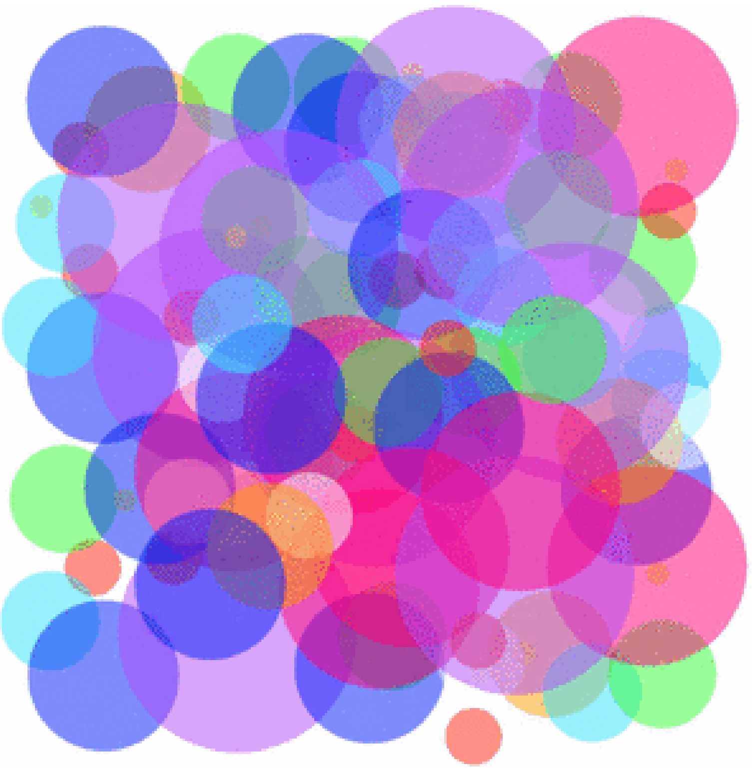
Rule Based Optimizations

1. Append 1234
2. Append 1950-2010
3. Append 0000-1233
4. Append 1235-1949
5. Append 2011-9999



John the Ripper's Rule Based Optimizations

1. Append 1234
2. Append 1950-2010
3. Append 0000-1233
4. Append 1235-1949
5. Append 2011-9999
6. Capitalize the first letter, Append 1234
7. Capitalize the first letter, Append 1950-2010
8. Capitalize the first letter, Append 0000-1233
9. Capitalize the first letter, Append 1235-1949
10. Capitalize the first letter, Append 2011-999
11. Replace 'a' with an '@', Append 1234
12. Replace 'a' with an '@', Append 1950-2010
13. Replace 'a' with an '@', Append 0000-1233
14. Replace 'a' with an '@', Append 1235-1949
15. Replace 'a' with an '@', Append 2011-9999
16. Uppercase the last letter, Append 1234
17. Uppercase the last letter, Append 1950-2010
18. Uppercase the last letter, Append 0000-1233
19. Uppercase the last letter, Uppercase the last letter, Append 1235-1949
20. Uppercase the last letter, Uppercase the last letter, Append 2011-9999



New Idea: Probabilities should be the focus

- Would like to try password guesses in highest probability order!
- Use the revealed password sets to determine the probabilities of different guesses
- We actually derive a grammar by training on the revealed data sets
- The grammar approach can be compared to the word mangling rules that previous approaches used
- Generate passwords in highest probability order

PCFG Approach

- **Training**: use revealed passwords sets to create a context-free grammar that gives structure to the passwords. The grammar rules derive strings (passwords) with probabilities based on the specific derivation
- **Cracking**: how can one derive the passwords in highest probability order based on the grammar
- **Patterns**: what are the patterns that can be effectively used?

Two Stages

- Training
  - Construct the grammar
- Cracking
  - Use the grammar to create password guesses

Information in the Datasets

Very little available except revealed passwords and revealed hashes

Information not available: how do individuals change passwords, how do they store them if they are difficult to remember, etc.

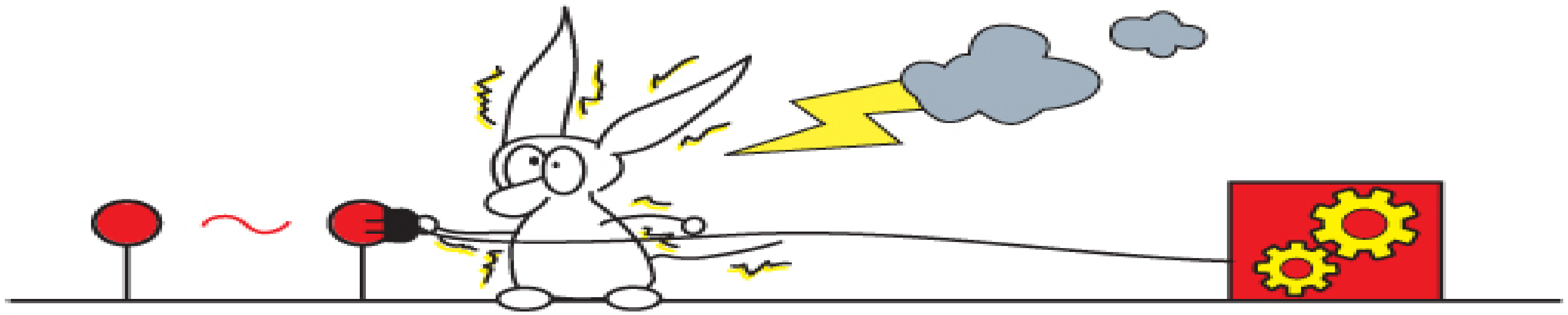
Training our Cracker

- Our password cracker is trained on known password lists

- We can use one or a set of appropriate training lists

- We train if possible on passwords similar to the target profiles

- What do we learn through the training? We actually learn a probabilistic context free grammar!



