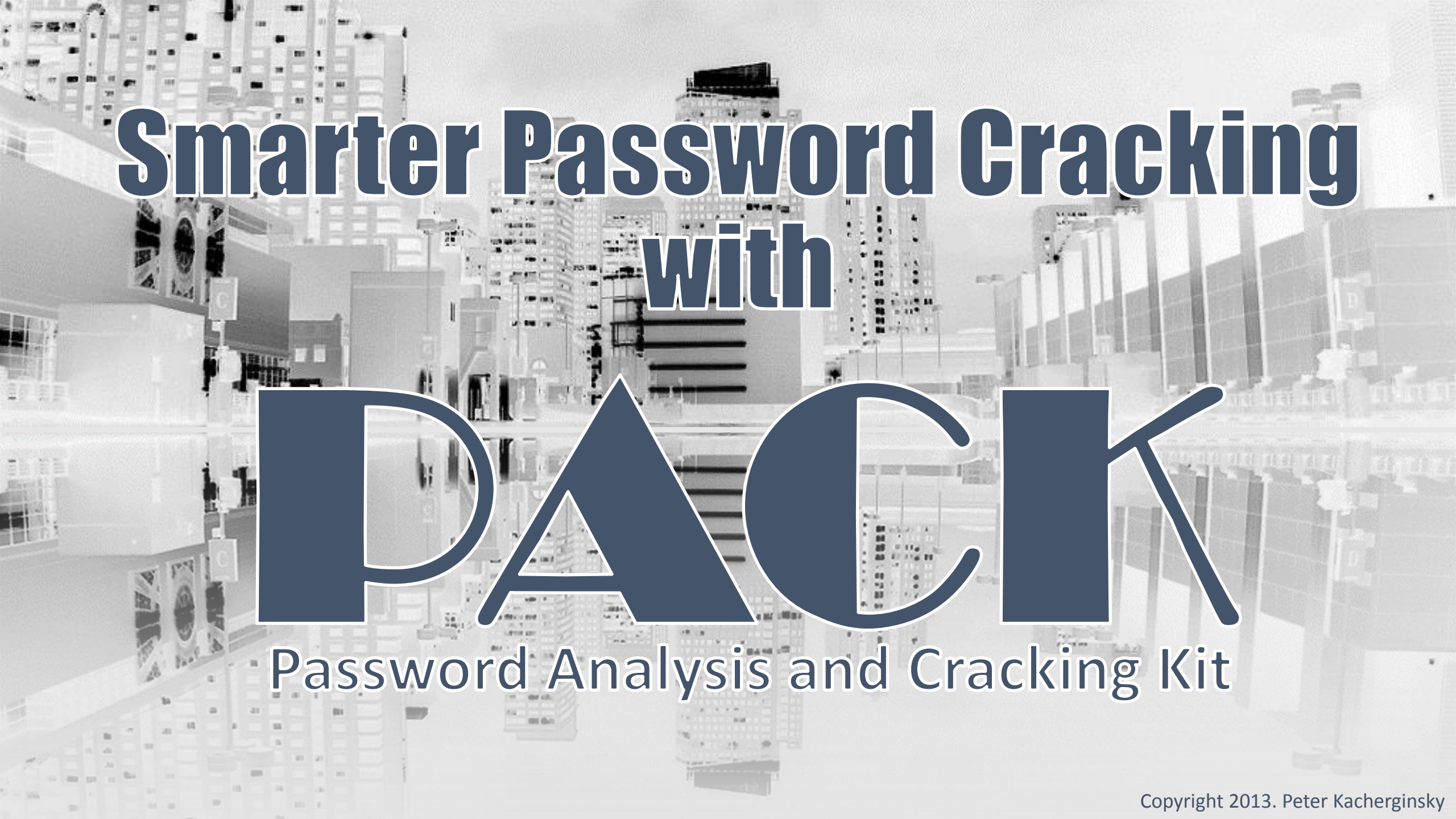
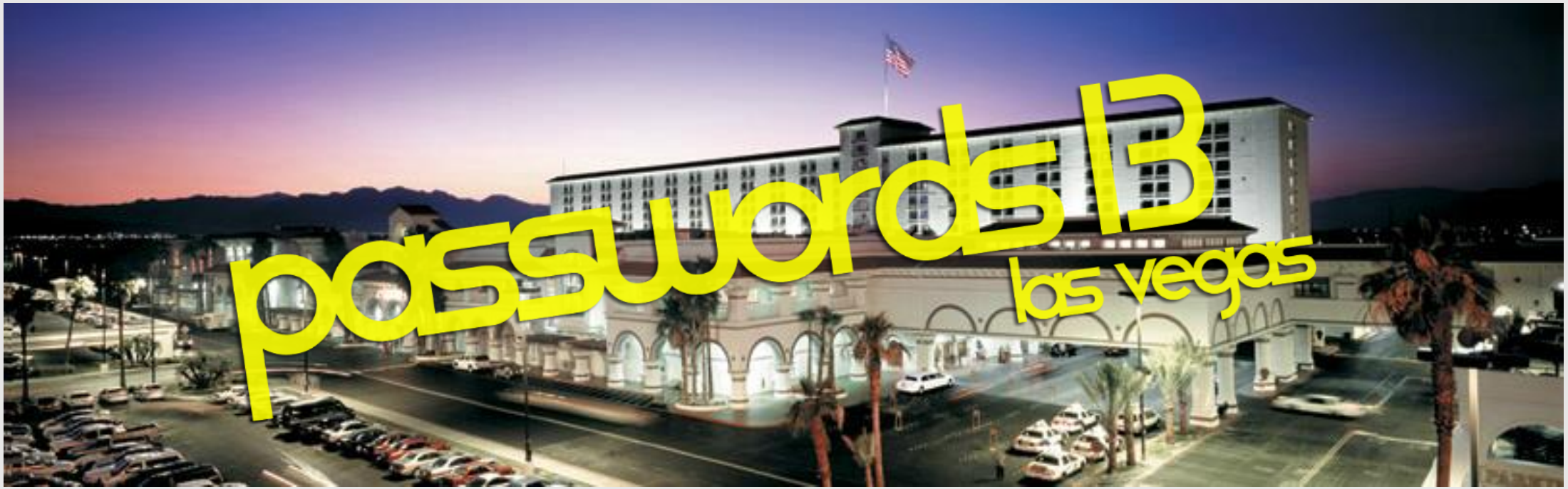


Welcome to PasswordsCon13



Speaker

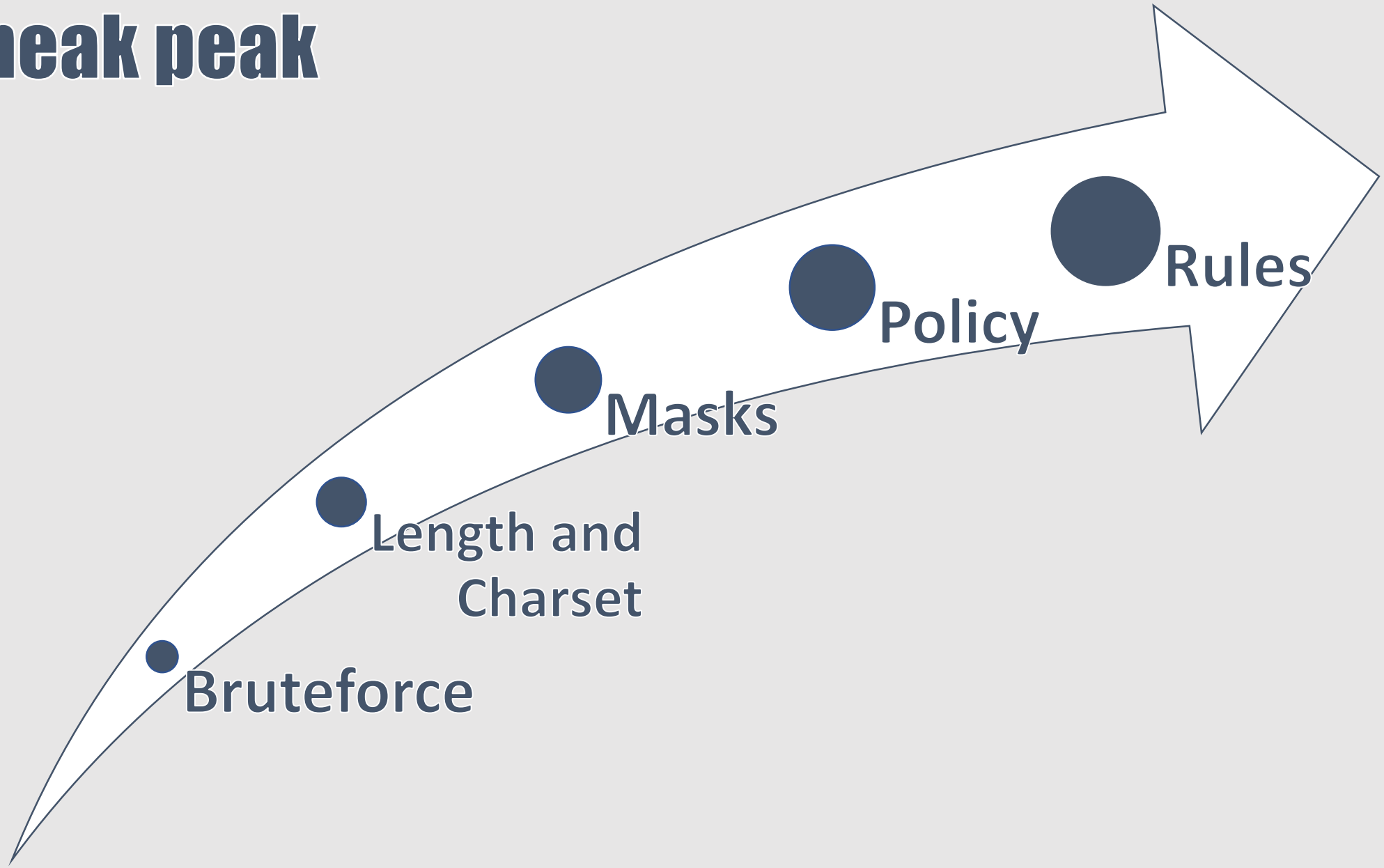
Peter Kacherginsky **@_iphelix**

http://thesprawl.org

Proud member of **Team Hashcat**

http://hashcat.net

Sneak peak



First attempt (CMIYC 2010 T-30 days)

+

+

MD5Decrypter.co.uk

Downloaded 2500 uncracked passwords

Expected results...

=

Brute-force attack

5.6 Billion Keys/Sec

Actual results...

=

24 Hours

300 passwords

Size is not everything...

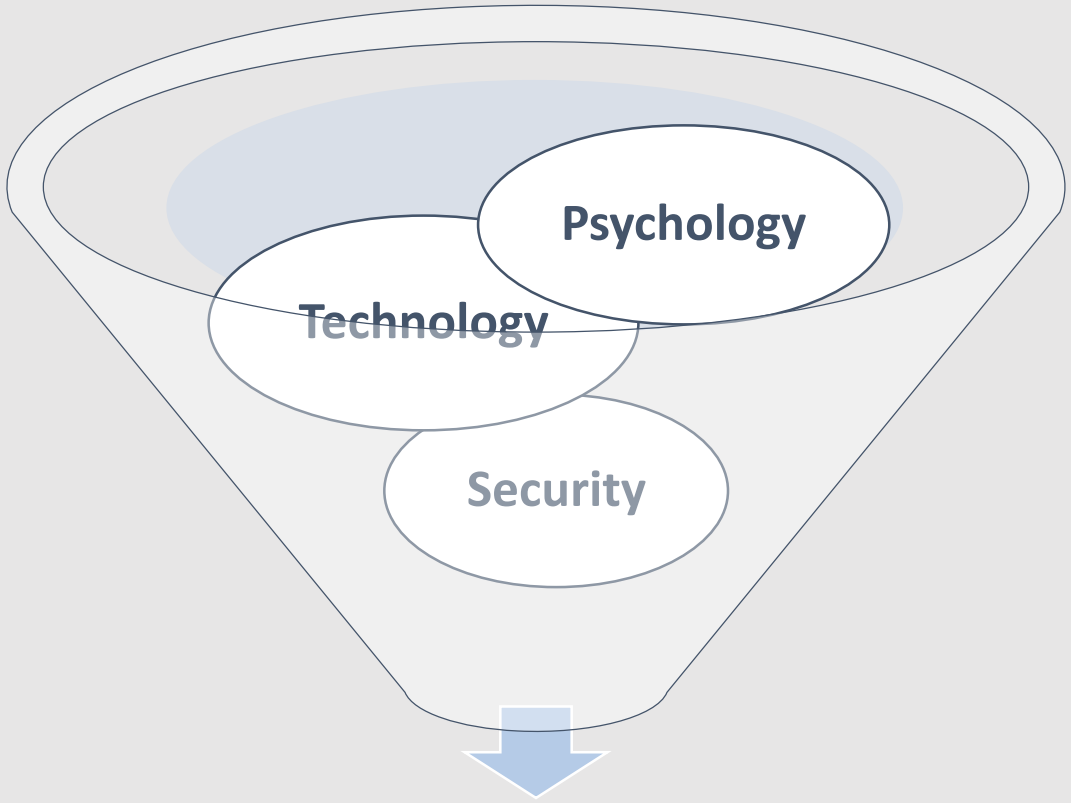
=

24 Hours

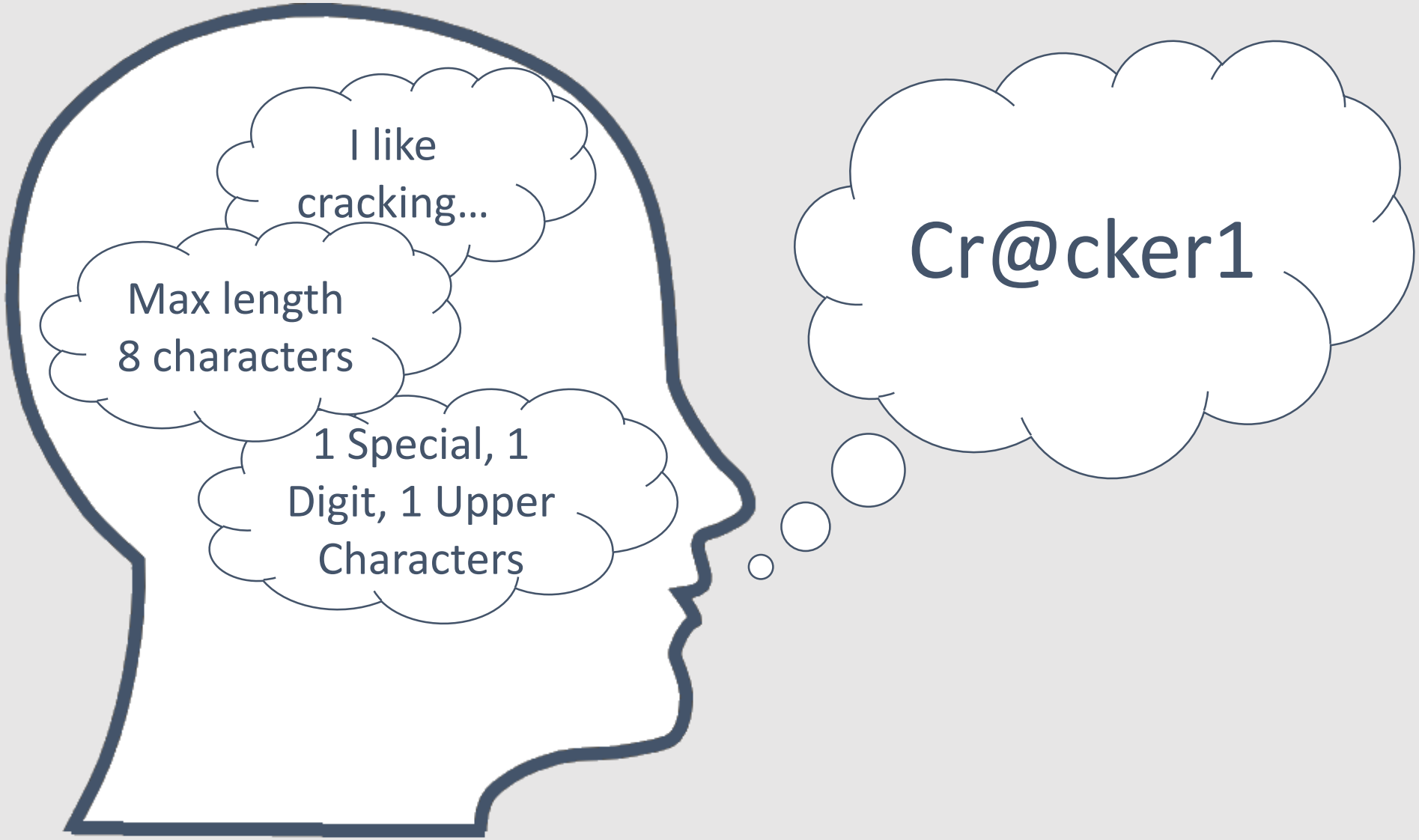
300 passwords

Smarter approach to password cracking

Password creation factors



Weak Password creation factors



Smarter password cracking…

… through **detection** and **analysis** of patterns in password creation factors to **reduce runtime** and **increase success rate**.

Password Analysis and Cracking Kit (PACK)

Password analysis tools

- Length and Character-Set
- Mask Patterns
- Policy Patterns
- Word Mangling Rules (new!)

