Reasoning Analytically About Password-Cracking Software

Enze “Alex” Liu, Amanda Nakanishi, Maximilian Golla, David Cash, Blase Ur
Attack Model

80d561388725fa74f2d03cd16e1d687c
1. \( h(“123456”) = e10adc3949ba59abbe56e057f20f883e \)
1. \( h(“123456”) = e10adc3949ba59abbe56e057f20f883e \)
2. \( h(“password”) = 5f4dcc3b5aa765d61d8327deb882cf99 \)
1. $h(“123456”) = e10adc3949ba59abbe56e057f20f883e$
2. $h(“password”) = 5f4dcc3b5aa765d61d8327deb882cf99$
3. $h(“monkey”) = d0763edaa9d9bd2a9516280e9044d885$
Attack Model

1. \( h(“123456”) = e10adc3949ba59abbe56e057f20f883e \)
2. \( h(“password”) = 5f4dcc3b5aa765d61d8327deb882cf99 \)
3. \( h(“monkey”) = d0763edaa9d9bd2a9516280e9044d885 \)
4. \( h(“letmein”) = 0d107d09f5bbe40cade3de5c71e9e9b7 \)
1. $h(123456) = e10adc3949ba59abbe56e057f20f883e$
2. $h(password) = 5f4dcc3b5aa765d61d8327deb882cf99$
3. $h(monkey) = d0763edaa9d9bd2a9516280e9044d885$
4. $h(letmein) = 0d107d09f5bbe40cade3de5c71e9e9b7$
5. $h(p@ssw0rd) = 0f359740bd1cda994f8b55330c86d845$
1. $h(“123456”) = e10adc3949ba59abbe56e057f20f883e$
2. $h(“password”) = 5f4dcc3b5aa765d61d8327deb882cf99$
3. $h(“monkey”) = d0763edaa9d9bd2a9516280e9044d885$
4. $h(“letmein”) = 0d107d09f5bbe40cade3de5c71e9e9b7$
5. $h(“p@ssw0rd”) = 0f359740bd1cda994f8b55330c86d845$
6. $h(“Chic4go”) = 80d561388725fa74f2d03cd16e1d687c$
Chic4go

Guess # 6
Chic4go

Guess # 6
Guess # 13,545,239,432
Password-Cracking Methods

Probabilistic Models

Software Tools

14
Password-Cracking Methods

Probabilistic Models

Software Tools

Chic4go

Guess #
Password-Cracking Methods

Probabilistic Models

Software Tools

Chic4go ➔ Guess #
Guess Number by Enumeration

1. 123456
2. password
3. monkey
4. letmein
5. p@ssw0rd
6. Chic4go

Does Not Scale !!!
Our Analysis Goals

1. Compute guess numbers efficiently
2. Configure guessing method systematically
Outline

● State of the art
● How software password-cracking tools work
● Our efficient techniques for guess numbers
● Our techniques for systematic configuration
Probabilistic Models

Markov Models [Narayanan and Shmatikov, CCS 2005]


Neural Networks [Melicher et al., Usenix Security 2016]

Guess #

Configuration
Probabilistic Models

Markov Models [Narayanan and Shmatikov, CCS 2005]
Neural Networks [Melicher et al., Usenix Security 2016]
Probabilistic Models

Markov Models [Narayanan and Shmatikov, CCS 2005]


Neural Networks [Melicher et al., Usenix Security 2016]

Monte Carlo Strength Evaluation: Fast and Reliable Password Checking [CCS 2015]

Matteo Dell’Amico
Symantec Research Labs, France
matteo_dellamico@symantec.com

Maurizio Filippone*
University of Glasgow, UK
maurizio.filippone@eurecom.fr
Probabilistic Models

Markov Models [Narayanan and Shmatikov, CCS 2005]


Neural Networks [Melicher et al., Usenix Security 2016]
Probabilistic Models

Markov Models [Narayanan and Shmatikov, CCS 2005]


Neural Networks [Melicher et al., Usenix Security 2016]

Guess-Efficient 🌟
Probabilistic Models

Markov Models [Narayanan and Shmatikov, CCS 2005]


Neural Networks [Melicher et al., Usenix Security 2016]