Password Structures

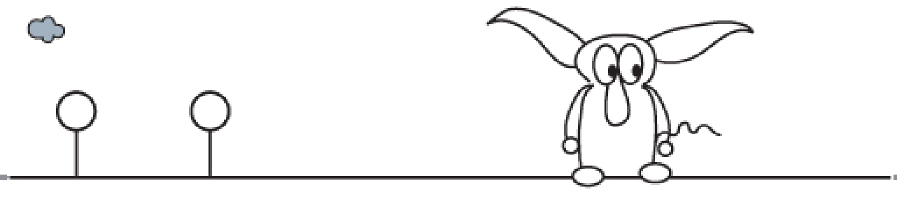
- Possibly, the most naive structure that can be inferred from passwords is the sequence of the character classes used
  - Letters = L
  - Digits = D
  - Symbols = S
- password12! --> LDS the "simple structure"

The Context-Free Assumption

- Context-free grammars lead to efficient algorithms, but simple structures are “too lossy” to allow for capturing sufficiently fine-grained human behavior in password choice in a context-free way
- “97” as a password element (a date) is more likely than would be expected by the independent probabilities of ‘9’ and ‘7’
- Some password lengths are preferred

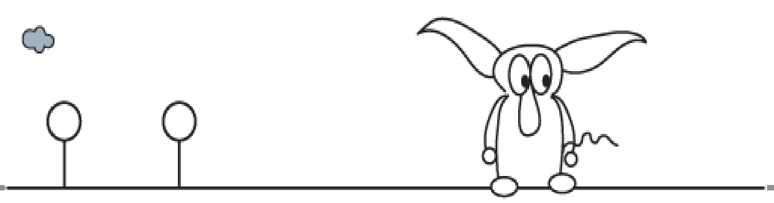
Learning the “Base structures”

- Extend the character class symbols to include length information
  - password12 = $L_8S_1D_2S_1$
  - Calculate the probabilities of all the base structures
- Base structures, while still very simple, empirically capture sufficient information to derive useful context-free grammar models from password datasets



Learning the Grammar (continued)

- The next step is to learn the probabilities of digits and special characters
- We record the probabilities of different length strings independently
- Picks up rules such as 007, 1234, !!, $$, !@#$
- We learn about capitalization
- We can also can learn about Keyboard combination and the L structures



Capitalization

Case Mask	Percentage of Total
N_6	93.206%
U_1N_5	3.1727%
U_6	2.9225%
N_3U_3	0.1053%
$U_1N_4U_1$	0.0078%

Probabilities of Top 5 Case Masks for Six Character Words

Assigning Probability to Dictionary Words

- By default we just assign a probability to each dictionary word of $1/n_k$
- n_k is the number of dictionary words of length k
- However, we can use multiple dictionaries with different assigned probabilities to model different probabilities of words

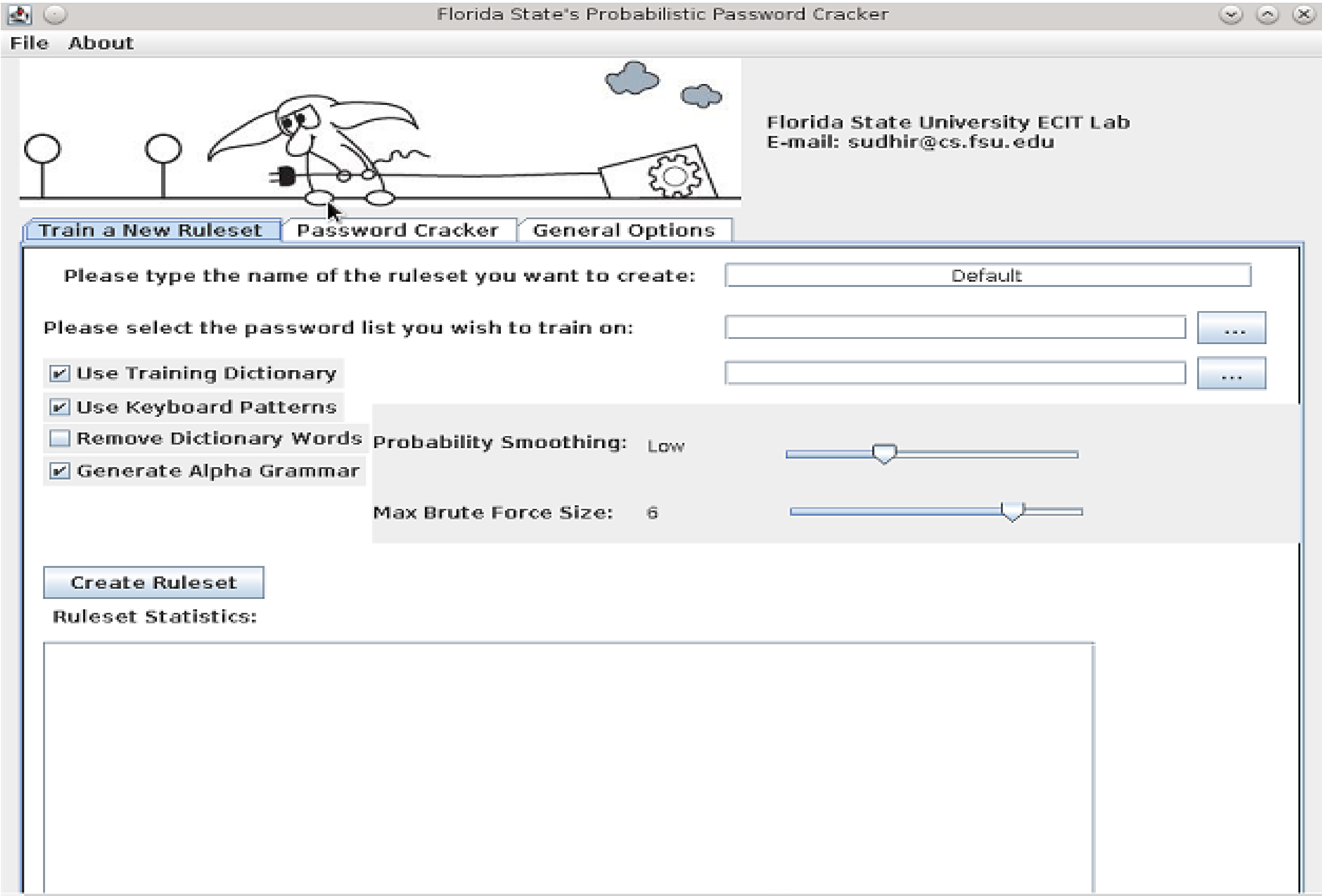
A Simple Example of the Learned Probabilistic Context-free Grammar

- Derive the production rules from the training set

- Derive the probabilities from the training set

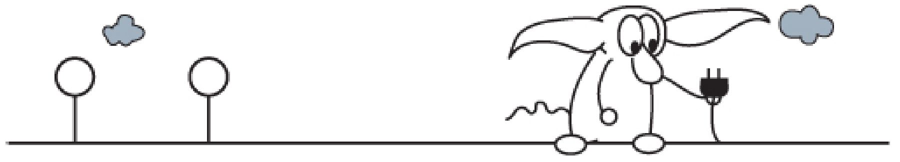
$S \rightarrow$	L_4D_2	.50
$S \rightarrow$	$D_1L_3D_1$	.25
$S \rightarrow$	$L_4D_1S_1$	.25
$D_2 \rightarrow$	99	.50
$D_2 \rightarrow$	98	.30
$D_2 \rightarrow$	11	.20
$D_1 \rightarrow$	1	.80
$D_1 \rightarrow$	2	.20
$S_1 \rightarrow$	!	1.0
$L_4 \rightarrow$	pass	.10
$S \rightarrow^*$ *pass11* with probability .5 x .1 x .2 = .01		

