Joshua Main-Smith

September 20, 2020

ONLINE USER AUTHENTICATION SECURITY LAB EXERCISES

PASSWORD STRENGTH

From CLARK Digital Library, www.clark.center

Lab Exercise Development Institution: Colorado School of Mines

Lab Manual Contributors:

Yi Qin[*], Cody Watters[+], Mykel Allen[+], Peer Seyferman[+], Chuan Yue[@]
(*: PhD student and research/teaching assistant; +: Undergraduate students
in the CSCI474 Introduction to Cryptography class of 2017; @: Faculty.)

PASSWORD STRENGTH LAB EXERCISE

Lab Description: You will experiment with an open-source password checker as well as the *John the Ripper* password cracker in this lab exercise. You will examine and create different passwords, analyze their strengths, as well as understand and evaluate the critical factors that influence the strengths of users' passwords.

The high-level **learning outcomes** and the corresponding **assessment** of this lab are summarized as follows. In other words, upon completion of this lab, students should be able to:

- **Examine** the functionality of some password checkers.
- **Construct** complex variations of passwords based on simple passwords.
- **Create** hashed passwords based on the plaintext passwords.
- **Use** the different methods in John the Ripper (JTR) password checker to crack hashed passwords.

Lab Files that are Needed: PasswordStrength.zip

Learning Setting: This lab module is for students to complete outside the classroom, so it can be used in either face to face or online courses.

LAB EXERCISE/STEP 1 (DOWNLOAD AND UNZIP THE FILE PASSWORDSTRENGTH.ZIP)

Download and unzip the file PasswordStrength.zip, which contains the materials that you will need to use in this lab exercise.

LAB EXERCISE/STEP 2 (ANALYZE A LIST OF BANNED PASSWORDS WITH THE PASSWORD METER)

The Password Meter contained in the .zip file is a web page for checking the strength of passwords in a list.

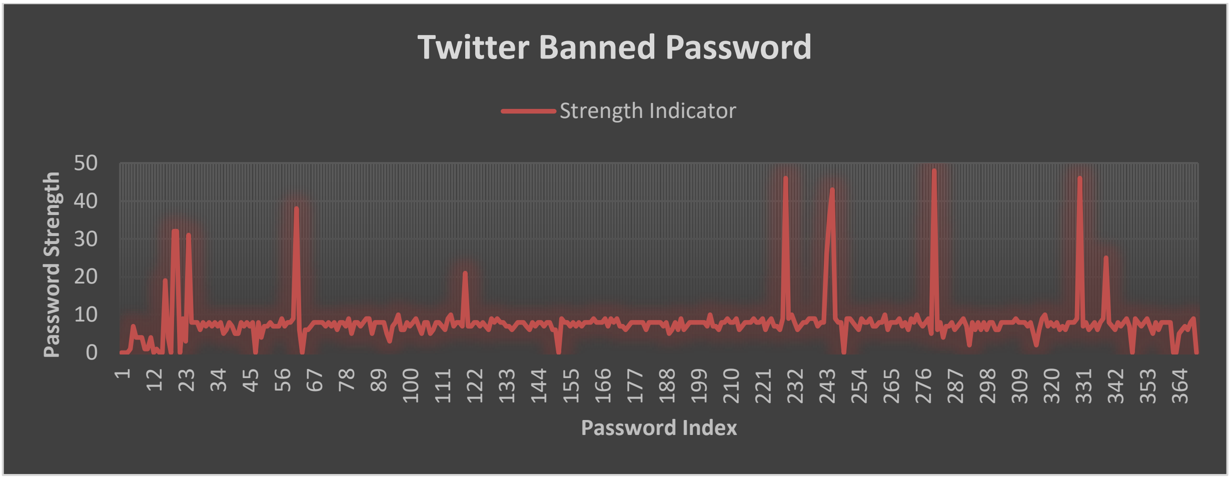
The text file *'twitter-banned.txt'* contains 370 simple passwords banned by Twitter. This file was taken from Skull Security (The skullsecurity.org website). Please use The Password Meter to analyze the strengths of all the 370 passwords, and answer the following two questions.

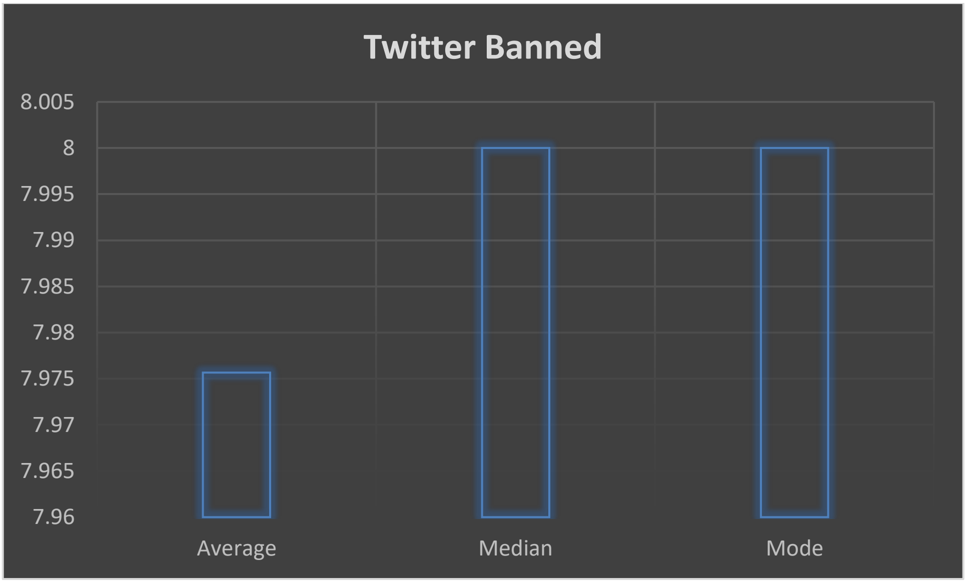
Please answer Questions 1 and 2:

Question 1:

What is the distribution of the password strength indicators or scores of the 370 passwords? You are expected to draw and present a figure for this distribution. Give each password an index (i.e. start with 1 label the passwords with an integer). Make a plot of password index on the x-axis and password strength indicator or score on the y-axis.

As can be seen in the two figures below, there are a few passwords that have a complexity rating reaching almost 50, but the majority fall well below that. In fact, the three statistics (mean, median, mode) are around 8 out of 100.





Question 2:

What criteria are used by the password checker to rate the strength of a password?

The password meter uses 16 factors from two dimensions in determining the complexity of a given password.

The first dimension being labeled additions (criteria for a more complex password), which include the number of characters, use of upper and lower case letters, numbers, symbols, numbers or symbols located in the middle of a password, and the overall requirements met by the complexity that merit it sufficient in the given domain.

The second dimension being labeled deductions (criteria indicating an insufficient scheme in password creation), which include a password containing only letters, numbers, having repeating characters, (including uppercase, lowercase and numbers), and sequential characters, numbers, or symbols (such as abcd, qwerty, 12345, !@#$%).

Each factor is given a different rate at which the password complexity is increased/decreased. One example being the letters only deduction will continue deducting the complexity score at a rate of *-n* until a non-letter item is introduced, at which point the total deductions in this category is set to zero.

One weakness in this approach is the meter doesn't consider commonly used passwords. For instance, when trying *Password1!*, it indicated that this is a strong password when, in all likelihood, is common and would be included among a wordlist for potential attackers. For the meter to be more accurate, it may be necessary to incorporate a wordlist of commonly used passwords.

LAB EXERCISE/STEP 3 (MODIFY PASSWORDS TO GENERATE NEW PASSWORD FILES)

Please modify the original password file *'twitter-banned.txt'* following the four requirements listed below. You should keep all the **four different plaintext password files** for use in the following up steps.

(1) Make no change. This will keep the original *'twitter-banned.txt'* as your first password file.

(2) Capitalize all the following letters (**a, d, h, j, l, p, q and z**) in the original password file *'twitter-banned.txt'* to generate your second password file.

(3) Make the following guessable replacements (**a to @, e to 3, and o to 0**) in the original password file *'twitter-banned.txt'* to generate your third password file.

(4) Make the following guessable replacements and capitalizations (**a to @, e to 3, o to 0, s to 5, l to 1, t to 7, h to H, p to P, w to W, d to D, r to R, and n to N**) in the original password file *'twitter-banned.txt'* to generate your fourth password file.

Please answer Question 3:

Question 3:

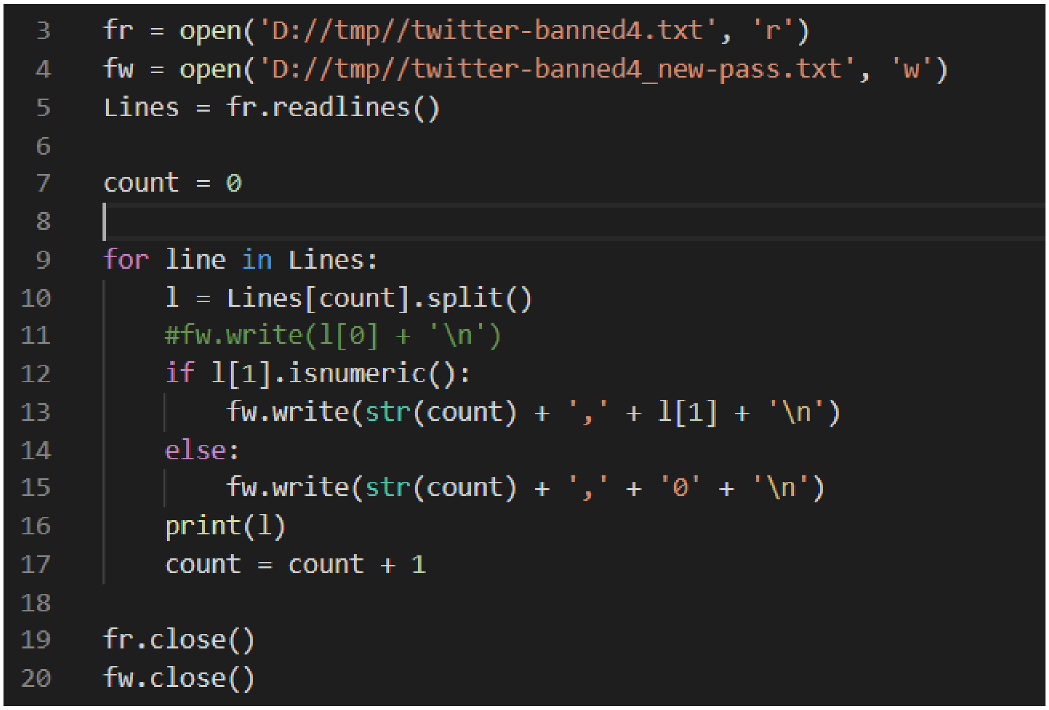
Analyze each of the three newly generated password files using the password checker. You are expected to draw and present three new figures like what you did for Question 1. Compare the four (three plus one from previous question) figures and describe your main observations.

