A Novel Hybrid Algorithm Based on Word and Method Ranking for Password Security

Berker Tasoluk[1†] and Zuhal Tanrikulu[2††],

Istanbul University Bogazici University

Abstract
It is a common practice to use a password in order to restrict access to information, or in a general sense, to assets. Right selection of the password is necessary for protecting the assets more effectively. Password finding/cracking try outs are performed for deciding which level of protection do used or prospective passwords offer, and password cracking algorithms are generated. These algorithms are becoming more intelligent and succeed in finding more number of passwords in less tries and in a shorter duration. In this study, the performances of possible password finding algorithms are measured, and a hybrid algorithm based on the performances of different password cracking algorithms is generated, and it is demonstrated that the performance of the hybrid algorithm is superior to the base algorithms.
Keywords:
Password cracking, password security, hybrid, brute force, dictionary attacks, password testing, information security, it security.

1. Introduction

It is a common practice to use a password in order to restrict access to information, or in a general sense, to assets. Right selection of the password is necessary for protecting the assets more effectively. Password finding/cracking try outs are performed for deciding which level of protection do used or prospective passwords offer, and password cracking algorithms are generated. These algorithms are becoming more intelligent and succeed in finding a greater number of passwords in less tries and in a shorter duration.

In this study, the performances of possible password finding algorithms are measured, and a hybrid algorithm based on different algorithms is generated. The study comprises of following sections:
In the related work section, general concepts about information security are examined; and the studies found on literature about password finding/cracking are briefly summarized.
In the methodology section, the hybrid algorithm generated is described.
In the results section, the performances of different password cracking algorithms are measured, a hybrid algorithm based on these algorithms is generated, and it is demonstrated that the performance of the hybrid algorithm is superior to the base algorithms.
In the conclusion and future work section, the results are summarized and suggestion for possible future research is specified.

2. Related Work

2.1. The 'Information Security' Concept

When information security is discussed, three features of information security show up. The first is confidentiality, which is the information will only be accessed by authorized ones. Integrity is another feature, and can be regarded as, information will only be changed by authorized ones. The last one is availability, can be summarized as being able to access information when needed. Those three features are regarded as the 'CIA Triad'. Those three features can contradict with each other, enhancing one feature may adversely impact other information security features [1, 2]

Data is very valuable asset for companies. Losing confidentiality of data may cost companies 100s of millions of dollars or more to companies [3]. According to IBM 2020 Cost of Data Breach Report, average data loss costs companies $3.86 Million [4]. There are numerous cases in which cyber attackers have successfully infiltrated data from companies [5-7].

We can recall several incidents relating to sensitive data breaches. Adobe had an incident in 2013 and lost between 38-153 million accounts [5, 8]. In 2012 incident, Linkedin is estimated to lose over 100 Million accounts [9]. In another incident, Marriott Hotels lost 500 Million customer data [10]. Other examples include Uber breach in 2016 which affected

Manuscript received March 5, 2023
Manuscript revised March 20, 2023
https://doi.org/10.22937/IJCSNS.2023.23.3.17

57 million users [11] and Friendfinder breach in 2017 which affected 412 million user accounts [12].

An incident happened in Epsilon is estimated to cost $4Billion [13]. Another good example regarding data security is cracking the Enigma machine code which is used by Germans in World War II [14]. This progress shifted the course of the war. Nowadays, military and national security systems are electronic based, the responsible parties must take all necessary precautions to protect these systems. Some protection mechanisms are added to systems in order to protect information. When a person or a program requests the data, the first step is, they must tell who they are (identification). Identification is an assertion of the requestor, the system must use extra mechanisms to validate if this assertion is correct. Those controls comprise of 'authentication' step. After authentication step, the system determines what can the requestor do on the system and data (read/write/delete/edit/etc.), this step is called 'authorization' [15, 16].

In the authentication step, three different 'factors' can be used. Those factors are identified as; something you know, something you have and something you are. If two of those options are used while authenticating users, it is said 'two factor authentication' is used. Likewise, when all three of those options are used, 'three factor authentication' is used. Both two and three-factor authentication are called 'strong authentication' schemes [16].

An example of 'something you know' which is used on authenticating users is passwords. Apart from this; birth place, birthdate, birth place, identification number, home address, cellular phone number, etc. are all examples of this authentication factor.

Tokens, cellular phones, credit/debit cards are all examples of 'something you have' factor. The password or push notification sent to cellular phone, the digital signature embedded in cellular sim card, or the dynamic number found on the soft token application are generally used in authentication as a second factor, on top of 'something you know' [17].

Fingerprint, retina scan, iris scan, palm scan, voice recognition, signature are all examples of the third factor, 'something you are'.

2.2. Passwords

Passwords are used extensively on user authentication. The passwords chosen by users may not always provide adequate protection. Research shows that a significant amount of user passwords can be found in short times [18]. A research made by Bonneau which examines 70 million Yahoo users reveals that user passwords provide 20-bit security on an offline attack [19].

The process of finding an unknown password is called 'password cracking'. Password cracking is used by attackers in order to gain unauthorized access to systems, and also used by information security professionals to decide the security level of a system, or to gain legitimate access to a system whose password is lost / forgotten.

Password cracking can be online of offline. In online password cracking, different passwords are tried through connecting to remote server's service. This service can be anything which allows remote connection: FTP, HTTP, IMAP, rlogin, SSH, NTP, VNC, Remote Desktop, Samba, etc. are a few service / protocol examples which can be victim of password cracking attacks [20]. This attack can be detected and reacted by the system owners. The system may lock out the user after a specific number of failed logon attempts, or disable the system for a specified period, or may request additional information like 'Captcha' [21] or another factor of authentication (probably something you have, by sending an sms to cellular phone or sending push notification to an application on phone). In the offline attack, typically the hash of the password is gained, and password cracking is attempted to determine the password which gives the same hash value as the gained password hash.