Helps craft pattern-based attacks against password lists

Uses **Hashcat** masks and rules format.

https://thesprawl.org/projects/pack

http://github.com/iphelix/PACK

Selecting input for analysis

Source	Date	Count	Cracked	Notes
PhpBB**	January 2009	184,389	97%*	MD5 Encrypted
RockYou**	December 2009	14,344,391	100%	Clear-text
Gawker***	December 2010	1,084,394	92%*	DES Encrypted
Stratfor***	December 2011	804,041	93%*	MD5 Encrypted
LinkedIn***	June 2012	5,374,200	94%*	SHA1 Encrypted
eHarmony***	June 2012	1,475,738	97%*	MD5 Encrypted
Gamigo***	July 2012	6,306,186	90%*	MD5 Encrypted

Note:

* **Important:** all statistics will be generated relative to the percentage cracked

** http://www.skullsecurity.org/wiki/index.php/Passwords

*** http://www.adeptus-mechanicus.com/codex/hashpass/hashpass.php

Selecting hardware and tools for analysis

oclHashcat-plus

AMD Radeon 7970

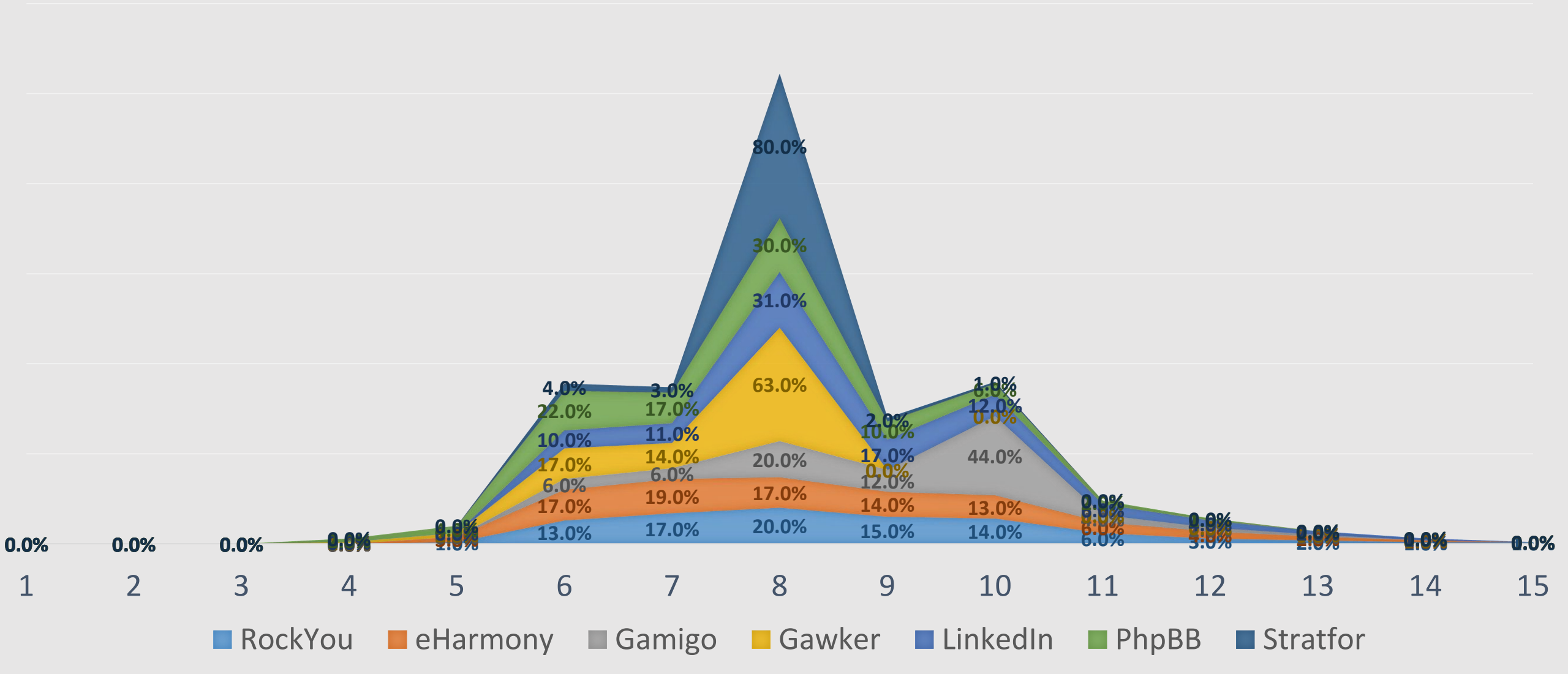
Hash Type	Performance*
NTLM	7501M c/s
MD5	5470M c/s
SHA1	2136M c/s
SHA256	1012M c/s
SHA512	76M c/s

* http://hashcat.net/oclhashcat-plus/
* http://golubev.com/gpuest.htm

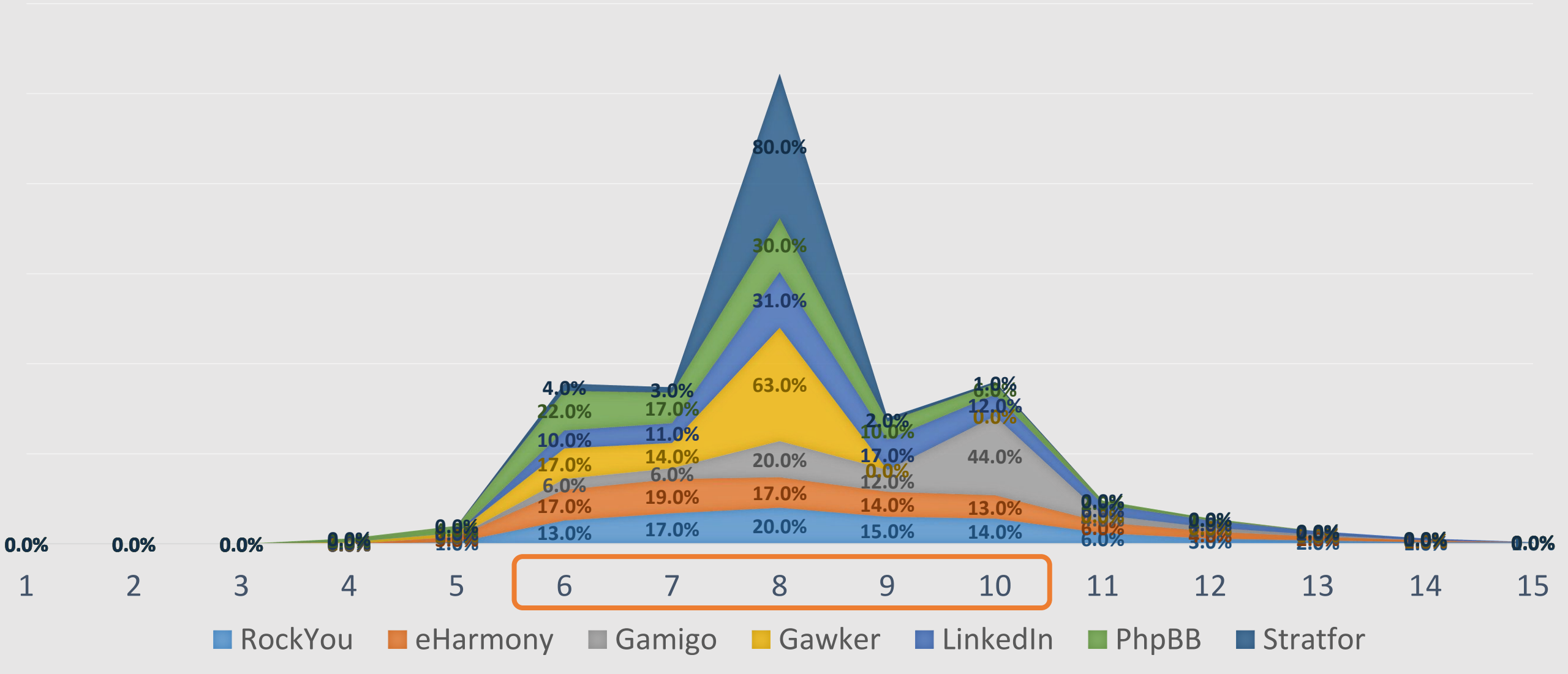
Password Length Analysis

```
$ python statsgen.py rockyou.txt
[*] Analyzing passwords: rockyou.txt
[+] Analyzing 100% (14344391/14344391) passwords

[*] Length Statistics...
[+]                  8: 20% (2966004)
[+]                  7: 17% (2506264)
[+]                  9: 15% (2191000)
[+]                 10: 14% (2013690)
[+]                  6: 13% (1947858)
[+]                 11: 06% (865973)
[+]                 12: 03% (555333)
[+]                 13: 02% (364169)
[+]                  5: 01% (259174)
[+]                 14: 01% (248514)
[+]                 15: 01% (161181)
...
```

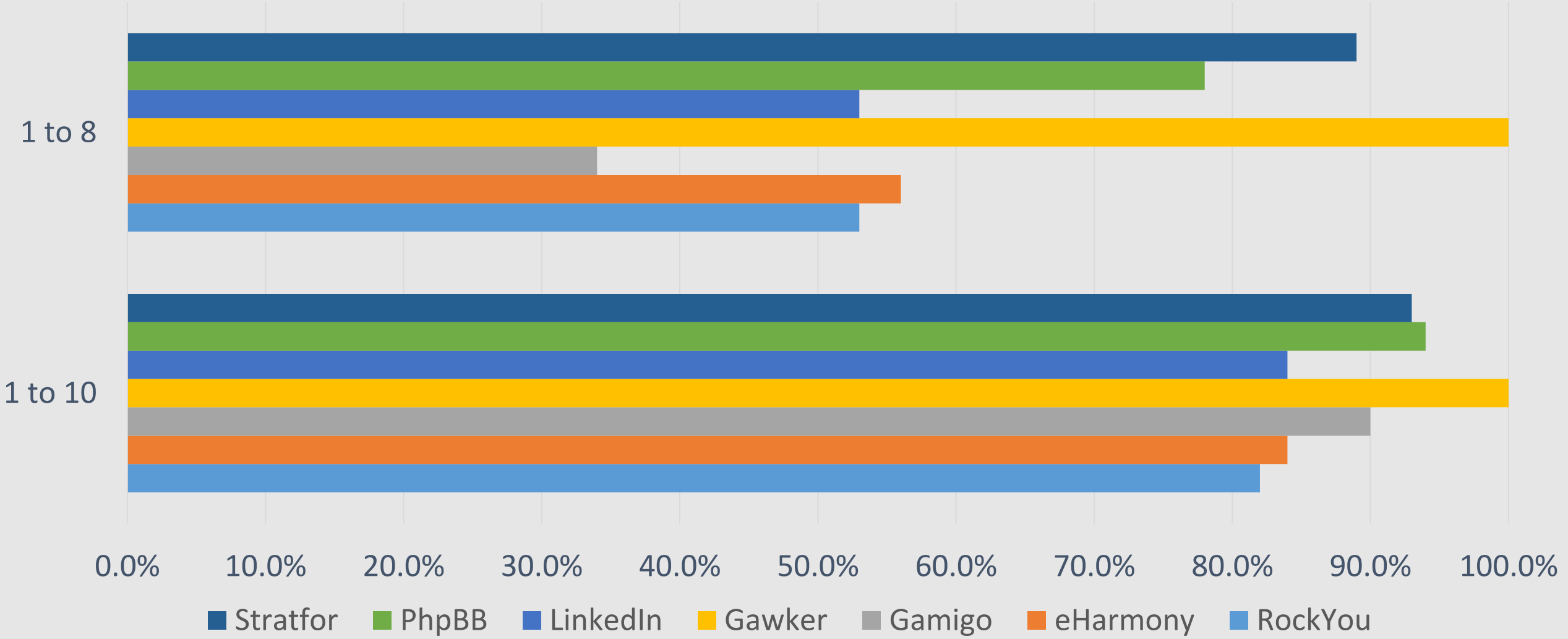
Password Length Statistics



Password Length Patterns



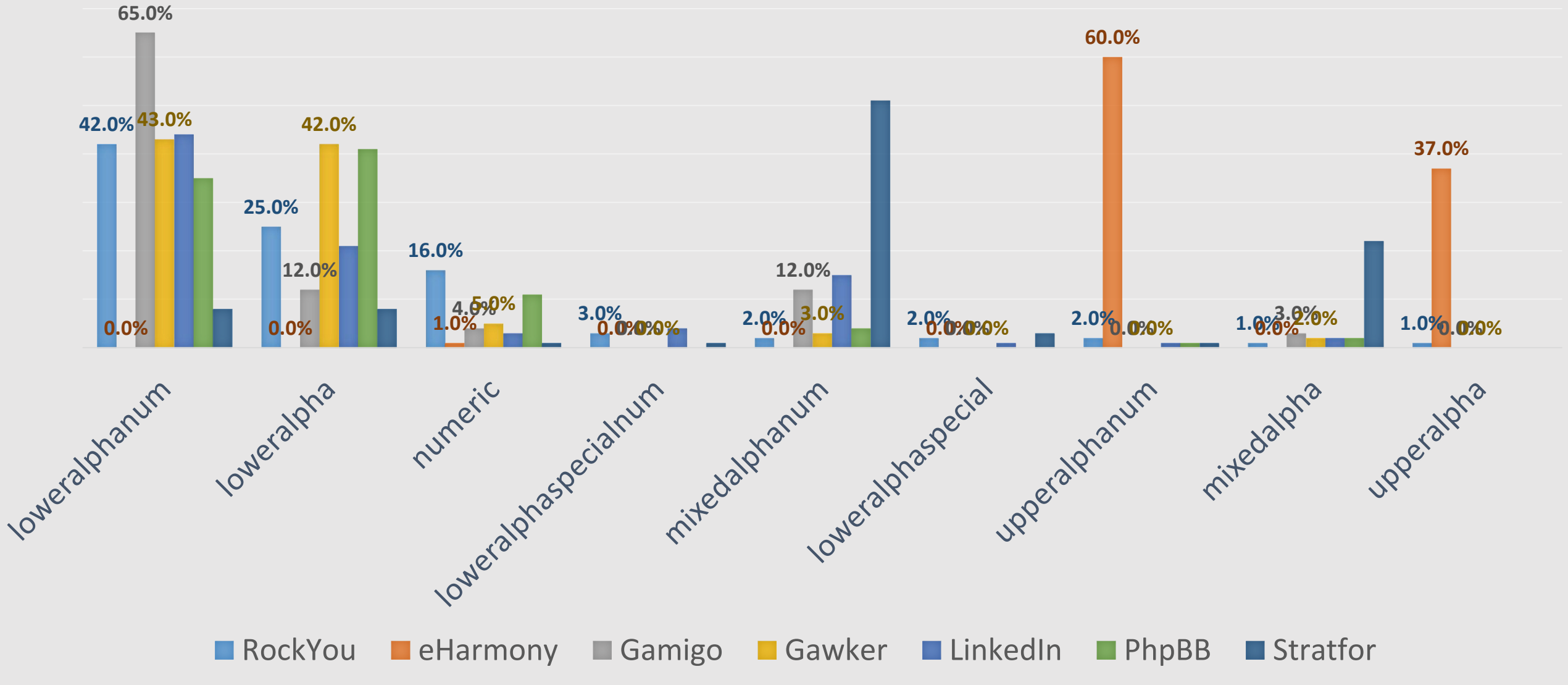
Password Length Patterns Brute-Force Attack



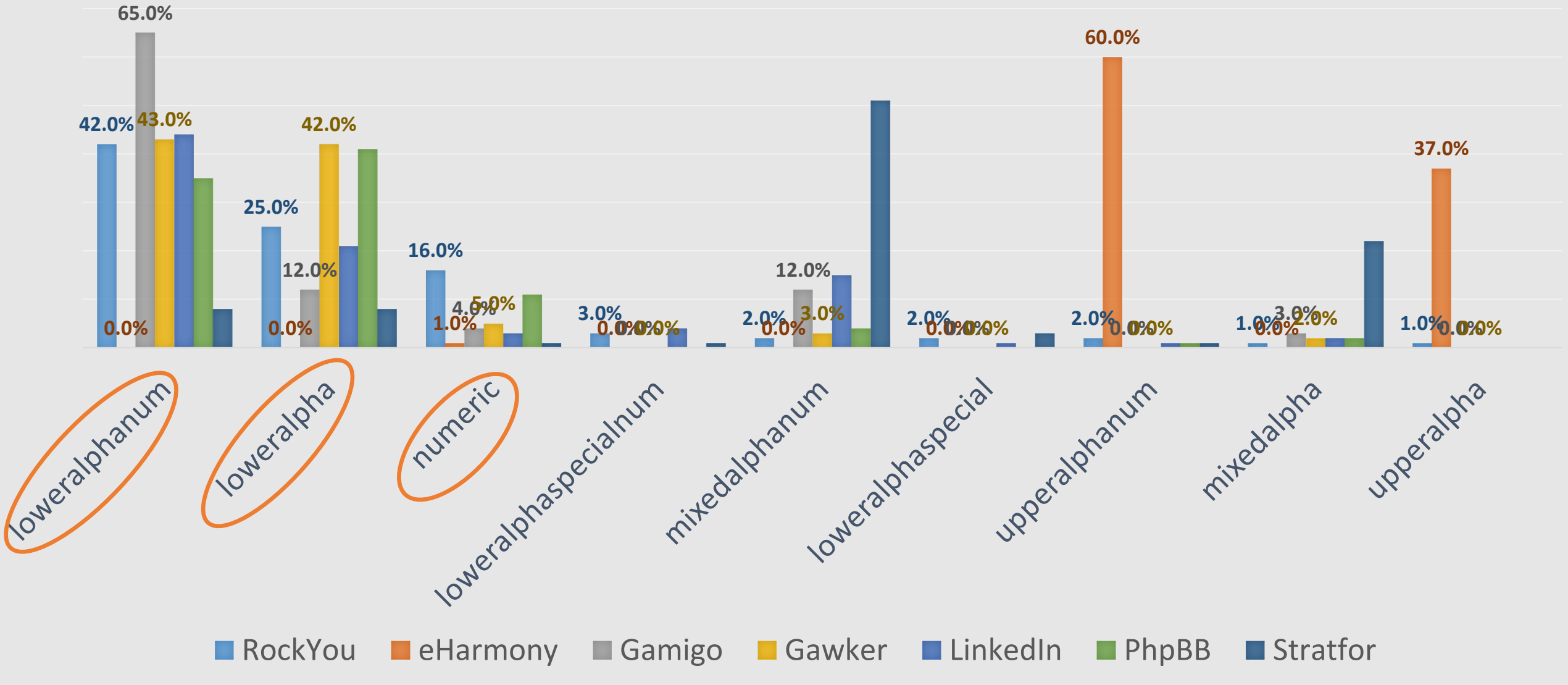
Password Character-Set Analysis

```
$ python statsgen.py rockyou.txt
[*] Analyzing passwords: rockyou.txt
[+] Analyzing 100% (14344391/14344391) passwords
...
[*] Charset statistics...
[+]             loweralphanum: 42% (6075055)
[+]                loweralpha: 25% (3726656)
[+]                   numeric: 16% (2346842)
[+]      loweralphaspecialnum: 03% (472673)
[+]             upperalphanum: 02% (407436)
[+]             mixedalphanum: 02% (382246)
[+]        loweralphaspecial: 02% (381095)
[+]                upperalpha: 01% (229893)
[+]                mixedalpha: 01% (159332)
[+]      mixedalphaspecialnum: 00% (53240)
[+]         mixedalphaspecial: 00% (49633)
[+]      upperalphaspecialnum: 00% (27732)
[+]         upperalphaspecial: 00% (26795)
[+]                   special: 00% (5763)
...
```