Guess-Efficient 🌿 Wall-Clock Time Slow 🥾
Software Tools

John the Ripper

Hashcat
Software Tools

chicago1
chicago2
chicago3
chicago6
chicago9

chicdog
chicagos
CHICAG
chicaga
Chicago
CHICAGO
CHIicago
Software Tools

John the Ripper
Hashcat

Guess-Inefficient
Wall-Clock Time Fast
Software Tools

John the Ripper
Hashcat

Guess-Inefficient 🔴
Wall-Clock Time Fast 🔵
Software Tools

John the Ripper
Hashcat

Guess #
Configuration

Reasoning Analytically About Password-Cracking Software
[S&P 2019]
Enze Liu, Amanda Nakanishi, Maximilian Golla†, David Cash, Blase Ur
University of Chicago, † Ruhr University Bochum
Outline

- State of the art
- How software password-cracking tools work
- Our efficient techniques for guess numbers
- Our techniques for systematic configuration
Mangled Wordlist Attack
Mangled Wordlist Attack

Wordlist

- Super
- Password
- Chicago
# Mangled Wordlist Attack

## Wordlist

<table>
<thead>
<tr>
<th>Super</th>
</tr>
</thead>
<tbody>
<tr>
<td>Password</td>
</tr>
<tr>
<td>Chicago</td>
</tr>
</tbody>
</table>

## Rulelist

1. Append “1”
2. Replace “a” → “4”
3. Lowercase all
Mangled Wordlist Attack

Wordlist
- Super
- Password
- Chicago

Rulelist
1. Append “1”
2. Replace “a” → “4”
3. Lowercase all

Guesses
- Super1
Mangled Wordlist Attack

**Wordlist**
- Super
- Password
- Chicago

**Rulelist**
1. Append “1”
2. Replace “a” → “4”
3. Lowercase all

**Guesses**
- Super1
- Password1
Mangled Wordlist Attack

Wordlist

Super Password Chicago

Rulelist

1. Append “1”
2. Replace “a” → “4”
3. Lowercase all

Guesses

Super1 Password1 Chicago1
Mangled Wordlist Attack

Wordlist

Super Password Chicago

Rulelist

1. Append “1”
2. Replace “a” → “4”
3. Lowercase all

Guesses

Super1 Password1 Chicago1 Super P4ssword Chic4go
Mangled Wordlist Attack

Wordlist

Super
Password
Chicago

Rulelist

1. Append “1”
2. Replace “a” → “4”
3. Lowercase all

Guesses

Super1
Password1
Chicago1
Super
P4ssword
Chic4go
super
password
chicago
Example Wordlists and Rulelists

Wordlist

PGS ($\approx 20,000,000$)
Linkedin ($\approx 60,000,000$)
HIBP ($\approx 500,000,000$)
## Example Wordlists and Rulelists

<table>
<thead>
<tr>
<th>Wordlist</th>
<th>Rulelist</th>
</tr>
</thead>
<tbody>
<tr>
<td>PGS ($\approx 20,000,000$)</td>
<td>Korelogic ($\approx 5,000$)</td>
</tr>
<tr>
<td>Linkedin ($\approx 60,000,000$)</td>
<td>Megatron ($\approx 15,000$)</td>
</tr>
<tr>
<td>HIBP ($\approx 500,000,000$)</td>
<td>Generated2 ($\approx 65,000$)</td>
</tr>
</tbody>
</table>
# Example Wordlists and Rulelists

<table>
<thead>
<tr>
<th>Wordlist</th>
<th>Rulelist</th>
<th>10^9 - 10^{15} guesses</th>
</tr>
</thead>
<tbody>
<tr>
<td>PGS (≈ 20,000,000)</td>
<td>Korelogic (≈ 5,000)</td>
<td></td>
</tr>
<tr>
<td>Linkedin (≈ 60,000,000)</td>
<td>Megatron (≈ 15,000)</td>
<td></td>
</tr>
<tr>
<td>HIBP (≈ 500,000,000)</td>
<td>Generated2 (≈ 65,000)</td>
<td></td>
</tr>
</tbody>
</table>
## Example Wordlists and Rulelists

<table>
<thead>
<tr>
<th>Wordlist</th>
<th>Rulelist</th>
</tr>
</thead>
<tbody>
<tr>
<td>PGS (≈ 20,000,000)</td>
<td>Korelogic (≈ 5,000)</td>
</tr>
<tr>
<td>Linkedin (≈ 60,000,000)</td>
<td>Megatron (≈ 15,000)</td>
</tr>
<tr>
<td>HIBP (≈ 500,000,000)</td>
<td>Generated2 (≈ 65,000)</td>
</tr>
</tbody>
</table>

+ Hackers’ private word/rule lists

$10^9 - 10^{15}$ guesses
Outline

- State of the art
- How software password-cracking tools work
- Our efficient techniques for guess numbers
- Our techniques for systematic configuration
Is This Password in the Guesses?