Training Demo



Now to the Cracking

- After training, the grammar can be distributed for purposes of password cracking (e.g., base structures can be distributed and the replacement tokens also)
- Size of grammar when trained on the MySpace set of 33,481 passwords
  - 1,589 base structures (with probabilities)
  - 4,410 digit components (with probabilities)
  - 144 symbol components (with probabilities)



Requirements for the Next Function

- Generate all possible guesses with no duplicates
- Generate the guesses in probability order
- Reasonable memory requirements
- Comparable time requirements to existing methods
- Able to support distributed password cracking

Pre-Terminal Structures

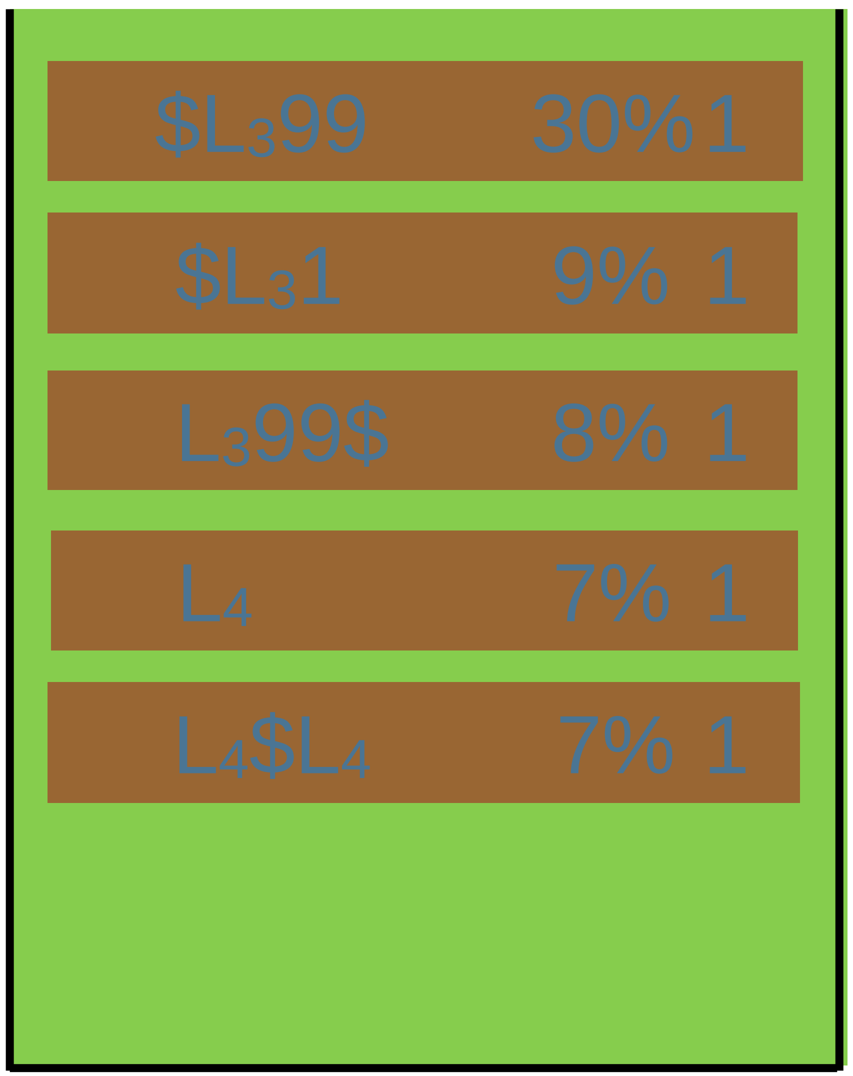
- Essentially the base structure with all the productions except for the dictionary words replaced with terminals

