Password Cracking Using Probabilistic Context Free Grammars

Emily Bennett & Henry O'Brien

April 17, 2019

COMS4507

Relevant Papers

M. Weir, S. Aggarwal, B. Medeiros and B. Glodek, "Password Cracking Using Probabilistic Context-Free Grammars," in *30th IEEE Symposium on Security and Privacy*, Berkeley, CA, 2009, pp. 391-405. Available: `http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5207658&isnumber=5207632`

S. Houshmand, S. Aggarwal and R. Flood. (2015, Aug). "Next Gen PCFG Password Cracking". *IEEE Transactions on Information Forensics and Security. vol. 10, no. 8*, pp. 1776-1791. Available: `http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7098389&isnumber=7127092`

Agenda

- Background
- Password cracking using PCFGs
  - Password preprocessing
  - Computing probabilities
  - Optimisation
- "Next Generation" Password Cracking
  - Keyboard patterns
  - Alpha strings
- Evaluation of techniques

Password Cracking Refresher

- Find an input, x, such that $h(x)$ matches the password hash

Password Cracking Refresher

- Find an input, x, such that $h(x)$ matches the password hash
- Brute force?
  - Slow!

Password Cracking Refresher

- Find an input, x, such that $h(x)$ matches the password hash
- Brute force?
  - Slow!
- Dictionary attacks
  - Mangling rules
  - How can we work out what passwords to try first?

Probabilistic Password Cracking

Probabilistic Password Cracking

- People are bad at coming up with passwords
  - How do we take advantage of this?

Probabilistic Password Cracking

- People are bad at coming up with passwords
  - How do we take advantage of this?
- Obtain existing leaked passwords to use as training data

Probabilistic Password Cracking

- People are bad at coming up with passwords
  - How do we take advantage of this?
- Obtain existing leaked passwords to use as training data
- Analyse this training set to discover patterns

Probabilistic Password Cracking

- People are bad at coming up with passwords
  - How do we take advantage of this?
- Obtain existing leaked passwords to use as training data
- Analyse this training set to discover patterns
- Assign these patterns probabilities based on how frequently they occur in the data

Probabilistic Password Cracking

- People are bad at coming up with passwords
  - How do we take advantage of this?
- Obtain existing leaked passwords to use as training data
- Analyse this training set to discover patterns
- Assign these patterns probabilities based on how frequently they occur in the data
- Allows us to try the most likely passwords first
  - 'password12' vs 'P@$$W0rd!23'

Context Free Grammars

- Context Free Grammars (CFGs) are a way of formally representing these patterns

- Context Free Grammars (CFGs) are a way of formally representing these patterns
- Consist of: Terminal symbols, non-terminal symbols, and productions

Context Free Grammars

- Context Free Grammars (CFGs) are a way of formally representing these patterns
- Consist of: Terminal symbols, non-terminal symbols, and productions
- Example: All possible strings of length two containing the letters a or b
    - $S \rightarrow YY$
    - $Y \rightarrow a$
    - $Y \rightarrow b$

Probabilistic Context Free Grammars

- Probabilistic CFGs assign a probability to each production based on how likely it is to occur

- Probabilistic CFGs assign a probability to each production based on how likely it is to occur
- If a is two times more likely to occur than b, then we have:

Production	**Probability**
$S \rightarrow YY$	1
$Y \rightarrow a$	$^{2}/_{3}$
$Y \rightarrow b$	$^{1}/_{3}$

Representing Passwords as PCFGs

- **Non-terminals**
  - L_n - Sequence of alphabet symbols (eg. "abc")
  - D_n - Sequence of digits (eg. "123")
  - S_n - Sequence of non-alpha/non-digits (eg. "#$%")

(each sequence of length n)

- **Non-terminals**
  - L_n - Sequence of alphabet symbols (eg. "abc")
  - D_n - Sequence of digits (eg. "123")
  - S_n - Sequence of non-alpha/non-digits (eg. "#$%")

(each sequence of length n)

$$\texttt{!!password1} \rightarrow S_2 L_8 D_1$$

1. **Training set $\rightarrow$ all possible base structures**

Training set $= \{!4!\mathit{dog}\$\$4, \mathit{dog}5!, \mathit{cat}5?, \mathit{cat}4!\}$

Production	Probability
$S \rightarrow S_1 D_1 S_1 L_3 S_2 D_1$	0.25
$S \rightarrow L_3 D_1 S_1$	0.75

2. Expand D and S non-terminals

Production	Probability
$S \rightarrow S_1 D_1 S_1 L_3 S_2 D_1$	0.25
$S \rightarrow L_3 D_1 S_1$	0.75
$D_1 \rightarrow 4$	0.6
$D_1 \rightarrow 5$	0.4
$S_1 \rightarrow !$	0.8
$S_1 \rightarrow ?$	0.2
$S_2 \rightarrow \$\$$	1

