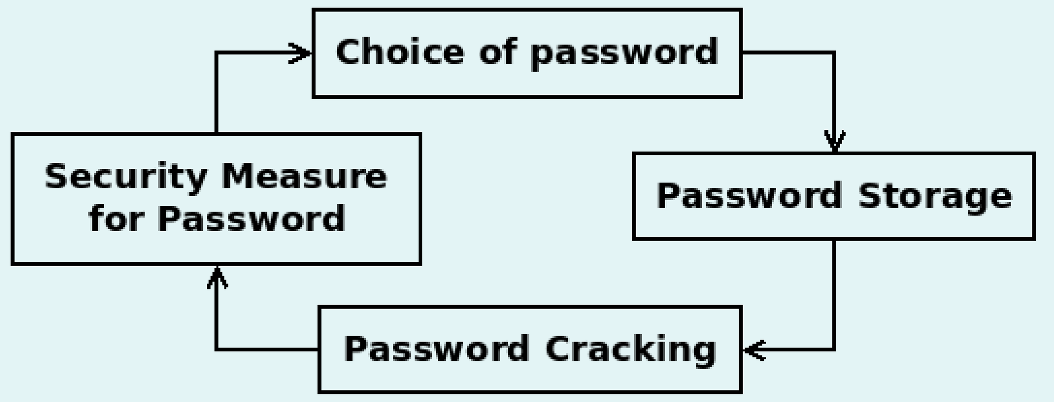
Password Guessability with Markov Model

Derek Tzeng, Yiming Zong
Advisors: Lujo Bauer, Blase Ur
School of Computer Science, Carnegie Mellon University, Pittsburgh, PA

Introduction & Problem Statement

- Passwords has been and will be the most commonly used method of authentication.
- Common question: how many times does an adversary need in order to guess a password?
- Our goal: Study state-of-the-art password cracking techniques, and implement a look-up table to allow efficient guessability analysis with Markov model.

