International Journal of Scientific Research in Science and Technology
Print ISSN: 2395-6011 | Online ISSN: 2395-602X (www.ijsrst.com)
doi : https://doi.org/10.32628/IJSRST

Evaluation of Leet Speak on Password Strength and Security

Medhansh Garg
American Embassy School, New Delhi, India

Article Info
Volume 9, Issue 5
Page Number : 410-422

Publication Issue
September-October-2022

Article History
Accepted : 02 Oct 2022
Published : 14 Oct 2022

ABSTRACT

Making secure passwords is one of the biggest challenges in everyday life. There are numerous rules and requirements making passwords complex and hard to remember, and keeping track of which password is for which account is a major hassle. According to an article from HelpNetSecurity, statistics show that an alarming "78% of respondents required a password reset in their personal life within the last 90 days" ("78% of people"). In recent years, leet speak has become increasingly popular as a way to create memorable passwords. Leet speak is a convenient method for users to create passwords that meet password requirements in many services. But there has been increasing debate on whether this approach is a secure and safe method or not. This paper aims to solve this debate by effectively evaluating the strength of ordinary passwords and leet passwords using various means. With the help of password cracking or recovery tools and password strength classifiers, this paper will compare the cracking time and strength scores of ordinary passwords and leet converted passwords. The paper will begin with a background information section explaining important concepts discussed in the paper, followed by the methodology of the experiment, a presentation of the data along with the evaluation of the results, and a conclusion at the end.

Keywords: Cyber Security, Password, Password Strength, Leet, Leet Speak

I. INTRODUCTION

Passwords are a form of providing authentication, a way of proving that one is who that claim to be. The use of secret words or phrases for the purpose of authentication has existed since ancient times ("Password"). In ancient Rome, guards would require people to say a watchword to prove their identity (ibid).

Passwords have long prevailed as the most common choice for user authentication. This is because they are simple to implement, without the need for special hardware; have to be entered precisely and correctly, as even a minor error results in an entirely incorrect password; are convenient for the user, as they don't require a user to carry an additional item with them;

protect user privacy, a password can be easily swapped and replaced if compromised (ibid).
These are some of the things that other forms of authentication, such as biometrics and physical security keys, fail to address.
However, this does not mean that passwords are perfect; they too have their flaws.

Primarily, making secure passwords. On the one hand, they have to be long and complex to be safe, but on the other hand, they must be simple enough to be memorable. The balance between secure passwords exists at the point where they are not easy to guess but not hard to remember, and this point is quite hard to arrive at.

Having long and complex passwords increases the likelihood of a user repeating the same password for multiple accounts or storing the password in a clear text written form electronically or physically, increasing the chances of them being discovered. It also often results in a user forgetting their password and being compelled to change it to something more simple yet unsafe.

There exist many techniques to combat this widespread issue, such as the Correct Horse Battery Staple Method, Revised Passphrase Method, Bruce Schneier Method, etc. (Schneier).

One such method in the debate is leet speak. Some sources claim that it is a viable and safe method of producing passwords, whereas others claim that it has little to no effect on password strength.

Leet Speak

Leet speak (a.k.a. “l337 speak”), derived from the word “elite” (used to refer to hackers), was first seen in the 1980s on bulletin board systems (BBS) (TechTarget Contributor). BBSs are servers that allow for the public communication and exchange of files and messages between users through the use of a terminal application, similar to modern forums such as Reddit or Quora ("Bulletin board"). Originating with hackers and techies, leet speak found its way to the mainstream culture with the rise of “Doom and Doom II”, a fairly influential game in the video game industry (TechTarget Contributor). Leet speak is the process of creatively replacing certain letters in a word or phrase with numbers or symbols that resemble the original letter (ibid).
Common examples of this are leet → 1337, noob → n00b, hacker → hax0r (ibid).

