Password Strength Analysis Against Brute Force and Dictionary Attacks

Shifa Salsabiila 13519106
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
13519106@std.stei.itb.ac.id

***Abstract*—The dilemma of choosing strong, yet hard-to-remember passwords or weak but easy-to-remember passwords has long haunted internet users. This paper shows that many people tend to reside to choosing the latter option. This fact is concerning and is an indicator that many internet users today are still lacking the awareness of the dangers of a weak password and how they are at risk to it. This paper explains how brute force attacks and dictionary attacks are executed as direct implementations of the brute force algorithm and regular expressions. A strong password should be tested against and be able to hold off all kinds of attacks.**

***Keywords*—password strength, password cracking, brute force attack, dictionary attack, regular expressions**

I. Introduction

Passwords has been and still is the main authentication mechanism for most websites and applications. Although it works, it is not completely flawless. In the past, there has been many incidents related to password cracking and data breach. It is highly encouraged that internet users start putting more thought and concern into choosing stronger passwords for their accounts. This includes a password of length greater than 8 characters, containing multiple assortments of lowercase and uppercase letters, numbers, and symbols, and for it not to be used for multiple accounts. A strong password is like a double-sided sword. On one side, it is good because it is very hard for crackers to guess, but on the other side, it is also more difficult to remember.

As this is the case, a lot of people tend to reside to either choosing weak passwords that are easier to rememeber, especially as they start owning more accounts and having different passwords for each and every one of them, or choosing a single strong universal password to use for all their accounts. It is surprisisng how many people still use common passwords, such as “password” or “password1” for their accounts. Raising awareness regarding the importance of password strength and how easy it is to crack a weak password should be conducted more often. The consequence of a password breach for a user may lead to a loss in time, money, personal information, and in extreme cases, possibly even a person’s physical safety.

II. Brute Force Algorithm

Brute force algorithm is a problem solving metehod that is known to be able to solve almost any problem. However, its approach is considered inefficient, as it takes up a lot of computation time. A brute force algorithm will attempt to try out every possible solution to a problem. Theoretically, if a certain problem has a known set of solution range that can be enumerated and written out as a set of rules, then that problem can be solved using a brute force approach.

A problem that comes along the brute force algorithm is that most of the time, for larger problems, it is not implementable. Perhaps in theory, a certain problem can be solved using the brute force algorithm, however today, most computers are not powerful enough to carry out the algorithm for larger problems within a reasonable amount of time. Therefore, this algorithm is rarely used to solve problems that have other known more efficient solutions that can be used to solve them.

An attempt to slightly improve the brute force algorithm, while still implementing the same concepts is to add heuristic rules to the set of possible solutions that is completely dependent on each specific problem. This is done to further minimize the set of possible solutions, hence also minimizing the number of solutions that needs to be checked.

The time complexity of the brute force algorithm highly depends on the problem. Two of the most popular problems that can be solved using brute force algorithm is the Knapsack problem and the assignement problem. The time complexity for the Knapsack problem using a brute force approach is $O(2^n)$ and for the assignment problem, it is $O(n!)$. The approach used to solve these two problems is later adapted to solve many other problems, including the password cracking method that will be further discussed in this paper.

III. Regular Expression

Regular expression is a sequence of characters that is used to denote a certain pattern. It is usually used to find matches of a pattern within a larger text. Below is a summary of the common regular expression syntax.

i. Predefined Characters

Expression	Description
`.`	Any character except newline
`\d`	Any digit [0-9]
`\w`	Any word character [a-zA-Z0-9_]
`\s`	Space character
`\D`	Any non digit characters
`\W`	Any non word characters
`\S`	Any non space characters

Table 3.1 Regular expression – predefined characters

ii. Characters

Expression	Description
`[abc]`	a, b, or c
`[^abc]`	Any character except for a, b, and c
`[a-z]`	Any lowercase alphabet characters
`[a-zA-Z]`	Any lowercase and uppercase alphabet characters
`[0-9]`	Any numerical digit
`(abc)`	The pattern ‘abc’ in that order
`[a-z&&[^s-u]]`	Any lowercase alphabet charater except s, t, and u.