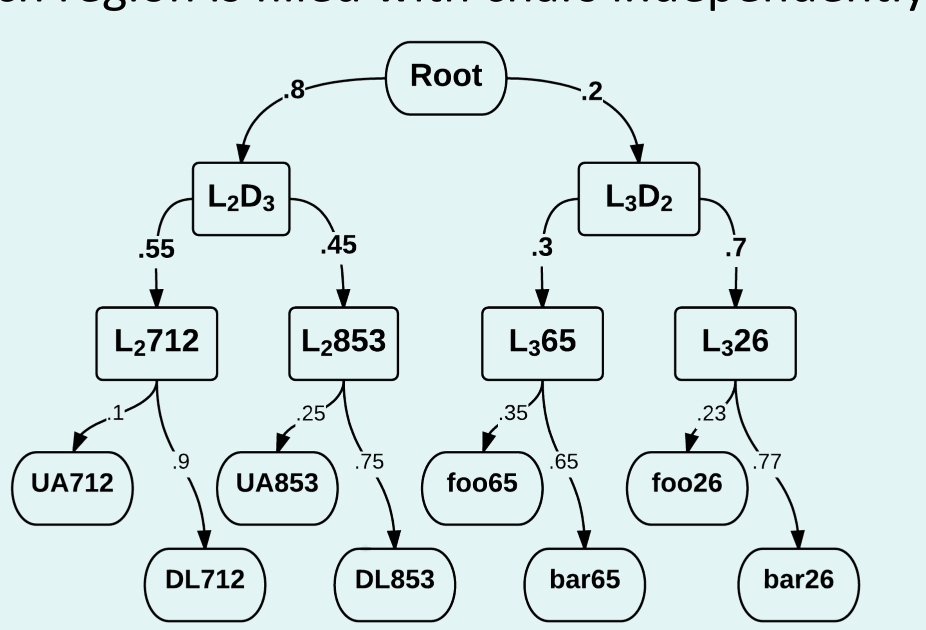


Password Cracking Techniques

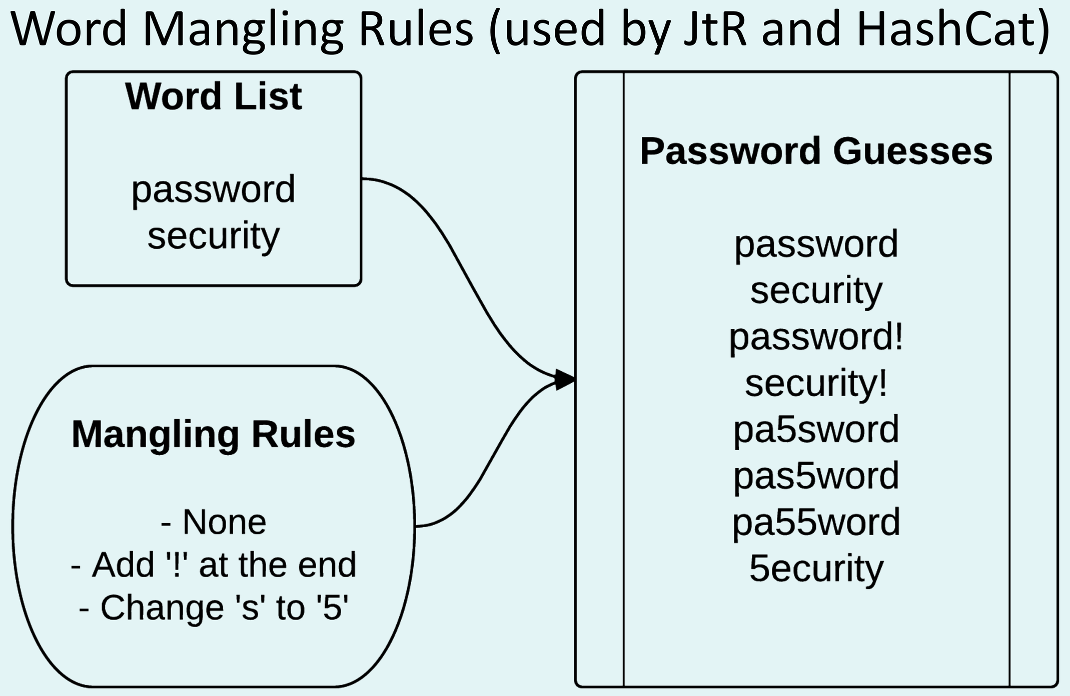
- Brute Force / Masked Brute Force:

```
aaa, aab, aac, ..., zzz
aaaa, aaab, aaac, ..., zzzz
```

- Probabilistic Conext-Free Grammar (PCFG):
  - Splits password into homogeneous "regions"
  - Different regions have different probability
  - Each region is filled with chars independently



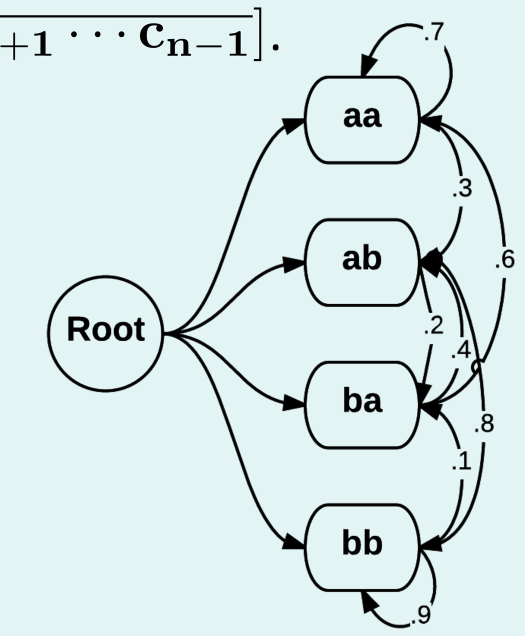
- Word Mangling Rules (used by JtR and HashCat)



Methodology

- Markov Model (k-gram):
  - Next character depends on its previous (k-1)-gram, i.e. $\Pr[\mathbf{c} \mid \overline{\mathbf{x_{n-k+1}\cdots c_{n-1}}}]$.

Pr[a | aa] = .7
Pr[b | aa] = .3
Pr[a | ab] = .2
Pr[b | ab] = .8
Pr[a | ba] = .6
Pr[b | ba] = .4
Pr[a | bb] = .1
Pr[b | bb] = .9



Step One: Learn Probability Parameters

$$\Pr[\mathbf{c} \mid \overline{\mathbf{x_{n-k+1}\cdots c_{n-1}}}] := \frac{\operatorname{count}(\overline{\mathbf{x_{n-k+1}\cdots c_{n-1}c}}) + \delta}{\operatorname{count}(\overline{\mathbf{x_{n-k+1}\cdots c_{n-1}\cdot}}) + \delta|\boldsymbol{\Sigma}|}$$

$$\Pr[\overline{\mathbf{x_0\cdots x_{k-2}}}] := \frac{\operatorname{count}(\overline{\mathbf{x_0\cdots x_{k-2}}}) + \delta}{|\mathcal{D}| + \delta|\boldsymbol{\Sigma}|^{\mathbf{k-1}}}$$

$$\Pr[\overline{\mathbf{x_{n-k+2}\cdots x_n}\perp}] := \frac{\operatorname{count}(\overline{\mathbf{x_{n-k+2}\cdots x_n}\perp}) + \delta}{|\mathcal{D}| + \delta|\boldsymbol{\Sigma}|^{\mathbf{k-1}}}$$