With the rise of restrictions on passwords, many have resorted to leet speak to meet all the requirements of uppercase letters, lowercase letters, numbers, and symbols. Users find leet speak as a convenient way of meeting password requirements while making them easy to

remember. Meeting these requirements gives users a sense of assurance that their password is strong and safe, but this sense of security may not always be true (). Password requirements and strength meters don’t take many things into account such as common unsafe password habits: using names, birthdates, favorite items, or common passwords (“123456”, “password123”, “qwerty”). Thus it is not enough to say that a password containing leet is secure just because a strength meter said so.

Password Security

Password security is important to protect online accounts and prevent unauthorized access. There is a lot of thought and effort that goes into managing and maintaining the security of passwords.

One of the methods used is password hashing. To protect passwords from being compromised in the event of a data breach, they are stored as password

hashes (Jung). These are scrambled texts that are returned by passing a password through an incredibly complex function called a hash function (ibid). Password hash functions have several criteria that they have to meet before they can be considered safe enough: they must be non-reversible, they must be deterministic, and even a small change in the input must result in an entirely new output (ibid).

Non-reversible hashes mean that there is no inverse function that will output the decrypted password when given a hash as an input (ibid). This is impossible in practicality so hash developers aim to make hash functions as complex as possible and make it impractical and infeasible to find the original password from just the hash (ibid).

Deterministic hash functions are functions that for the same input, the function will output the same exact hash every time (ibid). Collisions are passwords that result in the same exact hash, and although there will always be collisions, hash developers aim to make it as difficult as possible to find them (ibid).

During the password creation stage, a user creates a cleartext password which is stored as a hash in a database (ibid). Then when the user wants to log in to their account, they present the password in cleartext which is hashed using the same algorithm used during the creation stage, and if the two hashes match perfectly, the passwords are the same and the user is granted access to the account (ibid). However, this approach alone is vulnerable to brute force attacks (ibid).

A brute force attack is an attack where a malicious actor tries to gain access to a system by trying every possible combination of passwords, keys, or other authentication factors ("Brute Force"). This type of attack is usually conducted by automated software that can try thousands or even millions of different combinations very quickly.

Brute force attacks can be very difficult to defend against because they can be conducted very quickly and with little effort (ibid). The best defense against a brute force attack is to use strong authentication factors that are difficult to guess, such as long passwords or passphrases, and to limit the number of attempts that can be made to gain access to a system (ibid).

Another method used to combat brute force attacks is password salts (Jung). This method is more effective against offline brute force attacks than online ones (ibid). Offline brute attacks work by somehow obtaining the password hash, and trying every possible combination to match the hash (ibid). Offline attacks are generally more effective than online ones because many services have a lock-out mechanism, where after a certain number of incorrect attempts, the service either locks the account or notifies the user (ibid). With offline attacks, this risk is mitigated.

To combat offline attacks, security professionals use random salts that are appended to the password at the time of creation (ibid). When the password is hashed, it is hashed with a random salt, which is also stored separately elsewhere (ibid). When the user attempts to log in, the stored salt is re-appended to the provided cleartext password and if the hash of the user-provided password plus salt matches the stored hash, then the user is granted access (ibid). This nearly mitigates the offline attack risk because it is nearly impossible to verify the password without knowing the associated salt (ibid).

These methods and techniques help mitigate many issues and significantly reduce the likelihood of leaking a password, but they are not foolproof. The best way to maintain security is to start at the source, making strong passwords. There are many ways to evaluate the strength of a password: online services

often use password strength meters, there exist machine
learning-based password strength classifiers, and password entropy can be used to calculate the complexity of a password.

Many users also opt to use password generators that use algorithms to develop long, complex, and incredibly safe passwords, however, these are often nearly impossible to remember, so users choose to use a password manager alongside the generator ("Is it Safe"). Many password managers now feature a password generator that can automatically save the password to the manager for different accounts (ibid). Password managers are generally considered to be incredibly safe, however, it is important to use popular ones as they tend to be more secure (ibid). Some examples of popular managers include NordPass, DashLane, 1Password, etc. (ibid).