All four figures are presented below (the first being exactly the same as Q1).

The main take away from these graphs is the overall complexity according to the password meter (from lowest to highest) is: 1, 2, 3, 4. This can be seen in the graphs, and even more clearly in the bar charts provided. The median, for instance, increases from 8 in (1) to 56 in (4). Even before carrying out the test, it seemed obvious that (4) would rank highest in complexity with the amount of complexity that's added relative to the other files.

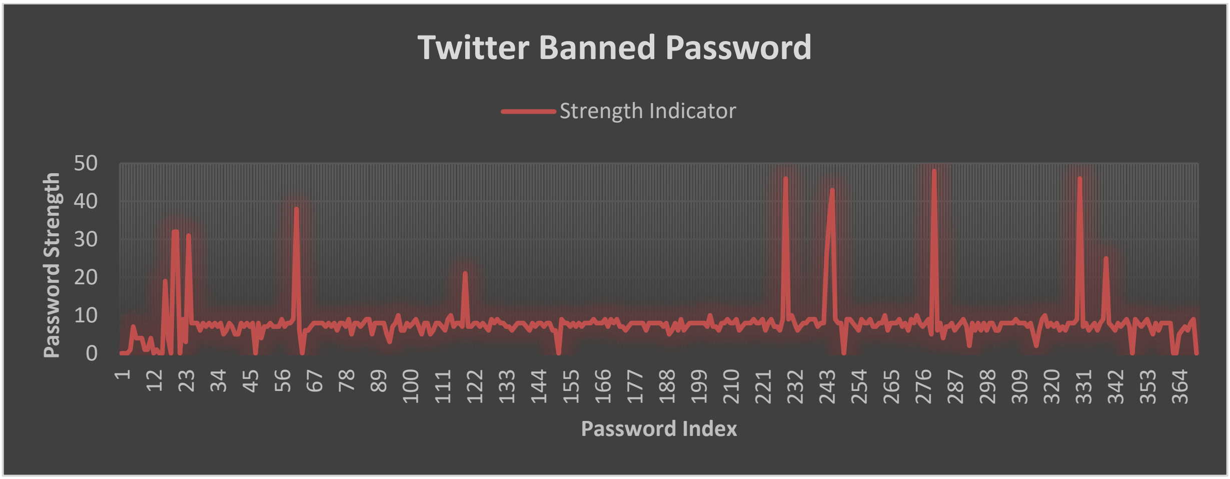
As mentioned in my previous answer, these graphs don't take into account passwords that are commonly used, but still satisfy the password meter as being sufficient (such as *P@ssword1*).

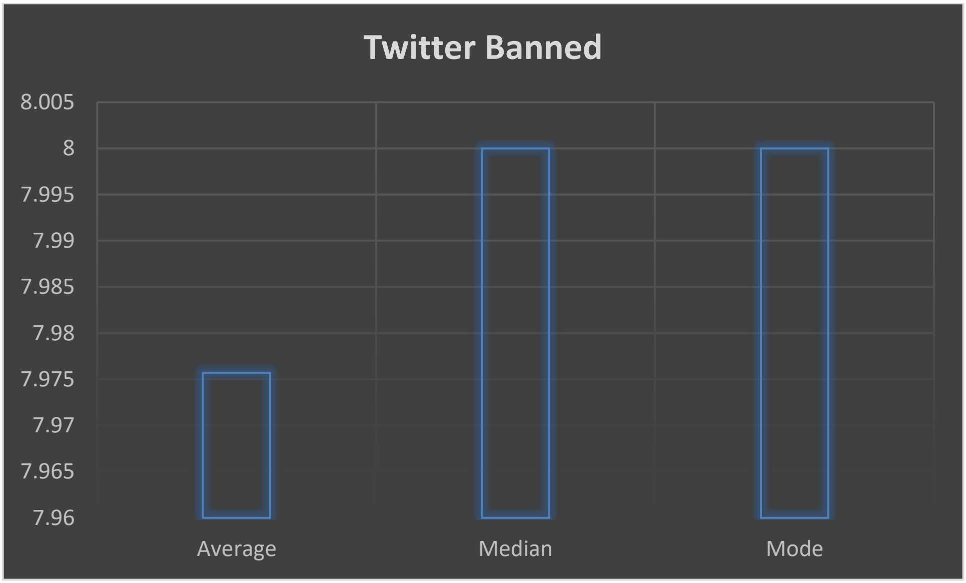
I employed the following Python script in obtaining the complexity score from the password meter:



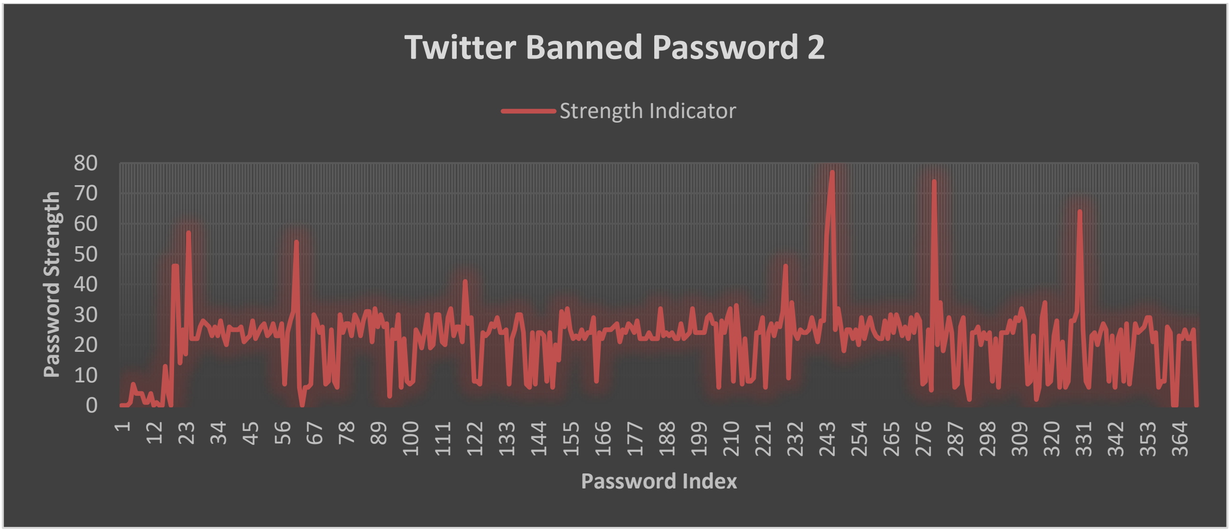
```python
fr = open('D://tmp//twitter-banned4.txt', 'r')
fw = open('D://tmp//twitter-banned4_new-pass.txt', 'w')
Lines = fr.readlines()

count = 0

for line in Lines:
    l = Lines[count].split()
    #fw.write(l[0] + '\n')
    if l[1].isnumeric():
        fw.write(str(count) + ',' + l[1] + '\n')
    else:
        fw.write(str(count) + ',' + '0' + '\n')
    print(l)
    count = count + 1

fr.close()
fw.close()
```

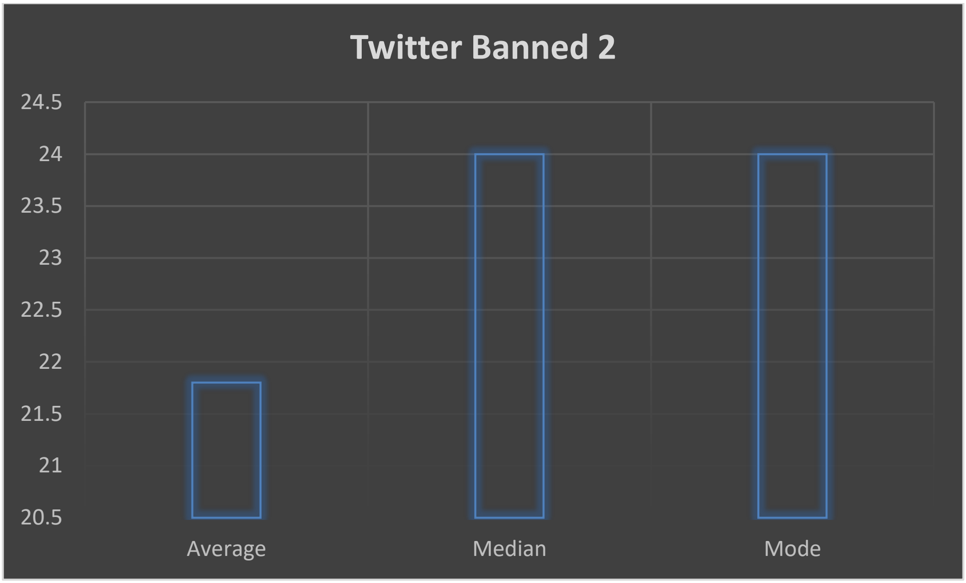
(1)


Twitter Banned Password
Strength Indicator
Password Strength
50
40
30
20
10
0
1
12
23
34
45
56
67
78
89
100
111
122
133
144
155
166
177
188
199
210
221
232
243
254
265
276
287
298
309
320
331
342
353
364
Password Index