Step Two: Discretize Probabilities

- Initially proposed by Duermuth, et al. In 2015
- Reduce continuous variables to discrete variables:

$$\hat{\ell}[\cdot] = -\operatorname{round}(\log(\mathbf{c_1 p + c_2})).$$

- Parameters: $\ell_{\max}, \mathbf{c_1}, \mathbf{c_2}$, satisfying:

$$\begin{cases} -\log(\mathbf{c_2}) = \ell_{\max} \\ \log(\mathbf{c_1 p_{\max} + c_2}) = \mathbf{0} \end{cases}$$

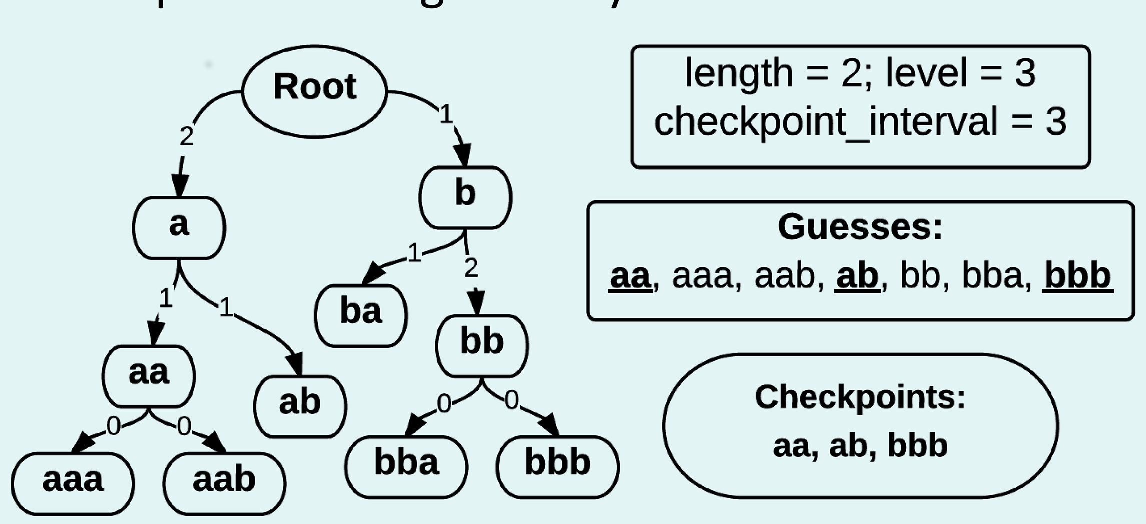
$$\implies (\mathbf{c_1}, \mathbf{c_2}) = \left(\frac{\mathbf{1} - \exp(-\ell_{\max})}{\mathbf{p_{\max}}}, \exp(-\ell_{\max})\right).$$

Step Three: Build Enumeration Index

- Level of a password = sum of the levels of its Markov probabilities, e.g:

$$\mathbf{lvl}(\text{"ab"}) := \hat{\ell}[\text{"a"}] + \hat{\ell}[\text{"b"} \mid \text{"a"}] + \hat{\ell}[\text{"a}\perp\text{"}].$$

For each (length, level)-pair, enumerate passwords with certain length and level with DFS and create "checkpoints" along the way for future use:



Methodology (Cont'd)

- **Step Four: Guess Count Calculation**
- Pick up the DFS process from the most recent checkpoint (if exists), and continue until finding the password. Overall, the guess number of password is:

$$\mathbf{g(x)} = \left(\sum_{\text{prior (len, lvl)}} |\mathcal{I}_{\text{len, lvl}}|\right) + \mathbf{g}_{\text{len, lvl}}(\mathbf{x}).$$

Results & Evaluation

- Input data: RockYou leaked passwords (weighted with counts), Unix English dictionary
- Performance evaluation on personal machine: Ubuntu 14.04; Intel i5-3210 @ 2.50GHz; 8G RAM.
- $\mathbf{k = 3}, \ell_{\max} = \mathbf{10}, \mathrm{max\_passwd\_len} = \mathbf{12}.$

Step One: Index Builder

- Time taken: 6m52s
- Memory usage: peak at *3.625 GB*
- Storage: 1.6 GB for all modes
- Output format:

$$\{\mathbf{1} \mapsto [\text{"aa"}, \text{"ab"}], \mathbf{2} \mapsto [\text{"bb"}], \cdots\}$$

$$\{\text{"aa"} \mapsto \{\mathbf{0} \mapsto [\text{'c'}], \cdots\}, \text{"ab"} \mapsto \{\mathbf{2} \mapsto [\perp]\}, \cdots\}$$