There is also a tool that allows people to deep scan the web for mentions over their password or credentials (Hunt). Created by Microsoft Regional Director, Troy Hunt, the "Have I Been Pwned" site allows users to enter any credential and scan online password lists, credential leaks, and more for the entered credentials (ibid). Although the site sounds suspicious and insecure, it has been researched and evaluated by many researchers and security professionals (ibid). It is used by governments, organizations, individual users, etc. (ibid). The website's code is also open-source through the .NET Foundation (ibid).

Methodology Zxcvbn

To test and compare the strength of regular passwords and leet combined passwords, I used a tool called "zxcvbn" (zxcvbn). This tool allows one to estimate the strength of a password using an algorithm that returns valuable information: a strength score (0 - 4), estimates of how long it would take to crack, and feedback for improving the password.

Before using the zxcvbn tool, I first downloaded a list of passwords from Daniel Miessler (Miessler). I then used the following command "sed -ne '/[^0-9]/p' passwords.txt > passwords_sifted.txt" to sift through the passwords and remove any password containing only numbers, storing the remaining passwords into a passwords_sifted.txt file. This was because number passwords cannot be converted to leet. The resultant file had 834792 passwords.

I then wrote a script to convert these sifted passwords into leet format. To create the script, I formulated a character map, mapping each alphabet to a set of leet symbols and characters obtained from GameHouse (Craenen). The character map is as follows:

```
char_map = {
    "a" : ["a", "4", "@"],
    "b" : ["b", "13", "8", "I3", "l3"],
    "c" : ["c", "(", ""],
    "d" : ["d", "|)", "[)", "cl", "c1"],
    "e" : ["e", "3"],
    "f" : ["f", "ph"],
    "g" : ["g", "9"],
    "h" : ["h", "#", "/-/"],
    "i" : ["i", "1", "!"],
    "j" : ["j"],
    "k" : ["k", "|<"],
    "l" : ["l", "1", "|", "|_"],
    "m" : ["m", "/\/\", "/V\", "|V|"],
    "n" : ["n", "/\/", "/V"],
    "o" : ["o", "0", "()", "ø"],
    "p" : ["p"],
    "q" : ["q"],
    "r" : ["r"],
    "s" : ["s", "$", "5"],
    "t" : ["t", "+", "7"],
    "u" : ["u"],
    "v" : ["v", "\/"],
    "w" : ["w", "\/\/", "VV", "vv"],
    "x" : ["x", "><"],
    "y" : ["y", "`/"],
    "z" : ["z", "2", "7_"]
}
```

I then used the leet_converter.py script (attached in the appendix) to convert the letters in each password

to a randomly chosen leet alternative, however not all letters were converted to increase the realisticness of the generated passwords. The script created a passwords_leet.txt file which contains the leet-converted passwords.

The passwords_leet.txt file and the passwords_sifted.txt file were then fed into a strength testing script, password_strength.py (attached in the appendix). The password_strength.py script uses the zxcvbn library to assess and evaluate each password in the two files provided and outputs the data into two new files, called password_normal_strength.txt (for non-leet passwords) and password_leet_strength.txt (for leet passwords) (insert citation). The results found that almost all leet passwords had lower strength scores than their non-leet counterparts.

There were a few exceptions where the leet password had been heavily modified to the point where it no longer easily resembled the original. I found that, in the original word list, approximately 52.3% of the passwords had been given a score of 3 or above, and about 82.4% of these passwords had been given a score of 2 or below when converted to leet.

Hashcat

To test the strength of the two types of passwords using Hashcat, I chose to conduct a brute force attack against every password (Hashcat). However, before I could use Hashcat to crack the passwords, I needed to convert them into hashes. So I composed a script to convert each password into a hash using the SHA-1 hashing algorithm, called hash_converter.py (attached in the appendix). After which I ran a brute force program to crack the passwords. The program lasted 3673.324 seconds and was able to crack 12.98% of the leet converted passwords, and it lasted 3774.213 seconds and was able to crack 13.01% of the original passwords.