Chic4go

Guesses

Super1
Password1
Chicago1
Super
P4ssword
Chic4go
super
password
chicago
Is This Password in the Guesses?

Wordlist
- Super
- Password
- Chicago

Rulelist
1. Append “1”
2. Replace “a” → “4”
3. Lowercase all

Guesses
- Super1
- Password1
- Chicago1
- Super
- P4ssword
- Chic4go
- super
- password
- chicago
Insight

We can work backwards!
Insight: Invert Rules
Insight: Invert Rules

Rulelist

1. Append “1”
2. Replace “a” → “4”
3. Lowercase all

Password

Chic4go
Insight: Invert Rules

Rulelist

1. Append “1”
2. Replace “a” → “4”
3. Lowercase all

Password

Chic4go
Insight: Invert Rules

Preimages

Chicago
Chic4go

Rulelist

1. Append “1”
2. Replace “a” → “4”
3. Lowercase all

Password

Chic4go
### Table 1: ALU Function Definitions

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
<th>Description</th>
<th>Example Rule</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nothing</td>
<td></td>
<td>do nothing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lower-case</td>
<td>i</td>
<td>Lowercase all letters</td>
<td>p@SSWord</td>
<td></td>
</tr>
<tr>
<td>Upper-case</td>
<td>u</td>
<td>Uppercase all letters</td>
<td>p@SSWord</td>
<td></td>
</tr>
<tr>
<td>Capitalize</td>
<td>c</td>
<td>Capitalize the first letter and lower the rest of the word</td>
<td>p@SSWord</td>
<td></td>
</tr>
<tr>
<td>Insert Capitalize</td>
<td>c</td>
<td>Lowercase first found character, upercase the rest</td>
<td>p@SSWord</td>
<td></td>
</tr>
<tr>
<td>Toggle Case</td>
<td>t</td>
<td>Toggle the case of all characters in word.</td>
<td>p@SSWord</td>
<td></td>
</tr>
<tr>
<td>Toggle @ N</td>
<td>t</td>
<td>Toggle the case of characters at position N</td>
<td>p@SSWord</td>
<td></td>
</tr>
<tr>
<td>Reverse</td>
<td>r</td>
<td>Reverse the entire word</td>
<td>p@SSWord</td>
<td></td>
</tr>
<tr>
<td>Duplicate</td>
<td>d</td>
<td>Duplicate entire word</td>
<td>p@SSWord</td>
<td></td>
</tr>
<tr>
<td>Duplicate N</td>
<td>pN</td>
<td>Append duplicated word N times</td>
<td>p@SSWord</td>
<td></td>
</tr>
<tr>
<td>Reflect</td>
<td>f</td>
<td>Duplicate word reversed</td>
<td>p@SSWord</td>
<td></td>
</tr>
<tr>
<td>Rotate Left</td>
<td>l</td>
<td>Rotates the word left</td>
<td>p@SSWord</td>
<td></td>
</tr>
<tr>
<td>Rotate Right</td>
<td>r</td>
<td>Rotates the word right</td>
<td>p@SSWord</td>
<td></td>
</tr>
<tr>
<td>Append Character</td>
<td>SX</td>
<td>Append character X to end</td>
<td>p@SSWord</td>
<td></td>
</tr>
<tr>
<td>Prepend Character</td>
<td>X</td>
<td>Prepend character X from front</td>
<td>p@SSWord</td>
<td></td>
</tr>
<tr>
<td>Truncate Left</td>
<td>i</td>
<td>Deletes first character</td>
<td>p@SSWord</td>
<td></td>
</tr>
<tr>
<td>Truncate Right</td>
<td>r</td>
<td>Deletes last character</td>
<td>p@SSWord</td>
<td></td>
</tr>
<tr>
<td>Delete @ N</td>
<td>D</td>
<td>Deletes character at position N</td>
<td>p@SSWord</td>
<td></td>
</tr>
<tr>
<td>Extract Range</td>
<td>x</td>
<td>Extracts characters, starting at position N</td>
<td>p@SSWord</td>
<td></td>
</tr>
<tr>
<td>Omit Range</td>
<td>o</td>
<td>Omit characters, starting at position N</td>
<td>p@SSWord</td>
<td></td>
</tr>
<tr>
<td>Insert @ N</td>
<td>i</td>
<td>Inserts character X at position N</td>
<td>p@SSWord</td>
<td></td>
</tr>
<tr>
<td>Overwrite @ N</td>
<td>o</td>
<td>Overwrites character at position N with X</td>
<td>p@SSWord</td>
<td></td>
</tr>
<tr>
<td>Truncate @ N</td>
<td>t</td>
<td>Truncate word at position N</td>
<td>p@SSWord</td>
<td></td>
</tr>
<tr>
<td>Replace s@X</td>
<td>s</td>
<td>Replace all instances of X with Y</td>
<td>p@SSWord</td>
<td></td>
</tr>
<tr>
<td>Purge @IX</td>
<td>p</td>
<td>Purge all instances of X</td>
<td>p@SSWord</td>
<td></td>
</tr>
</tbody>
</table>