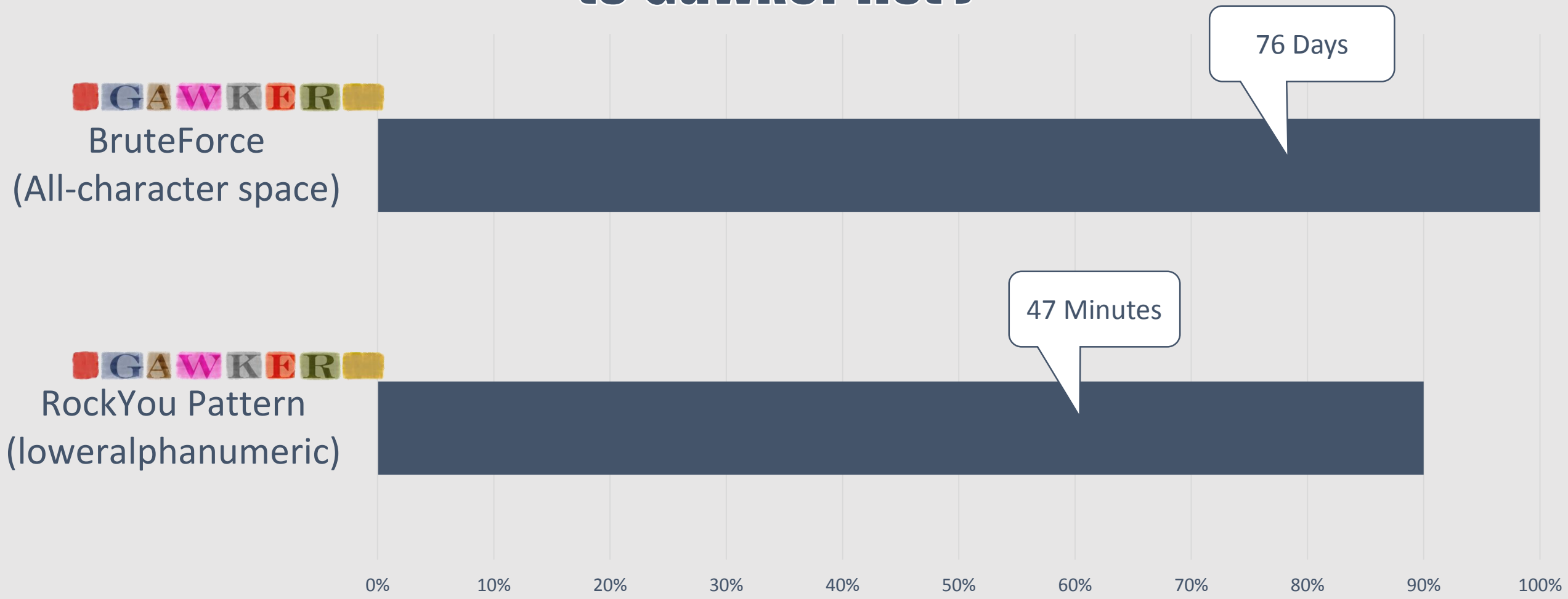
Password Character-Set Statistics



Password Character-Set Patterns



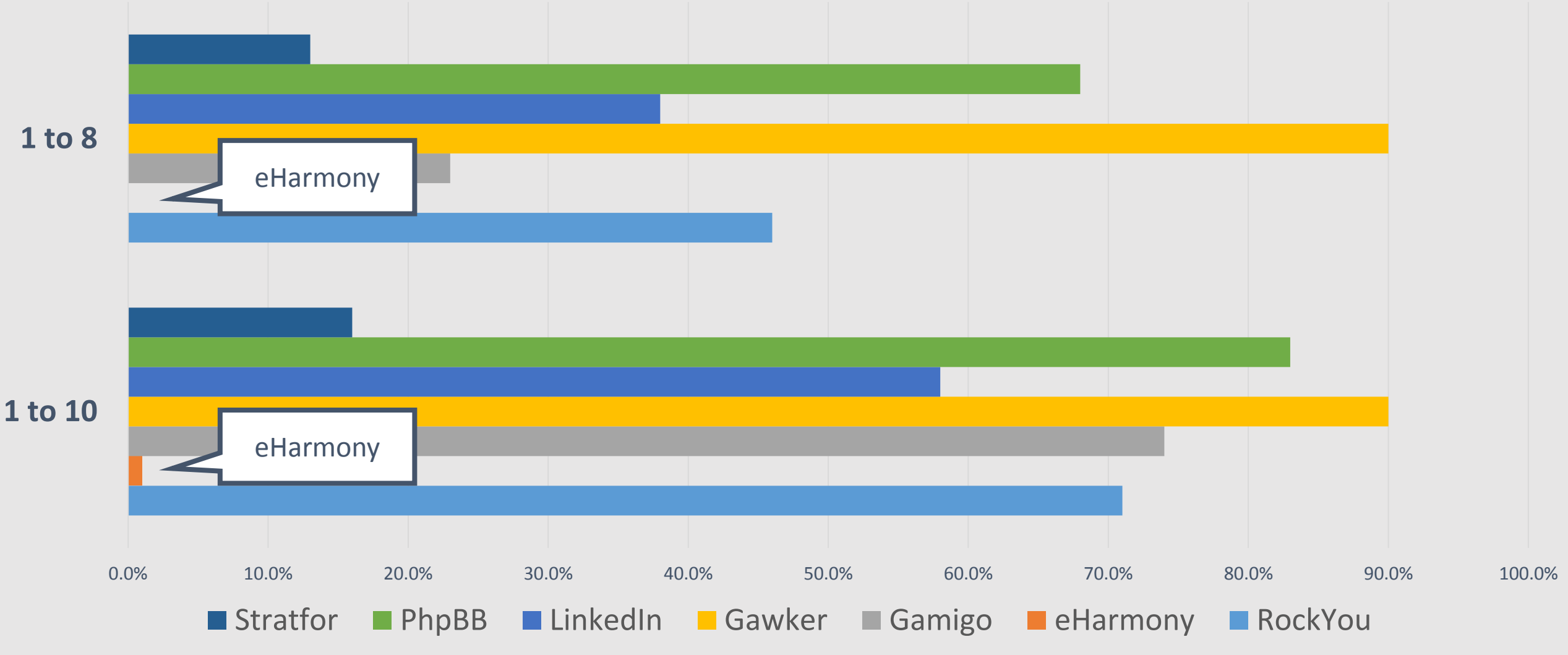
Password Character-Set Patterns Brute-Force (applying RockYou loweralphanumeric pattern to Gawker list)



```
$ python statsgen.py --maxlength=8
--charset="loweralphanum,numeric,loweralpha” gawker.txt

[+] Analyzing 90% (986425/1084394) of passwords
    NOTE: Statistics below is relative to the number of analyzed passwords, not
total number of passwords

[*] Length Statistics...
[+]                          8: 62% (612522)
[+]                          6: 18% (183307)
[+]                          7: 14% (146152)
[+]                          5: 02% (26438)
[+]                          4: 01% (15088)
[+]                          3: 00% (2497)
[+]                          2: 00% (308)
[+]                          1: 00% (113)

[*] Charset statistics...
[+]              loweralphanum: 47% (470580)
[+]                 loweralpha: 46% (459208)
[+]                    numeric: 05% (56637)
```

Analysis filters

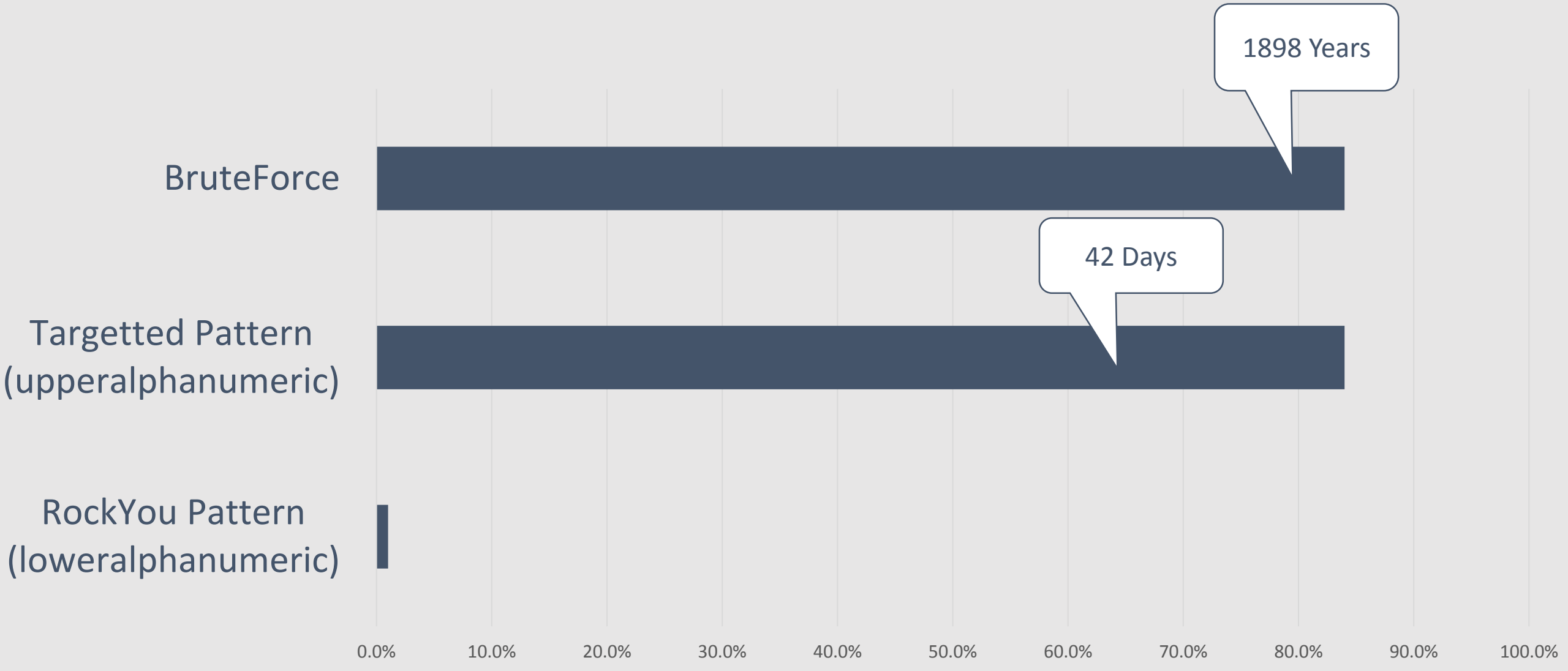
Password Character-Set Patterns Brute-Force (applying RockYou loweralphanum pattern)



eHarmony targeted attack (1-6 words)

```
[*] Length Statistics...
[+]                          6: 84% (254004)
[+]                          5: 15% (46821)

[*] Charset statistics...
[+]           upperalphanum: 57% (173459)
[+]               upperalpha: 42% (126954)
[+]                  numeric: 00% (187)
[+]      upperalphaspecialnum: 00% (118)
[+]         upperalphaspecial: 00% (101)
[+]      loweralphaspecialnum: 00% (5)
[+]                  special: 00% (1)
```

Applying targeted charset pattern to eHarmony (1 to 10 character passwords)

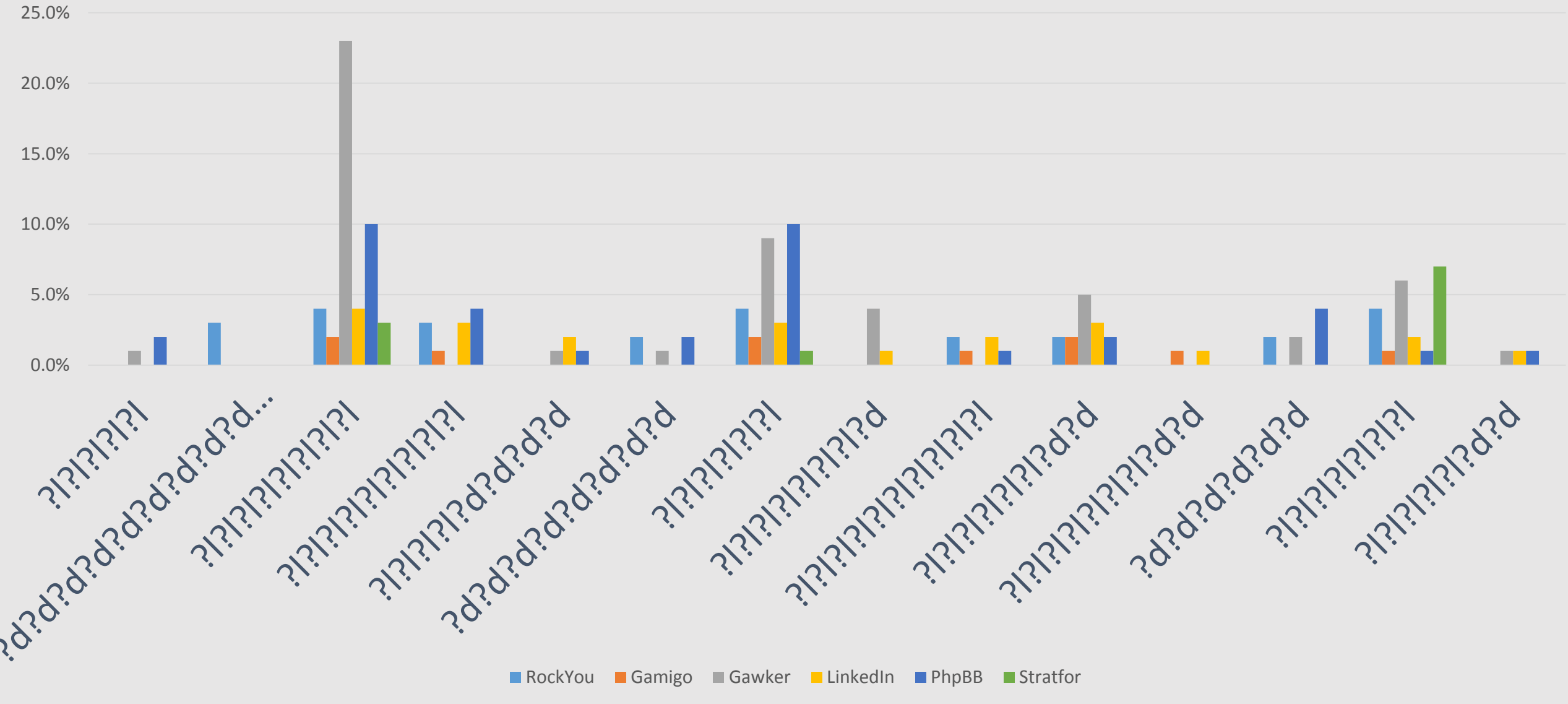


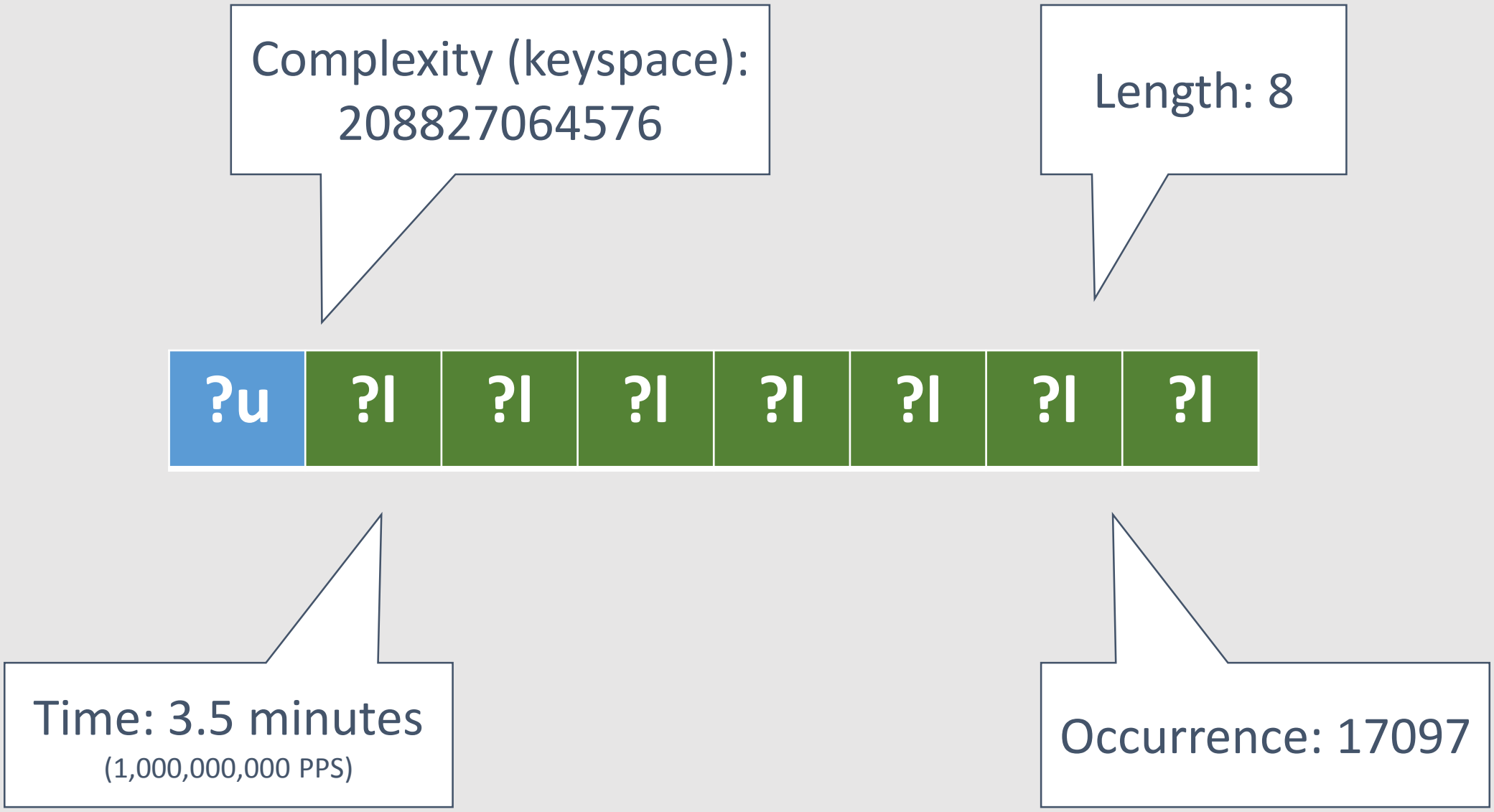
Password Mask Analysis

P	a	s	s	w	o	r	d	1	!
?u	?l	?l	?l	?l	?l	?l	?l	?d	?s

```
$ python statsgen.py rockyou.txt -o rockyou.masks
...
[*] Advanced Mask statistics...
[+]          ?l?l?l?l?l?l?l?l: 04% (688053)
[+]             ?l?l?l?l?l?l: 04% (601257)
[+]           ?l?l?l?l?l?l?l: 04% (585093)
[+]         ?l?l?l?l?l?l?l?l?l: 03% (516862)
[+]             ?d?d?d?d?d?d?d: 03% (487437)
[+]       ?d?d?d?d?d?d?d?d?d?d: 03% (478224)
[+]          ?d?d?d?d?d?d?d?d: 02% (428306)
[+]          ?l?l?l?l?l?l?d?d: 02% (420326)
[+]       ?l?l?l?l?l?l?l?l?l?l: 02% (416961)
[+]               ?d?d?d?d?d?d: 02% (390546)
[+]         ?d?d?d?d?d?d?d?d?d: 02% (307540)
[+]             ?l?l?l?l?l?d?d: 02% (292318)
[+]         ?l?l?l?l?l?l?l?d?d: 01% (273640)
...
```