$$S_1 L_3 D_2$$

$$\rightarrow \$L_3 99$$

D_2	D_2 Prob.	S_1	S_1 Prob.
99	50%	$	60%
12	30%	%	40%
33	20%		

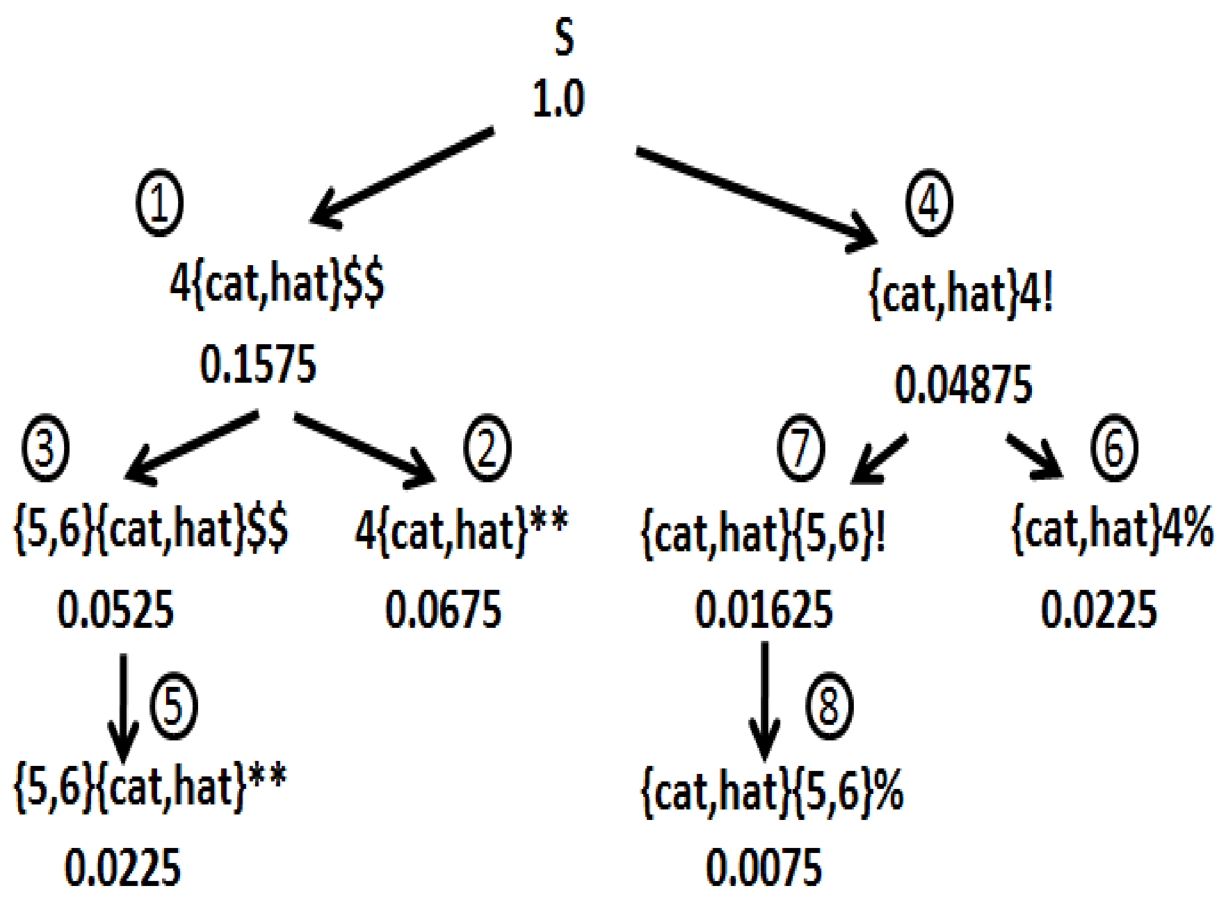
Generating Guesses



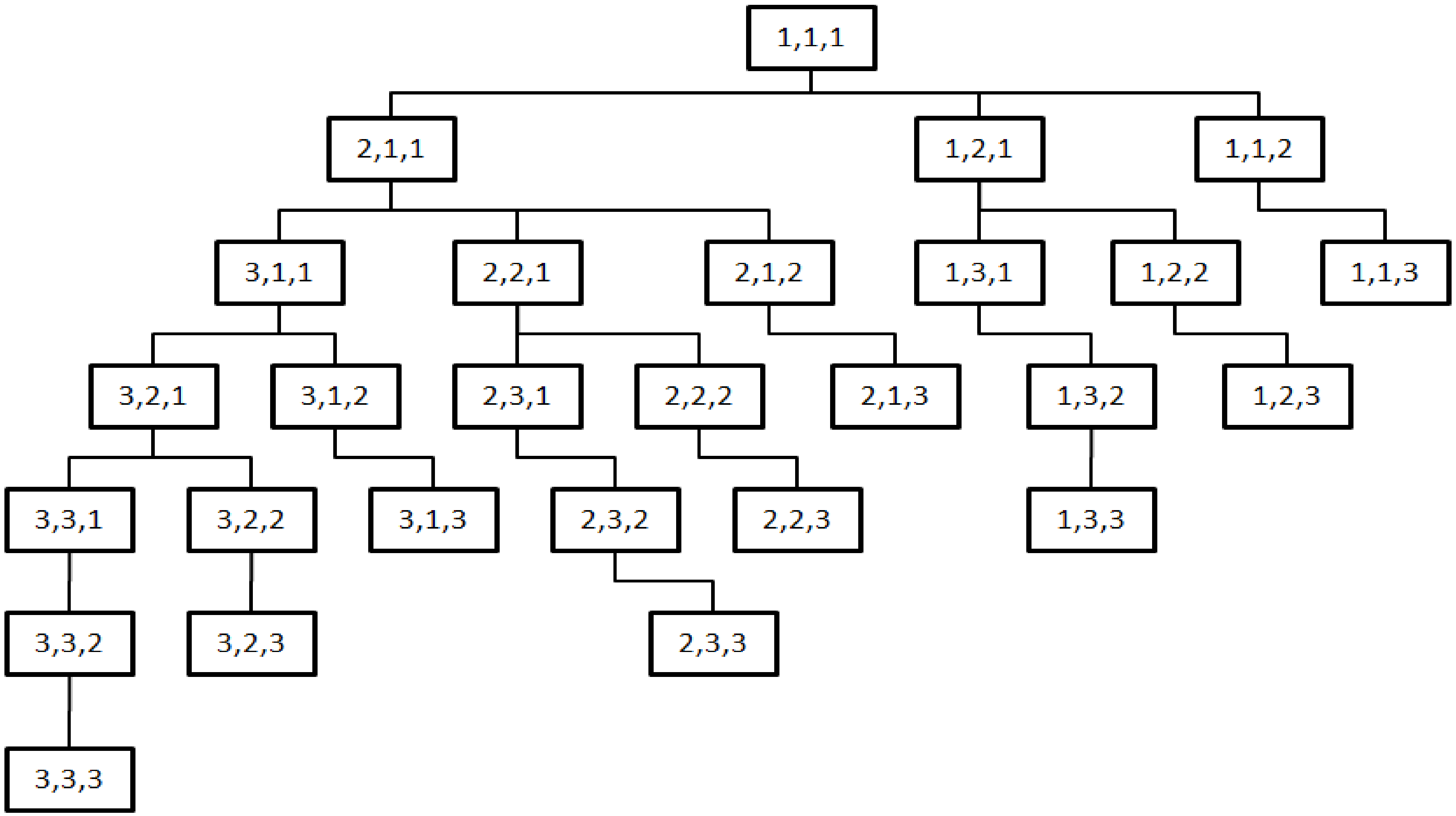
- Pop the top value (30%) and check the guesses: \$dog99, \$cat99, etc.
- Create children of the popped value: $\$L_{3}12$ (18%) and $\%L_{3}99$ (20%) and push them into the p-queue
- Pop the next top value
- Continue until queue is empty

The Pivot Next Function

- We needed an efficient next function algorithms to generate guesses in probabilistic order. Our first function was called a pivot function. Basically we limited which node would create children