Step Two: Checkpoint Builder

- Data range: All passwords with total level ≤ 22, totaling approx. 10^{11} passwords
- Time taken: 20+ hours
- Memory usage: peak at *150 MB / thread*
- Storage: 55MB for all checkpoints
- Output format: passwords & count

<u>len5-lvl10.out</u>

12da0
meiia
mizwa
roe87
dau14
… … …

1125991

Step Three: Guess Count Calculator

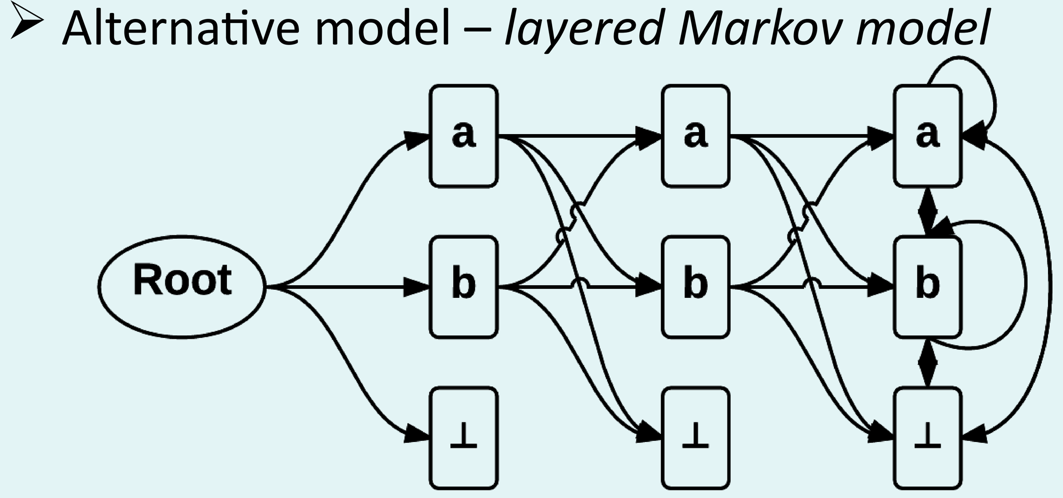
- Time per password: < 15 seconds (mostly loading index / checkpoints)
- Memory usage: peak at 150 MB
- Storage: None
- Output format:

```
> python guess.py
Input password to guess > juniker
Password length: 7; Total level: 9
Using binary search to calculate guess count...
Result: 14586299
```

Overall: Memory likely to hit resource limit first; guess count calculation time strongly correlated to checkpoint frequency.

Remarks & Future Work

- Time-space tradeoff:
  - More storage used for checkpointing gives faster run-time for guess number calculation; vice versa.
- Alternative model – *layered Markov model*



- Other smoothing techniques, e.g. Good-Turing:

$$\hat{c}(s) = \begin{cases} 0.22 & \text{for } \mathbf{c(s) = 1} \\ \mathbf{0.88} & \text{for } \mathbf{c(s) = 2} \\ \mathbf{c(x) - 1} & \text{for } \mathbf{c(s) > 2} \\ \text{previously deducted count} & \text{otherwise} \end{cases}$$

- Fine-tuned implementation based on our algorithm can be incorporated in a Guessability-as-a-Service module, like in [2].
- Source code and final report for the project is available in our Github repo [1], and intermediate output data (> 2GB) is available upon request.

Selected References

- M. Durmuth, F. Angelstorf, C. Castelluccia, D. Perito, and A. Chaabane. OMEN: faster password guesing using an ordered markov enumerator. In *Proc. ESSoS*, 2015.
- J. Ma, W. Yang, M. Luo, and N. Li. A study of probabilistic password models. In *Proc. IEEE Symp. On Security and Privacy*, 2014.
- B. Ur, S. M. Segreti, L. Bauer, N. Christin, L. F. Cranor, S. Komanduri, D. Kurilova, M. L. Mazurek, W. Melicher, and R. Shay. Measuring real-world accuracies and biases in modeling password guessability. In *Proc. USENIX Security*, 2015.

(Full list of references is available in the final report.)

Acknowledgements

The authors thank our advisors Lujo Bauer and Blase Ur for proposing the research topic and holding regular meetings to track our progress and review our report. We also thank Blase Ur for pointing us to many of the references, which introduced us to the area of password security.

[1] https://github.com/ymzong/password-guessability
[2] https://pgs.ece.cmu.edu

Password Guessability | 08-731 F15 | Dec 10, 2015