Password Masks Analysis (Failed attempt)




Complexity (keyspace):
208827064576
Length: 8
?u
?l
?l
?l
?l
?l
?l
?l
Time: 3.5 minutes
(1,000,000,000 PPS)
Occurrence: 17097

```
$ python maskgen.py rockyou.masks -t 86400 --showmasks --optindex -q
[*] Analyzing masks in [rockyou.masks]
[*] Using 1,000,000,000 keys/sec for calculations.
[*] Sorting masks by their [optindex].
[L:] Mask:                              [ Occ:  ] [ Time:  ]
...
[ 8] ?u?u?d?d?d?d?s?s                   [30     ] [ 0:00:07]
[ 9] ?l?l?l?l?l?l?d?d?s                 [4149   ] [ 0:16:59]
[ 8] ?s?d?d?l?l?l?l?l                   [159    ] [ 0:00:39]
[ 9] ?l?l?l?l?d?d?d?d?l                 [480    ] [ 0:01:58]
[ 8] ?d?d?d?d?s?d?l?l                   [9      ] [ 0:00:02]
[ 8] ?l?d?d?d?d?d?l?s                   [9      ] [ 0:00:02]
[ 9] ?l?l?d?d?l?l?l?d?d                 [478    ] [ 0:01:58]
[ 8] ?l?l?l?l?s?d?d?s                   [200    ] [ 0:00:49]
...
[!] Target time exceeded.
[*] Finished generating masks:
    Masks generated: 5285
    Masks coverage:  75% (10794350/14344380)
    Masks runtime:   1 day, 0:32:08
```

```
$ python maskgen.py rockyou.masks -t 86400 --showmasks --optindex -q
[*] Analyzing masks in [rockyou.masks]
[*] Using 1,000,000,000 keys/sec for calculations.
[*] Sorting masks by their [optindex].
[L:] Mask:                              [ Occ:   ]
...
[ 8] ?u?u?d?d?d?d?s?s                   [30      ]
[ 9] ?l?l?l?l?l?l?d?d?s                 [4149    ] [ 0:16:59]
[ 8] ?s?d?d?l?l?l?l?l                   [159     ] [ 0:00:39]
[ 9] ?l?l?l?l?d?d?d?d?l                 [480     ] [ 0:01:58]
[ 8] ?d?d?d?d?s?d?l?l                   [9       ] [ 0:00:02]
[ 8] ?l?d?d?d?d?d?l?s                   [9       ] [ 0:00:02]
[ 9] ?l?l?d?d?l?l?l?d?d                 [478     ] [ 0:01:58]
[ 8] ?l?l?l?l?s?d?d?s                   [200     ] [ 0:00:49]
...
[!] Target time exceeded.
[*] Finished generating masks:
    Masks generated: 5285
    Masks coverage:  75% (10794350/14344380)
    Masks runtime:   1 day, 0:32:08
```

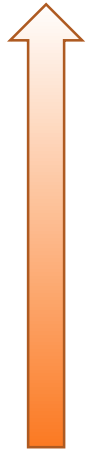
Masks sorting mode

OptIndex = Complexity/Occurrence

```
$ python maskgen.py rockyou.masks -t 86400 --showmasks --optindex -q
[*] Analyzing masks in [rockyou.masks]
[*] Using 1,000,000,000 keys/sec for calculations.
[*] Sorting masks by their [optindex].
[L:] Mask:                              [ Occ:  ] [ Time:  ]
...
[ 8] ?u?u?d?d?d?d?s?s                   [30     ] [ 0:00:07]
[ 9] ?l?l?l?l?l?l?d?d?s                 [4149   ] [ 0:16:59]
[ 8] ?s?d?d?l?l?l?l?l                   [159    ] [ 0:00:39]
[ 9] ?l?l?l?l?d?d?d?d?l                 [480    ] [ 0:01:58]
[ 8] ?d?d?d?d?s?d?l?l                   [9      ] [ 0:00:02]
[ 8] ?l?d?d?d?d?d?l?s                   [9      ] [ 0:00:02]
[ 9] ?l?l?d?d?l?l?l?d?d                 [478    ] [ 0:01:58]
[ 8] ?l?l?l?l?s?d?d?s                   [200    ] [ 0:00:49]
...
[!] Target time exceeded.
[*] Finished generating masks:
    Masks generated: 5285
    Masks coverage:  75% (10794350/14344380)
    Masks runtime:   1 day, 0:32:08
```

Target runtime (seconds)

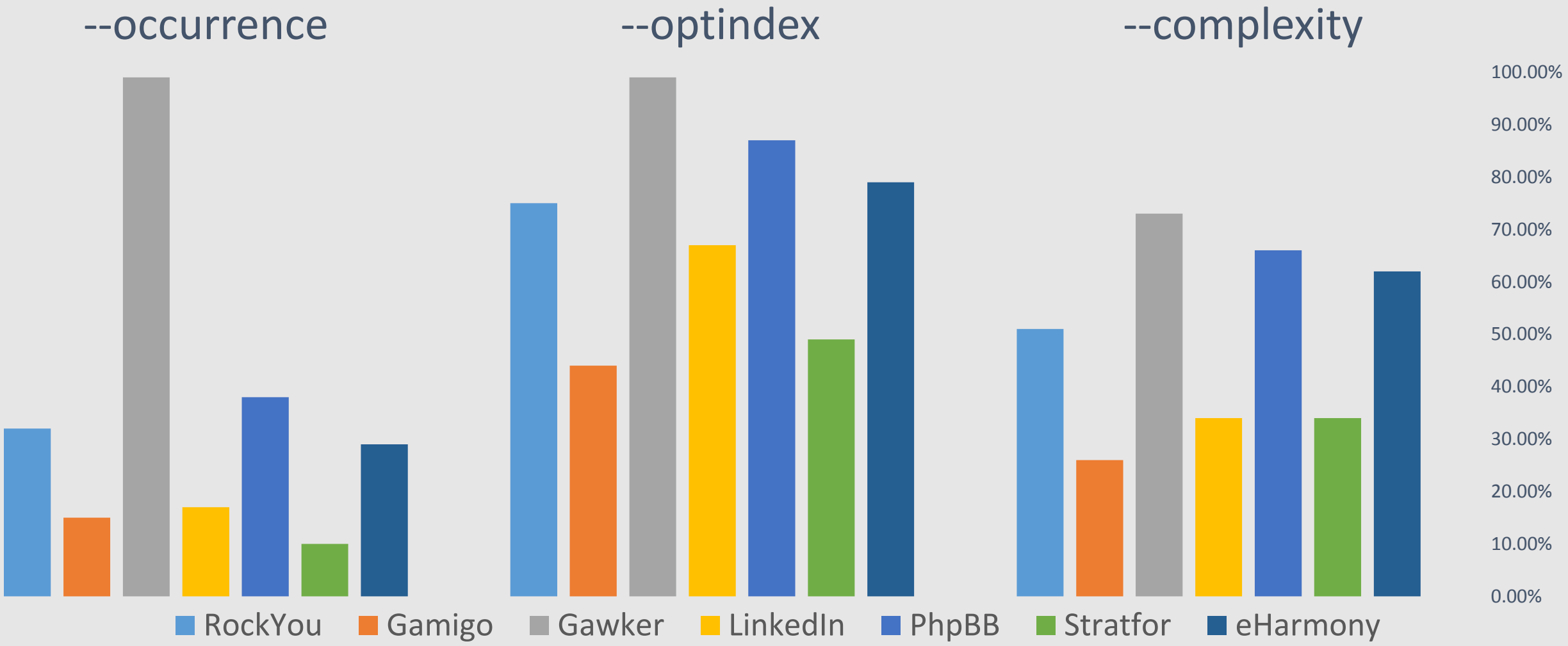
```
$ python maskgen.py rockyou.masks -t 86400 --showmasks --optindex -q
[*] Analyzing masks in [rockyou.masks]
[*] Using 1,000,000,000 keys/sec for calculations.
[*] Sorting masks by their [optindex].
[L:] Mask:                              [ Occ:   ] [ Time:   ]
...
[ 8] ?u?u?d?d?d?d?s?s                   [30      ] [ 0:00:07]
[ 9] ?l?l?l?l?l?l?d?d?s                 [4149    ] [ 0:16:59]
[ 8] ?s?d?d?l?l?l?l?l                   [159     ] [ 0:00:39]
[ 9] ?l?l?l?l?d?d?d?d?l                 [480     ] [ 0:01:58]
[ 8] ?d?d?d?d?s?d?l?l                   [9       ] [ 0:00:02]
[ 8] ?l?d?d?d?d?d?l?s                   [9       ] [ 0:00:02]
[ 9] ?l?l?d?d?l?l?l?d?d                 [478     ] [ 0:01:58]
[ 8] ?l?l?l?l?s?d?d?s                   [200     ] [ 0:00:49]
...
[!] Target time exceeded.
[*] Finished generating masks:
    Masks generated: 5285
    Masks coverage:  75% (10794350/14344380)
    Masks runtime:   1 day, 0:32:08
```



Sorted Masks (higher is better)

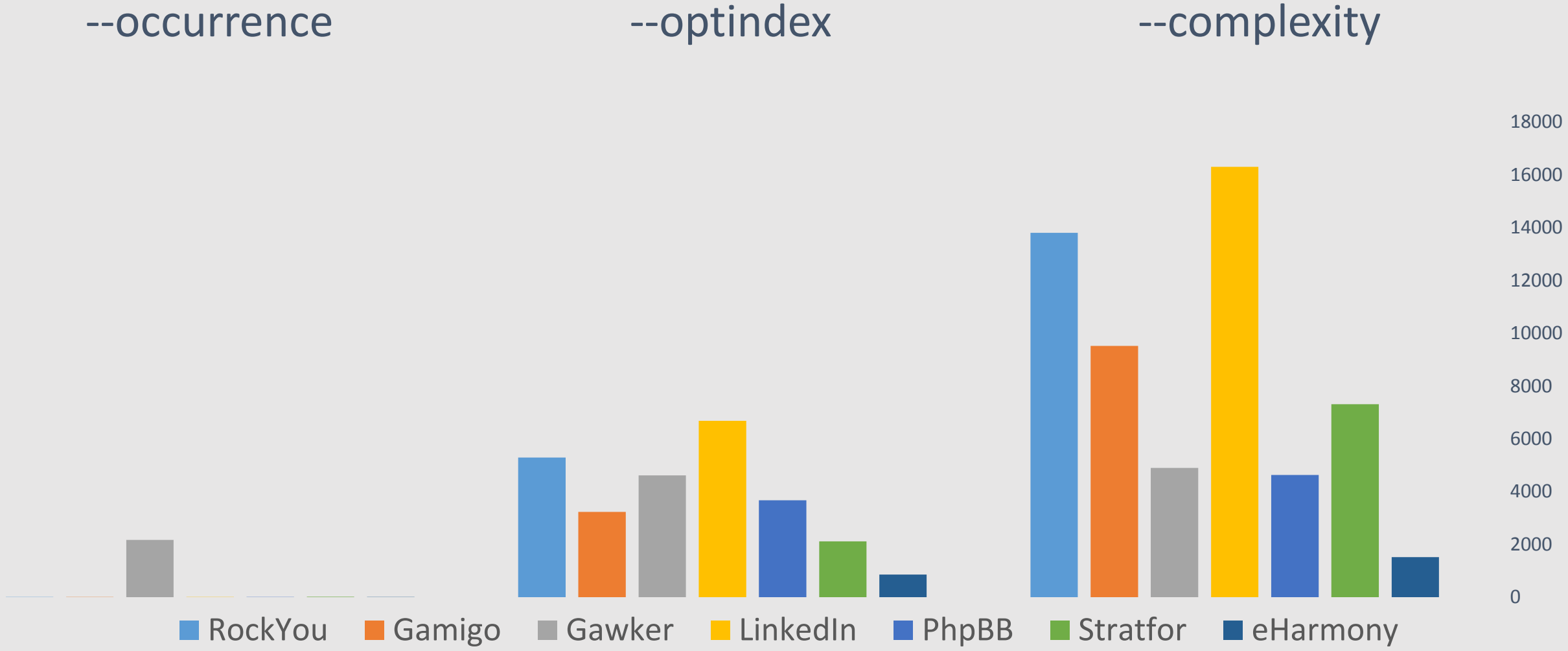
Masks coverage and total runtime

Coverage comparison (1 day* target runtime)



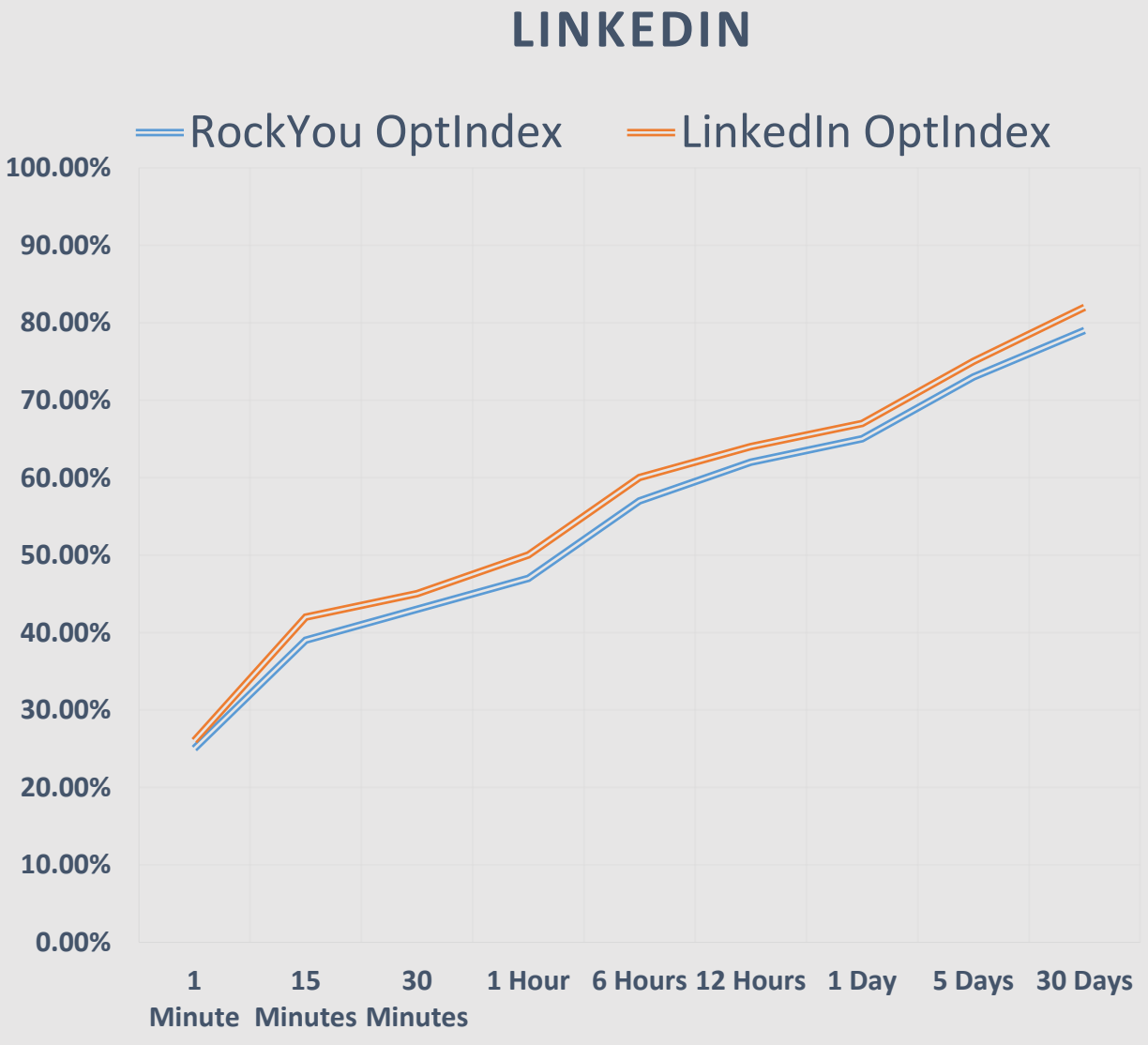
* 1,000,000,000 keys/sec

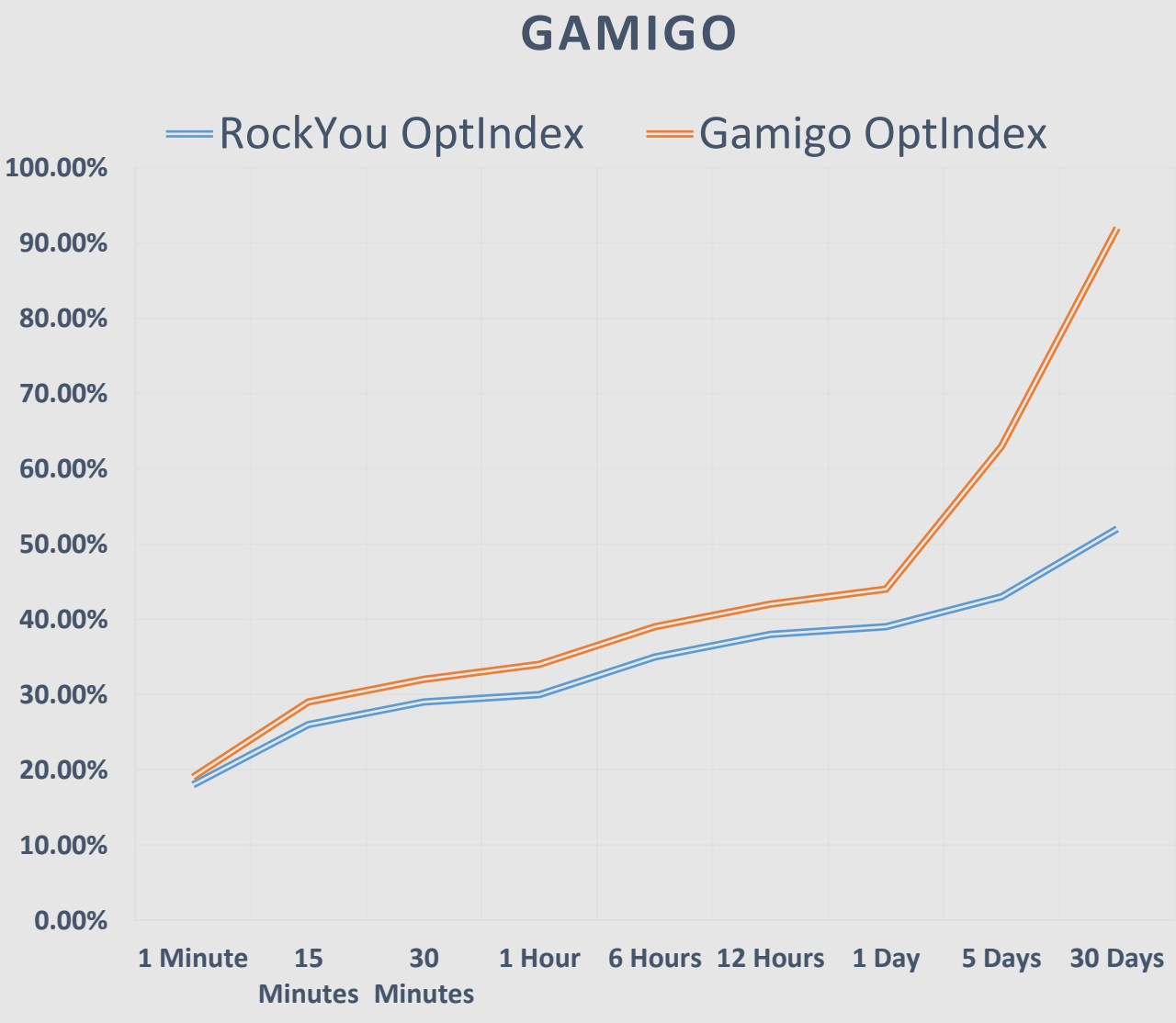
Masks generated (1 day* target runtime)