Table 3.2 Regular expressions - characters

iii. Boundary Matchers

Expression	Description
`^`	Start of line
`$`	End of line
`\b`	Word boundary
`\B`	Not a word boundary
`\A`	Beginning of string
`\G`	Beginning of string or end of previous match
`\Z`	End of string
`?=`	Look ahead

Table 3.3 Regular expressions – boundary matchers

iv. Quantifiers

Expression	Description
`*`	Zero or more appearance
`+`	One or more appearance
`?`	Zero or one appearance
`{n}`	Exactly n appearance
`{n, m}`	Appearance between n to m times (inclusive)
`{, m}`	Appearance at most m times
`{n, }`	Appearance at least n times

Table 3.4 Regular expressions – quantifiers

IV. Dataset Analysis

In the past, there have been several cases of data breach which resulted in multiple account passwords being released to the public. This is especially bad when it happens to companies who store their user passwords in plain text, not encypted using any hashing algorithms. A major data breach that may have changed the world of password cracking was one that occurred in 2009 to a company called *RockYou*.

RockYou started off as a company that provides slideshow services for other websites, but gained its major success in becoming a widget provider for many other multinational companies. It was known to be the main widget maker for facebook, as rated by the number of total installations. In 2009, the company experienced a data breach, exposing more thatn 32 million unencrypted user account passwords from the service itself and also from companies that use its services such as Facebook and MySpace. The password list contained over 14 million lines of real unique passwords. The sheer size of this list made it the largest known password list to exist at the time. To make it worse, the current CEO of the company failed to inform the public of this incident and thus remained hidden until years later. This incident provided a massive new list of passwords that can be made of use by crackers to enhance their cracking success rate.

Using regular expressions, this paper has summarized some of the most common patterns being used in passwords based on the RockYou password list.

```
regex = "^[A-Z]+$" #To be changed for
different tests
pattern = re.compile(regex)

found = 0
lines = 0

for line in open("rockyou.txt",
errors="ignore"):
    for match in re.finditer(pattern,
line):
        found += 1
    lines += 1
```

The purpose of this block of code is to test out various regular expression patterns to see which yields the highest percentage result when being tested against the RockYou password list. The given regex variable above filters only passwords containing all uppercase alphabet characters of any length and nothing else. When testing, the regex variable was altered multiple times to test for different patterns as well, such as the use of combiantions between lowercase, uppercase alphabets and numeric characters, among many others.

The results are summarized in the table below:

No	regex	Percentage Yield
1	`^[a-z]+$`	25.977%
2	`^[A-Z]+$`	1.603%
3	`^[0-9]+$`	16.360%
4	`^[a-z0-9]+$`	84.687%
5	`^[A-Z0-9]+$`	20.803%
6	`^[A-Za-z]+$`	28.690%
7	`^[A-Za-z0-9]+$`	92.906%
8	`^[A-Za-z0-9]{,8}$`	51.068%
9	`^[a-z]+[0-9]+$`	32.906%
10	`^[A-Za-z]+[0-9]+$`	37.224%
11	`[A-Za-z]+\d\d\d\d`	13.419%
12	`^(?=.*[A-Z])(?=.*[a-z])(?=.*[0-9])(?=.*[^A-Za-z0-9])`	3.036%

Table 4.1 Common patterns from RockYou password list

Another analysis was done to see the average length of the passwords from the RockYou list.

```
totalLength = 0
lines = 0

for line in open("rockyou.txt",
errors="ignore"):
    totalLength += len(line)
    lines += 1
```

This block of code yielded that the average length of passwords from the RockYou password list is 9.74846 characters. From these results, there are several points that can be concluded about the common patterns and trends in user passwords. This average is not bad, being judged by solely the password length. Longer passwords do take greater time to crack, as will be explained in more detail later.