2. Expand D and S non-terminals

Production	Probability
$S \rightarrow S_1 D_1 S_1 L_3 S_2 D_1$	0.25
$S \rightarrow L_3 D_1 S_1$	0.75
$D_1 \rightarrow 4$	0.6
$D_1 \rightarrow 5$	0.4
$S_1 \rightarrow !$	0.8
$S_1 \rightarrow ?$	0.2
$S_2 \rightarrow \$\$$	1

$S \rightarrow L_3 D_1 S_1 \rightarrow L_3 4 S_1 \rightarrow L_3 4!$ ("pre-terminal")

2. Expand D and S non-terminals

Production	**Probability**
$S \rightarrow S_1 D_1 S_1 L_3 S_2 D_1$	0.25
$S \rightarrow L_3 D_1 S_1$	0.75
$D_1 \rightarrow 4$	0.6
$D_1 \rightarrow 5$	0.4
$S_1 \rightarrow !$	0.8
$S_1 \rightarrow ?$	0.2
$S_2 \rightarrow \$\$$	1

$$S \rightarrow L_3 D_1 S_1 \rightarrow L_3 4 S_1 \rightarrow L_3 4! \text{ (“pre-terminal”)}$$

$$P = 0.75 \times 0.6 \times 0.8 = 0.36$$

3. Identify pre-terminal with highest probability and perform dictionary attack

$$D = \{cat, dog, monkey, rat\}$$

3. Identify pre-terminal with highest probability and perform dictionary attack

$$D = \{cat, dog, monkey, rat\}$$

$$S \rightarrow L_3 4! \rightarrow cat4!$$

$$S \rightarrow L_3 4! \rightarrow dog4!$$

$$S \rightarrow L_3 4! \rightarrow rat4!$$

Optimisation

- Trivial approach
  1. Build list of *all* pre-terminal structures
  2. Sort in order of descending probability
  3. Perform dictionary attacks in an iterative manner
- This is inefficient in both time and memory!

Optimisation

- Trivial approach
  1. Build list of *all* pre-terminal structures
  2. Sort in order of descending probability
  3. Perform dictionary attacks in an iterative manner
- This is inefficient in both time and memory!
- Can optimise by running the probability calculations and dictionary attacks in parallel
  1. Build up a distributed priority queue of pre-terminals
  2. Concurrently pop the best entry from the PQ and perform dictionary attack

“Next Generation” Password Cracking

- The original PCFG technique encounters issues in certain scenarios

- The original PCFG technique encounters issues in certain scenarios
- 'qwertyuiop' would be represented as $S \rightarrow L_{10}$

- The original PCFG technique encounters issues in certain scenarios
- 'qwertyuiop' would be represented as $S \rightarrow L_{10}$
- 'hellohello123' becomes $S \rightarrow L_{10}D_3$

- The original PCFG technique encounters issues in certain scenarios
- ‘qwertyuiop’ would be represented as $S \rightarrow L_{10}$
- ‘hellohello123’ becomes $S \rightarrow L_{10}D_3$
  - Neither of these are common 10 letter words, but should still be easy to crack

Keyboard Patterns

- A keyboard pattern is a memorable sequence of keystrokes on a physical keyboard

Keyboard Patterns

- A keyboard pattern is a memorable sequence of keystrokes on a physical keyboard
- Modify PCFGs so that they identify these patterns during training

Keyboard Patterns

- A keyboard pattern is a memorable sequence of keystrokes on a physical keyboard
- Modify PCFGs so that they identify these patterns during training
- Introduce new non-terminal K_n
    - 'qwertyuiop' becomes $S \rightarrow K_{10}$

Resolving Ambiguity

- A grammar is ambiguous if there are multiple productions which can lead to the same terminal string

Resolving Ambiguity

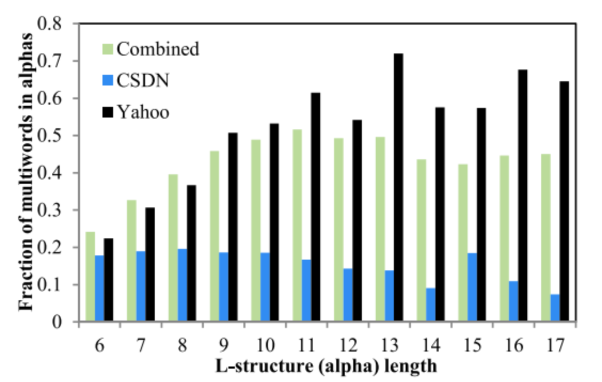
- A grammar is ambiguous if there are multiple productions which can lead to the same terminal string
- 'qw34!99' would have been $S \rightarrow L_2 D_2 S_1 D_2$, but can now also be represented by $S \rightarrow K_4 S_1 D_2$