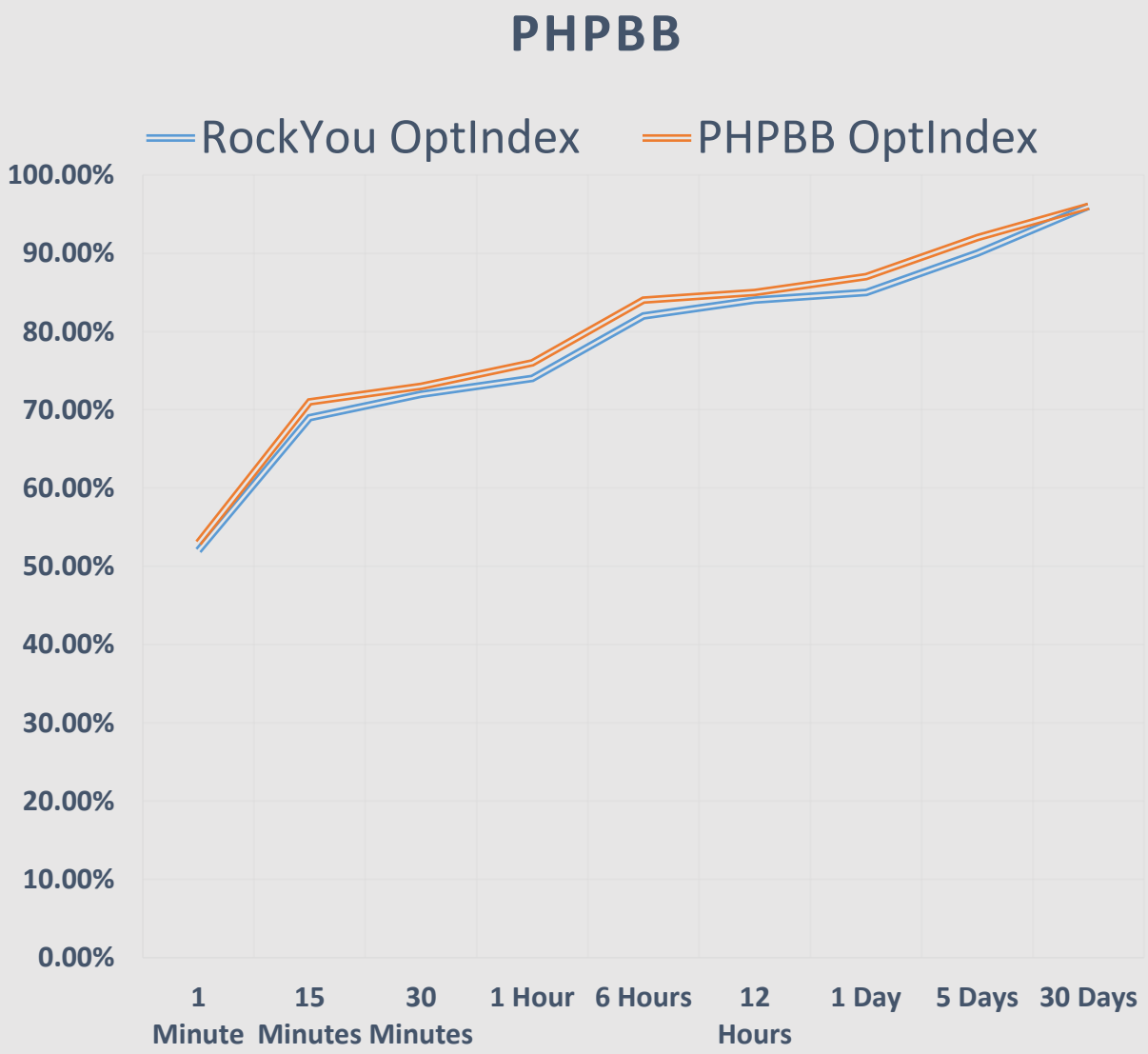
* 1,000,000,000 keys/sec

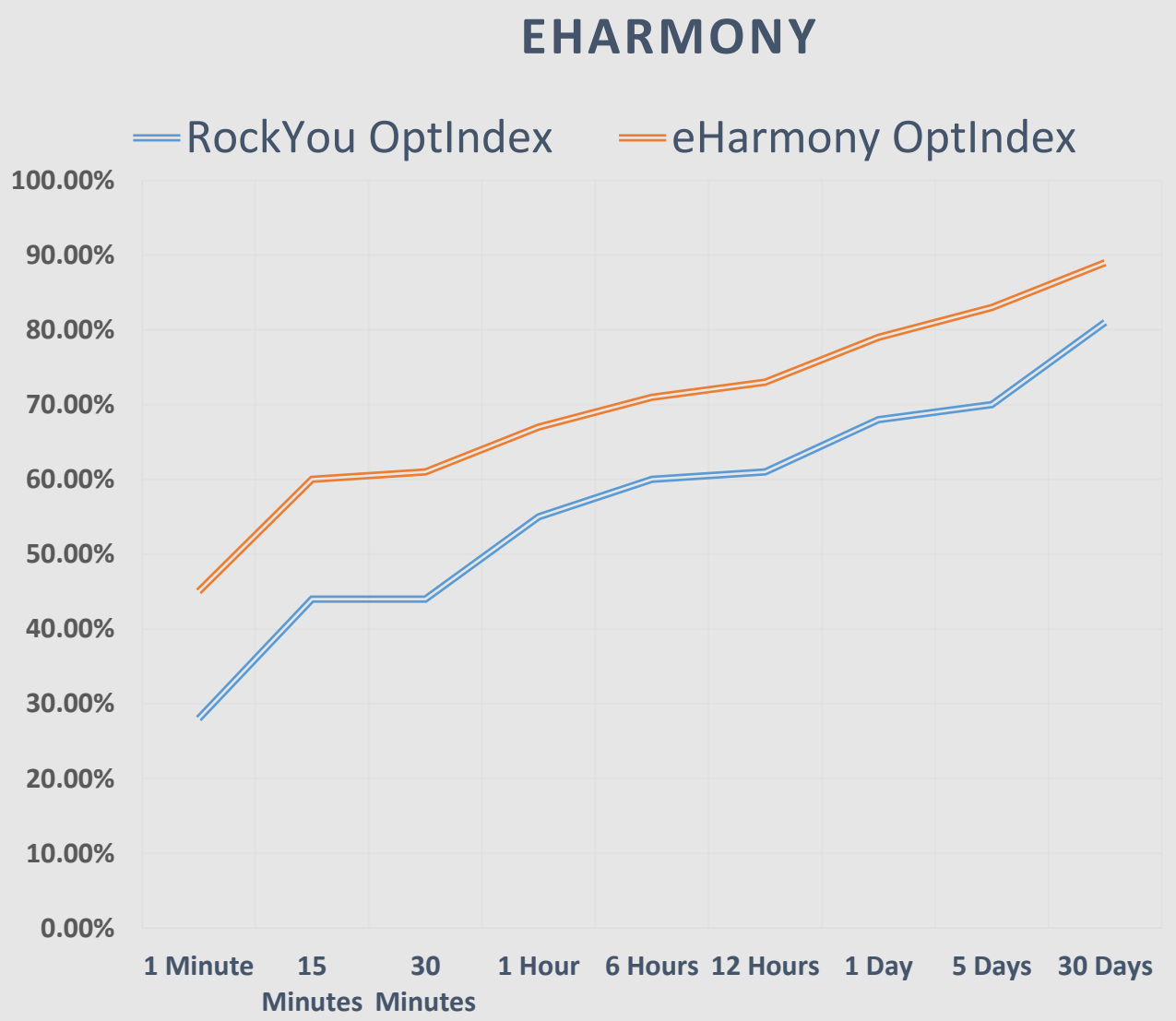
RockYou OptIndex Masks vs LinkedIn and Gamigo



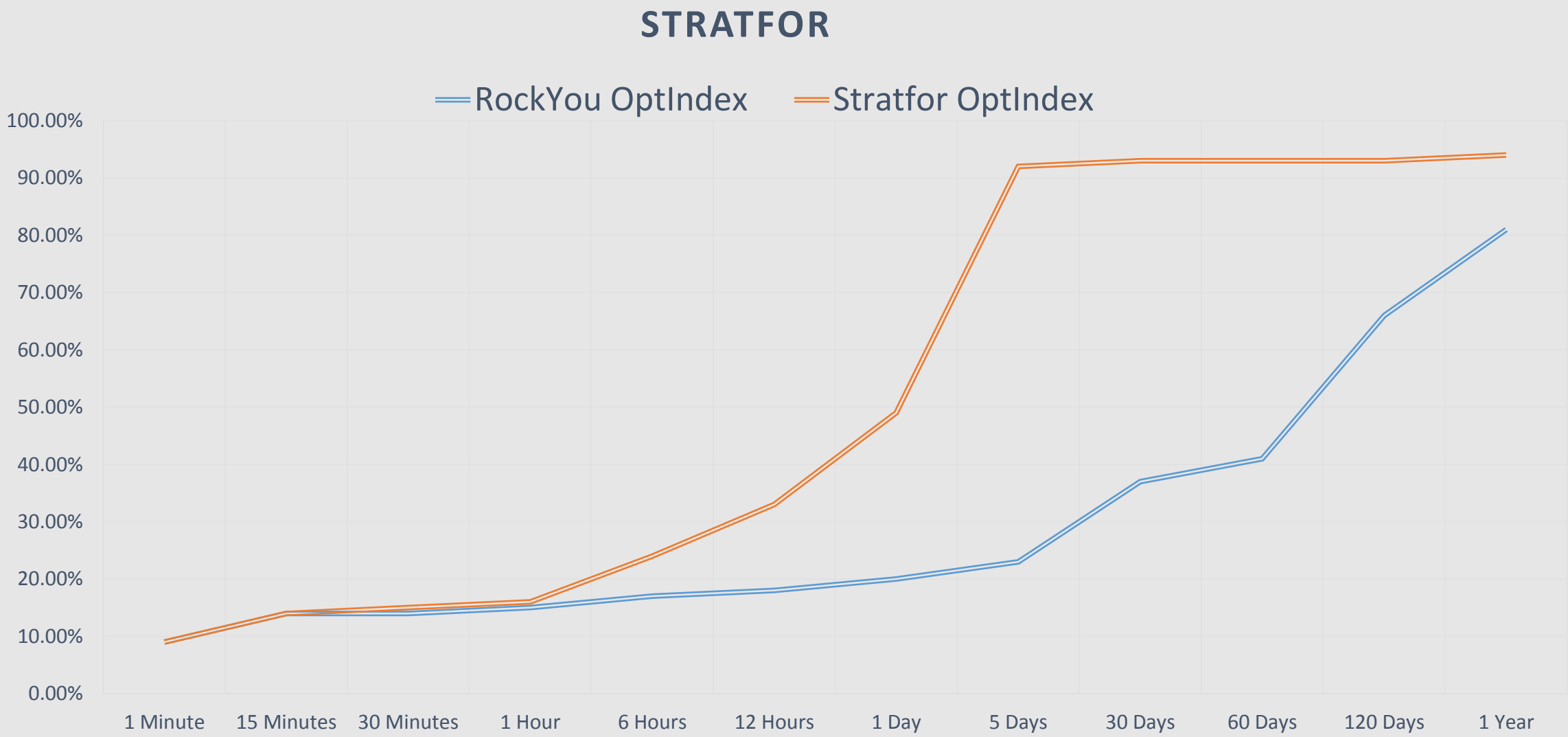


RockYou OptIndex Masks vs PHPBB and eHarmony



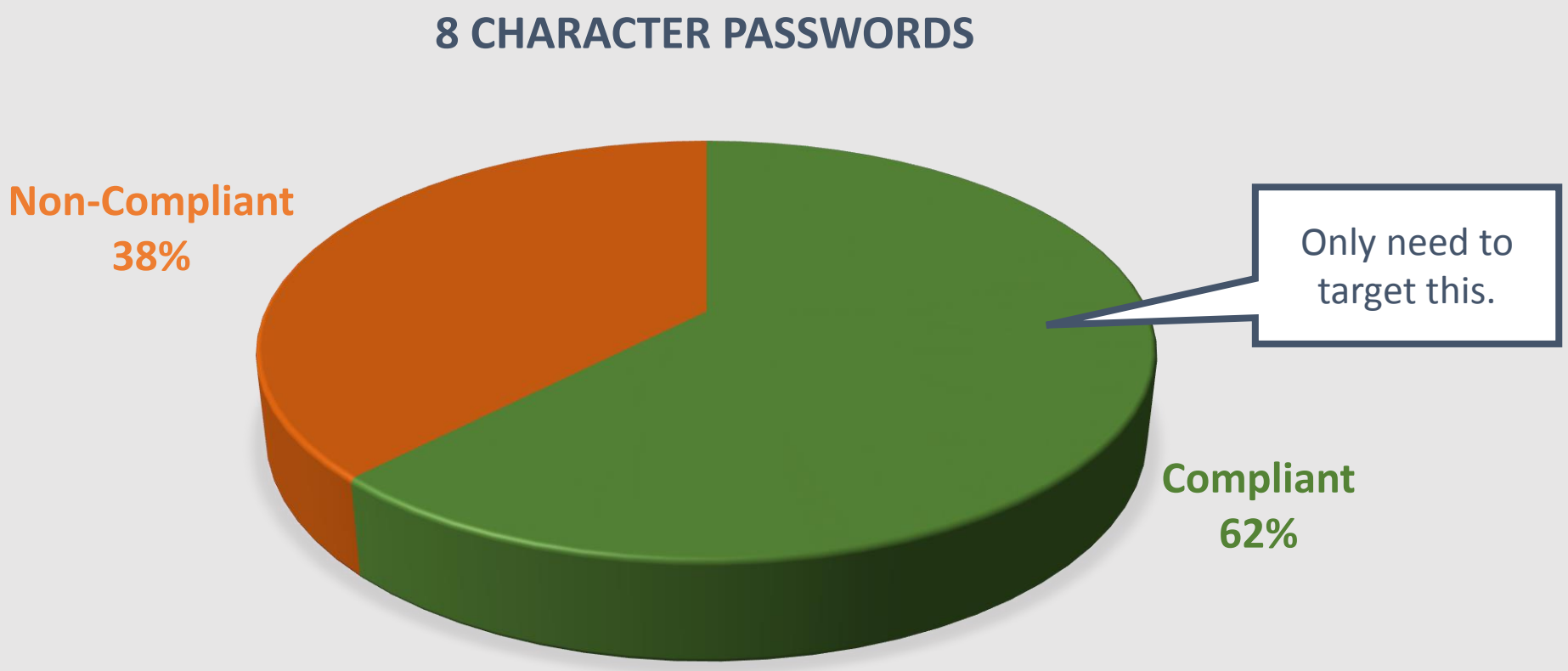


RockYou OptIndex Masks vs Stratfor

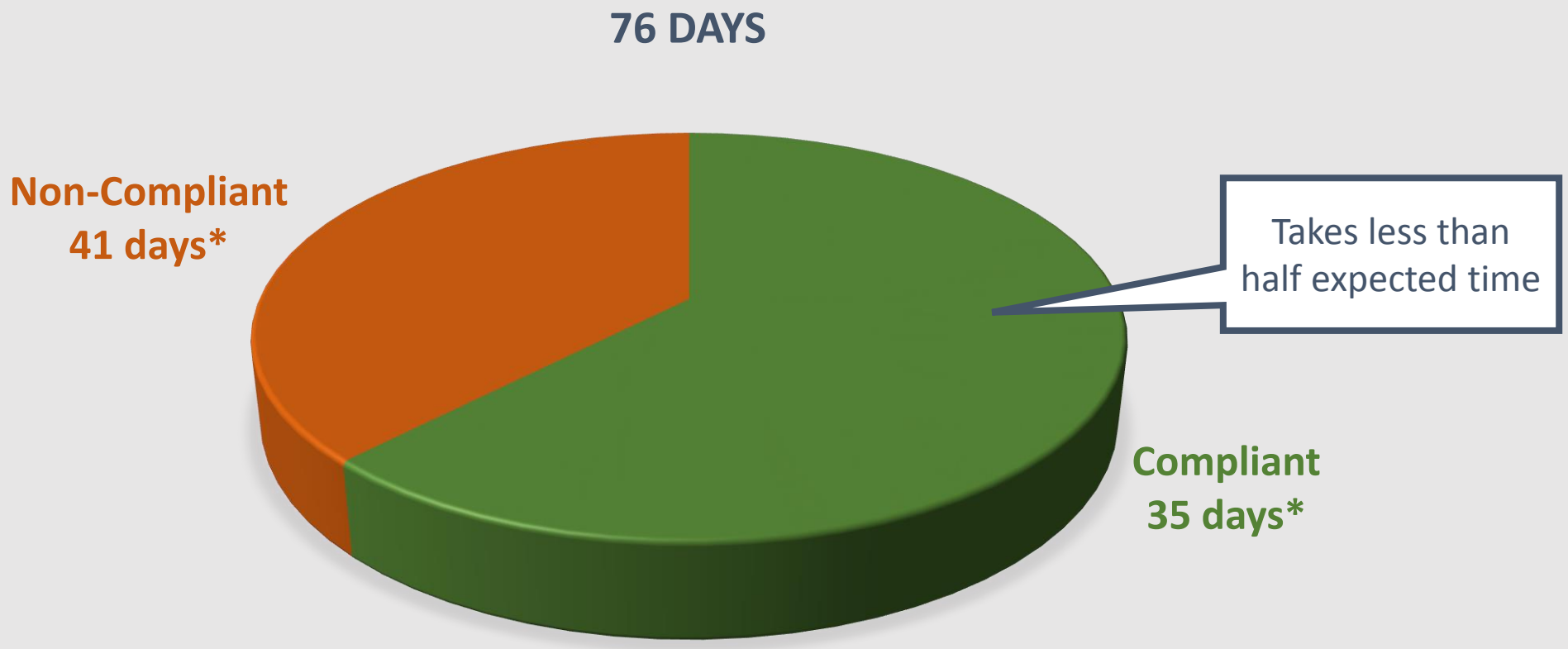


Password Policy Mask Analysis

Char:8 Lower:1 Upper:1 Digit:1 Special:1



Char:8 Lower:1 Upper:1 Digit:1 Special:1



* 1,000,000,000 keys/sec

Minimum password policy definition

```
$ python policygen.py --mindigit=1 --minlower=1 --minupper=1 --
minspecial=1 --showmasks -q -o policy-compliant.hcmask

[*] Using 1,000,000,000 keys/sec for calculations.
[*] Password policy:
    Pass Lengths: min:8 max:8
    Min strength: l:1 u:1 d:1 s:1
    Max strength: l:None u:None d:None s:None
[*] Generating [compliant] masks.
[*] Generating 8 character password masks.
[ 8] ?d?d?d?d?d?l?u?s                [l: 1 u: 1 d: 5 s: 1] [ 0:00:02]
[ 8] ?d?d?d?d?d?l?s?u                [l: 1 u: 1 d: 5 s: 1] [ 0:00:02]
[ 8] ?d?d?d?d?d?u?l?s                [l: 1 u: 1 d: 5 s: 1] [ 0:00:02]
...
[*] Total Masks:  65536 Time: 76 days, 18:50:04
[*] Policy Masks: 40824 Time: 35 days, 0:33:09
```