Entropy Values

Entropy measures the complexity of a password by assessing the number of passwords that can be generated given a character pool, and it is calculated by multiplying the password length by the log base 2 of the character pools (E = L * log2(R)) (Szczepanek). A character pool is the type of characters such as lowercase letters, uppercase letters, numbers, special characters, etc. Each character pool corresponds to the number of characters in that pool. The table shown below demonstrates this:

Lowercase Letters [a-z]	26
Uppercase Letters [A-Z]	26
Numbers [0-9]	10
Special Characters [“!”, “”””, “””, “#”, “$”, “%”,“&”, “’”, “”, “(“, “)”, “*”, “+”, “,”, “-”, “.”, “/”,“:”, “;”, “<”, “=”, “>”, “?”, “@”, “[“, “\”, “]”, “^”,“_”, “`”, “{“, “|”, “}”,” “~”, “ ”]	35

The value of R in the formula is the sum of the numbers corresponding to the character pools that are present in the password. For example, the password “password123” would have an R value of 36 whereas “Password123” would have an R value of 62. Using this formula, I programmed a python script to multiply the length of each password by the log base of the R value of the password and stored the results in the following files, password_normal_entropy.txt, and password_leet_entropy.txt. I found that approximately 82.9% of the leet passwords had a higher entropy value than their original counterparts, and 5.7% of the normal passwords had a higher entropy value than their leet-converted counterparts.

II. RESULTS AND DISCUSSION

	zxcvbn (Given a score of	Hashcat (Percent cracked)	Entropy (Greater value than

	3 and above)		the counterpart)
Normal	52.3%	13.01%	5.7%
Leet	22.4%	12.98%	82.9%

As the table above demonstrates, zxcvbn demonstrates that leet speak has an adverse effect on password strength, and password entropy demonstrates that it significantly benefits the password complexity. However, I think Hashcat's results hold the greatest weightage and most accurately portray the conclusion among the three methods as this method best demonstrates the process of hackers. Hashcat's results demonstrate that it was able to crack more normal passwords but by a minuscule margin. I believe that that margin can be attributed to the margin of error, because, given another dataset, the percentage cracked could be reversed. The conclusion that can be drawn from the results overall is that leet speak has a minimal effect on the security of passwords as hackers are aware of these techniques and have developed programs that allow for them to circumvent these tricks and techniques. Leet speak may be able fool password strength meters in many websites but it is not a viable method to increase the security or strength of passwords.

III. REFERENCES

[1]. "Brute Force Attack: Definition and Examples." kaspersky, www.kaspersky.com/resource-center/definitions/brute-force-attack. "Bulletin board system." Wikipedia, en.wikipedia.org/wiki/Bulletin_board_system. Craenen, Roald. "L33T SP34K CH34T SH33T." GameHouse,

[2]. www.gamehouse.com/blog/leet-speak-cheat-sheet/.

[3]. Hashcat. Hashcat, hashcat.net/hashcat/.

[4]. Hunt, Troy. "Who, what & why." Have I Been Pwned, haveibeenpwned.com/About. "Is it Safe to Use Random Password Generators?" Best Reviews, password-managers.bestreviews.net/faq/is-it-safe-to-use-random-password-generators/.

[5]. Jung, Jason. "What is Password Hashing and Salting?" Okta, 7 May 2021, www.okta.com/uk/blog/2019/03/what-are-salted-passwords-and-password-hashing/.

[6]. Miessler, Daniel. "10-million-password-list-top-1000000.txt." text file. "Password." Wikipedia, en.wikipedia.org/wiki/Password. Accessed 21 July 2022. Schneier, Bruce. "Choosing Secure Passwords." Schneier on Security, 3 Mar. 2014,

[7]. www.schneier.com/blog/archives/2014/03/choosing_secure_1.html.

[8]. "78% of people forgot a password in the past 90 days." Help Net Security, 11 Dec. 2019, www.helpnetsecurity.com/2019/12/11/forgot-password/. Accessed 1 Aug. 2022.

[9]. Szczepanek, Anna. "Password Entropy Calculator." Omni Calculator, 13 Apr. 2022, www.omnicalculator.com/other/password-entropy.

[10]. TechTarget Contributor. "leet speak (leet)." WhatIs, Aug. 2016, www.techtarget.com/whatis/definition/leet-speak-leet-leetspeak-leetspeek-or-hakspeak. Accessed 1 Aug. 2022.