2.3. Hash Function

The passwords are typically not stored as cleartext in systems databases, they are hashed, and hash values are stored. There are many different hash algorithms, some of the widespread ones being used are MD5 and SHA1 [22]. Hash functions provide a fixed output independent of the size of the input. Even one bit change in input may result significant changes in hash output. Hash functions are one-way, one can not decide input by just looking at the output. MD5 produces 128-bit output, while SHA-1 produces 160-

bit output. Below is a table which summarizes some hash algorithms and their respective outputs for the input word 'berker':

Example outputs of some hashing algorithms.

Algorithm	Hash
MD2	DB95F5FB244301298AF06A98EF73F0DA
MD4	A958EF83DB1CA9627C28A506B9B35988
MD5	B9B209C7129285852F61F807F332725A
SHA-1	F1FA12439950A7CB0FE46EE5B8911ADD C8E94104
SHA-2 (256)	FC6635A10054A2533935191F43A6906710 3FE33492288B454 98BC6E49312A9CD
SHA-2 (384)	D7952C2446A6893FE3AB25A68518A1000 65FAE0246CA305 077E91ABDD68291145FDD666569EFEBCEC1BE C36325FE7B57
SHA-2 (512)	D74BA971799C91E23B78A9DE42E294CF 9C2B9AA7714FDA 1AD43DD0BD55A57525C3D9EB690B1653C0482 DB95E31434 FE4329C51F6B73CB3D36C3A1E43BB30120D
RIPEMD-160	B63D71A24BE89C4E650ADFE71926421E 51EC89E1
LM	870BE7B49860AF40
NT	F70AC408F8922D97038256067D18BABB
MySQL323	7BED7155063CEB94
MySQLSHA1	B1234211B767DE84C67277071FDCCEC0 B055DCF2
Cisco PIX	wHJNRpyB.b4CKMp5
VNC Hash	EFFF9CB65095D1C2
Base64	YmVya2Vy

2.4. Password Cracking Attacks

To crack a password, different approaches can be employed. These can be summarized as, brute force attacks, dictionary attacks, hybrid attacks and pre computed hash (rainbow) tables [23]. In the brute force attack, all possible combinations are tried sequentially. In the dictionary attack, dictionary files are used, and the items in the dictionary are used. The dictionaries can be language words, names, most used passwords, etc. Hybrid attack is the combination of brute force and dictionary attack. A dictionary is used for base words, and some rules are applied to those words, and then those modified words are used in the attack. In the rainbow table attack, pre-computed hash tables are used, only a lookup to the is performed, if the hash value of the target password is on the table, then the password is cracked at that time, if not, then this attack is said to be not successful [24-26]. Some rainbow tables come free within a password cracking software, and some are sold separately. There are also some paid services that give you the original source value of the given hash.

Many methods are advised to increase the efficiency and effectiveness of password cracking attacks. Chou et al. used a dictionary based on character placement on keyboard, and cracked 114% more passwords compared to standard dictionary [27]. Schweitzer et al. used keyboard patterns and was able to find 2/11 more passwords than standard dictionary attacks [28]. Weir et al. modeled the used passwords using context-free grammars, created a dictionary and cracked 28% to 129% more passwords compared to John the Ripper program [29, 30]. Narayanan and Shmatikov used Markov models, which is used for language processing, for password cracking and got better results than Oechslin's rainbow attack [31]. Simon used Markov chains and cracked 80% of the passwords in less than 24 hours [32].

3. Methodology

3.1. Uniqueness of the study

The referenced studies propose one method to improve the performance of password cracking, and compare the performance with standard dictionary attacks or John the Ripper password cracking program [30]. In this research, one aim is to propose a method which compares the success criteria of the dictionary and word generation / modification rules. Another aim is to create a custom dictionary that will be used in the hybrid attack, by combining many different dictionaries and word generation / modification rules, whose success criteria is better than them all. The proposed success criteria and dictionary / rule combination method constitute the novelty brought to the field.

3.2. Structure of the hybrid algorithm

There are two success criterion for the password cracking algorithm. One is how early the algorithm managed to crack the password. The less tries the algorithm performs before revealing the password, the better its performance is. Let's name this number as r. If the algorithm can not crack the password, it will not try forever, after a certain number of times, it will try to crack the next password. Let's name maximum try number, as q. If we try to crack p different passwords, then our total try count becomes:

r(t)= r(1)+r(2)+⋯r(p).

This is our first success criteria. The lesser this number is, more successful the algorithm is. Second criteria is what percentage of the passwords are cracked in first q try. Let's call this number as s:

s = passwords revelaed / total passwords

This number is anything between 0 and 1, and can be shown as percentage as well. The greater this number is, the more successful the algorithm is.
After having those numbers for different algorithms, we have two performance numbers for every one of them. We will first normalize those numbers among the algorithms, let's call the normalized versions as r(t,n) and s(n). Then we will weigh those numbers according to needs of the password cracking operation. Let's give a and b for the weights of r(t,n) and s(n) respectively, given that a+b=1. These weights make the composed performance score as:

x = a * r(t,n) + b * s(n)

If the aim of the algorithm is to crack a specific password, then more weight would be given to 'a' coefficient, if the aim is to crack as many passwords as possible in a specific time frame, then more weight would be given to 'b' coefficient. Different weight scenarios are considered during performance tests.
After having all algorithms graded, a final word list / algorithm would be composed based on the findings, and this list's performance is expected to be superior to base algorithms.

