Beyond the Dictionary Attack:

Enhancing Password Cracking Efficiency through Machine Learning-Induced Mangling Rules

Radek Hranický Lucia Šírová Viktor Rucký

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Why rules?

FACT: Sysadmins want strong passwords!
Password policies: At least X characters, at least 1 special symbol, ...

People frequently use common easy-to-remember patterns:

- Uppercase letter? Most frequently it is the first character.
- Numbers or special symbols? Frequently at the end: Summer2023#
- CamelCase
- L33t$p3@k
- Multiple-words-separated by.special.characters
- Keyboard walks: Qwerty123!, Asdf2020$

Examples of mangling rules

Applied to „Password“

Name	Function	Example Rule	Output Word
Lowercase all letters	l	l	password
Toggle case	T	t	pASSWORD
Duplicate 1st letter N times	zN	Z2	PPPassword
Append character X to the end	$X	$1	Password1
Replace Xes with Ys	sXY	ss$	Pa$$word
Delete first character	[	[	assword

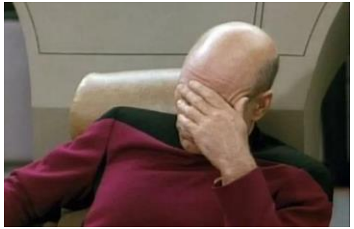
CRACK (1991) - First password cracker with mangling-rule support
JOHN THE RIPPER adopted Crack‘s rules and added more.
HASHCAT supports 56 unique rule commands, all applied on GPU.

How does a ruleset look like?

```
i59 o64
o40 R5
o81 i92 oA3
i50 ,6 o77
i69 o75
^4 ^2 r
o1o $1
l $f
^0 ^0 ^1
^8 ^0 r
^3 ^5 r
o4g
l $7 $3
o61 i72 i83 o94
i42 i50 i60 o74
i61 i72 o83 o94
ss1 $9 $8 $9
$7 $1 $4
o51 ss0 $1
$1 $0 $8
i59 o61
i63 o72
^3 ^3 ^3 r
i59 o60
^2 ^0 ^0 ^1 r
[ $2 $0 $0 $8
i3i $1
o78 o85
^3 ^1 T2
o3y ]
i42 ss0 $0 $5
o0g i1i
o0j $2 $0 $0 $8
```

How does a ruleset look like?

```
i59 o64
o40 R5
o81 i92 oA3
i50 ,6 o77
i69 o75
^4 ^2 r
o1o $1
l $f
^0 ^0 ^1
^8 ^0 r
^3 ^5 r
o4g
l $7 $3
o61 i72 i83 o94
i42 i50 i60 o74
i61 i72 o83 o94
ss1 $9 $8 $9
$7 $1 $4
o51 ss0 $1
$1 $0 $8
i59 o61
i63 o72
^3 ^3 ^3 r
i59 o60
^2 ^0 ^0 ^1 r
[ $2 $0 $0 $8
i3i $1
o78 o85
^3 ^1 T2
o3y ]
i42 ss0 $0 $5
o0g i1i
o0j $2 $0 $0 $8
```



Manual creation is possible... but it is PAIN ☹
How to make a ruleset that is actually „good“?

HOW TO create rulesets? (automatically)

01

Hashcat's generate-rules.c
Works but rules are purely RANDOM ☹

02

Marechal's rulesfinder
Works but require an existing ruleset ☹

03

Iphelix's PACK/rulegen
Based on password similarity

04

Clustering?

01 – Take an existing (training) password dictionary

DFRWS, hello, h3llo, dfrws, DFRW$

02 – Create clusters of similar passwords

by (Damerau-) Levenshtein distance

hello, h3llo   DFRWS, dfrws, DFRW$

03 – Select a (representative) password from each cluster

hello, h3llo   DFRWS, dfrws, DFRW$

04 – Create mangling rules that transform the representative to other passwords in the cluster

hello -> h3llo | **Replace all „e“ with „3“**

dfrws -> DFRWS | **Uppercase all letters**

dfrws -> DFRW$ | **Uppercase all letters AND Replace all „s“ with „$“**

Use as few commands as possible. If multiple are usable, use those **with the highest priority**.