[11]. Zxcvbn. Version v4.4.2. Github, 7 Feb. 2017, github.com/dropbox/zxcvbn.

Cite this article as :

Medhansh Garg, "Evaluation of Leet Speak on Password Strength and Security", International Journal of Scientific Research in Science and Technology (IJSRST), Online ISSN : 2395-602X, Print ISSN : 2395-6011, Volume 9 Issue 5, pp. 410-422, September-October 2022. Available at doi : https://doi.org/10.32628/IJSRST229567

Journal URL : https://ijsrst.com/IJSRST229567

Appendix

leet_converter.py:

```
# Import necessary modules
import random

# Creating the character map with letters corresponding to leet speak characters
# Because not all letters are always converted to leet, each letter also corresponds
to the same letter twice to create a weighted probability of a letter being converted.
char_map = {
    "a" : ["a", "a", "4", "@"],
    "b" : ["b", "b", "13", "8", "I3", "13"],
    "c" : ["c", "c", "(", ""],
    "d" : ["d", "d", "|)", "())", "cl", "cl"],
    "e" : ["e", "e", "3"],
    "f" : ["f", "f", "ph"],
    "g" : ["g", "g", "9"],
    "h" : ["h", "h", "#", "/-/"],
    "i" : ["i", "i", "1", "!"],
    "j" : ["j", "j"],
    "k" : ["k", "k", "|<"],
    "l" : ["l", "l", "1", "|", "|_"],
    "m" : ["m", "m", "/\\/\\", "/V\\", "|V|"],
    "n" : ["n", "n", "/\\/", "/V"],
    "o" : ["o", "o", "0", "()", "m"],
    "p" : ["p", "p"],
    "q" : ["q", "q"],
    "r" : ["r", "r"],
    "s" : ["s", "s", "5", "$"],
    "t" : ["t", "t", "+", "7"],
    "u" : ["u", "u"],
    "v" : ["v", "v", "\\/"],
    "w" : ["w", "w", "\\/\\/", "VV", "vv"],
    "x" : ["x", "x", "><"],
    "y" : ["y", "y", "`/"],
    "z" : ["z", "z", "2", "7_"]
}

# Opening the sifted passwords file and reading it into the text_original variable
text_original = open("./passwords_sifted.txt", 'r').read()
text_leet = ""
# Looping through ever character in the original text to convert to leet
```

```
# Import necessary
modulesimport random
```

```
# Creating the character map with letters corresponding to leet speak characters
# Because not all letters are always converted to leet, each letter also
corresponds to the same letter twice to create a weighted probability of a letter
being converted.char_map = {
    "a" : ["a", "a", "4", "@"],
    "b" : ["b", "b", "13", "8", "I3", "l3"],
    "c" : ["c", "c", "(", ""],
    "d" : ["d", "d", "|)", "[)", "cl",
    "cl"],"e" : ["e", "e", "3"],
    "f" : ["f", "f", "ph"],
    "g" : ["g", "g", "9"],
    "h" : ["h", "h", "#", "/-/"],
    "i" : ["i", "i", "1",
    "!"],"j" : ["j", "j"],
    "k" : ["k", "k", "|<"],
    "l" : ["l", "l", "1", "|", "|_"],
    "m" : ["m", "m", "/\\/\\", "/v\\", "|v|"],
    "n" : ["n", "n", "/\\/", "/v"],
    "o" : ["o", "o", "0", "()",
    "ø"],"p" : ["p", "p"],
    "q" : ["q", "q"],
    "r" : ["r", "r"],
    "s" : ["s", "s", "$", "5"],
    "t" : ["t", "t", "+",
    "7"],"u" : ["u", "u"],
    "v" : ["v", "v", "\\/"],
    "w" : ["w", "w", "\\/\\/", "vv",
    "vv"],"x" : ["x", "x", "><"],
    "y" : ["y", "y", "`/"],
    "z" : ["z", "z", "2", "7_"]
}


# Opening the sifted passwords file and reading it into the text_original
variabletext_original = open("./passwords_sifted.txt", 'r').read()
text_leet = ""
# Looping through ever character in the original text to convert to leet
```

```
for char in text_original.lower():
   # If the character is a letter and exists in the char map, then choose a random
leet character
   if char in char_map:
       leet_chars = char_map[char]
       leet_char = random.choice(leet_chars)
   # If the character is not a letter (numbers or symbols), then keep it the same
   else:
       leet_char = char
   # Append the leet characters to the transformed leet text
   text_leet += leet_char


# Opening the passwords_leet.txt file and writing the newly converted passwords to
this file
text_leet_file = open("./passwords_leet.txt", 'w')
text_leet_file.write(text_leet)
```