As an example, let's have the following algorithms and corresponding word lists:

Algorithm / List 1: Combined word lists
Algorithm / List 2: Combined revelaed password lists
Algorithm / List 3: A custom made password list
Algorithm / List 4: Lists that use keyboard placement of letters
Algorithm / List 5: Lists which use frequency analysis

Let's name those algorithms as A1, A2, A3, A4 and A5. p = 1000 is the total number of target passwords, q = 100.000 is maximum try for each of the passwords. After running the script, let's have the corresponding scores for the algorithms:

Algorithm	r(t)	s
A1	62,173,468	17%
A2	36,225,779	37%
A3	28,113,016	16%
A4	58,330,849	64%
A5	36,514,250	32%

After normalizing the scores, we get:

Algorithm	r(t,n)	s(n)
A1	45.21706	26.5625
A2	77.605	57.8125
A3	100	25
A4	48.19579	100
A5	76.9919	50

If we get the a = b = 0.5 and normalize again, we get:

Algorithm	x	x(n)
A1	35.88978	48.43563
A2	67.70875	91.37742
A3	62.5	84.34787
A4	74.0979	100
A5	63.49595	85.69198

And finally, we will construct a final table based on the x(n) scores.

4. **Results**

4.1. Source and Destination Lists

Both source and destination lists are revealed passwords from real-world. As source list, following files are used. Line number counts are same as password numbers. These lists are used to decide which rules would be applied in the final algorithm.

```
Command Prompt - python
>>> countline_dir(folder = 'C:\\Users\\bt\\OneDrive\\TEZ\\PYTHON\\python codes\\leaked')
alypaa.txt            has     1,384 lines
carders.cc.txt        has     1,904 lines
elitehacker.txt       has       895 lines
facebook-pastebay.txt has        55 lines
facebook-phished.txt  has     2,442 lines
faithwriters.txt      has     8,347 lines
hak5.txt              has     2,351 lines
hotmail.txt           has     8,931 lines
myspace.txt           has    37,144 lines
porn-unknown.txt      has     8,089 lines
singles.org.txt       has    12,234 lines
tuscl.txt             has    38,820 lines
>>>
```

For the destination (target) lists, the following lists are used:

```
Command Prompt - python
Microsoft Windows [Version 10.0.17763.1339]
(c) 2018 Microsoft Corporation. All rights reserved.

c:\py>python
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 22:45:29) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> from ex import *
>>> countline_dir(folder = 'C:\\Users\\bt\\OneDrive\\TEZ\\PYTHON\\python codes\\targetlist')
phpbb.txt             has    184,389 lines
rockyou.txt           has 14,344,391 lines
>>>
>>>
```

All lists are preprocessed; first checked for uniqueness then randomized. Rockyou.txt list further divided into sub-lists, such that the number of entries in the lists would be same as the number of entries in phpbb.txt (which is 184.389).

```
Command Prompt - python
Microsoft Windows [Version 10.0.17763.1339]
(c) 2018 Microsoft Corporation. All rights reserved.

c:\py>python
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 22:45:29) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> from ex import *
>>> countline_dir(folder = 'C:\\Users\\bt\\OneDrive\\TEZ\\PYTHON\\python codes\\targetlist2')
rockyou-first 184389 entries.txt has     184,389 lines
rockyou-fourth 184389 entries.txt has     184,389 lines
rockyou-second 184389 entries.txt has     184,389 lines
rockyou-third 184389 entries.txt has     184,389 lines
>>>
```

4.2. Source lists unprocessed

In this scenario, source lists are unmodified and they are used as they are. The results are below:

Source: Unprocessed lists, **Target**: phpbb.txt

List name	Match count	Try count	Count per match	Match count (normalized)	1 / Count per match	1 / Count per match (Normalized)	Final Score
alypaa.txt	583	254,771,047	437,000	8	0.0000022883	70	39
carders.cc.txt	522	350,446,434	671,353	7	0.0000014895	45	26
elitehacker.txt	540	164,766,020	305,122	7	0.0000032774	100	54
facebook-pastebay.tx	11	10,141,083	921,917	0	0.0000010847	33	17
facebook-phished.txt	267	449,914,131	1,685,072	4	0.0000005934	18	11
faithwriters.txt	1777	1,529,731,126	860,850	24	0.0000011616	35	30
hak5.txt	271	433,109,153	1,598,189	4	0.0000006257	19	11
hotmail.txt	959	1,641,356,684	1,711,529	13	0.0000005843	18	16
myspace.txt	2545	6,788,598,640	2,667,426	35	0.0000003749	11	23
porn-unknown.txt	3277	1,475,560,891	450,278	45	0.0000022209	68	56
singles.org.txt	3048	2,232,201,994	732,350	42	0.0000013655	42	42
tuscl.txt	7275	6,972,158,311	958,372	100	0.0000010434	32	66

Source: Unprocessed lists, **Target**: rockyou.txt

List name	Match count	Try count	Count per match	Match count (normalized)	1 / Count per match	1 / Count per match (Normalized)	Final Score
alypaa.txt	993	19,851,908,939	19,991,852	5	0.0000000500	80	43
carders.cc.txt	1308	27,310,380,224	20,879,496	7	0.0000000479	77	42
elitehacker.txt	803	12,837,859,547	15,987,372	4	0.0000000625	100	52
facebook-pastebay.tx	29	788,940,623	27,204,849	0	0.0000000368	59	29
facebook-phished.txt	968	35,027,748,990	36,185,691	5	0.0000000276	44	25
faithwriters.txt	4823	119,710,883,524	24,820,834	24	0.0000000403	64	44
hak5.txt	568	33,722,905,999	59,371,313	3	0.0000000168	27	15
hotmail.txt	4042	128,090,576,955	31,689,900	20	0.0000000316	50	35
myspace.txt	18517	532,435,383,489	28,753,869	94	0.0000000348	56	75
porn-unknown.txt	6057	116,005,794,772	19,152,352	31	0.0000000522	83	57
singles.org.txt	7721	175,438,476,189	22,722,248	39	0.0000000440	70	55
tuscl.txt	19737	556,430,223,902	28,192,239	100	0.0000000355	57	78
TOTAL		**1,757,651,083,153**					

In both phpbb and rockyou lists, tuscl list got the maximum score.

4.3. Add character to the items of the list

! ? . @ * 0 and 1 characters are added to the end of the list items. As source list, tuscl.txt is used, which got the maximum score for both target lists, phpbb and rockyou.