05 – Count rule occurence, deduplicate and select N most frequent rules. DONE

General idea

Drdák & Hranický (2019-2020), Li et al. (2022)

Timeline of clustering-based approaches

Drdák & Hranický (2019-2020)

- Affinity propagation clustering method
- Works & provides decent results

- Distance matrix calculation „each x each“ required – **$O(n^2)$** time & space complexity
  -> not usable for bigger training dictionaries

Li et al. (2022)

- MDBSCAN (modified DBSCAN) clustering -> better handling of outliers -> better rules
- SymSpell fuzzy search algorithm instead of full distance matrix -> faster, less memory

- Cluster representative selection is not optimal
- Limited number of rule commands
- No other clustering methods tested
- No PoC implementation available

Let‘s improve the representative selection

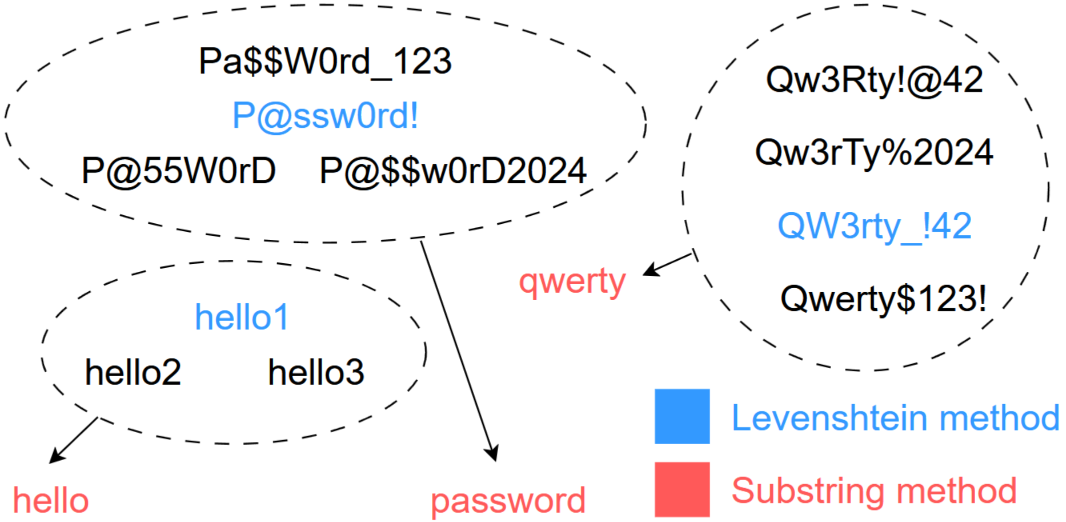
ISSUE: In the classic „Levenshtein method“ (Drdák et al., Li et al.), the representative is **ALWAYS AN EXISTING PASSWORD -> not always good** ☹

... and thus, we came with

The SUBSTRING method

1. Revert leetspeak transformations
2. Convert all letters to lowercase
3. Find the longest common substring
4. The substring is the representative

In theory, this should provide more accurate representations of the „base word“



The COMBO method

Was the SUBSTRING method better?

Yes, but… not always!

Our final **COMBO METHOD**

1. Create clusters from passwords
2. For each cluster:
   - Select a representative using the LEVENSHTEIN method & generate rules accordingly
   - Select a representative using the SUBSTRING method & generate rules accordingly
3. The top *n* most frequent rules form the final ruleset

Other contributions of this work

More rule commands added!

- Toggle case
- Word rotation commands
- Word reversals

Rule-command priorities updated accordingly

RuleForge

- PoC implementation
- Password research & experiment tool
- Rule creation for an actual forensics use
- Open-source (MIT License): https://github.com/nesfit/RuleForge/

Alternate clustering methods

Overall, RuleForge support the following methods:

- Affinity Propagation (AP)
- Hierarchical Agglomerative Clustering (HAC)
- Density-based spatial clustering with noise (DBSCAN)
- Modified DBSCAN (MDBSCAN) by Li et al.

Experiments

- Benchmarking of clustering & rule creation
- Comparison of MDBSCAN implementations
- Comparison with alternate methods
- Comparison with popular rulesets