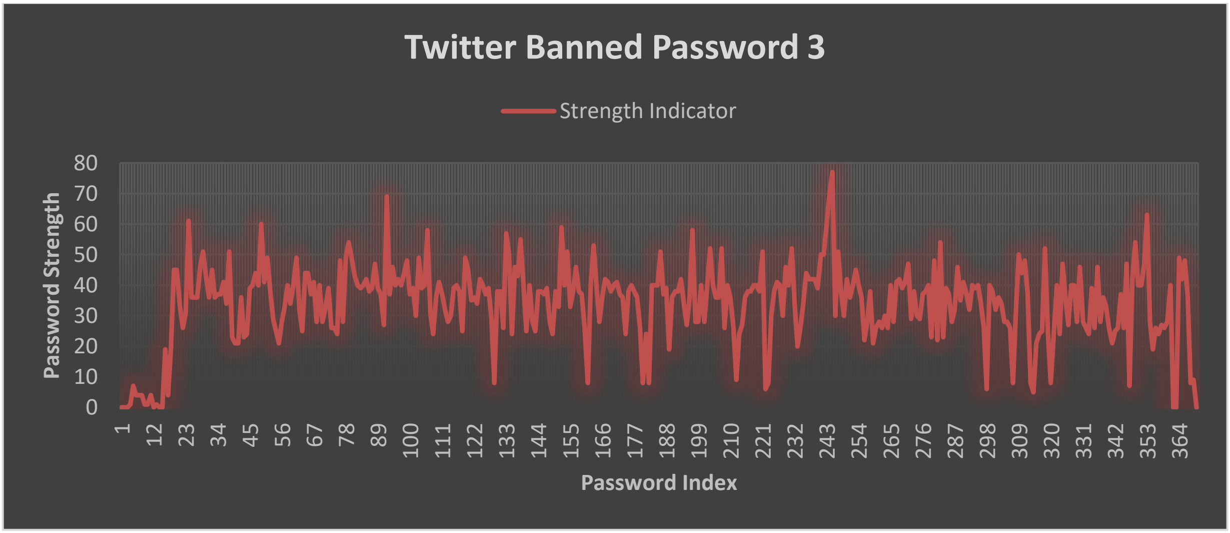

Twitter Banned
8.005
8
7.995
7.99
7.985
7.98
7.975
7.97
7.965
7.96
Average
Median
Mode

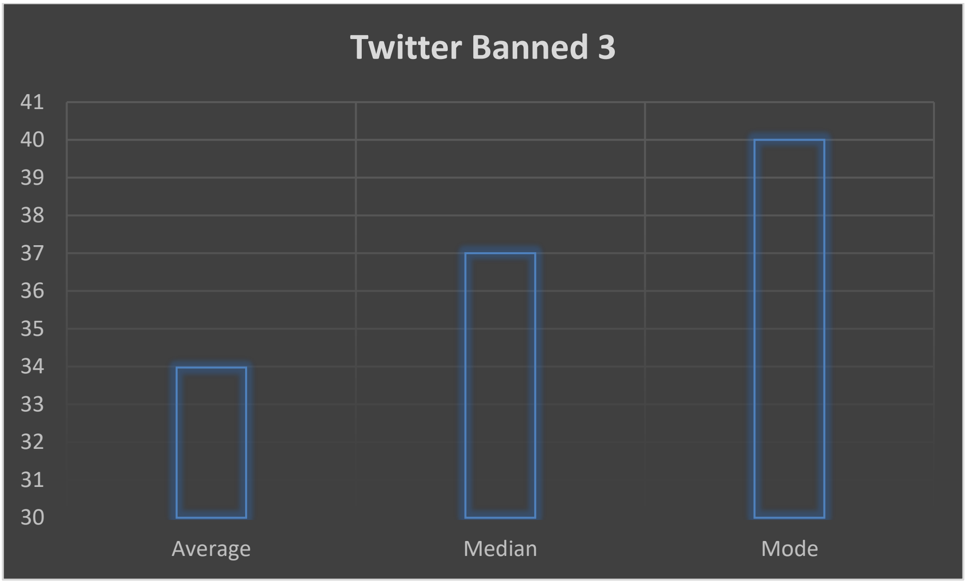
(2)


Twitter Banned Password 2
Strength Indicator
Password Strength
80
70
60
50
40
30
20
10
0
1
12
23
34
45
56
67
78
89
100
111
122
133
144
155
166
177
188
199
210
221
232
243
254
265
276
287
298
309
320
331
342
353
364
Password Index