The scores of characters:

```
Command Prompt
c:\py>
c:\py>ex2.py

Performance report:

testing with list: tuscl.txt and with special character: !
match count:   53 | try count: 7,156,382,826 | count per match: 135,026,091 | took 835.72 seconds
Final performance report:
{'tuscl.txt': [53, 7156382826]}
It took: 13 minutes and 56 seconds
testing with list: tuscl.txt and with special character: ?
match count:    2 | try count: 7,157,904,977 | count per match: 3,578,952,488 | took 817.64 seconds
Final performance report:
{'tuscl.txt': [2, 7157904977]}
It took: 27 minutes and 33 seconds
testing with list: tuscl.txt and with special character: .
match count:   15 | try count: 7,157,558,291 | count per match: 477,170,553 | took 810.86 seconds
Final performance report:
{'tuscl.txt': [15, 7157558291]}
It took: 41 minutes and 4 seconds
testing with list: tuscl.txt and with special character: @
match count:    2 | try count: 7,157,905,498 | count per match: 3,578,952,749 | took 806.53 seconds
Final performance report:
{'tuscl.txt': [2, 7157905498]}
It took: 54 minutes and 31 seconds
testing with list: tuscl.txt and with special character: *
match count:    8 | try count: 7,157,760,386 | count per match: 894,720,048 | took 827.49 seconds
Final performance report:
{'tuscl.txt': [8, 7157760386]}
It took: 68 minutes and 18 seconds
testing with list: tuscl.txt and with special character: 0
match count:  140 | try count: 7,154,444,280 | count per match: 51,103,173 | took 874.77 seconds
Final performance report:
{'tuscl.txt': [140, 7154444280]}
It took: 82 minutes and 53 seconds
testing with list: tuscl.txt and with special character: 1
match count: 1346 | try count: 7,117,571,384 | count per match: 5,287,943 | took 904.59 seconds
Final performance report:
{'tuscl.txt': [1346, 7117571384]}
It took: 97 minutes and 58 seconds

c:\py>
```

Normalized:

Append Char	Match count	Try count	Count per match	Match count (normalized)	1 / Count per match	1 / Count per match (Normalized)	Final Score
Char: !	53	7,156,382,826	135,026,091	4	0.0000000074	4	4
Char: ?	2	7,157,904,977	3,578,952,489	0	0.0000000003	0	0
Char: .	15	7,157,558,291	477,170,553	1	0.0000000021	1	1
Char: @	2	7,157,905,498	3,578,952,749	0	0.0000000003	0	0
Char: *	8	7,157,760,386	894,720,048	1	0.0000000011	1	1
Char: 0	140	7,154,444,280	51,103,173	10	0.0000000196	10	10
Char: 1	1346	7,117,571,384	5,287,943	100	0.0000001891	100	100
TOTAL		50,059,527,642					

Add 1 rule got the maximum score, add 0 rule is following.

4.4. Change character of the items of the list

Change character rules of ('a', '@'), ('o', '0'), ('i', '1'), ('s', '5') and ('e', '3') are applied. As source list, tuscl.txt is used, which got the maximum score for both target lists, phpbb and rockyou.
The scores of change operations:

```
Command Prompt
c:\py>ex3.py

Performance report:

testing with list: tuscl.txt and with replace characters: ('a', '@')
match count: 3930 | try count: 7,058,926,261 | count per match: 1,796,164 | took 1135.64 seconds
Final performance report:
{'tuscl.txt': [3930, 7058926261]}
It took: 18 minutes and 56 seconds
testing with list: tuscl.txt and with replace characters: ('o', '0')
match count: 5058 | try count: 7,030,318,084 | count per match: 1,389,940 | took 1141.53 seconds
Final performance report:
{'tuscl.txt': [5058, 7030318084]}
It took: 37 minutes and 57 seconds
testing with list: tuscl.txt and with replace characters: ('i', '1')
match count: 4946 | try count: 7,033,612,072 | count per match: 1,422,081 | took 1127.66 seconds
Final performance report:
{'tuscl.txt': [4946, 7033612072]}
It took: 56 minutes and 45 seconds
testing with list: tuscl.txt and with replace characters: ('s', '5')
match count: 5014 | try count: 7,035,828,002 | count per match: 1,403,237 | took 1236.95 seconds
Final performance report:
{'tuscl.txt': [5014, 7035828002]}
It took: 77 minutes and 22 seconds
testing with list: tuscl.txt and with replace characters: ('e', '3')
match count: 3861 | try count: 7,063,584,165 | count per match: 1,829,470 | took 1129.18 seconds
Final performance report:
{'tuscl.txt': [3861, 7063584165]}
It took: 96 minutes and 11 seconds
```

Normalized:

Replace Char	Match count	Try count	Count per match	Match count (normalized)	1 / Count per match	1 / Count per match (Normalized)	Final Score
('a', '@')	3930	7,058,926,261	1,796,164	78	0.0000005567	77	78
('o', '0')	5058	7,030,318,084	1,389,940	100	0.0000007195	100	100
('i', '1')	4946	7,033,612,072	1,422,081	98	0.0000007032	98	98
('s', '5')	5014	7,035,828,002	1,403,237	99	0.0000007126	99	99
('e', '3')	3861	7,063,584,165	1,829,470	76	0.0000005466	76	76

('o', '0'), ('i', '1') and ('s', '5') change operations got the maximum score.