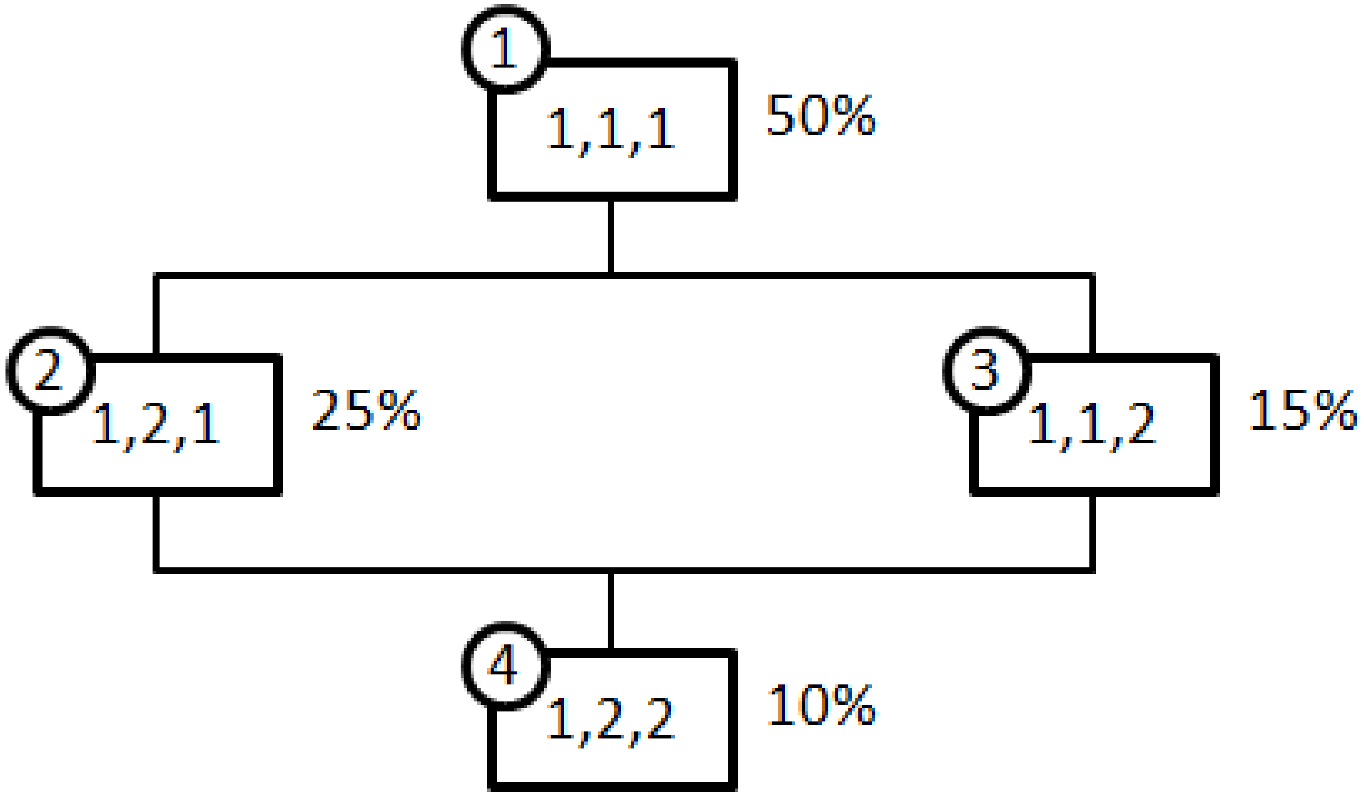


Example Tree for Generating Guesses



We actually have a much better algorithm that we have implemented and use: dead-beat dad

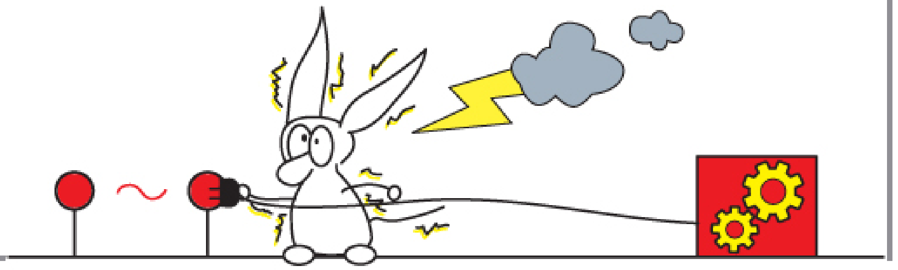
Better Algorithm: Deadbeat Dad



When node 1 is popped nodes 2,3 pushed in the original pivot algorithm (the children of 1). When 2 is next popped, its child node 4 is pushed. But in the deadbeat dad algorithm, 4 is not pushed since 2 knows there is another dad 3 responsible for 4 and will let 3 push 4 when 3 is popped.

Size of Potential Search Space

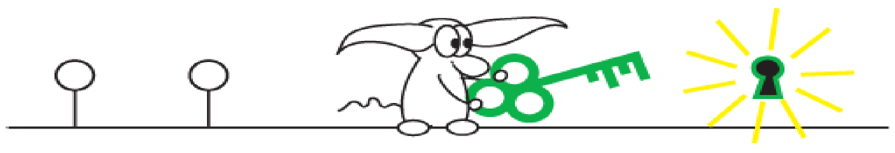
Structure	Number of Structure in the MySpace Training Set
Base	1,589
Pre-Terminal	34 trillion



Generating guesses: we use a priority queue

$\$L_3 99$	30%	1
$\$L_5 1$	9%	1
$L_3 99\$$	8%	1
L_4	7%	1
$L_4 \$ L_4$	7%	1

- Pop the top value (30%) and check the guesses: $dog99, $cat99, etc.
- Create children of the popped value: $\$L_3 12$ (18%) and $\%L_3 99$ (20%) and push them into the p-queue
- Pop the next top value
- Continue until queue is empty

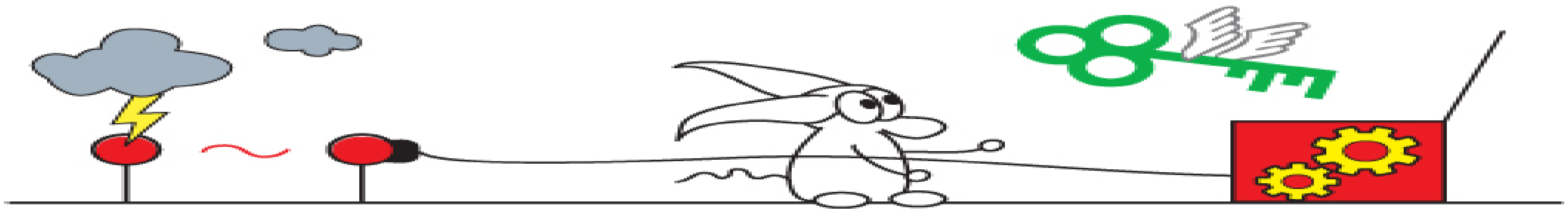


Smoothing – using the Laplacian

- Training set may not have all possible values of some type of set, say D_3, with the value 732.
- Probability smoothing allows all non-used values to have some probability of being chosen based on the smoothing parameters.
- Consider values in K different categories (1000) in the above example. Let N_i be the number in category i with $N = \sum N_i$. Smoothing parameter $0 \leq$ **$\alpha \leq 1$.**
- **Prob (i) = $(N_i + \alpha) / (N + K * \alpha)$**

Algorithm optimization – Using Containers

- If many items have the same values (say a bunch of smoothed values) we can aggregate them into containers.
- In fact, each pre-terminal that we discussed previously is actually a “container” with many values having that exact probability.
- This permits many guesses to be tried without stressing the priority queue.