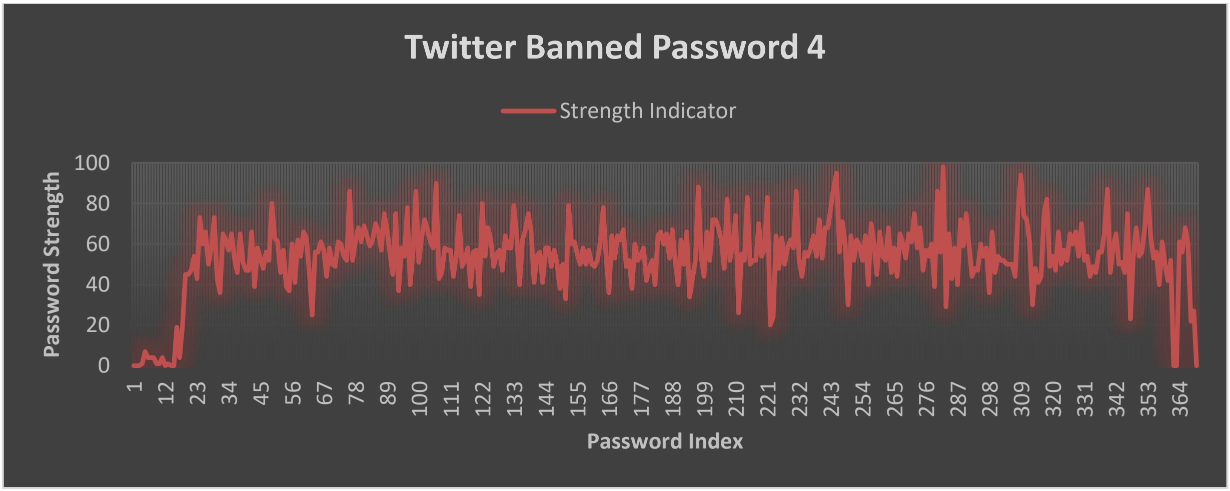

Twitter Banned 2
24.5
24
23.5
23
22.5
22
21.5
21
20.5
Average
Median
Mode

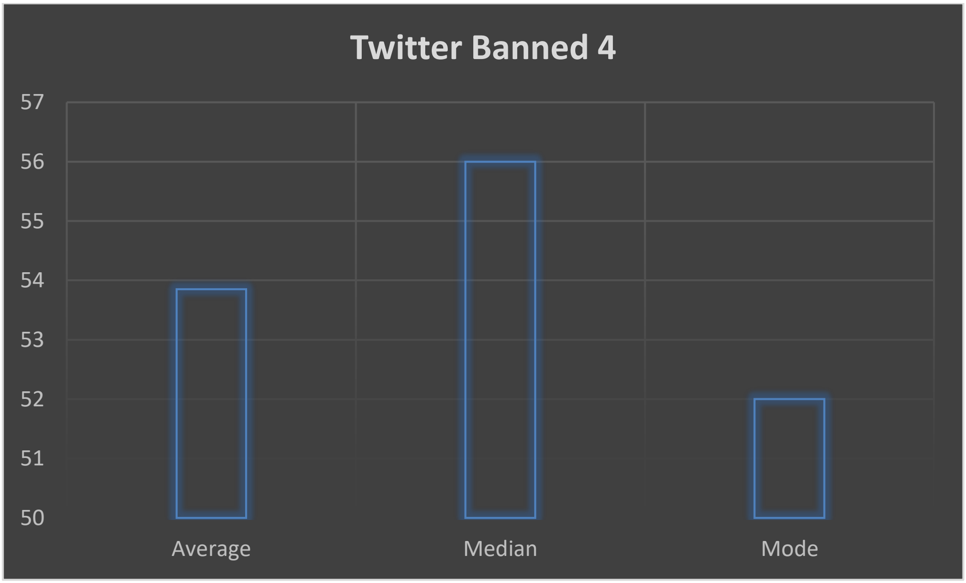
(3)


Twitter Banned Password 3
Strength Indicator
Password Strength
80
70
60
50
40
30
20
10
0
1
12
23
34
45
56
67
78
89
100
111
122
133
144
155
166
177
188
199
210
221
232
243
254
265
276
287
298
309
320
331
342
353
364
Password Index


Twitter Banned 3
41
40
39
38
37
36
35
34
33
32
31
30
Average
Median
Mode

(4)


Twitter Banned Password 4
Strength Indicator
Password Strength
100
80
60
40
20
0
1
12
23
34
45
56
67
78
89
100
111
122
133
144
155
166
177
188
199
210
221
232
243
254
265
276
287
298
309
320
331
342
353
364
Password Index


Twitter Banned 4
57
56
55
54
53
52
51
50
Average
Median
Mode

LAB EXERCISE/STEP 4 (INSTALL AND LEARN ABOUT THE JOHN THE RIPPER (JTR))

John the Ripper (JTR) is a fast password cracking application which is publicly available from Openwall at The openwall.com website.

LAB EXERCISE/STEP 5 (GENERATE AND SAVE THE HASHED PASSWORDS INTO FOUR NEW HASHED PASSWORD FILES)

Servers or services should only save the hashed passwords instead of the original plaintext passwords in their user authentication systems. Furthermore, they should use strong cryptographic hash functions, and should use random salts together with the original plaintext passwords as the inputs to the hash functions. In this lab exercise, we use the weak and unsalted (or raw) MD5 hash function to generate hashed passwords because our focus is on examining the cracking capabilities of JTR on passwords with different strengths.

Therefore, please use an MD5 utility installed on your operating system or hosted on some trustworthy website to generate ***four correspondingly hashed password files*** based on the four plaintext password files that you generated and kept in Step 3.

LAB EXERCISE/STEP 6 (USE JTR TO CRACK THE PASSWORDS IN THE FOUR HASHED PASSWORD FILES)

You will use two modes of JTR to perform five different experiments on each of the nine hashed password files. One mode is the ***Incremental Method*** and the other mode is the ***Wordlist Method***.

(a) The ***Incremental Method*** has no variation in this lab exercise.

```
$./john --incremental --format=[FORMAT] [PASSWD_FILE]
```

(b) The ***Wordlist Method*** allows you to specify both the wordlist file and the word mangling rules to use in the cracking.

```
$./john --wordlist=[WORD_FILE] --rules=[RULES] --
format=[FORMAT] [PASSWD_FILE]
```

In this lab exercise, you will use ***two different wordlist files***. One is the default file *'john.txt'* that contains 3107 passwords. The other is the large file *'rockyou.txt'* that contains 14,344,357 actual Internet passwords that were leaked. Both files were taken from Skull Security (The skullsecurity.org website).

You will use ***two different rules*** to specify the modifications that would be applied to each word in the wordlist to produce other likely passwords for cracking. One rule is *'none'* which means no modification rule will be applied. The other rule is *'all'* means that all the modifications in the JTR's rule list will be applied.

Please learn more about these modes and rules using the command './run/john' or visiting the following link: The openwall.com website.