4.5. All three scenarios combined

When we combine all three scenarios, we get:

```
Select Command Prompt
c:\py>ex4.2.py

Performance report:

testing with list: tuscl.txt | target list is: phpbb.txt
match count: 7275 | try count: 6,972,158,311 | count per match:   958,372 | took 625.75 seconds
testing with list: tuscl.txt and with special character: 1 | target list is: phpbb.txt
match count: 1042 | try count: 6,999,604,749 | count per match: 6,717,471 | took 1140.96 seconds
testing with list: tuscl.txt and with special character: ! | target list is: phpbb.txt
match count:   53 | try count: 7,153,986,034 | count per match: 134,980,869 | took 1064.93 seconds
testing with list: tuscl.txt and with special character: ? | target list is: phpbb.txt
match count:    2 | try count: 7,157,720,589 | count per match: 3,578,860,294 | took 1327.64 seconds
testing with list: tuscl.txt and with special character: . | target list is: phpbb.txt
match count:   15 | try count: 7,157,005,148 | count per match: 477,133,677 | took 1135.70 seconds
testing with list: tuscl.txt and with special character: @ | target list is: phpbb.txt
match count:    2 | try count: 7,157,905,498 | count per match: 3,578,952,749 | took 1060.29 seconds
testing with list: tuscl.txt and with special character: * | target list is: phpbb.txt
match count:    8 | try count: 7,157,391,618 | count per match: 894,673,952 | took 1039.95 seconds
testing with list: tuscl.txt and with special character: 0 | target list is: phpbb.txt
match count:  133 | try count: 7,145,801,723 | count per match: 53,727,833 | took 984.47 seconds
testing with list: tuscl.txt and with replace characters: ('a', '@') | target list is: phpbb.txt
match count:   20 | try count: 2,959,382,305 | count per match: 147,969,115 | took 593.33 seconds
testing with list: tuscl.txt and with replace characters: ('o', '0') | target list is: phpbb.txt
match count:  241 | try count: 2,137,398,754 | count per match: 8,868,874 | took 432.74 seconds
testing with list: tuscl.txt and with replace characters: ('i', '1') | target list is: phpbb.txt
match count:  146 | try count: 2,104,211,042 | count per match: 14,412,404 | took 421.82 seconds
testing with list: tuscl.txt and with replace characters: ('s', '5') | target list is: phpbb.txt
match count:   40 | try count: 2,053,808,274 | count per match: 51,345,207 | took 411.40 seconds
testing with list: tuscl.txt and with replace characters: ('e', '3') | target list is: phpbb.txt
match count:  110 | try count: 2,868,335,250 | count per match: 26,075,775 | took 573.99 seconds
Final performance report:
{'tuscl.txt': [9087, 69024709295]}
It took: 180 minutes and 13 seconds

c:\py>
```

Normalized:

OPERATION	match count	try count	count per match	1/cpm	mc (normalized)	1/cpm (normalized)	final score
none	7275	6,972,158,311	958,372	0.00000104343585952749	100.00	100.00	100.00
add 1	1042	6,999,604,749	6,717,471	0.00000014886554846527	14.32	14.27	14.29
add !	53	7,153,986,034	134,980,869	0.00000000740845729194	0.73	0.71	0.72
add ?	2	7,157,720,589	3,578,860,295	0.00000000027941856281	0.03	0.03	0.03
add .	15	7,157,005,148	477,133,677	0.00000000209584870904	0.21	0.20	0.20
add @	2	7,157,905,498	3,578,952,749	0.00000000027941134464	0.03	0.03	0.03
add *	8	7,157,391,618	894,673,952	0.00000000111772562226	0.11	0.11	0.11
add 0	133	7,145,801,723	53,727,833	0.00000001861232723152	1.83	1.78	1.81
replace ('a', '@')	20	2,959,382,305	147,969,115	0.00000000675816705608	0.27	0.65	0.46
replace ('o', '0')	241	2,137,398,754	8,868,874	0.00000011275387877390	3.31	10.81	7.06
replace ('i', '1')	146	2,104,211,042	14,412,404	0.00000006938467534189	2.01	6.65	4.33
replace ('s', '5')	40	2,053,808,274	51,345,207	0.00000001947601463407	0.55	1.87	1.21
replace ('e', '3')	110	2,868,335,250	26,075,775	0.00000003834977100393	1.51	3.68	2.59
TOTAL	**9087**	**69024709295**	**7,595,984**	**0.00000013164850808953**	**124.91**	**12.62**	**68.76**

4.6. Selection of Rules, (a, b) = (0.5, 0.5)

If we analyze the marginal utility of operations, we get the following table for values (a, b) = (0.5, 0.5):

OPERATION	match count	try count	count per match	1/cpm	mc (normalized)	1/cpm (normalized)	final score	improvement		
none	7275	6,972,158,311	958,372	0.00000104343585952749	100.00	100.00	100.00			
add 1	1042	6,999,604,749	6,717,471	0.00000014886554846527	14.32	14.27	14.29			
add !	53	7,153,986,034	134,980,869	0.00000000740845729194	0.73	0.71	0.72		a	0.5
add ?	2	7,157,720,589	3,578,860,295	0.00000000027941856281	0.03	0.03	0.03		b	0.5
add .	15	7,157,005,148	477,133,677	0.00000000209584870904	0.21	0.20	0.20			
add @	2	7,157,905,498	3,578,952,749	0.00000000027941134464	0.03	0.03	0.03			
add *	8	7,157,391,618	894,673,952	0.00000000111772562226	0.11	0.11	0.11			
add 0	133	7,145,801,723	53,727,833	0.00000001861232723152	1.83	1.78	1.81			
replace ('a', '@')	20	2,959,382,305	147,969,115	0.00000000675816705608	0.27	0.65	0.46			
replace ('o', '0')	241	2,137,398,754	8,868,874	0.00000011275387877390	3.31	10.81	7.06			
replace ('i', '1')	146	2,104,211,042	14,412,404	0.00000006938467534189	2.01	6.65	4.33			
replace ('s', '5')	40	2,053,808,274	51,345,207	0.00000001947601463407	0.55	1.87	1.21			
replace ('e', '3')	110	2,868,335,250	26,075,775	0.00000003834977100393	1.51	3.68	2.59			
TOTAL	**9087**	**69024709295**	**7,595,984**	**0.00000013164850808953**	**124.91**	**12.62**	**68.76**			
none + add 1	**8317**	**13971763060**	**1,679,904**	**0.00000059527204722007**	**114.32**	**57.05**	**85.69**	24.61%		
plus replace ('o', '0')	**8558**	**14153590783**	**1,653,843**	**0.00000060465221378868**	**117.64**	**57.95**	**87.79**	27.68%		
plus replace ('i', '1')	**8704**	**16257801825**	**1,867,854**	**0.00000053537372971392**	**119.64**	**51.31**	**85.48**	24.31%		
plus replace ('e', '3')	**8814**	**19126137075**	**2,169,972**	**0.00000046083534617771**	**121.15**	**44.17**	**82.66**	20.21%		
plus add 0	8947	26271938798	**2,936,396**	**0.00000034055347299610**	**122.98**	**32.64**	**77.81**	13.16%		
plus replace ('s', '5')	8987	28325747072	**3,151,858**	**0.00000031727318531640**	**123.53**	**30.41**	**76.97**	11.94%		
plus add !	9040	35479733106	**3,924,749**	**0.00000025479334844464**	**124.26**	**24.42**	**74.34**	8.11%		
plus replace ('a', '@')	9060	38439115411	**4,242,728**	**0.00000023569741142918**	**124.54**	**22.59**	**73.56**	6.98%		
plus add .	9075	45596120559	**5,024,366**	**0.00000019903009047134**	**124.74**	**19.07**	**71.91**	4.58%		
plus add *	9083	52753512177	**5,807,939**	**0.00000017217810957353**	**124.85**	**16.50**	**70.68**	2.78%		
plus add ?	9085	59911232766	**6,594,522**	**0.00000015164101255409**	**124.88**	**14.53**	**69.71**	1.37%		
plus add @	9087	67069138264	**7,380,779**	**0.00000013548705463057**	**124.91**	**12.98**	**68.95**	0.27%		