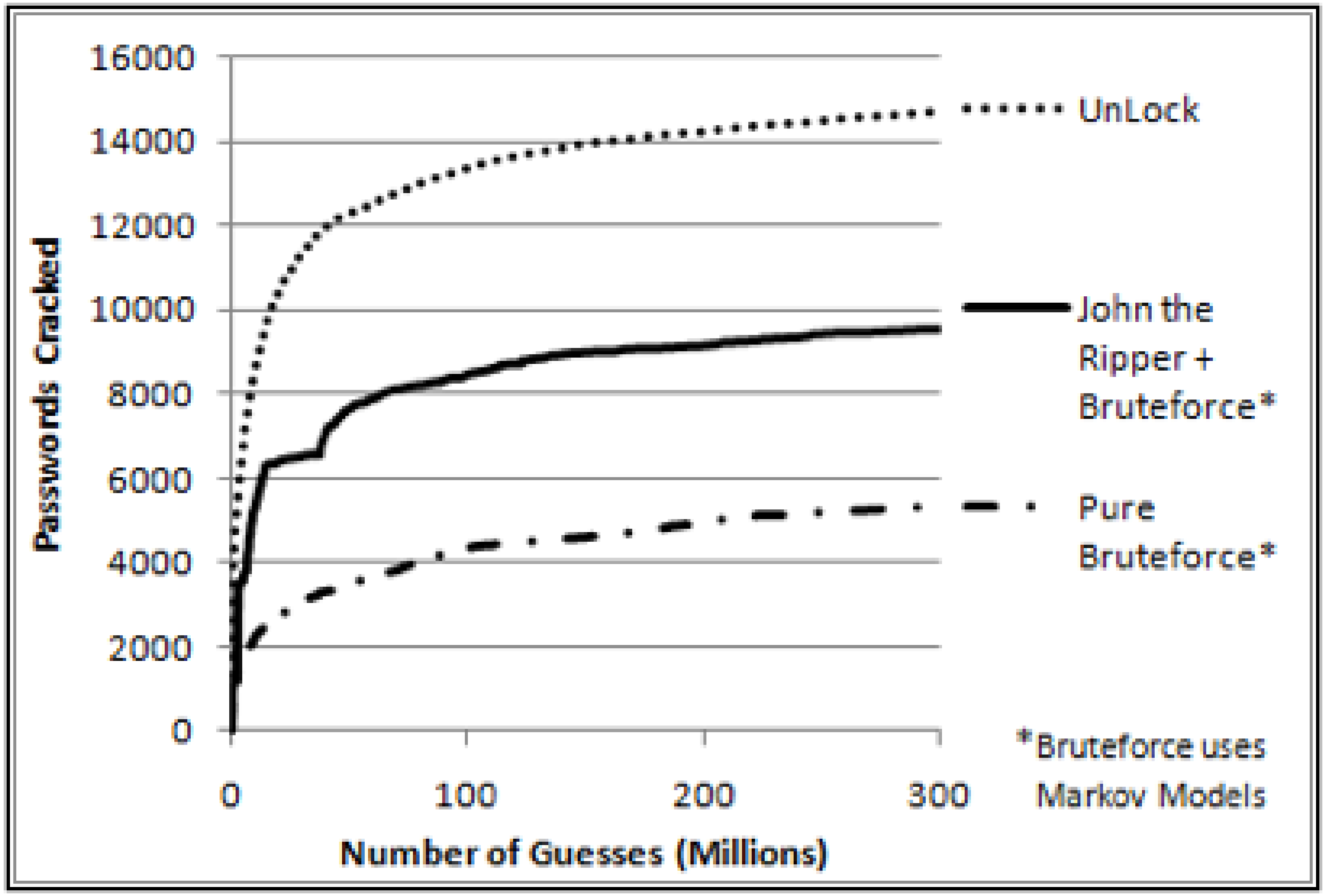
The MySpace List

Split it into a training list and a test list

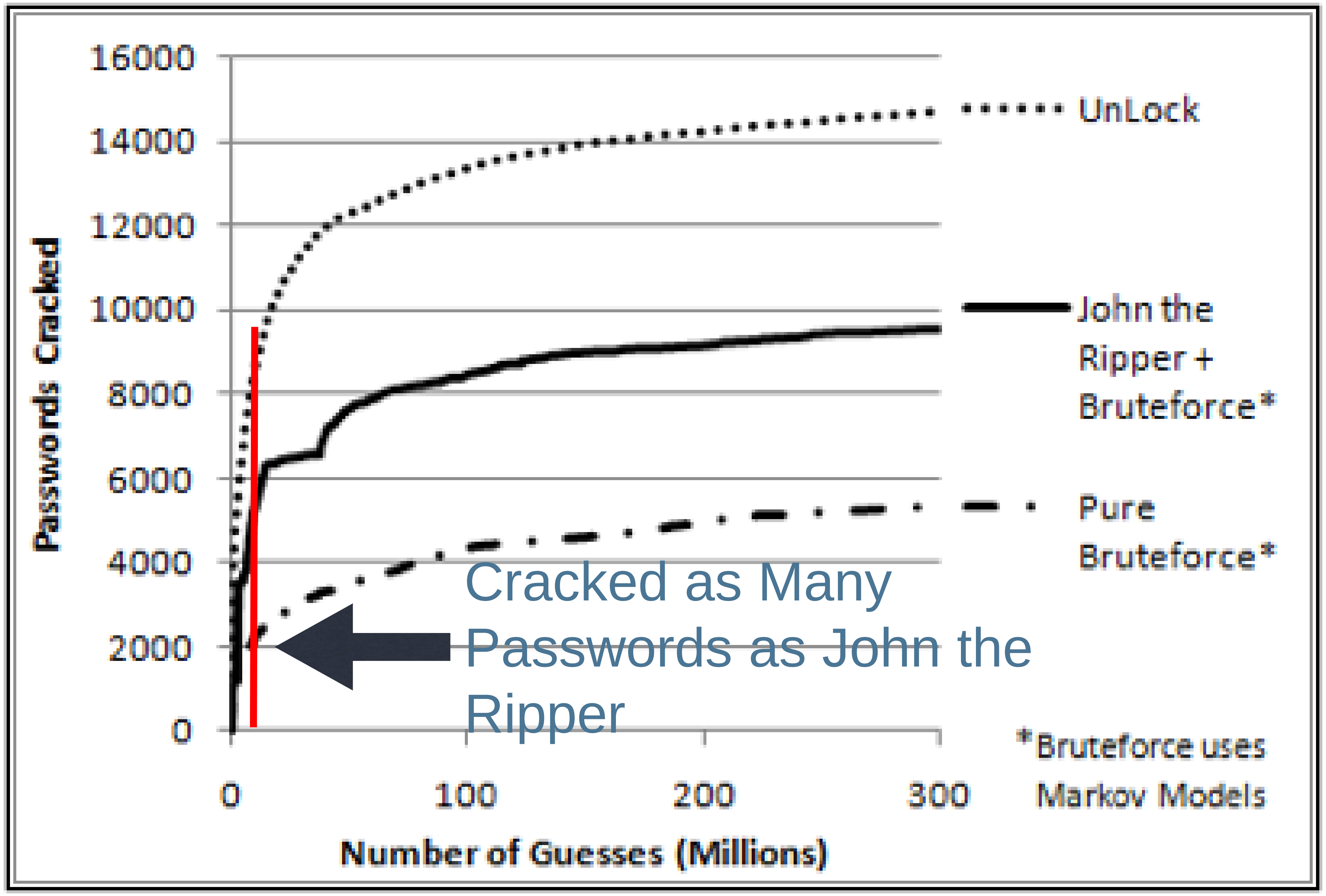
-Training List: 33,561

-Test List: 33,481