Please answer Questions 4, 5, and 6:

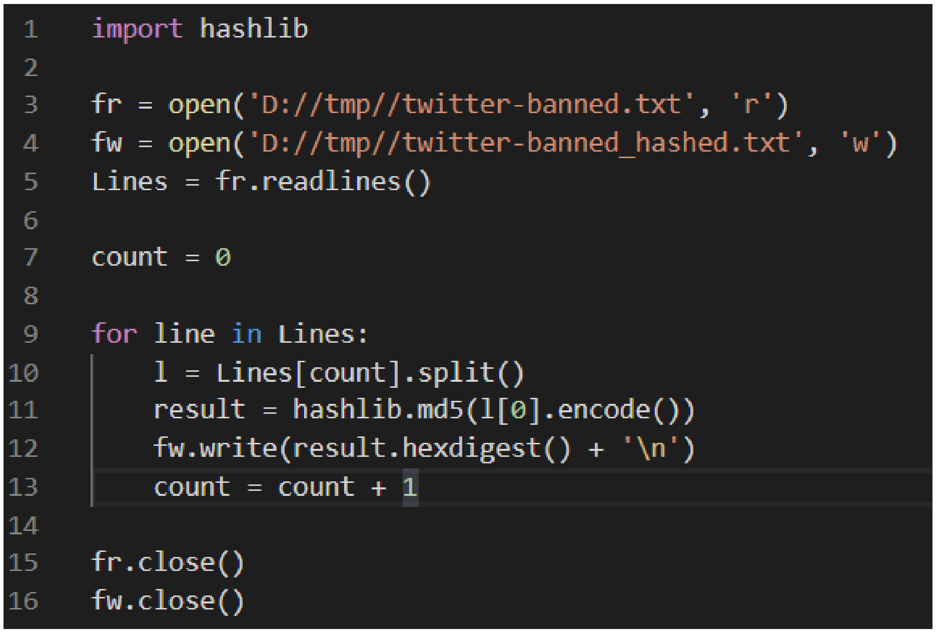
Question 4:

What are the essential differences between the Incremental Method and the Wordlist Method?

The incremental method uses a bunch of different combinations in brute forcing at what the password could be. This method comes with an option in limiting the character length when brute forcing. For this lab, we just used the default.

The wordlist method utilizes a file containing several commonly used passwords to assist with cracking. This method comes with the option of using rules, such as "mangling" (using different combinations of the given passwords in finding a probable match), that may assist in the cracking process. Enabling rules increases processing resources but may be fruitful if time is not limited.

I employed the following Python script in hashing the password files:



```python
import hashlib

fr = open('D://tmp//twitter-banned.txt', 'r')
fw = open('D://tmp//twitter-banned_hashed.txt', 'w')
Lines = fr.readlines()

count = 0

for line in Lines:
    l = Lines[count].split()
    result = hashlib.md5(l[0].encode())
    fw.write(result.hexdigest() + '\n')
    count = count + 1

fr.close()
fw.close()
```

Question 5:

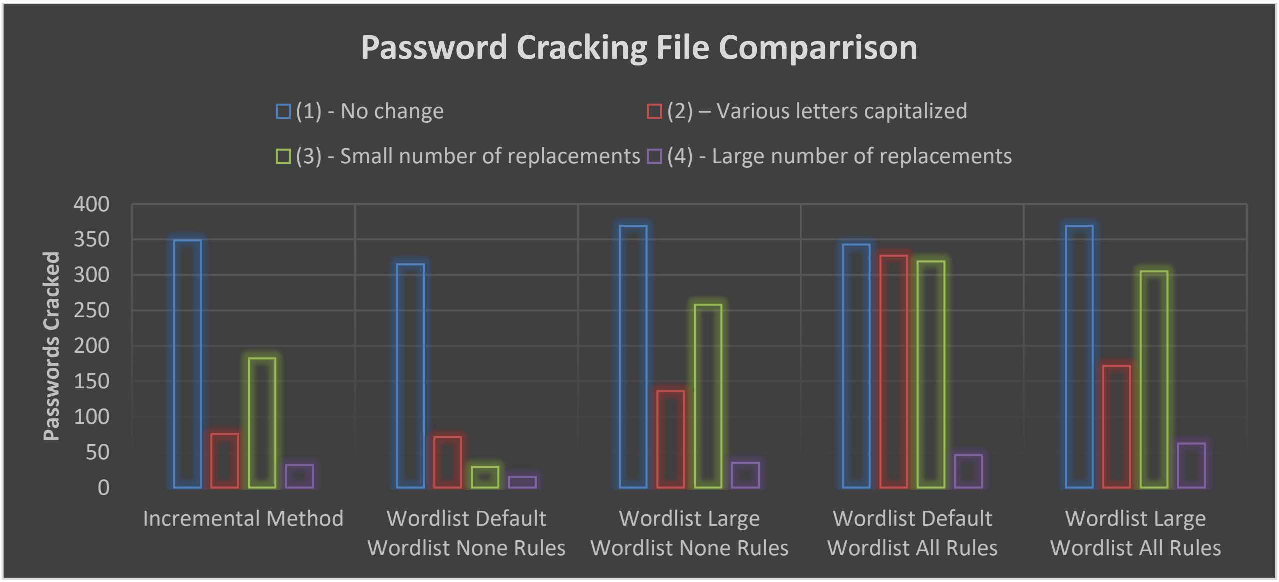
Use the Incremental Method and the Wordlist Method (with its four combinations of the aforementioned two wordlist files and two rules) to perform ***five different experiments*** on each of the four hashed password files that you generated in Step 5. In each experiment, please record the number of passwords cracked in a maximum time window of 120 seconds, and fill your recorded information into the following table.