Resolving Ambiguity

- A grammar is ambiguous if there are multiple productions which can lead to the same terminal string
- 'qw34!99' would have been $S \rightarrow L_2 D_2 S_1 D_2$, but can now also be represented by $S \rightarrow K_4 S_1 D_2$
- We introduce rules that deterministically choose one non-terminal over another
  - 'qw34!99' is now always $S \rightarrow K_4 S_1 D_2$

Alpha Strings

- Can break up L non-terminal into:
  - A_n - single dictionary word or pattern (eg. cat)
  - R_n - word or pattern repeated once (eg. catcat)
  - M_n - two or more consecutive A-words, excluding R-words (eg. iloveyou)



Alpha Strings Example

Training set $= \{catcat123, passwd123, iloveyou456\}$

Without alpha strings

Production	**Probability**
$S \rightarrow L_6 D_3$	$^2/_3$
$S \rightarrow L_8 D_3$	$^1/_3$
$D_3 \rightarrow 123$	$^2/_3$
$D_3 \rightarrow 456$	$^1/_3$

Alpha Strings Example

Training set $= \{catcat123, passwd123, iloveyou456\}$

With alpha strings

Production	**Probability**
$S \to R_6 D_3$	$1/3$
$S \to A_6 D_3$	$1/3$
$S \to M_8 D_3$	$1/3$
$D_3 \to 123$	$1/2$
$D_3 \to 456$	$1/2$

Alpha Strings Cracking Phase

- Each alpha non-terminal requires a different approach during the dictionary attack phase
  - A_n category - replace with single words of length n from attack dictionary
  - R_n category - replace with two occurrences of each word of length $n/2$ from attack dictionary
  - M_n category - incorporated into grammar

Production	**Probability**
$S \rightarrow M_8$	1
$M_8 \rightarrow iloveyou$	0.35
$M_8 \rightarrow someword$	0.00004
...	...

Evaluation

Evaluation

- PCFG password cracking has been tested against John the Ripper's default mangling rules

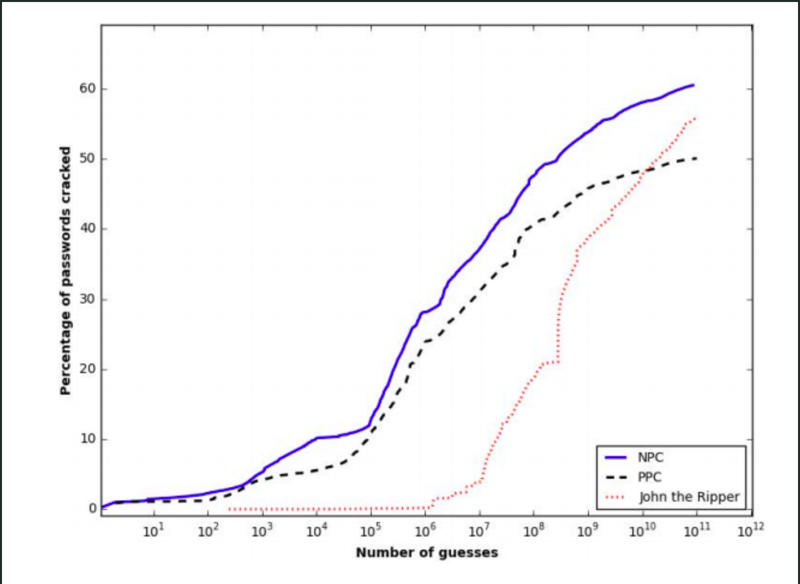
Evaluation

- PCFG password cracking has been tested against John the Ripper's default mangling rules
- PCFG cracked between 28% and 129% more passwords given the same number of guesses when it was trained on the same data set as it was cracking

Evaluation

- PCFG password cracking has been tested against John the Ripper's default mangling rules
- PCFG cracked between 28% and 129% more passwords given the same number of guesses when it was trained on the same data set as it was cracking
- Performed better than JTR when trained on a different set to the one it was cracking (except when the sets were of different complexities)

Evaluation



Percentage of passwords cracked
Number of guesses
0
10
20
30
40
50
60
10^1
10^2
10^3
10^4
10^5
10^6
10^7
10^8
10^9
10^{10}
10^{11}
10^{12}
NPC
PPC
John the Ripper

Conclusion

- PCFGs provide a new way to think about password cracking, and a way to formalise the patterns we can observe amongst passwords

Conclusion

- PCFGs provide a new way to think about password cracking, and a way to formalise the patterns we can observe amongst passwords
- This is an area of password cracking which is continuing to grow, and will likely become more and more effective as further work on it is done.

Questions?