RuleForge

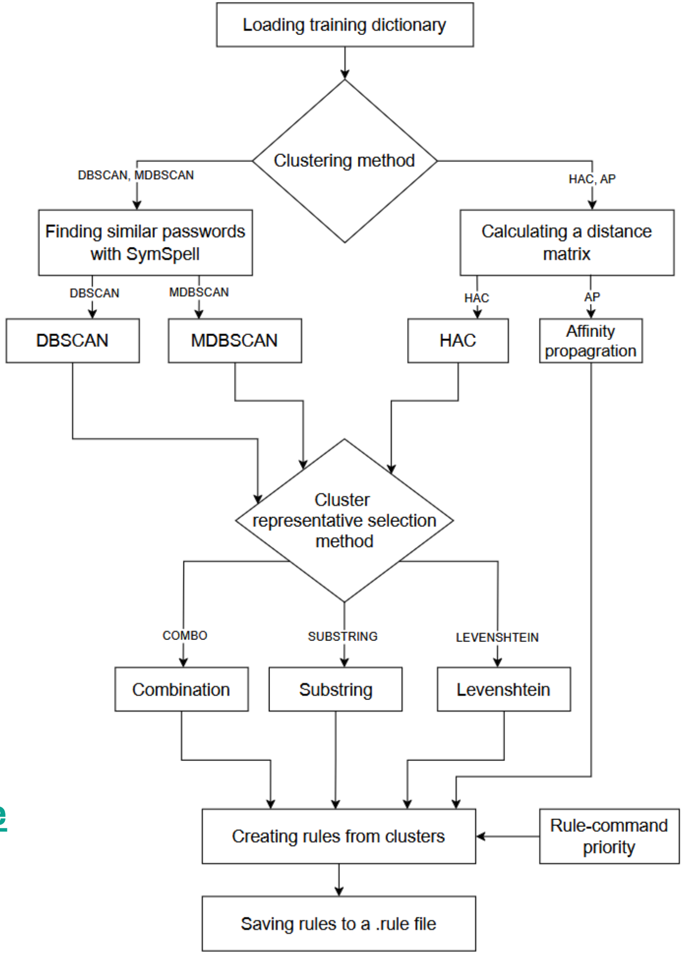
Features

4 clustering methods
3 representative selection methods
2 distance calculation methods
1 ruleset on the output

First Release

Python 3 + **C#** for critical calculations
Open-source (MIT License)

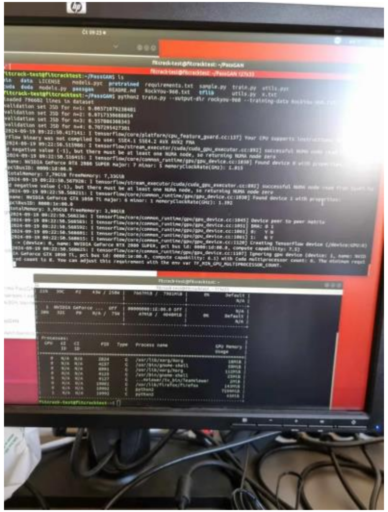
https://github.com/nesfit/RuleForge

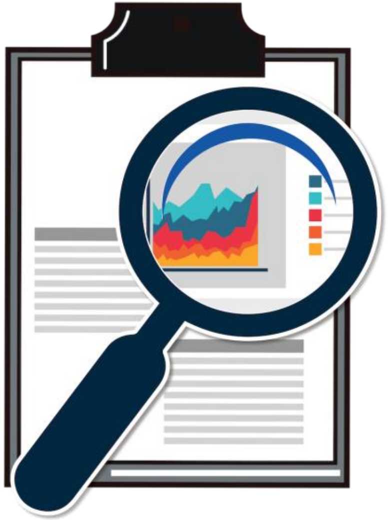


We did Benchmarks & Hit rate testing

Observations

- **MDBSCAN & AP** => best-quality rulesets
- **HAC & DBSCAN & MDBSCAN**
  => Lowest CPU requirements
- **DBSCAN & MDBSCAN + SymSpell**
  => Lowest memory requirements
- **DBSCAN** => sometimes suboptimal clustering due to a large cluster of outliers