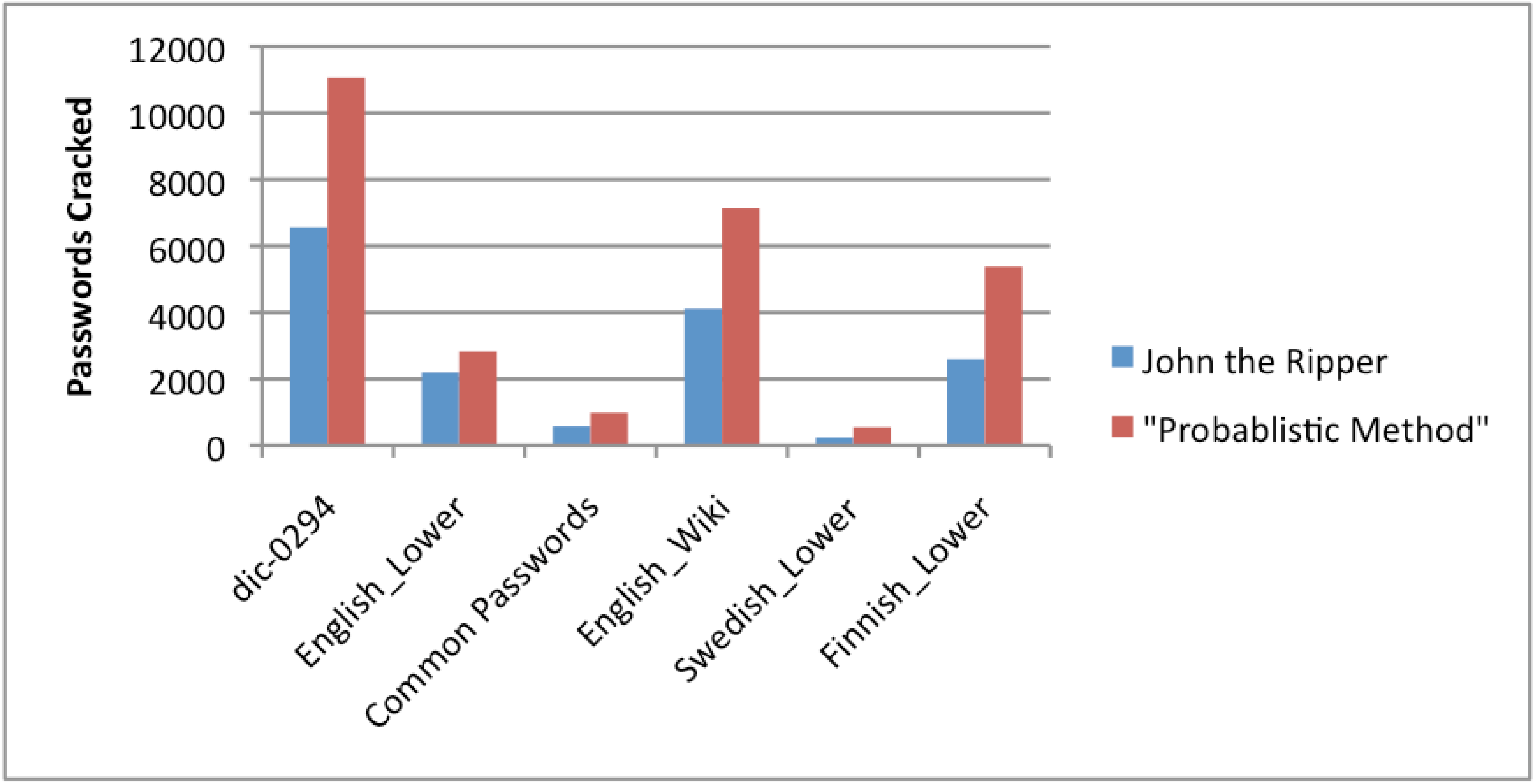
Results: Original Grammar



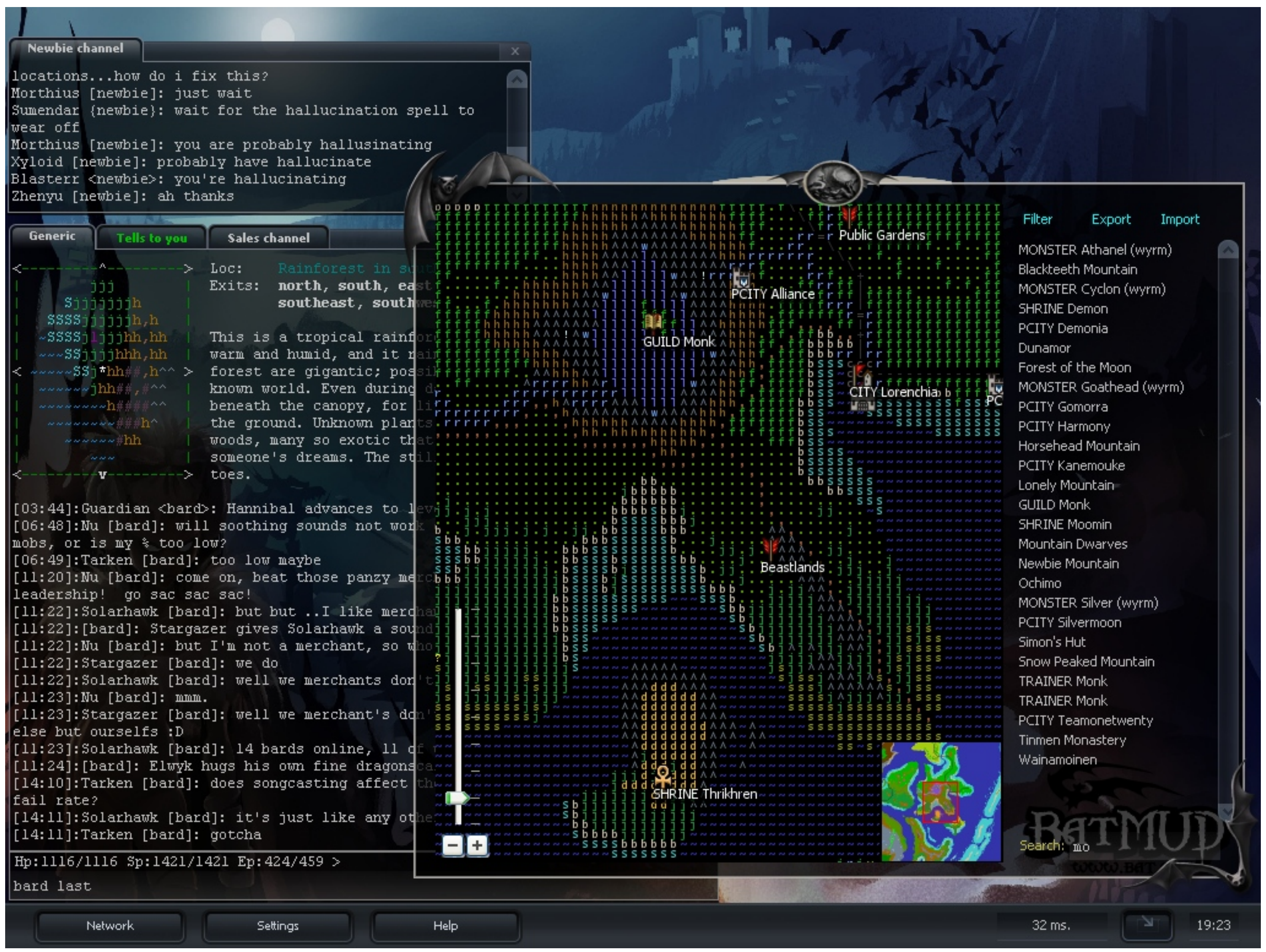
Results: Original Grammar



Real World Results - MySpace List