Here maximum score is for the rule ‘add 1’ (14.29). Second score is for the rule replace (‘o’ with ‘0’) (7.06), and third score is for the rule replace (‘i’ with 1) (4.33).
Here, ‘add 1’ rule increased the overall score by 24.61%, on top of this, replace (‘o’ with ‘0’) rule increased this number to 27.68%, but replace (‘i’ with 1) rule decreased this number to 24.31%. So, we can conclude that the rules which must be selected are ‘add 1’ and replace (‘o’ with ‘0’).

4.7. Selection of Rules, (a, b) = (0.8, 0,2)

OPERATION	match count	try count	count per match	1/cpm	mc (normalized)	1/cpm (normalized)	final score	improvement		
none	7275	6,972,158,311	958,372	0.00000104343585952749	100.00	100.00	100.00			
add 1	1042	6,999,604,749	6,717,471	0.00000014886554846527	14.32	14.27	14.31			
add !	53	7,153,986,034	134,980,869	0.00000000740845729194	0.73	0.71	0.72		a	0.8
add ?	2	7,157,720,589	3,578,860,295	0.00000000027941856281	0.03	0.03	0.03		b	0.2
add .	15	7,157,005,148	477,133,677	0.00000000209584870904	0.21	0.20	0.21			
add @	2	7,157,905,498	3,578,952,749	0.00000000027941134464	0.03	0.03	0.03			
add *	8	7,157,391,618	894,673,952	0.00000000111772562226	0.11	0.11	0.11			
add 0	133	7,145,801,723	53,727,833	0.00000001861232723152	1.83	1.78	1.82			
replace ('a', '@')	20	2,959,382,305	147,969,115	0.00000000675816705608	0.27	0.65	0.35			
replace ('o', '0')	241	2,137,398,754	8,868,874	0.00000011275387877390	3.31	10.81	4.81			
replace ('i', '1')	146	2,104,211,042	14,412,404	0.00000006938467534189	2.01	6.65	2.94			
replace ('s', '5')	40	2,053,808,274	51,345,207	0.00000001947601463407	0.55	1.87	0.81			
replace ('e', '3')	110	2,868,335,250	26,075,775	0.00000003834977100393	1.51	3.68	1.94			
TOTAL	**9087**	**69024709295**	**7,595,984**	**0.00000013164850808953**	**124.91**	**12.62**	**102.45**			
none + add 1	**8317**	**13971763060**	**1,679,904**	**0.00000059527204722007**	**114.32**	**57.05**	**102.87**	0.41%		
plus replace ('o', '0')	**8558**	**14153590783**	**1,653,843**	**0.00000060465221378868**	**117.64**	**57.95**	**105.70**	3.17%		
plus replace ('i', '1')	**8704**	**16257801825**	**1,867,854**	**0.00000053537372971392**	**119.64**	**51.31**	**105.98**	3.44%		
plus replace ('e', '3')	**8814**	**19126137075**	**2,169,972**	**0.00000046083534617771**	**121.15**	**44.17**	**105.76**	3.23%		
plus add 0	8947	26271938798	**2,936,396**	**0.00000034055347299610**	**122.98**	**32.64**	**104.91**	2.41%		
plus replace ('s', '5')	8987	28325747072	**3,151,858**	**0.00000031727318531640**	**123.53**	**30.41**	**104.91**	2.40%		
plus add !	9040	35479733106	**3,924,749**	**0.00000025479334844464**	**124.26**	**24.42**	**104.29**	1.80%		
plus replace ('a', '@')	9060	38439115411	**4,242,728**	**0.00000023569741142918**	**124.54**	**22.59**	**104.15**	1.66%		
plus add .	9075	45596120559	**5,024,366**	**0.00000019903009047134**	**124.74**	**19.07**	**103.61**	1.13%		
plus add *	9083	52753512177	**5,807,939**	**0.00000017217810957353**	**124.85**	**16.50**	**103.18**	0.72%		
plus add ?	9085	59911232766	**6,594,522**	**0.00000015164101255409**	**124.88**	**14.53**	**102.81**	0.35%		
plus add @	9087	67069138264	**7,380,779**	**0.00000013548705463057**	**124.91**	**12.98**	**102.52**	0.07%		

Here maximum score is for the rule 'add 1' (14.31). Second score is for the rule replace ('o' with '0') (4.81), and third score is for the rule replace ('i' with 1) (2.94).