Winner? MDBSCAN

Best Hitrate / overhead tradeoff

AMD Ryzen 5 2600X workstation with 64 GB RAM, cracking on dual RTX 3070

Representative selection comparison

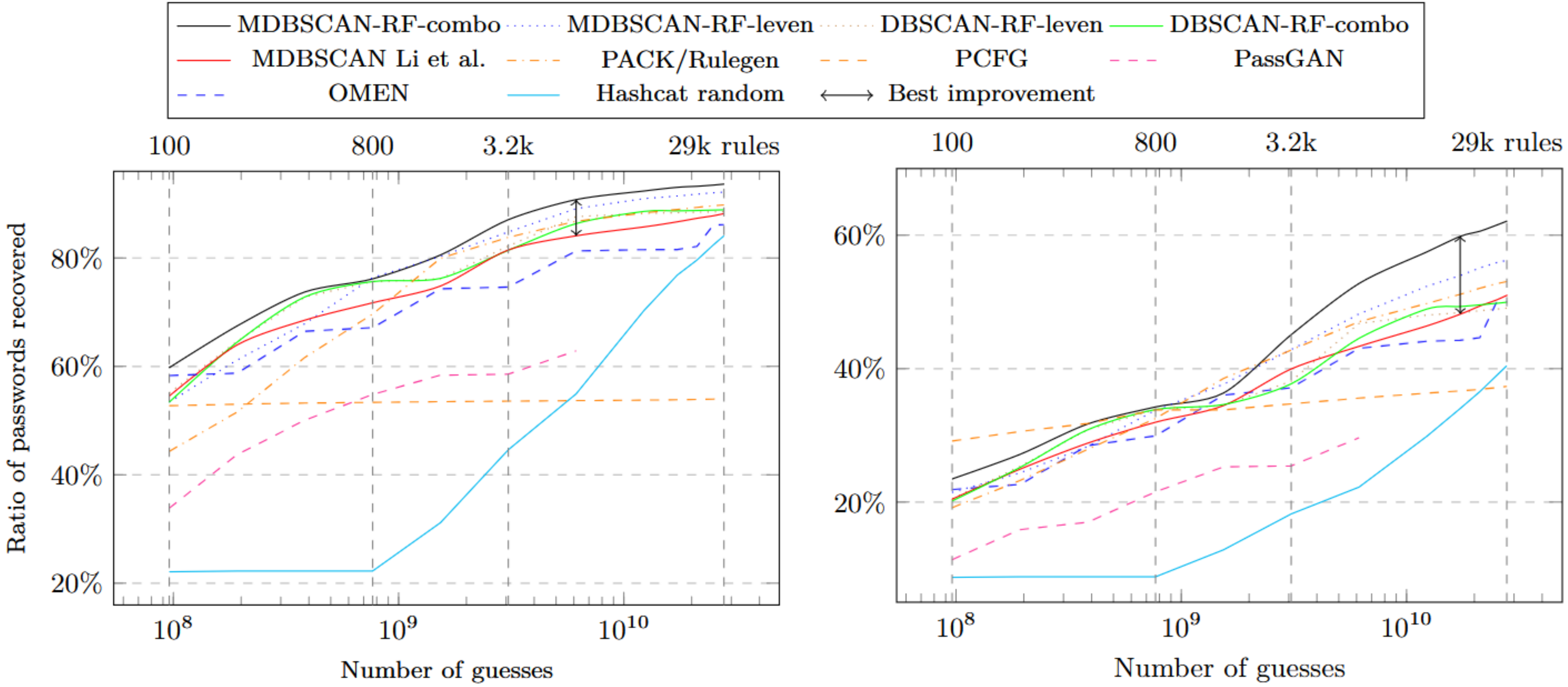
Rules		Hit ratio			
t^a	Method	pr	tm	en	dp
tl	Li et al.	52.44%	46.04%	18.55%	2.19%
	RF-leven	55.12%	51.45%	21.10%	2.53%
	RF-substr	53.42%	48.22%	22.34%	2.36%
	RF-combo	56.54%	51.56%	22.60%	2.60%
r65	Li et al.	55.14%	50.49%	19.41%	2.30%
	RF-leven	55.83%	51.70%	21.44%	2.50%
	RF-substr	53.65%	47.69%	23.76%	2.51%
	RF-combo	57.43%	53.23%	23.22%	2.66%
ms	Li et al.	51.19%	43.96%	17.26%	2.10%
	RF-leven	51.06%	44.41%	18.04%	2.06%
	RF-substr	52.76%	48.08%	20.12%	2.26%
	RF-combo	55.85%	50.15%	21.30%	2.43%
dw	Li et al.	52.49%	45.87%	18.42%	2.27%
	RF-leven	54.01%	49.84%	20.91%	2.58%
	RF-substr	50.99%	44.69%	20.48%	2.24%
	RF-combo	55.99%	52.05%	23.02%	2.72%

Legend

- **Li et al.** – The **original** MDBSCAN with the Levenshtein method
- **RF-leven** – RuleForge‘s implementation with expanded rule command set & **Levenshtein**
- **RF-substr** – RuleForge‘s implementation of MDBSCAN with the **Substring** method
- **RF-combo** – RuleForge‘s implementation of MDBSCAN with the **Combo** method

t^a – training dictionary | columns – wordlists on which rules were applied | target „hashlist“ dictionary: rockyou-75-m