### Table 2: ALU Example Rules

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
<th>Description</th>
<th>Example Rule</th>
<th>Input Word</th>
<th>Output Word</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reject less</td>
<td>&lt;N</td>
<td>Reject plain if their length is greater than N</td>
<td>p@SSWord</td>
<td>p@SSWord</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reject greater</td>
<td>&gt;N</td>
<td>Reject plain if their length is less or equal to N</td>
<td>p@SSWord</td>
<td>p@SSWord</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reject equal</td>
<td>_N</td>
<td>Reject plain of length not equal to N</td>
<td>p@SSWord</td>
<td>p@SSWord</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reject contain</td>
<td>%X</td>
<td>Reject plain if contains a character X</td>
<td>p@SSWord</td>
<td>p@SSWord</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reject not contain</td>
<td>%X</td>
<td>Reject plain if does not contain a character X</td>
<td>p@SSWord</td>
<td>p@SSWord</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reject equal first</td>
<td>(X</td>
<td>Reject plain which does not start with X</td>
<td>p@SSWord</td>
<td>p@SSWord</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reject equal last</td>
<td>(X</td>
<td>Reject plain which does not end with X</td>
<td>p@SSWord</td>
<td>p@SSWord</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reject equals at</td>
<td>=N</td>
<td>Reject plain which do not have char X at position N</td>
<td>p@SSWord</td>
<td>p@SSWord</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reject contains</td>
<td>%NX</td>
<td>Reject plain if contains a character X less than N times</td>
<td>p@SSWord</td>
<td>p@SSWord</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reject contains</td>
<td>Q</td>
<td>Reject plain where the memory saved matches current word</td>
<td>p@SSWord</td>
<td>p@SSWord</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Example Rules:**