Password policy compliant masks

Minimum password policy

```
$ python policygen.py --mindigit=1 --minlower=1 --minupper=1
--minspecial=1 --showmasks -q -o policy-compliant.hcmask --noncompliant
[*] Using 1,000,000,000 keys/sec for calculations.
[*] Password policy:
    Pass Lengths: min:8 max:8
    Min strength: l:1 u:1 d:1 s:1
    Max strength: l:None u:None d:None s:None
[*] Generating [non-compliant] masks.
[*] Generating 8 character password masks.
[ 8] ?d?d?d?d?d?d?d?d                  [l: 0 u: 0 d: 8 s: 0] [ 0:00:00]
[ 8] ?d?d?d?d?d?d?d?l                  [l: 1 u: 0 d: 7 s: 0] [ 0:00:00]
[ 8] ?d?d?d?d?d?d?d?u                  [l: 0 u: 1 d: 7 s: 0] [ 0:00:00]
...
[*] Total Masks:  65536 Time: 76 days, 18:50:04
[*] Policy Masks: 24712 Time: 41 days, 18:16:55
```

Password policy **non-compliant** masks

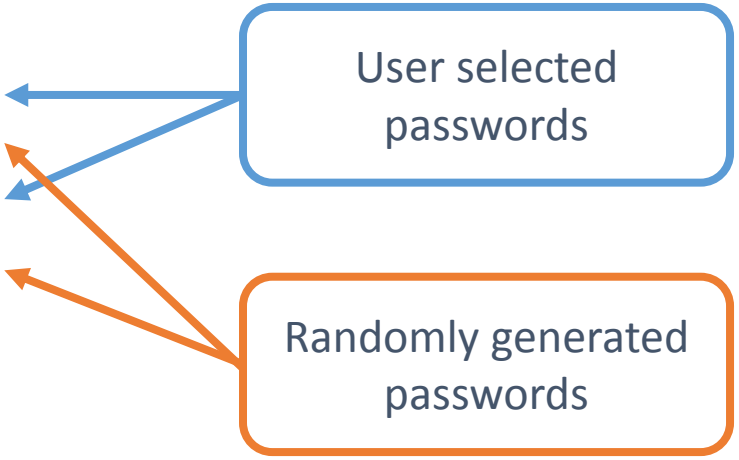
Targeted policy attack against Stratfor

1. Obtain a small sample with **bruteforce**:

```
./oclHashcat-plus64.bin -n 160 --runtime 3600 -m 0 -a 3 -o stratfor.dict --
outfile-format=2 stratfor.hash ?a?a?a?a?a?a?a?a
Session.Name...: oclHashcat-plus
Status.........: Running
Input.Mode.....: Mask (?a?a?a?a?a?a?a?a)
Hash.Target....: File (stratfor.hash)
Hash.Type......: MD5
Time.Started...: Mon Jul 22 14:35:30 2013 (8 secs)
Time.Estimated.: Fri Sep  6 14:52:09 2013 (46 days, 0 hours)
Speed.GPU.#1...:  2528.8 MH/s
Recovered......: 1016/822657 (0.12%) Digests, 0/1 (0.00%) Salts
Progress.......: 13736345600/6634204312890625 (0.00%)
Rejected.......: 0/13736345600 (0.00%)
HWMon.GPU.#1...: 79% Util, 50c Temp, 39% Fan
```

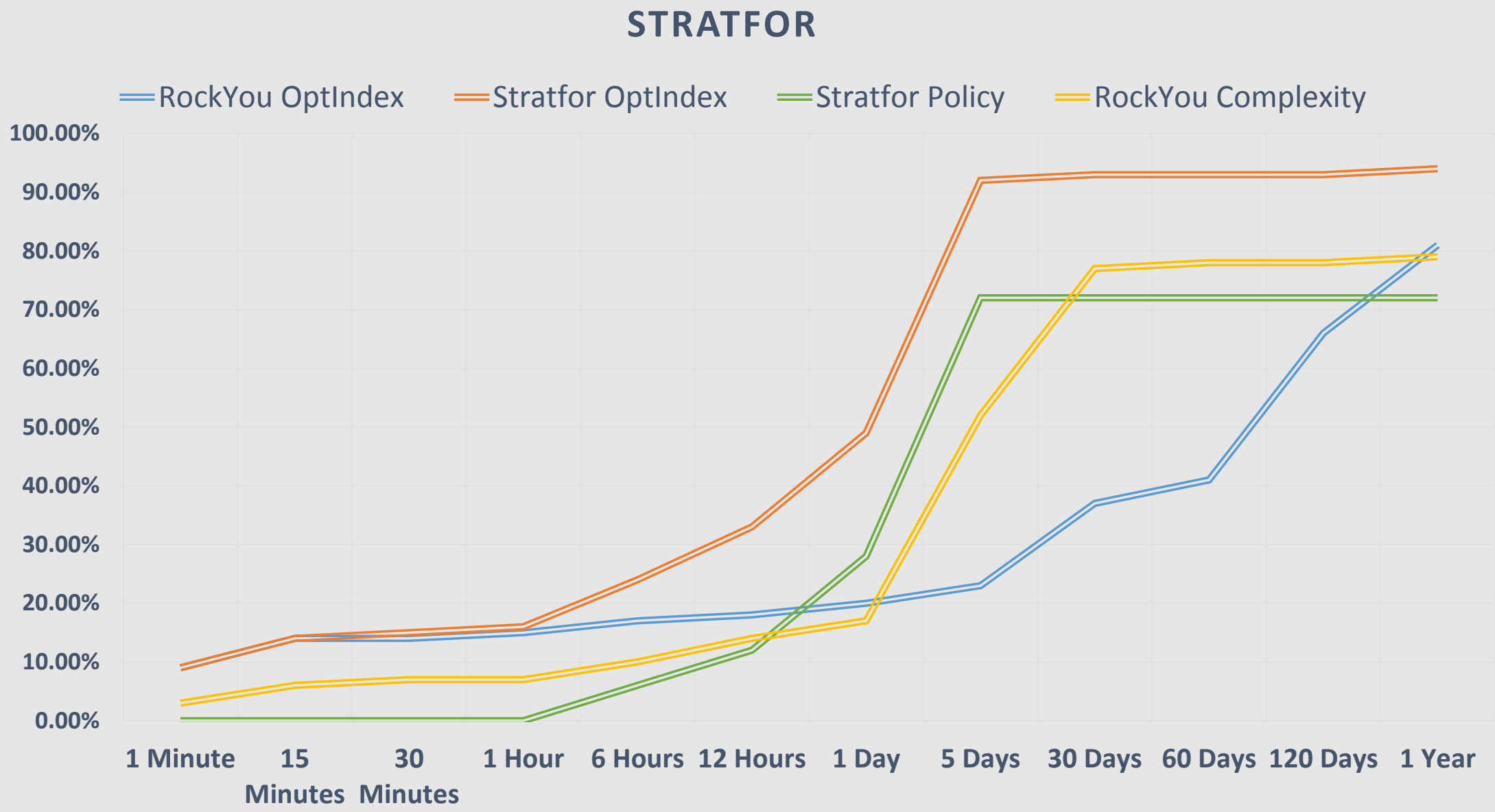
2. Analyze recovered passwords with **statsgen**:

```
$ python statsgen.py stratfor.dict
...
[*] Length Statistics...
[+]                          8: 100% (6961)

[*] Charset statistics...
[+]              loweralphanum: 34% (2417)
[+]              mixedalphanum: 23% (1613)
[+]                 loweralpha: 22% (1596)
[+]                 mixedalpha: 11% (825)
[+]                    numeric: 02% (199)
...
```



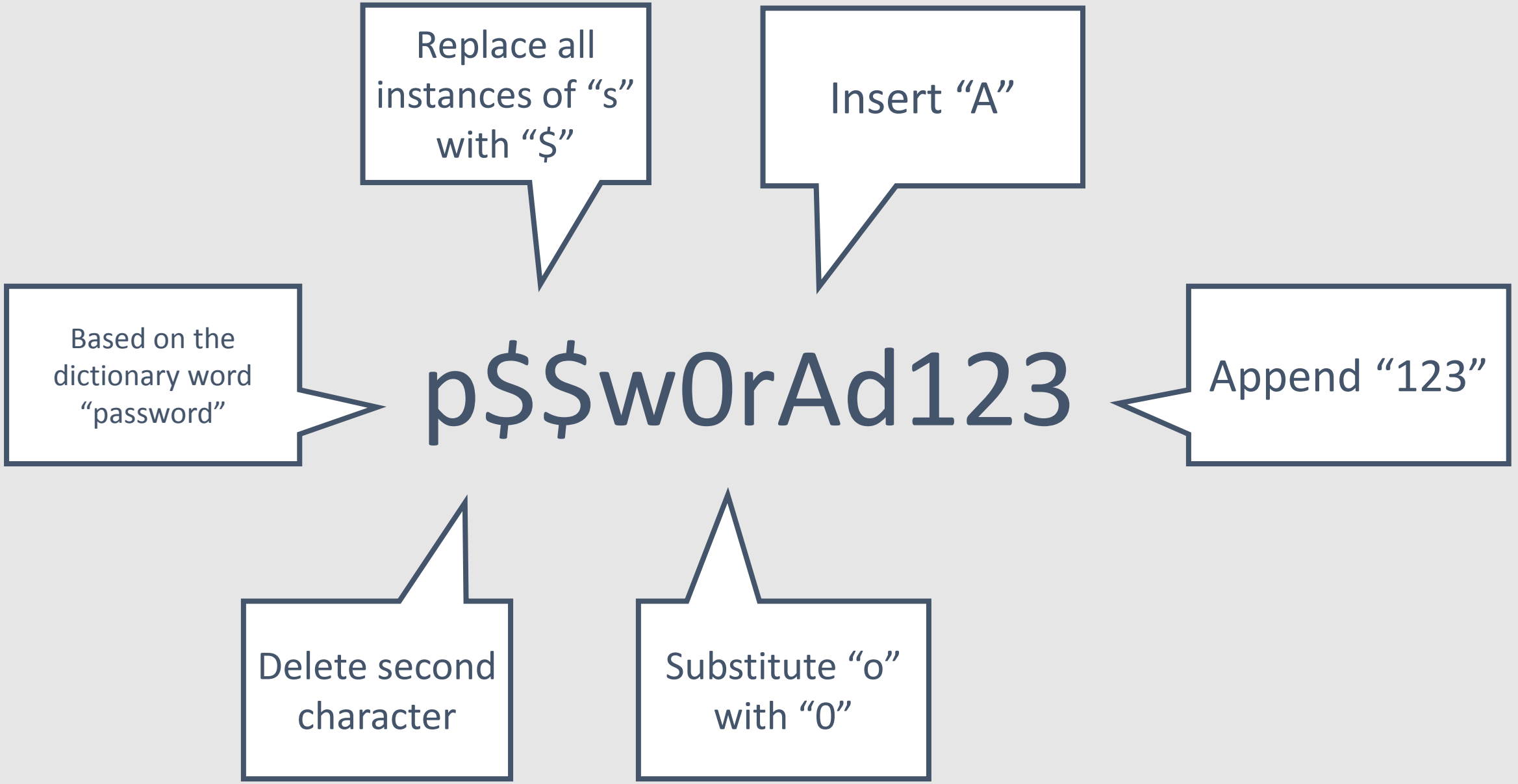
3. Generate targeted policy masks with **policygen**.

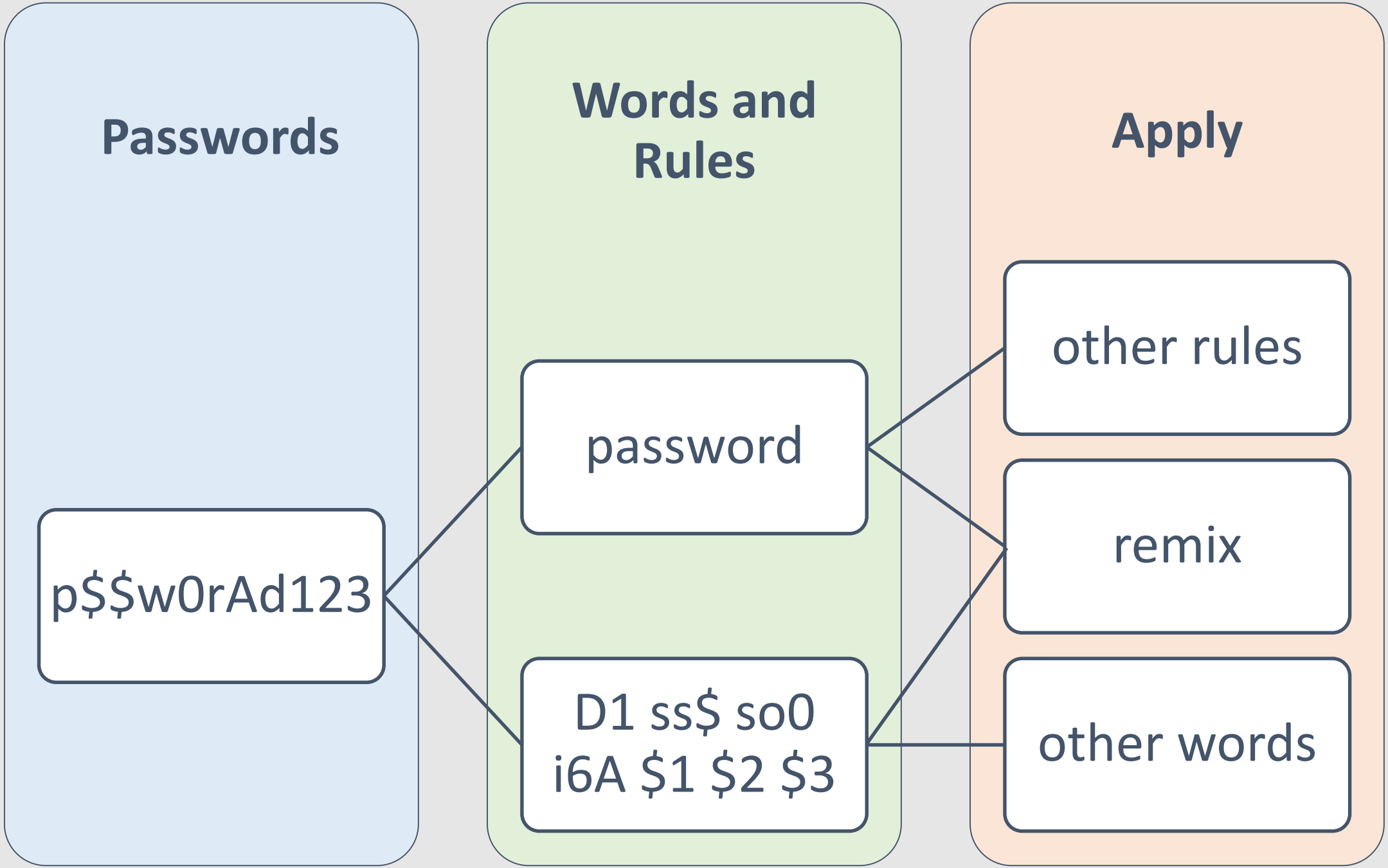
```
$ python policygen.py --mindigit=0 --minlower=1 --minupper=1 --maxspecial=0
-o stratfor-policy.hcmask
[*] Saving generated masks to [stratfor-policy.hcmask]
[*] Using 1,000,000,000 keys/sec for calculations.
[*] Password policy:
    Pass Lengths: min:8 max:8
    Min strength: l:1 u:1 d:0 s:None
    Max strength: l:None u:None d:None s:0
[*] Generating [compliant] masks.
[*] Generating 8 character password masks.
[*] Total Masks:  65536 Time: 76 days, 18:50:04
[*] Policy Masks: 6050 Time: 2 days, 11:04:57
```


STRATFOR
RockYou OptIndex
Stratfor OptIndex
Stratfor Policy
RockYou Complexity
100.00%
90.00%
80.00%
70.00%
60.00%
50.00%
40.00%
30.00%
20.00%
10.00%
0.00%
1 Minute
15 Minutes
30 Minutes
1 Hour
6 Hours
12 Hours
1 Day
5 Days
30 Days
60 Days
120 Days
1 Year

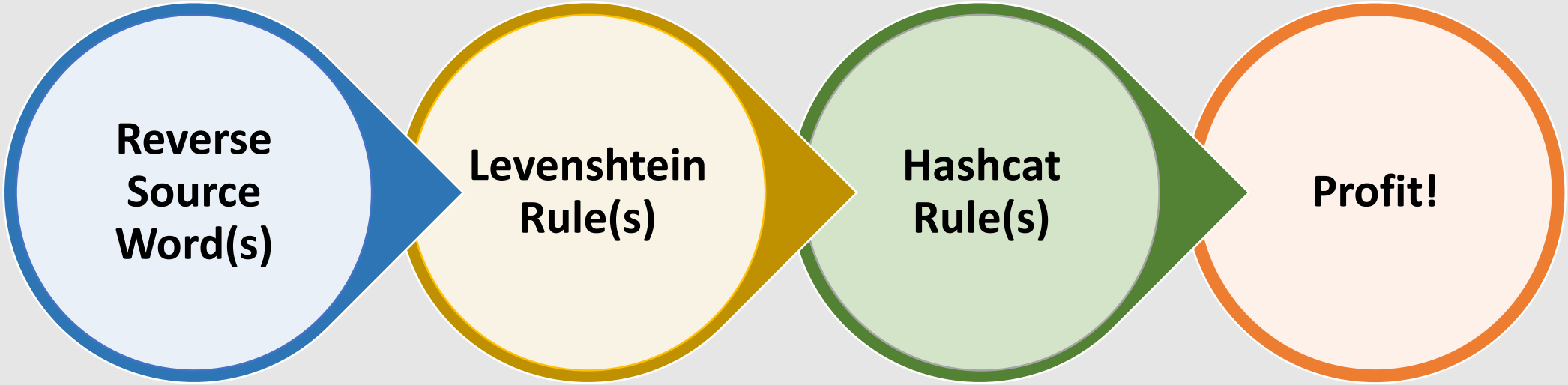
Word Mangling Rules Analysis

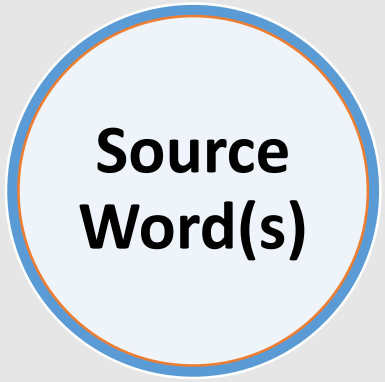
p$$w0rAd123


Replace all instances of “s” with “$”
Insert “A”
Based on the dictionary word “password”
p$$w0rAd123
Append “123”
Delete second character
Substitute “o” with “0”