Hit ratio: RuleForge vs. other methods

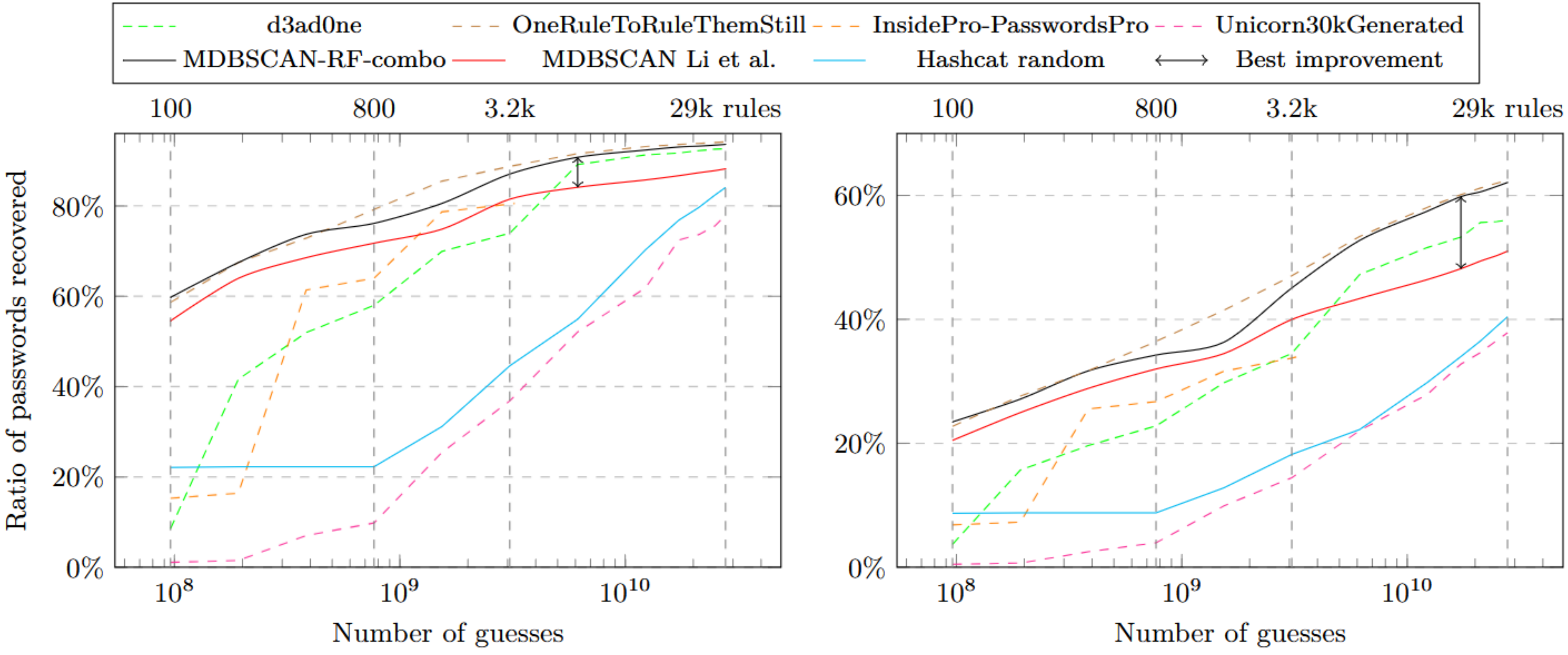


(a) Target: Xato-net-100k

(b) Target: phpbb-m

training AND attack dictionary: RockYou-960

Hit ratio: RuleForge vs. popular rulesets



(a) Target: Xato-net-100k

(b) Target: phpbb-m

training AND attack dictionary: RockYou-960

Summary

- **Clustering-based rule creation is usable** for password cracking
- **MDBSCAN** provides the best success/overhead tradeoff
- The **COMBO METHOD** outperformed the original work in all cases
- We achieved up to an **11.67 %pt.** improvement over known best-performing rule creation method (MDBSCAN Li et al.)
- **We outperformed almost all** widely-used rulesets.

Future work in progress

- Optimized Affinity Propagation and HAC
- GPU-accelerated version of RuleForge
- GenAI-based approaches (like PassGAN, PassGPT, VAEPass, ...)

Thank you for your attention!

Feel free to contact us!

Radek Hranický
hranicky@fit.vut.cz
Discord: radekhranicky

Lucia Šírová
xsirov01@stud.fit.vutbr.cz
Discord: sirrluc.

Viktor Rucký
rucky01@stud.fit.vutbr.cz
Discord: alpatron