However, password length alone cannot determine the strength of a password. As mentioned earlier, a good password should contain various characters within them, and ideally, this would include a combination of lowercase letter, uppercase letter, number, and a symbol. Based on the test results using regex 12 in Table 4.1, only 3.036% of the passwords from the list is considered strong.

Referring to Table 4.1, regex 1 shows that around a quarter of the passwords from the list only consists of lowercase letters. This type of password is considered the weakest type and is very easy to crack. The fact that over a quarter of passwords from the list fall into this category is concerning, because that is a large portion of people and hence shows that the awareness of password vulnerability is still not clear to many people. Below is a more descriptive venn diagram of the specific distribution between common password patterns containing lowercase, uppercase alphabets, and numeric characters.

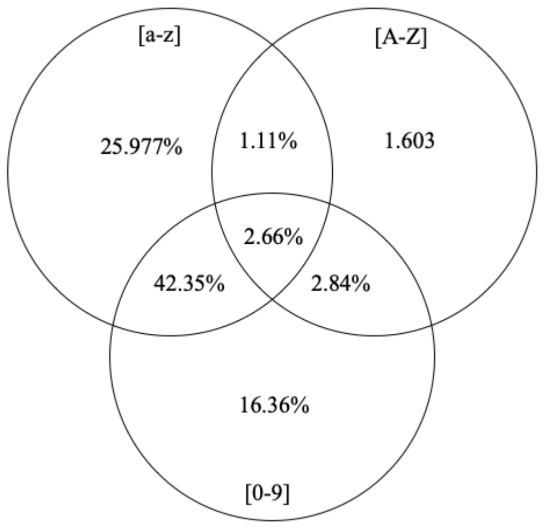

Fig 4.1 Venn diagram of common password pattern

As it can be seen from the venn diagram, most passwords contain either only lowercase letters, only numbers, or only a combination of lowercase letters and numbers. Unfortunately, today, with all the dictionaries available, this is not enough to be a strong password.

Another pattern that was tested out was regex 11. This pattern yielded around 13%, which is quite significant for a specific pattern. This pattern denotes a series of letters followed by 4 numeric digits at the end. This is a common password pattern, as people tend to use their name or a word followed by a date as a password and its popularity makes it a vulnerable password as well, despite the possibility that it contains a combination of lowercase, uppercase letter, and a numeric digit.

V. Password Cracking Methods

A. *Brute Force Attack*

As can be seen from its name, a brute force attack makes use of the brute force algorithm. How it works is that it tries out every possible solution until it has found a match or it has checked all of the possible solutions. In order to execute a brute force attack, first of all, one must define the set of possible solutions, usually denoted using regular expressions or other scripts. This set of possible solutions can have rules as general as "check for all combinations of lowercase letters, uperrcase letters, and numbers" `^[a-zA-Z0-9]+$` or can be as specific as "check for all combinations of strings containing a series of lowercase or uppercase letters followed by 4 digits, with a minimum length of 6 and a maximum length of 8 characters" `([A-Za-z]+\d\d\d\d){6,8}`.

The attempt to make the possible solution smaller by applying certain rules is the example of implementing heuristics in a brute force algorithm. The time complexity for a brute force attack depends on the heuristic rules applied, but is in average $O(m^n)$, where m is the number of characters being checked and n is the length of the string.

Consider the following example: A brute force is trying to be done with the target password being of length 4 and only consisting of lowercase letters. Hence, we can get the maximum amount of tries to be 26^4, as there are 26 characters of lowercase letters and there are 4 slots. This means that every slot can be filled in with a single character from the alphabet and repetitions are allowed. All the possible cobinations of the 4 letters form the possible solution set.

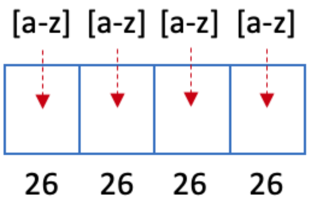

Fig 5.1 Combination for brute force algorithm illustration

From this, it can be derived that the brute force attack method speed is highly dependent on m and n. Here, n has a greater effect in the time escalation, which means that in general, making a password longer makes it harder to crack than adding more symbols to it. Below is a representation of the number of checks that has to be done based on the number of characters and the length of the string.