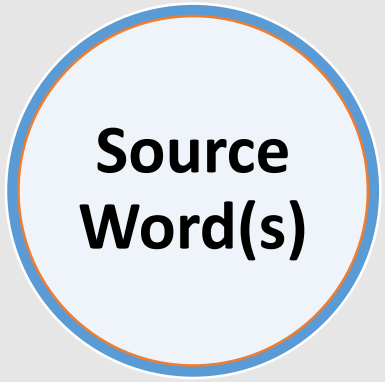

Passwords
Words and Rules
Apply
p$$w0rAd123
password
D1 ss$ so0
i6A $1 $2 $3
other rules
remix
other words

PACK: rulegen approach



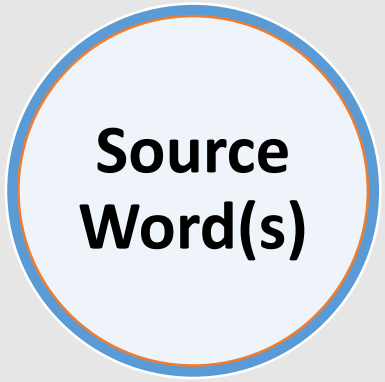


- Is a word mangled password a case of intentional misspelling?
  - p4ssword1 => **password**, swordplay, crossword
- Does it work for more complex passwords?
  - 1pa$$word1 => pulpwood, upwardness
- Can we improve results with a bit of pre-analysis?
  - 1pa$$word1 => **password**, broadsword, swordplayer



- What other rules can be detected readily by pre-analysis?
  - **Rotation**: wordpass => password
  - **Reversal:** drowssap => password
  - **Prefix/Appendix**: 1password1 => password
  - **Duplication**: passwordpassword => password
  - **Combination**: super!man => super + man
  - **Patterns**: password@gmail.com => password

- How about custom dictionaries for targeted attacks?
  - Iloveu123 => spillover, allover, alleviate
  - Iloveu123 => **iloveu**



- How can we prioritize generated source words?

  Levenshtein Edit Distance

  - distance between two words is the minimum number of single-character edits (insertion, deletion, substitution) required to change one word into the other.

    -Wikipedia (http://en.wikipedia.org/wiki/Levenshtein_distance)

  p4ssw0rd =>

  - **password** Edit distance 2
  - Pissaro Edit distance 5
  - assured Edit distance 5

Levenshtein Rule(s)

Levenshtein Edit Distance Algorithm

$$\operatorname{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \operatorname{lev}_{a,b}(i-1,j) + 1 \\ \operatorname{lev}_{a,b}(i,j-1) + 1 \\ \operatorname{lev}_{a,b}(i-1,j-1) + [a_i \neq b_j] \end{cases} & \text{otherwise.} \end{cases}$$

Blank Levenshtein Matrix

		p	a	s	s	w	o	r	d
p									
4									
s									
s									
w									
0									
r									
d									

Edit distance empty string to “password”

		p	a	s	s	w	o	r	d
	0	1	2	3	4	5	6	7	8
p									
4									
s									
s									
w									
0									
r									
d									

Edit distance letter “p” to “password”

		p	a	s	s	w	o	r	d
	0	1	2	3	4	5	6	7	8
p	1	0	1	2	3	4	5	6	7
4									
s									
s									
w									
0									
r									
d									

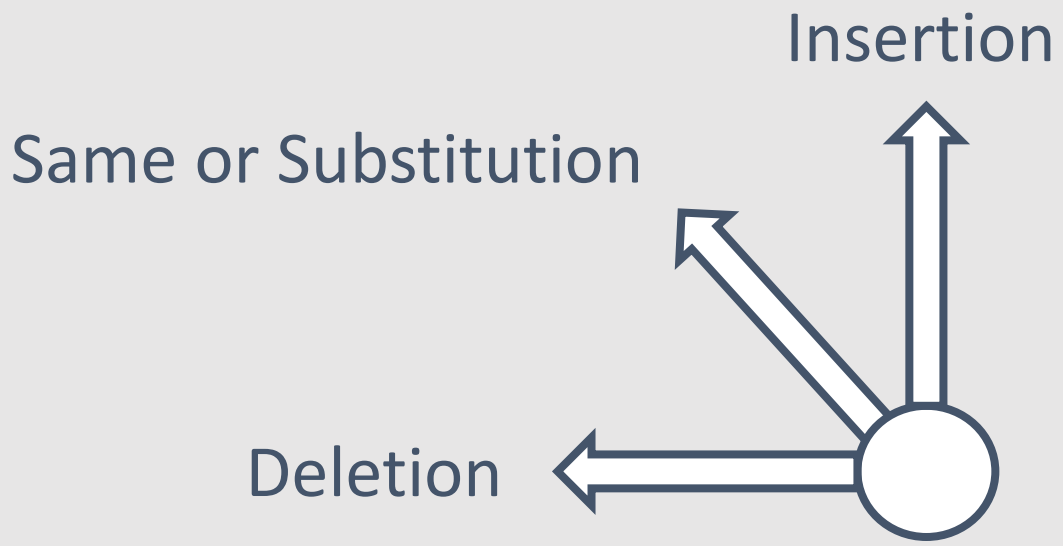
Edit distance of “p4” to “password”

		p	a	s	s	w	o	r	d
	0	1	2	3	4	5	6	7	8
p	1	0	1	2	3	4	5	6	7
4	2	1	1	2	3	4	5	6	7
s									
s									
w									
0									
r									
d									

Edit distance of “p4sswOrd” to “password”

		p	a	s	s	w	o	r	d
	0	1	2	3	4	5	6	7	8
p	1	0	1	2	3	4	5	6	7
4	2	1	1	2	3	4	5	6	7
s	3	2	2	1	2	3	4	5	6
s	4	3	3	2	1	2	3	4	5
w	5	4	4	3	2	1	2	3	4
0	6	5	5	4	3	2	2	3	4
r	7	6	6	5	4	3	3	2	3
d	8	7	7	6	5	4	4	3	2

Reverse Levenshtein Paths Algorithm
(Recursive, Depth First, Short-circuited)



Taking the first step backwards

		p	a	s	s	w	o	r	d
	0	1	2	3	4	5	6	7	8
p	1	0	1	2	3	4	5	6	7
4	2	1	1	2	3	4	5	6	7
s	3	2	2	1	2	3	4	5	6
s	4	3	3	2	1	2	3	4	5
w	5	4	4	3	2	1	2	3	4
0	6	5	5	4	3	2	2	3	4
r	7	6	6	5	4	3	3	2	3
d	8	7	7	6	5	4	4	3	2

First the change

		p	a	s	s	w	o	r	d
	0	1	2	3	4	5	6	7	8
p	1	0	1	2	3	4	5	6	7
4	2	1	1	2	3	4	5	6	7
s	3	2	2	1	2	3	4	5	6
s	4	3	3	2	1	2	3	4	5
w	5	4	4	3	2	1	2	3	4
0	6	5	5	4	3	2	2	3	4
r	7	6	6	5	4	3	3	2	3
d	8	7	7	6	5	4	4	3	2

Substitution ‘o’-’0’

		p	a	s	s	w	o	r	d
	0	1	2	3	4	5		7	8
p	1	0	1	2	3	4		6	7
4	2	1	1	2	3	4		6	7
s	3	2	2	1	2	3		5	6
s	4	3	3	2	1	2		4	5
w	5	4	4	3	2	1		3	4
0							2	3	4
r	7	6	6	5	4	3	3	2	3
d	8	7	7	6	5	4	4	3	2

Complete reverse path
Substitute ‘a’-’4’, Substitute ‘o’-’0’

		p	a	s	s	w	o	r	d
	0	1		3	4	5		7	8
p	1	0		2	3	4		6	7
4			1	2	3	4		6	7
s	3	2	2	1	2	3		5	6
s	4	3	3	2	1	2		4	5
w	5	4	4	3	2	1		3	4
0							2	3	4
r	7	6	6	5	4	3	3	2	3
d	8	7	7	6	5	4	4	3	2

Multiple Reverse Paths

Delete ‘a’, Substitute ‘o’-’0’, Insert ‘1’

		p	a	s	s	w	o	r	d
	0	1	2	3	4	5	6	7	8
p	1	0	1	2	3	4	5	6	7
s	2	1	1	1	2	3	4	5	6
s	3	2	2	1	1	2	3	4	5
w	4	3	3	2	2	1	2	3	4
0	5	4	4	3	3	2	2	3	4
r	6	5	5	4	4	3	3	2	3
d	7	6	6	5	5	4	4	3	2
1	8	7	7	6	6	5	5	4	3

Multiple Reverse Paths

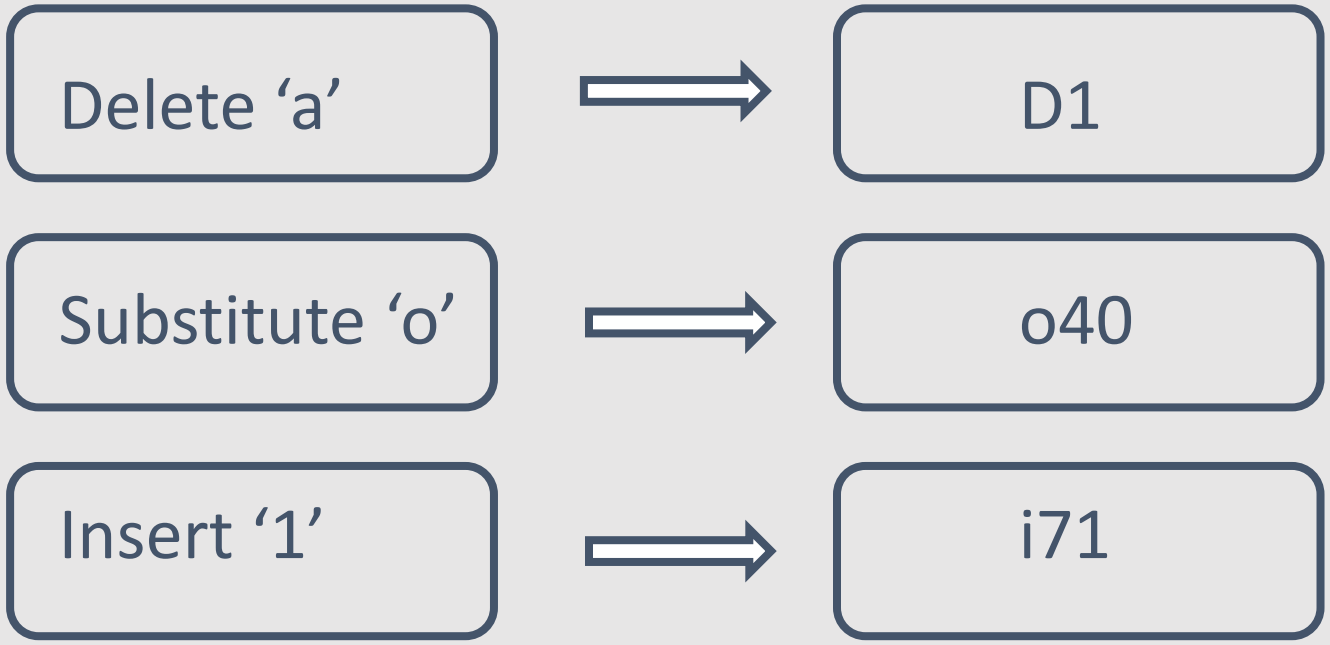
Substitute ‘a’-’s’, Delete ‘s’, Substitute ‘o’-’0’, Insert ‘1’

		p	a	s	s	w	o	r	d
	0	1			4	5		7	8
p	1	0			3	4		6	7
s			1		2	3		5	6
s	3	2	2	1	1	2		4	5
w	4	3	3	2	2	1		3	4
0							2	3	4
r	6	5	5	4	4	3	3	2	3
d	7	6	6	5	5	4	4	3	2
1									3

Hashcat Rule(s)

Simple approach

Hashcat rules to transform ‘password’ to ‘pssw0rd1’



Levenshtein to Hashcat Conversion Table

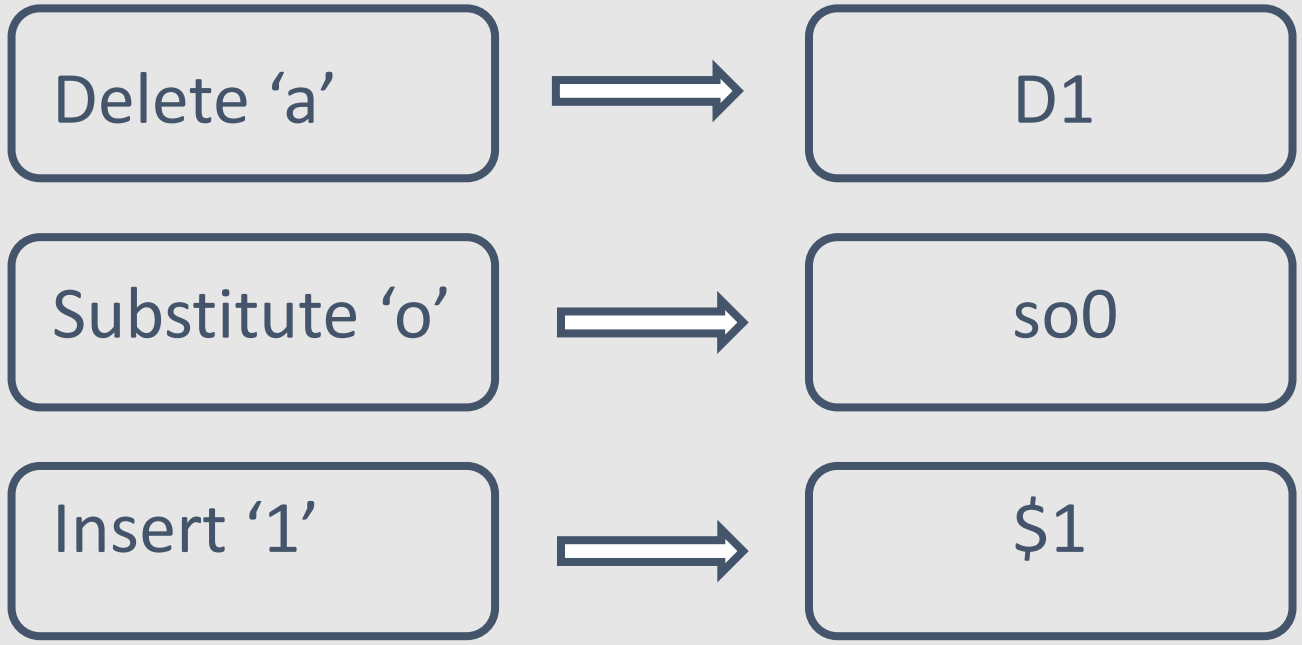
Levenshtein OP	Rule	Description
(insert,i,j) where i is at the end of the word.	**$c**	Append character to end.
(insert,i,j) where i is at the beginning of the word.	**^c**	Prepend character to end.
(delete,i,j) where i is at the end of the word.	**]**	Delete last character.
(replace,i,j) where i and i+1 were swapped.	***XY**	Swap character X with Y.
(replace,i,j) where i and i+1 were swapped where i at the beginning of the word	**k**	Swap the first two characters.
(replace,i,j) where i and i+1 were swapped where i is at the end of the word	***XY**	Swap the last two characters.
(replace,i,j) where a case changed for the character i	**TN**	Toggle the case of characters at position N.
(replace,i,j) where all characters at the index location i in the source word were replaced in the password	**sXY**	Replace all instances of X with Y.
(replace,i,j) where replacement character at i is an ASCII increment	**+N**	Increment character @ N by 1 ascii value.
(replace,i,j) where replacement character at i is a left bitwise shift	**LN**	Bitwise shift left character @ N.