Here, 'add 1' rule increased the overall score by 0.41%, on top of this, replace ('o' with '0') rule increased this number to 3.44%, and replace ('i' with 1) rule inreased this number to 3.44%. So, we can conclude that the rules which must be selected are 'add 1', replace ('o' with '0') and replace ('i' with '1').

4.8. Selection of Rules, (a, b) = (0.2, 0.8)

OPERATION	match count	try count	count per match	1/cpm	mc (normalized)	1/cpm (normalized)	final score	improvement		
none	7275	6,972,158,311	958,372	0.00000104343585952749	100.00	100.00	100.00			
add 1	1042	6,999,604,749	6,717,471	0.00000014886554846527	14.32	14.27	14.28			
add !	53	7,153,986,034	134,980,869	0.00000000740845729194	0.73	0.71	0.71		a	0.2
add ?	2	7,157,720,589	3,578,860,295	0.00000000027941856281	0.03	0.03	0.03		b	0.8
add .	15	7,157,005,148	477,133,677	0.00000000209584870904	0.21	0.20	0.20			
add @	2	7,157,905,498	3,578,952,749	0.00000000027941134464	0.03	0.03	0.03			
add *	8	7,157,391,618	894,673,952	0.00000000111772562226	0.11	0.11	0.11			
add 0	133	7,145,801,723	53,727,833	0.00000001861232723152	1.83	1.78	1.79			
replace ('a', '@')	20	2,959,382,305	147,969,115	0.00000000675816705608	0.27	0.65	0.57			
replace ('o', '0')	241	2,137,398,754	8,868,874	0.00000011275387877390	3.31	10.81	9.31			
replace ('i', '1')	146	2,104,211,042	14,412,404	0.00000006938467534189	2.01	6.65	5.72			
replace ('s', '5')	40	2,053,808,274	51,345,207	0.00000001947601463407	0.55	1.87	1.60			
replace ('e', '3')	110	2,868,335,250	26,075,775	0.00000003834977100393	1.51	3.68	3.24			
TOTAL	**9087**	**69024709295**	**7,595,984**	**0.00000013164850808953**	**124.91**	**12.62**	**35.07**			
none + add 1	**8317**	**13971763060**	**1,679,904**	**0.00000059527204722007**	**114.32**	**57.05**	**68.50**	95.31%		
plus replace ('o', '0')	**8558**	**14153590783**	**1,653,843**	**0.00000060465221378868**	**117.64**	**57.95**	**69.89**	99.25%		
plus replace ('i', '1')	**8704**	**16257801825**	**1,867,854**	**0.00000053537372971392**	**119.64**	**51.31**	**64.98**	85.25%		
plus replace ('e', '3')	**8814**	**19126137075**	**2,169,972**	**0.00000046083534617771**	**121.15**	**44.17**	**59.56**	69.82%		
plus add 0	8947	26271938798	**2,936,396**	**0.00000034055347299610**	**122.98**	**32.64**	**50.71**	44.57%		
plus replace ('s', '5')	8987	28325747072	**3,151,858**	**0.00000031727318531640**	**123.53**	**30.41**	**49.03**	39.79%		
plus add !	9040	35479733106	**3,924,749**	**0.00000025479334844464**	**124.26**	**24.42**	**44.39**	26.55%		
plus replace ('a', '@')	9060	38439115411	**4,242,728**	**0.00000023569741142918**	**124.54**	**22.59**	**42.98**	22.53%		
plus add .	9075	45596120559	**5,024,366**	**0.00000019903009047134**	**124.74**	**19.07**	**40.21**	14.63%		
plus add *	9083	52753512177	**5,807,939**	**0.00000017217810957353**	**124.85**	**16.50**	**38.17**	8.83%		
plus add ?	9085	59911232766	**6,594,522**	**0.00000015164101255409**	**124.88**	**14.53**	**36.60**	4.35%		
plus add @	9087	67069138264	**7,380,779**	**0.00000013548705463057**	**124.91**	**12.98**	**35.37**	0.84%		

Here maximum score is for the rule 'add 1' (14.28). Second score is for the rule replace ('o' with '0') (9.31), and third score is for the rule replace ('i' with 1) (5.72).

Here, 'add 1' rule increased the overall score by 95.31%, on top of this, replace ('o' with '0') rule increased this number to 99.25%, but replace ('i' with 1) rule decreased this number to 85.25%. So, we can conclude that the rules which must be selected are 'add 1' and replace ('o' with '0').

5. Conclusion and Future Work

Password usage is a very prevalent security control used to protect information assets. To assess the effectiveness of this control, security professionals may use password cracking tools and algorithms. This study proposes a method to assess success criterion of different password cracking algorithms; and based on the assessment scores, proposes another method to combine these algorithms in a specific order, to increase the likelihood of password cracking in a most efficient and effective manner. In this study it is shown that, based on the aim of the password cracking (to crack a specific password vs to crack as many passwords as possible in a specific time frame), different strategies may be followed. And this study offers a way to determine which algorithms / lists to include in the final cracking endeavor, and which ones to exclude.

This work studied add and replace rules. Further study may include keyboard patterns, frequency analysis, etc. Also this study used (a,b) weights of (0.2, 0.8), (0.5, 0.5) and (0.8, 0.2). Different weights, different source lists, different target lists, different add / replace rules, different password try cutoff points may be tried. If data can be obtained, user demographics, target system features, password length, etc. may be used as a decision point in the final algorithm.

References

[1] Samonas, S. and D. Coss, THE CIA STRIKES BACK: REDEFINING CONFIDENTIALITY, INTEGRITY AND AVAILABILITY IN SECURITY. Journal of Information System Security, 2014. 10(3).