	Possible characters		
Length	`[a-z]`	`[a-zA-Z]`	`[a-zA-Z0-9]`
0	1	1	1
1	26	52	62
2	676	2704	3844
3	17576	140608	238328
4	456976	7311616	14776336
5	11881376	380204032	916132832
6	308915776	1.9771E+10	5.68E+10
7	8031810176	1.0281E+12	3.5216E+12
8	2.0883E+11	5.346E+13	2.1834E+14
9	5.4295E+12	2.7799E+15	1.3537E+16
10	1.4117E+14	1.4456E+17	8.393E+17

Table 5.1 Number of checks comparison

In passwords of longer lengths, the role of heuristics become more important. Take a password of length 8 for example. Without heuristics, if we were to attempt a brute force attack that assumes a password of length 8 and containing lowercase letters, uppercase letters, and numbers, from Table 5.1, it can be derived that the total number of checks is at 2.1834E+14 checks at maximum. However, this approach assumes that lowercase letters, uppercase letters, and numbers have equal probabilities of occupying all slots. However, in reality, based on social analysis, we have gathered up information that people tend to start their passwords with uppercase letters, followed by several lowercase letters (in this case 3, because we are searching for a string of length 8) and ending with a date, consisting of 4 numbers.

With this assumption in mind, we can construct a narrower possible solution set with the regular expression

`^[A-Z]([a-z]{3})\d\d\d\d$.`

The total number of checks for this pattern is:

$$26 \times 26^3 \times 10^4 = 456976000$$

As it can be seen, this solution is around 100 million times faster. When designed correctly, a heuristic rule can greatly improve the search of a brute force algorithm.

The speed at which passwords are being cracked using the brute force algorithm also very strongly depends on the computer the cracking program is being run on. A computer running on multiple GPUs will be able to crack passwords at a much higher rate than one running on a single CPU. To give an approximation of password cracking time, suppose a computer is capable of running 1 million checks per second. With this, to check a password consisting of only lowercase letters of length 5, the computer only needs 11 seconds. However, when the password length increase to 10, it will now need 1.4117E+8 seconds, or equivalent to 39213 hours or 1633 days. Hence, this method of password cracking becomes unrealistic for passwords of character length greater than 8.

B. *Dictionaries*

As it can be seen from the previous section, the brute force attack is not at all effective against lengthy passwords or passwords with a greater variation of characters within them. Hence, exists another password cracking method, that is to use *dictionaries*. Dictionaries in password cracking is simply a list of known common passwords, usually a compilation of passwords from data breaches. In fact, the RockYou password list that was discussed in previous chapters can actually be used as a dictionary in dictionary attacks.

A dictionary attack works by first of all trying out all the passwords in the dictionary to see if a match is found. Generally speaking, this part of a dictionary attack is also an implementatin of the brute force algorithm, only here, the possible solution set is all the words or strings within the dictionary. However, if no match is successfully found by matching all the strings exactly as they are, certain modifications are done towards the strings in the dictionary, and the testing process is repeated. These modifications are carried out bsaed on certain rules that had been predefined prior to the attack. These rules can also be seen as heuristic rules that in this case attempts to expand the possible solution set.

It is very common for people to substitute certain characters for other characters in passwords. Some of the most common substitutions include replacing an ‘A’ with ‘@’ or ‘4’, or replacing an ‘E’ with ‘3’. A different rule may be to capitalise the first letter of each string. In dictionary attacks, it is very easy to, with the help of regular expressions, implement these substitution rules. Therefore, using the password “p@ssw0rd” instead of “password” does not actually make the password any more secure, because of how easy it is to replace the latter with the implementation of the right rules. If the first round of parsing fails to yield a match, then new parses will be executed with the implementation of these different rules.