Password File	**Incremental Method**	**Wordlist Default Wordlist None Rules**	**Wordlist Large Wordlist None Rules**	**Wordlist Default Wordlist All Rules**	**Wordlist Large Wordlist All Rules**
(1) - No change	349	315	369	343	369
(2) – Various letters capitalized	75	71	136	327	172
(3) - Small number of replacements	182	29	258	319	305
(4) - Large number of replacements	32	15	35	46	62

Question 6:

What are your main observations from the table that you filled above?

The most obvious observation is that file (1) was the least secure relative to the other files while file (4) was the most secure. File (2) did better than in more tests than (3), even though (3) was rated as being more secure with the password meter. The reason for this is probably due to (2) containing for changes than (3). If a similar number of changes were made, perhaps (3) would have done better. Another possibility is perhaps people tend to use symbols and numbers (as opposed to capitalized letters) when the goal is greater password complexity. So, instead of *JoshPassword* one might be inclined to use *J0shP@ssword*. This wouldn't explain the difference in the incremental results, so I suspect the reason is more variation (regardless of character set) leads to greater complexity in terms of cracking.
When looking at utilizing the ruleset, there seemed to be anywhere from a moderate to great advantage. The default wordlist benefited the most from the addition of the ruleset. This may indicate that if an attacker only has access to a limited number of commonly used password, flagging for rules may be a wise decision. The large wordlist did benefit as well, although not as much. If more time is available, it may be worth enabling all the rules when cracking, especially if the objective is cracking passwords for persons of interest (executives, CEOs, etc.). Below is a graph showing the effectiveness of each method in relation to the file tested on.



LAB EXERCISE/STEP 7 (ANALYZE PASSWORD FREQUENCY AT THE SERVER-SIDE)

Assume you are a website administrator. The file 'ServerSidePwd.txt' contains a list of plaintext passwords chosen by ***different users*** for registering accounts on your website.

Please answer Question 7:

Question 7:

(a) What are the two most frequently used passwords in this file? Are they strong passwords *(e.g., based on your identified password checker in Step 2)?*

The two most common passwords are *june1097* and *t8Ky/<^mTB* (determined from https://www.somacon.com/p568.php). According to the password meter:

Password	Score	Strength
t8Ky/<^mTB	100	Very Strong
june1097	50	Good

The first one was exceptional in almost every category and had minimal deductions. This one would have been considered among one of the best passwords in our analysis from Q3(4). The second, although good, could be better. There were a few factors giving this password an exceptional score, but failed in other areas, such as the lack of uppercase characters and symbols. Overall, a failure in meeting the overall requirements. This password would have been considered among the best in Q3(1), but below average in Q3(4).

(b) What could be some reasons for different users to choose the same password, either strong or not, for registering their online accounts?

There could be several reasons of how someone could arrive at the same password. If someone works for Microsoft, they could have Micr0soft2020 (changing the year every time they have to renew their password, if it's an annual process). In other words, relating what their doing to their password for ease of remembering. A more likely scenario is perhaps these are default passwords given to new hires. When I works as an IT Technician, we had a default password for new hires and gave the new hires the option to keep the default or change it. Something like *t8Ky/<^mTB* could be a default password that hasn't been changed, and is probably a more likely scenario than two employees coincidentally generating the same complex password.

(c) Is it necessary to deploy a server-side password checker to detect and then prevent a new user from choosing a same password that has been chosen by some or many other users? Please explain your answer.

It depends on where the *"many other users"* are coming from. If it's a generic wordlist of commonly used passwords, then this would be a useful approach in mitigating attackers from using dictionary attacks. If the server-side script is comparing a new hires password to passwords already stored on the server, this presents a few issues worth considering. First, this could give a potential attacker sensitive information, leading to an attempt in trying various username/common password combinations. Further, if the server-side script is able to compare various passwords, this would give the attacker the knowledge that the passwords are either being stored in plaintext, with unsalted hashes, or with static salts. Overall, alerting a new hire (or user) in which passwords are commonly used within the organization would have unintended consequences, and should probably be avoided.

LAB EXERCISE/STEP 8 (SUMMARIZE WHAT YOU LEARNED)

Please answer Question 8:

Question 8:

Please write a summary (100 ~ 150 words) about your understanding of using JTR to crack hashed passwords. Based on your observations in this lab exercise, please also summarize and explain what critical factors influence the strengths of users' passwords.

John the Ripper is a great tool to use in the automation of password cracking. There are a few different modes to select from, but the ones we focused on were incremental and wordlist (both of which were discussed above). Even though JTR supports automatic crypto detection, I have found with both modes that it is much more effective to specify the format being cracked. The wordlist methods used above were not as effective in cracking without specifying a format. Further, the tool is also very useful in it storing previously cracked hash functions, making it quicker in cracking other hash files obtained in the future. This could be very helpful for a pentester if time is limited.

Two factors that determine the strength of a user's password is complexity and uniqueness. The password file generated in Q3(4) is a great example of the use of complex passwords. To achieve this, a company could require users to generate a password the must contain a combination of uppercase letter, lowercase letters, numbers and symbols. The second factor is uniqueness. It doesn't matter how strong a password is if it's commonly used, as an attacker could include it in a dictionary attack. So, uniqueness is also an important consideration in the strength of a password. A company could implement this by employing a wordlist in preventing users from using commonly used passwords.