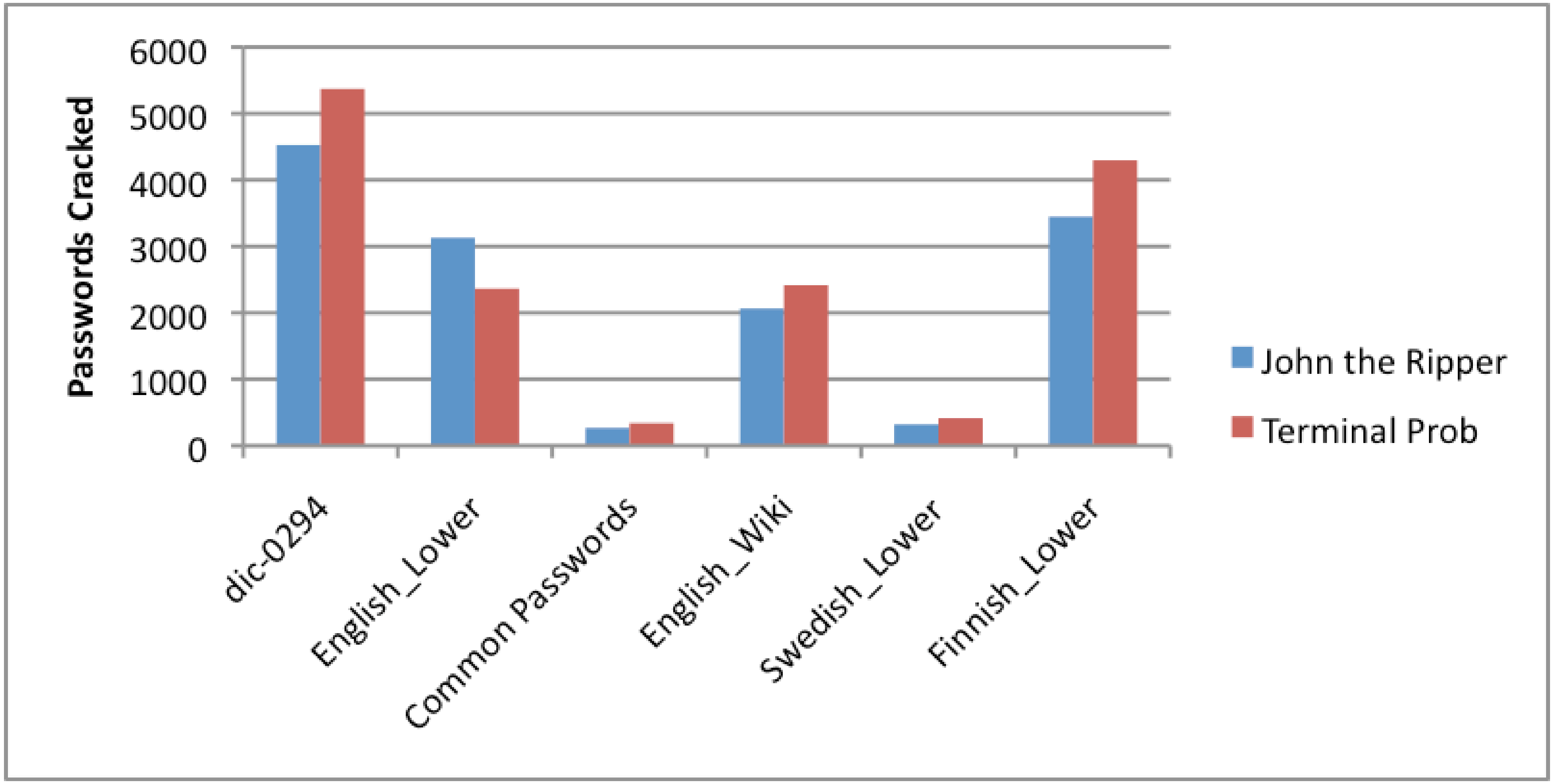


The Finnish List



- Hackers broke into several sites via SQL injection
- 15,699 Plain Text
- 29,853 MD5 Hashes

Finnish List



Cracking Demo