- `<G>`
- `>8`
- `_7`
- `/e`
- `(h)
- `?t`
- `=1a`
- `%2a`
- `rMrQ`

**Notes:**

- *For palindromes*

---

**Table 3: Bitwise ALU Example Rules**

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
<th>Description</th>
<th>Example Rule</th>
<th>Input Word</th>
<th>Output Word</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Swap front</td>
<td>k</td>
<td>Swaps first two characters</td>
<td>k</td>
<td>p@SSWord</td>
<td>p@SSWord</td>
<td></td>
</tr>
<tr>
<td>Swap back</td>
<td>K</td>
<td>Swaps last two characters</td>
<td>K</td>
<td>p@SSWord</td>
<td>p@SSWord</td>
<td></td>
</tr>
<tr>
<td>Swap @ N</td>
<td>*NM</td>
<td>Swaps character at position N with character at position M</td>
<td>*34</td>
<td>p@SSWord</td>
<td>p@SSWord</td>
<td></td>
</tr>
<tr>
<td>Bitwise shift left</td>
<td>L</td>
<td>Bitwise shift left character @ N</td>
<td>L2</td>
<td>p@SSWord</td>
<td>p@SSWord</td>
<td></td>
</tr>
<tr>
<td>Bitwise shift right</td>
<td>R</td>
<td>Bitwise shift right character @ N</td>
<td>R2</td>
<td>p@SSWord</td>
<td>p@SSWord</td>
<td></td>
</tr>
<tr>
<td>Ascii increment</td>
<td>+N</td>
<td>Increment character @ N by 1 ascii value</td>
<td>+2</td>
<td>p@SSWord</td>
<td>p@SSWord</td>
<td></td>
</tr>
<tr>
<td>Ascii decrement</td>
<td>-N</td>
<td>Decrement character @ N by 1 ascii value</td>
<td>-1</td>
<td>p@SSWord</td>
<td>p@SSWord</td>
<td></td>
</tr>
<tr>
<td>Replace N + 1</td>
<td>N</td>
<td>Replaces character @ N with value at @ N plus 1</td>
<td>.1</td>
<td>p@SSWord</td>
<td>p@SSWord</td>
<td></td>
</tr>
<tr>
<td>Replace N – 1</td>
<td>N</td>
<td>Replaces character @ N with value at @ N minus 1</td>
<td>.1</td>
<td>p@SSWord</td>
<td>p@SSWord</td>
<td></td>
</tr>
<tr>
<td>Duplicate block</td>
<td>yN</td>
<td>Duplicates first N characters</td>
<td>y2</td>
<td>p@SSWord</td>
<td>p@SSWord</td>
<td></td>
</tr>
<tr>
<td>Duplicate block</td>
<td>yN</td>
<td>Duplicates last N characters</td>
<td>y2</td>
<td>p@SSWord</td>
<td>p@SSWord</td>
<td></td>
</tr>
<tr>
<td>Title</td>
<td>E</td>
<td>Lower case the whole line, then upper case the first letter and every letter after a space</td>
<td>E</td>
<td>p@SSWord</td>
<td>p@SSWord</td>
<td></td>
</tr>
<tr>
<td>Title w/separat</td>
<td>eX</td>
<td>Lower case the whole line, then upper case the first letter and every letter after a custom separator character</td>
<td>e–</td>
<td>p@SSWord</td>
<td>p@SSWord</td>
<td></td>
</tr>
</tbody>
</table>

**Example Rules:**

- `p@SSWord W0rid`
- `p@SSWord W0rid`
- `p@SSWord W0rid`
- `p@SSWord W0rid`

**Notes:**

- *For palindromes*
*05 O03 d '7

Switch the first and the sixth char;
Delete the first three chars;
Duplicate the whole word;
Truncate the word to length 7;

Preimages?

Chic4go
Where in the Stream?

Wordlist

- Super
- Password
- Chicago

Rulelist

1. Append “1”
2. Replace “a” → “4”
3. Lowercase all

Guesses

- Super1
- Password1
- Chicago1

- Super
- P4ssword
- Chic4go
Where in the Stream?

**Wordlist**
- Super
- Password
- Chicago

**Rulelist**
1. Append “1”
2. Replace “a” → “4”
3. Lowercase all

**Guesses**
- Super1
- Password1
- Chicago1
- Super
- P4ssword
- Chic4go
Counting Guesses For Each Rule

Wordlist

Super Password
Chicago

Rule

Reject if no “a”;
Replace a→ 4

Guesses

2
Our First Contribution

- Fast Guess Number Estimation
Fast Guess Number Estimation

Linkedin + SpiderLab
Fast Guess Number Estimation

Linkedin + SpiderLab $\equiv 3.01 \times 10^{14}$ Guesses
Fast Guess Number Estimation

\[
\text{Linkedin + SpiderLab} \equiv 3.01 \times 10^{14} \quad \text{Guesses}
\]

<table>
<thead>
<tr>
<th></th>
<th>Enumeration</th>
<th>Our Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>~ 3 PB</td>
<td>~ 10 GB</td>
</tr>
</tbody>
</table>
Fast Guess Number Estimation