[2] Harris, S., CISSP All-in-One Exam Guide, 6th Edition. 2012: McGraw-Hill Osborne Media. 1456.

[3] The 2012 Data Breach Investigations Report - VerizonBusiness. 2013 2020/04/22/; Available from: http://www.verizonbusiness.com/resources/reports/rp_data-breach-investigations-report-2012_en_xg.pdf.

[4] Cost of a Data Breach Report 2020 | IBM. 2020 2020/08/18/; Available from: https://www.ibm.com/security/digital-assets/cost-data-breach-report/#.

[5] Swinhoe, D. The 15 biggest data breaches of the 21st century. CSO Online 2020 2020/04/17/; Available from: https://www.csoonline.com/article/2130877/the-biggest-data-breaches-of-the-21st-century.html.

[6] Top 10 Largest Data Breaches. 2020 2020/04/22/; Available from: https://www.securitymagazine.com/articles/92201-top-10-largest-data-breaches.

[7] Kiesnoski, K. 5 of the biggest data breaches ever. CNBC 2019 2020/08/22/; Available from: https://www.cnbc.com/2019/07/30/five-of-the-biggest-data-breaches-ever.html.

[8] Adobe Breach Impacted At Least 38 Million Users — Krebs on Security. 2020 2020/08/22/; Available from: https://krebsonsecurity.com/2013/10/adobe-breach-impacted-at-least-38-million-users.

[9] Scott, C. Protecting Our Members. 2020 2020/08/22/; Available from: https://blog.linkedin.com/2016/05/18/protecting-our-members.

[10] Sanger, D.E., et al. Marriott Data Breach Is Traced to Chinese Hackers as U.S. Readies Crackdown on Beijing. N.Y. Times 2018 2020/08/22/; Available from: https://www.nytimes.com/2018/12/11/us/politics/trump-china-trade.html.

[11] Newcomer, E. Uber Paid Hackers to Delete Stolen Data on 57 Million People. Bloomberg 2017 2020/08/22/; Available from: https://www.bloomberg.com/news/articles/2017-11-21/uber-concealed-cyberattack-that-exposed-57-million-people-s-data.

[12] McMillan, R. FriendFinder Investigates Report of Breached Accounts. WSJ 2016 2020/08/22/; Available from: https://www.wsj.com/articles/friendfinder-investigates-report-of-breached-accounts-1479160660.

[13] Swinhoe, D. The 15 worst data security breaches of the 21st Century - CSO Online - Security and Risk. 2012 2020/08/22/; Available from: http://www.csoonline.com/article/700263/the-15-worst-data-security-breaches-of-the-21st-century.

[14] Adee, S., The Hunt For The Kill Switch. Spectrum, IEEE, 2008. 45(5): p. 34-39.

[15] Metz, C., AAA protocols: authentication, authorization, and accounting for the Internet. IEEE Internet Computing, 1999. 3(6): p. 75-79.

[16] Paolini, A., et al., Authentication, Authorization, and Accounting, in Towards Interoperable Research Infrastructures for Environmental and Earth Sciences. 2020, Springer. p. 247-271.

[17] Aloul, F., S. Zahidi, and W. El-Hajj. Two factor authentication using mobile phones. in 2009 IEEE/ACS International Conference on Computer Systems and Applications. 2009. IEEE.

[18] El Emam, K., K. Moreau, and E. Jonker, How Strong are Passwords Used to Protect Personal Health Information in Clinical Trials? Journal of Medical Internet Research, 2011. 13(1): p. 13-22.

[19] Bonneau, J. and Ieee, The science of guessing: analyzing an anonymized corpus of 70 million passwords, in 2012 Ieee Symposium on Security and Privacy. 2012, Ieee: New York. p. 538-552.

[20] McClure, S., et al., Hacking exposed: network security secrets and solutions. 2009: McGraw-Hill.

[21] Bursztein, E., M. Martin, and J. Mitchell. Text-based CAPTCHA strengths and weaknesses. in Proceedings of the 18th ACM conference on Computer and communications security. 2011.

[22] Ducloyer, S., et al. Hardware implementation of a multi-mode hash architecture for MD5, SHA-1 and SHA-2. in Proceedings on the Design and Architectures for Signal and Image Processing Workshop (DASIP'07). 2007. Citeseer.

[23] Easttom, W., II, Computer Security Fundamentals (2nd Edition). 2011: Pearson IT Certification. 352.

[24] Kalenderi, M., et al. Breaking the GSM A5/1 cryptography algorithm with rainbow tables and high-end FPGAS. in 22nd International conference on field programmable logic and applications (FPL). 2012. IEEE.

[25] Theocharoulis, K., I. Papaefstathiou, and C. Manifavas. Implementing rainbow tables in high-end fpgas for super-fast password cracking. in 2010 International Conference on Field Programmable Logic and Applications. 2010. IEEE.

[26] Kumar, H., et al. Rainbow table to crack password using MD5 hashing algorithm. in 2013 IEEE Conference on Information & Communication Technologies. 2013. IEEE.

[27] Chou, H.C., et al., PASSWORD CRACKING BASED ON SPECIAL KEYBOARD PATTERNS. International Journal of Innovative Computing Information and Control, 2012. 8(1A): p. 387-402.

[28] Schweitzer, D., et al., Visualizing keyboard pattern passwords. Information Visualization, 2011. 10(2): p. 127-133.

[29] Weir, M., et al. Password cracking using probabilistic context-free grammars. in 2009 30th IEEE Symposium on Security and Privacy. 2009. IEEE.

[30] John the Ripper password cracker. 2020 2020/08/22/; Available from: https://www.openwall.com/john.

[31] Narayanan, A. and V. Shmatikov, Fast dictionary attacks on passwords using time-space tradeoff, in Proceedings of the 12th ACM conference on Computer and communications security. 2005, ACM: Alexandria, VA, USA. p. 364-372.

[32] Marechal, S., Advances in password cracking. Journal in Computer Virology, 2008. 4(1): p. 73-81.