A comprehensive dictionary coupled with a good rule set can be aa very effective method of password cracking, considering a lot of user account passwords still follow a common pattern that can be or has been translated into a rule in a rule set.

VI. Suggestions For Password Security

A. *Suggestions for Developers*

As passwords are still the main authentication system until today, developers providing such authentication system for their clients should treat it with utmost care. Learning from past experiences, such as the RockYou incident, developers should know not to store user passwords in plain text. Passwords should always be hashed prior to storing. Hashing is the process of transforming a text into another value of fixed size. It is a one-way process, meaning that once a password has been hashed, then there is no way to return it to its original value. The hashing function is constant, in a sense that the same input passed through the same hashing function will always produce the same output. Therefore, upon logging in, the string that the user entered in the password field will be passed into the same hashing function and the output will be compared to the one that is stored in the database. Hence, not even the administrator has access to see any user passwords.

Hashing has two main benefits. The first one, is that if a data breach were to happen and user information, including passwords, get exposed to the public, it will still be in a hashed form, meaning that unless it iss cracked, then it will not be very useful to anyone. The second one is that it increases the time required when an attack is happening. Good hashing functions are designed to be slow. This is because if for example, a data breach occurred to a database storing hashed passwords, then to crack the passwords, one must use one of the methods above, with the addition of passing it through the hashing functions before comparing the strings. Again, hashing rate depends on both the hashing function and CPU or GPU strength of a computer.

For a rough comparison, here is a couple of hashing algorithm to hash rate data on a PC with two ATI Radeon 7970 graphic cards.

Hashing Algorithm	**Rate (Hashes/second)**
MD5	23070.07 M/s
SHA-1	7973.8 M/s
SHA-256	3112.0 M/s
SHA-512	267.1 M/s
NTLM	44035.3 M/s

Table 6.1 Hash rate comparison

From Table 6.1, it can be seen that hashing algorithms such as NTLM or MD5 are not very good, because they have a high hash rate. This means that a computer would be able to produce a lot more hashes in a short amount of time under these two hash functions. As of today, when developing an authentication system, developers should choose hashing algorithms such as SHA-512. When hashing singular strings, the difference between all of these functions are irrelevant, however only when hashing millions or billions of strings will the differences matter.

B. *Suggestions for Internet Users*

It is not only the developer’s job to give an effort towards the security of passwords. As users, it is important to be able to choose strong passwords. There have been many research on what a strong password should look like. It is important to judge a password strength based on its vulnerability towards any cracking method. For example, the password “passwordpassword”, even though only consists of lowercase characters, it is of length 16, which means that it is probably safe against a brute force attack. However, such password is very common, that it might be one of the weakest passwords in a dictionary attack. Passwords such as “H3//0vv0r/d” might also seem like a password that would stand strong against a brute force attack due to its uses of various symbols. However, a dictionary attack with a good set of rules can also break this password quite easily.

A good password first of all should not have any ties to a person’s personal informarion, such as name or date of birth. This is to avoid it being vulnerable to social engineering. Including various symbols and characters is definitely good, as it will make the password stronger against brute force attacks. However, if implementing substitution of characters, make sure to use o'nes that is not commonly used. For example, instead of using a ‘3’ to substitute the letter ‘E’, maybe use it to substitute the letter 'D' or use a different symbol to substitute the letter 'E' that has not become common knowledge to the public.

Another approach to creating strong passwords is to simply make it really long, without really having to worry about the character assortment within it. A good way to choose such password iss by appending 4 or 5 words together. The key here is to use words that would not normally go together, otherwise it might still come up quite frequently in a dictionary attack. An example of such password would be "donkeyalgorithmwaterstsore". Though it might not look like it, this is actually a pretty strong password, as it is very long, therefore it is safe to assume that it is good agains brute force attacks. It is also most likely sasfe against dictionary attacks, because these 4 words do not commonly go together and until today, crackers probably do not have a specific rule set up yet to crack passwords designed this way.