(Total 20 Hashcat rules detected)

Hashcat Rule(s)

Advanced approach

Hashcat rules to transform ‘password’ to ‘pssw0rd1’



Hashcat Rule(s)

```
$ python rulegen.py --password 'p$$w0rAd123' -v -q
[*] Using Enchant 'aspell' module. For best results please install
    'aspell' module language dictionaries.
[*] Saving rules to analysis.rule
[*] Saving words to analysis.word
[*] Press Ctrl-C to end execution and generate statistical analysis.
[*] Analyzing password: p$$w0rAd123
...
[+] password => D1 ss$ so0 i6A $1 $2 $3 => p$$w0rAd123
[+] passwords => D1 ss$ so0 i6A o81 $2 $3 => p$$w0rAd123
[+] passwords => D1 ss$ so0 i6A i81 o92 $3 => p$$w0rAd123
[+] passwords => D1 ss$ so0 i6A i81 i92 oA3 => p$$w0rAd123
[*] Finished analysis in 0.00 seconds
```

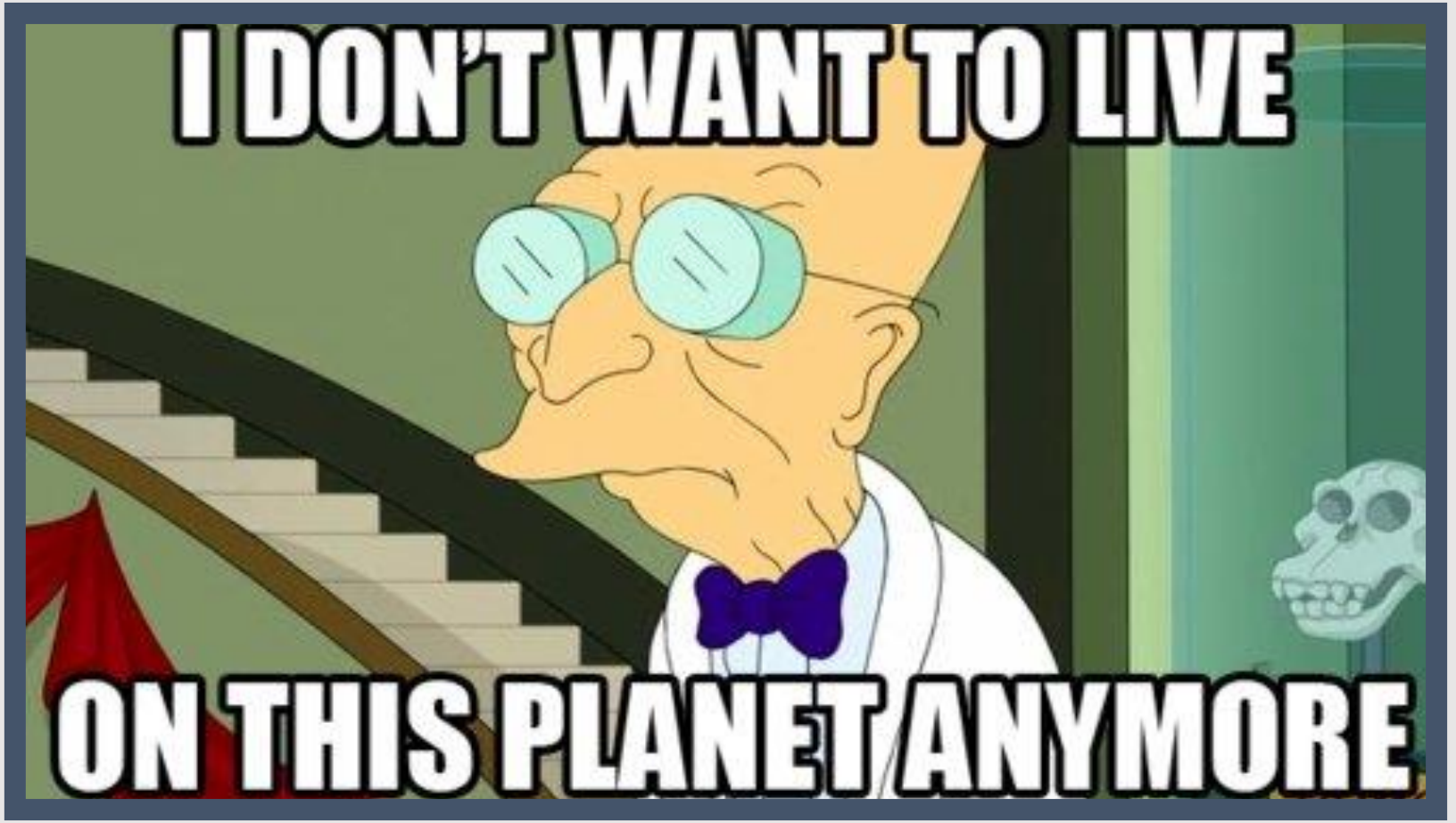
```
$ python rulegen.py gawker.dic -b gawker --bruterules
...
[*] Finished processing 1084394 passwords in 593.15 seconds at the rate of
1828.20 p/sec
[*] Word worker stopped.
[*] Generating statistics for [gawker] rules and words.
[-] Skipped 56637 all numeric passwords (5.22%)
[-] Skipped 13703 passwords with less than 25% alpha characters (1.26%)
[-] Skipped 26 passwords with non ascii characters (0.00%)

[*] Top 10 rule statistics
[+] : - 167201 (3.00%)
[+] T0 - 24531 (0.00%)
[+] $1 - 17619 (0.00%)
[+] r - 13412 (0.00%)
[+] ] - 7237 (0.00%)
[+] $2 - 4792 (0.00%)
[+] $1 $2 - 3718 (0.00%)
[+] l $1 - 3243 (0.00%)
[+] $1 $2 $3 - 3146 (0.00%)
[+] o71 - 2902 (0.00%)
```

Generating rules for some leaks

Source	Top Words (2+ chars)	Top Rules
RockYou*	lover, baby, lovey, maria, jean, loves, loved, angel, mike, kebab, mayo, angels, marina	: $1 r $2 $1 $2 $3 $1 $2 $3 $7 ^1 $1 $3

Source	Top Words (2+ chars)	Top Rules
LinkedIn	linked, linkedin, password, alex, mike, jim, jfs, jam, jack, job	: $1 $1 $2 $3 $0 $1 $2 $1 $2 $1 $1 $7 $3 ^1

* Generated using new --bruterules flag.


I DON'T WANT TO LIVE
ON THIS PLANET ANYMORE

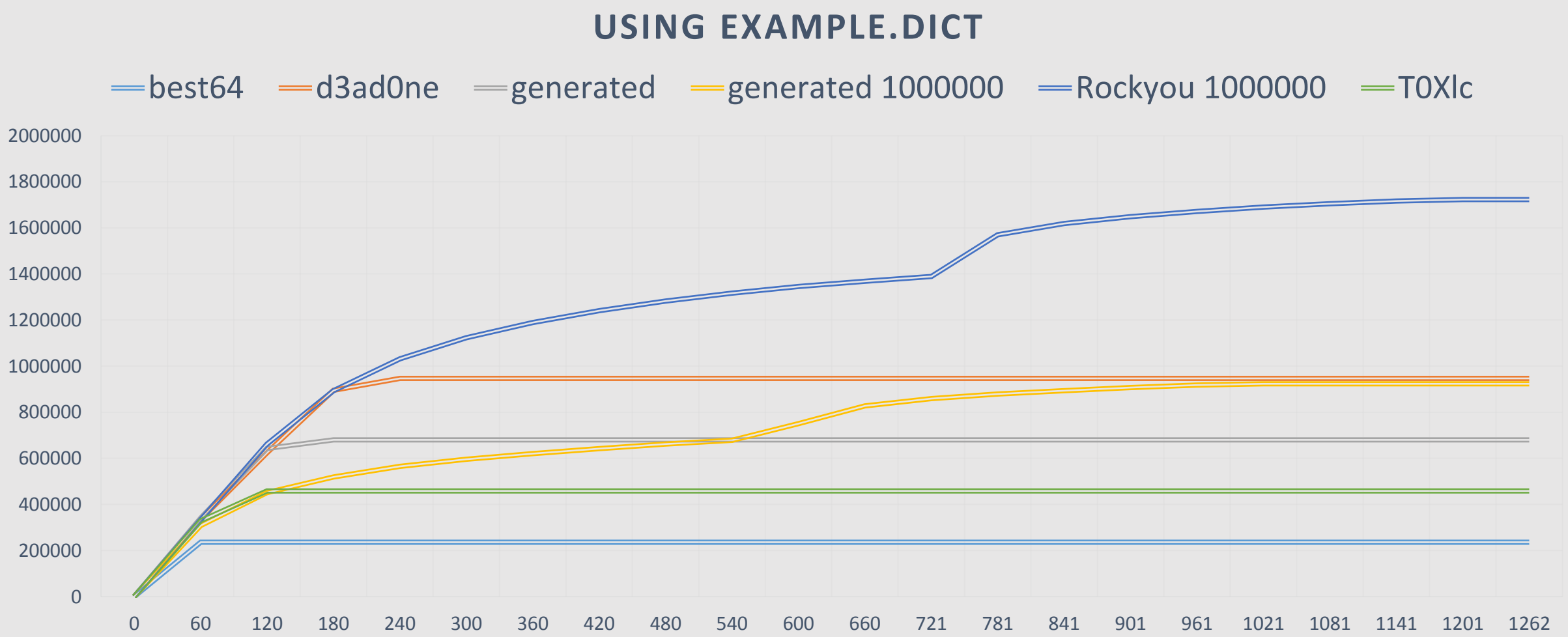
… some more leaks

Source	Top Words (2+ chars)	Top Rules
MySpace	password, olive, myspace, love, hearts, baseball, softball, football, cutie, chicken	$1 : $2 $! $3 $1 $2 $3 l $1 $7 $1 $2 $5

Source	Top Words (2+ chars)	Top Rules
Singles.org	Jesus, love, angel, loveme, faith	: T0 $1] $2 $7 l $1 $1 $2 $3 $1 $2 $4

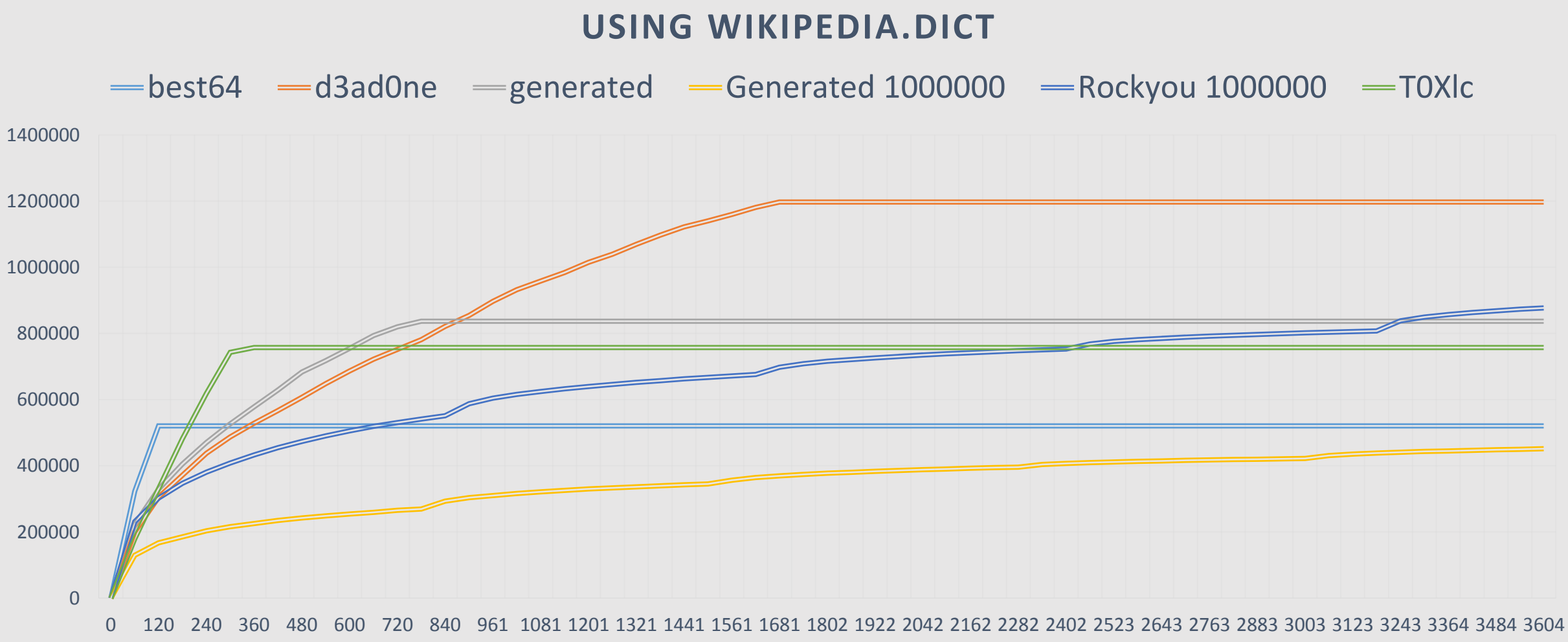
Applying RockYou 1,000,000 rules vs. Gamigo

(out of total 67,000,000 sorted unique rules)



Applying RockYou 1,000,000 rules vs. Gamigo

(out of total 67,000,000 sorted unique rules)

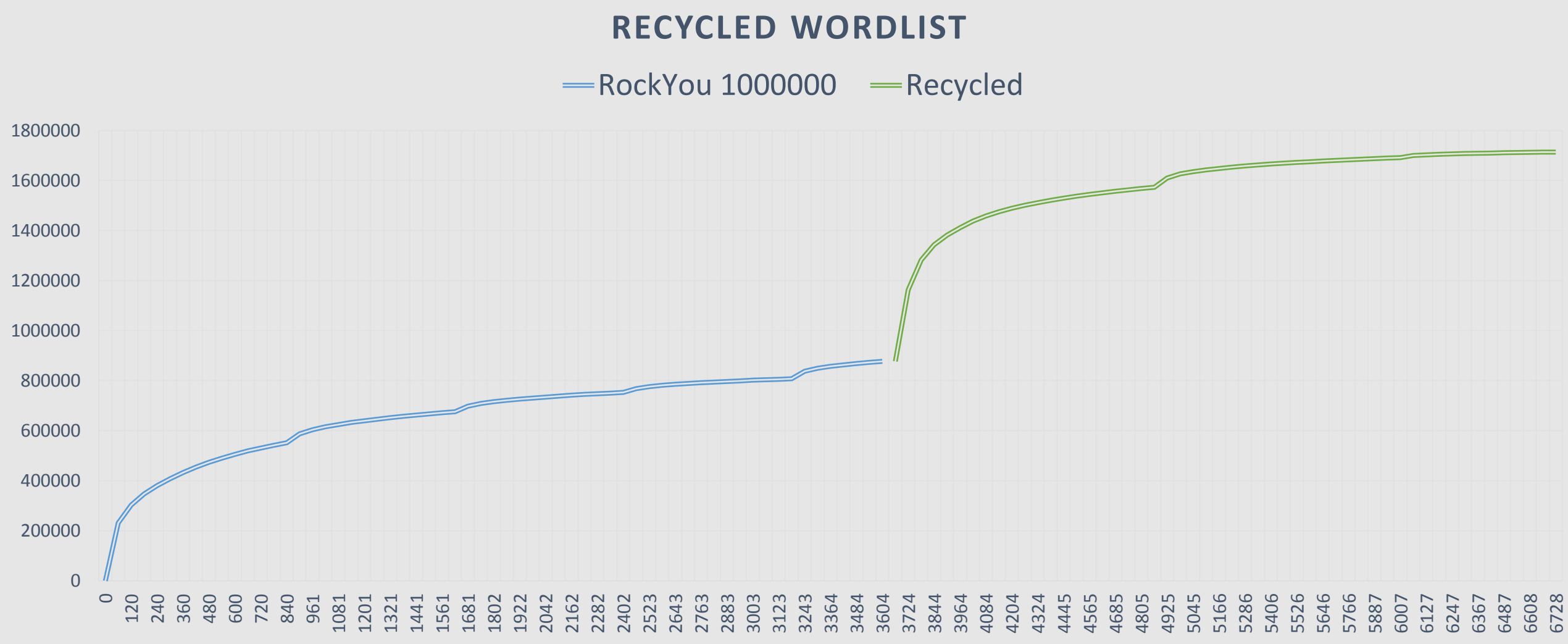


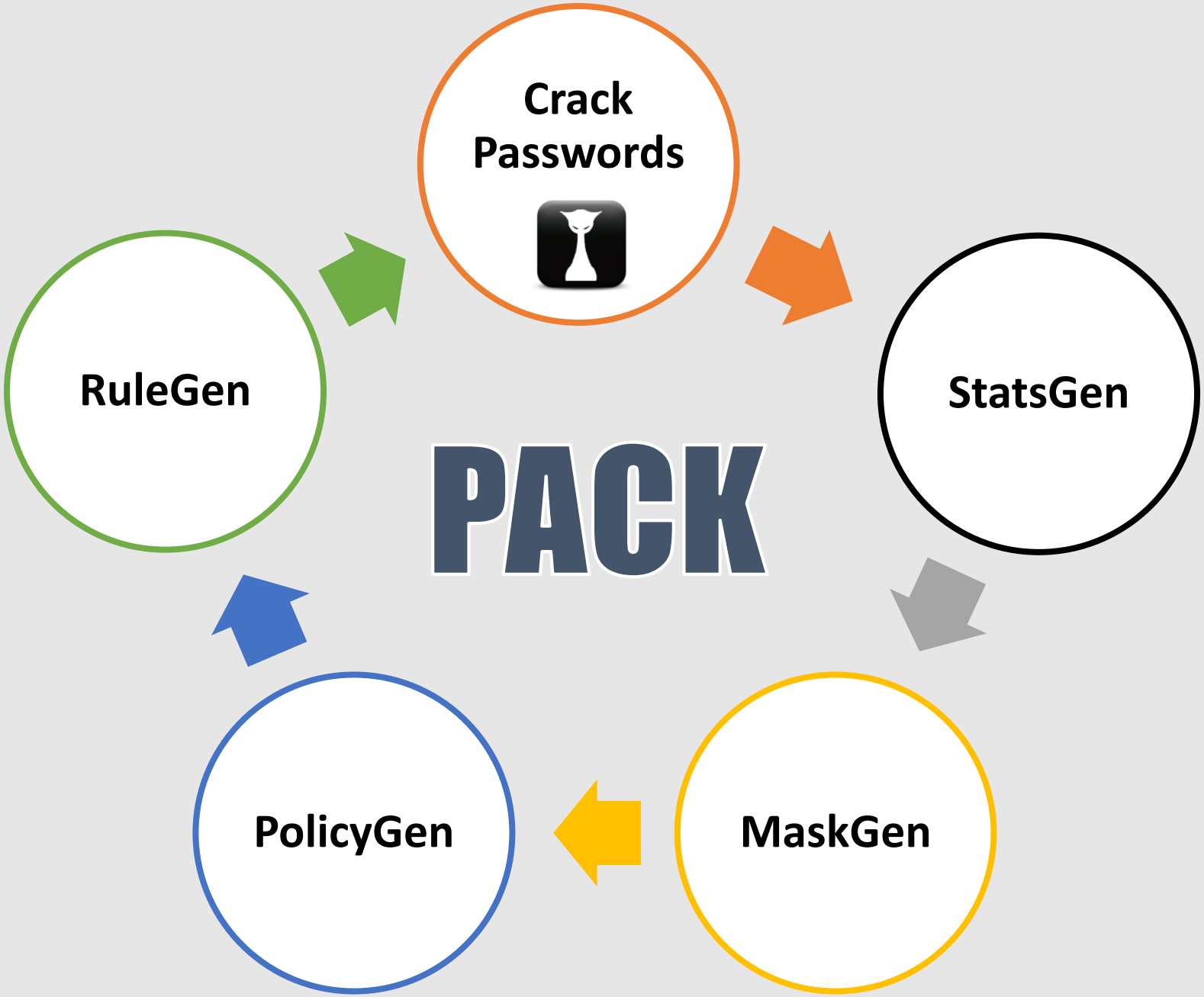
Recycle cracked passwords...

```
$ python rulegen.py gamigo-rockyou1000000-
wikipedia.cracked -b gamigo-recycled
```

```
$ oclHashcat-plus64.bin gamigo.hash
gamigo-recycled-sorted.word
-r gamigo-recycled-sorted.rule
```

Applying recycled rules and source words.




Crack Passwords
StatsGen
MaskGen
PolicyGen
RuleGen
PACK

Takeaways

Defenders & Developers

Security Researchers

Penetration Testers

Gratitude

Jens and Team Hashcat

KoreLogic SECURITY

Crack Me If You Can

passwords 13 las vegas

Per and Jeremi


ANY QUESTIONS ?