Linkedin + SpiderLab \( \equiv \) \( 3.01 \times 10^{14} \) Guesses

<table>
<thead>
<tr>
<th></th>
<th>Enumeration</th>
<th>Our Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>(~ 3 \text{ PB})</td>
<td>(~ 10 \text{ GB})</td>
</tr>
<tr>
<td>Preprocessing</td>
<td>(~ 2 \text{ years})</td>
<td>(~ &lt; 1 \text{ day} )</td>
</tr>
</tbody>
</table>
Fast Guess Number Estimation

Linkedin + SpiderLab $\equiv 3.01 \times 10^{14}$ Guesses

<table>
<thead>
<tr>
<th></th>
<th>Enumeration</th>
<th>Our Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Size</strong></td>
<td>~ 3 PB</td>
<td>~ 10 GB</td>
</tr>
<tr>
<td><strong>Preprocessing</strong></td>
<td>&gt; 2 years</td>
<td>&lt; 1 day</td>
</tr>
<tr>
<td><strong>Mean Lookup</strong></td>
<td>???</td>
<td>&lt; 1 second</td>
</tr>
</tbody>
</table>
Outline

● State of the art
● How software password-cracking tools work
● Our efficient techniques for guess numbers
● Our techniques for systematic configuration
Software Tools Depend On

- Order of rules
- Contents of the rulelist
- Order of words
- Contents of the wordlist
Insight: Data-Driven Configuration

Wordlist

Rulelist

Password Set
Insight: Data-Driven Configuration

Wordlist  Rulelist  Password Set

New configuration
Data-Driven Configuration

- Order of rules
- Contents of the rulelist
- Order of words
- Contents of the wordlist
Rule Ordering

- Should the rules be in a different order?

1. Append “1”
2. Replace “a” → “4”
3. Lowercase all

1. Replace “a” → “4”
2. Lowercase all
3. Append “1”
Rule Ordering

- Should the rules be in a different order?
- Key idea: Order by # cracks per guess

1. Append “1”
2. Replace “a” → “4”
3. Lowercase all

1. Replace “a” → “4”
2. Lowercase all
3. Append “1”
Rule Ordering Results

![Graph showing percentage of rules cracked against total number of guesses. The x-axis represents the total number of guesses in the range of $10^7$ to $10^{13}$, and the y-axis represents the percent cracked in the range of 40% to 80%. The graph shows a stepwise increase in the percentage of rules cracked as more guesses are made, with a label indicating "Original." ]
Rule Ordering Results

Graph showing the percent cracked against the total number of guesses for two datasets: Ideal and Original.
Rule Ordering Results

Percent Cracked vs Total # of Guesses graph showing three lines:
- Ideal
- Data-driven
- Original
Word Completeness

- Should other words be in the wordlist?
- Key idea: Add frequent preimage “misses”

Preimages: Oakland

Rulelist:
1. Append “1”
2. Replace “a” → “4”
3. Lowercase all

Oakland1
O@kl@nd
oakland
Word Completeness (Sample Results)

<table>
<thead>
<tr>
<th>Category</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set-specific</td>
<td>bfheros; ilovmyneopets”””</td>
</tr>
</tbody>
</table>
## Word Completeness (Sample Results)

<table>
<thead>
<tr>
<th>Category</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set-specific</td>
<td>bfheros; ilovmyneopets&quot;&quot;&quot;&quot;</td>
</tr>
<tr>
<td>Meaningful</td>
<td>MaSterBrain; la la la</td>
</tr>
</tbody>
</table>
## Word Completeness (Sample Results)

<table>
<thead>
<tr>
<th>Category</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set-specific</td>
<td>bfheros; ilovmyneopets””””</td>
</tr>
<tr>
<td>Meaningful</td>
<td>MaSterBrain; la la la</td>
</tr>
<tr>
<td>Short strings</td>
<td>a2; a23; 7a; b2; q2</td>
</tr>
</tbody>
</table>
Takeaway

Analytical Tools
Takeaway

**Analytical Tools**

**Guess Number**

**Analytical Tools**
Takeaway

- Guess Number
- Configuration Tools
- Analytical Tools
Takeaway

https://github.com/UChicagoSUPERgroup/

Guess Number  Configuration Tools

Analytical Tools
Takeaway

https://github.com/UChicagoSUPERgroup/

- Guess Number
- Configuration Tools
- Analytical Tools

Reasoning Analytically About Password-Cracking Software

Enze “Alex” Liu, Amanda Nakanishi, Maximilian Golla, David Cash, Blase Ur