The last criteria for a good password is that it should never be reused. A different password for every account may reach a point where it is unrealistic for people to remember everything without having to write it down, as it is also a very bad idea to write down passwords. Luckily, today we are presented with what is known as a password manager. Password managers is a digital vault that stores a user's password, only accessible using a master password. The password manager also provides a password generator that will create strong, random, unique passwords everytime a sign up is done. The password manager also keeps track of what password belongs to what site so that users no longer have to remember or even know what their passwords are.

VII. Conclusion

The subject of password strength should be taken more seriously by most people. As analysed in this paper, taken from the RockYou password list, the results show that majority of passwords cannot be considered secure and strong against password cracking attacks. It was shown that brute force attacks are most effective against password of short length and a low range of variety of characters, whereas a dictionary attack is most effective against common passwords, regardless of its length. In order to create a secure password, it must be able to withstand all forms of attacks.

This paper suggests that for developers of websites and applications that use a password authentication system to pay more attention to the storage mechanism of the user passwords. This includes choosing the most secure hashing algorithm and to never store the passwords in plain text. As for users of these sites and applications, it is recommended to choose a password that is long, uncommon, and consists of various assortment of characters. A user must also never use the same password for multiple accounts and should consider using a password manager for easier access to their passwords.

Video Link

Please kindly check this video for further explanation on the topic: https://youtu.be/8UNfnnxyNYc

Acknowledgment

The author would like to thank first of all, God for the guidance throughout the process of writing this paper. The author would also like to thank the lecturer of ITB Algorithm Strategies class 02 IF2211 Dr. Nur Ulfa Maulidevi, S.T., M.Sc. and Dr. Ir. Rinaldi Munir, MT. for the comprehensive website that the author has referred to multiple times. Not to forget, the author would also like to thank her family and friends who have given constant support throughout the entire semester that gave the author the strength and ability to complete it.

A friendly note to anyone reading this, please make sure to reevaluate your current passwords and make sure it is far from being crackable. Also please make sure to stay healthy and always follow the health protocols amidst this pandemic.

References

[1] Bosnjak, L., et al. "Brute-Force and Dictionary Attack on Hashed Real-World Passwords." *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 2018, doi:10.23919/mipro.2018.8400211.
[2] Charras, Christian, and Thierry Lecroq. "Brute Force Algorithm." *Brute Force Algorithm*, www-igm.univ-mlv.fr/~lecroq/string/node3.html.
[3] Dell'Amico, Matteo, et al. "Password Strength: An Empirical Analysis." *2010 Proceedingss IEEE INFOCOM,* 2010, doi:10.1109/infcom.2010.5461951.
[4] Hu, Gongzhu. "On Password Strength: A Survey and Analysis." *Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, 2017, pp. 165–186., doi:10.1007/978-3-319-62048-0_12.
[5] Kaspersky, Kapersky. "Brute Force Attack: Definition and Examples." *Www.kaspersky.com*, 13 Jan. 2021, www.kaspersky.com/resource-center/definitions/brute-force-attack.
[6] Mozilla. "Regular Expressions - JavaScript: MDN." *JavaScript | MDN*, developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions.
[7] Munir, Rinaldi."http://informatika.stei.itb.ac.id/~rinaldi.munir/"
[8] Triella, "Weak Passwords Are Still the Biggest Security Risk." *Triella*, 24 Sept. 2018, www.triella.com/weak-passwords/.
[9] Thriveni, C.A., and K. Madhavi. "A Secure Authentication Scheme against Password Guessing Attacks." *International Journal of Computer Sciences and Engineering*, vol. 6, no. 6, 2018, pp. 162–166., doi:10.26438/ijcse/v6i6.162166.
[10] Weber, James E., et al. "Weak Password Security: An Empirical Study." *Information Security Journal: A Global Perspective*, vol. 17, no. 1, 2008, pp. 45–54., doi:10.1080/10658980701824432.

Pernyataan

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Jakarta, 10 Mei 2021

[illegible]

Shifa Salsabiila - 13519106