password_strength.py:

```
from zxcvbn import zxcvbn


password_normal_str = open("./passwords_sifted.txt", 'r').read()
password_leet_str = open("./passwords_leet.txt", 'r').read()
password_normal = password_normal_str.split("\n")
password_leet = password_leet_str.split("\n")


password_normal_strength = open("./password_normal_strength.txt", 'a')
password_leet_strength = open("./password_leet_strength.txt", 'a')


try:
   for passwd in password_normal:
       password_normal_strength.write(str(zxcvbn(passwd)));
       password_normal_strength.write("\n")
except:
   pass


try:
   for passwd in password_leet:
       password_leet_strength.write(str(zxcvbn(passwd)));
```

hash_converter.py:

```
import hashlib


# Open files containing the normal and leet passwords
password_normal_str = open("./passwords_sifted.txt", 'r').read()
password_leet_str = open("./passwords_leet.txt", 'r').read()

# Split the passwords by "\n" into an array of passwords
password_normal = password_normal_str.split("\n")
password_leet = password_leet_str.split("\n")

# Create two files that will contain the hashes
password_normal_hash = open("./password_normal_hash.txt", 'a')
password_leet_hash = open("./password_leet_hash.txt", 'a')

# For each password in the normal password list, hash it using sha1 and write it to
the normal password hash file
for passwd in password_normal:
   hash_object = hashlib.sha1(bytes(passwd, 'utf-8'))
   hex_dig = hash_object.hexdigest()
   password_normal_hash.write(hex_dig + "\n")

# For each password in the leet password list, hash it using sha1 and write it to the
leet password hash file
for passwd in password_leet:
```

hashcat_cracker.py:

```
import os


password_normal_hash_str = open("./password_normal_hash.txt", 'r').read()
password_leet_hash_str = open("./password_leet_hash.txt", 'r').read()

password_normal_hash = password_normal_hash_str.split("\n")
password_leet_hash = password_leet_hash_str.split("\n")
```

```
character_set = "?a"


os.system("echo \"\" > passwd_normal_hashcat.txt")
os.system("echo \"\" > passwd_leet_hashcat.txt")

for passwd in password_normal_hash:
   os.system("echo \"" + passwd + "\" > passwd.txt")
   os.system("hashcat -m 100 -a 3 passwd.txt " + ("?a" * passwd.len()) + " >>
passwd_normal_hashcat.txt")
   os.system("echo \n" + passwd + "\n >> passwd_normal_hashcat.txt")


for passwd in password_leet_hash:
   os.system("echo \"" + passwd + "\" > passwd.txt")
   os.system("hashcat -m 100 -a 3 passwd.txt " + ("?a" * passwd.len()) + " >>
```

entropy_calculator.py:

```
from asyncio.proactor_events import _ProactorBaseWritePipeTransport
import math
from operator import truediv
from sre_parse import SPECIAL_CHARS
from tokenize import Special
import math


password_normal_str = open("./passwords_sifted.txt", 'r').read()
password_leet_str = open("./passwords_leet.txt", 'r').read()
password_normal = password_normal_str.split("\n")
password_leet = password_leet_str.split("\n")


password_normal_entropy = open("./password_normal_entropy.txt", 'a')
password_leet_entropy = open("./password_leet_entropy.txt", 'a')


for passwd in password_normal:
   has_uppercase = False
   has_lowercase = False
   has_numbers = False
   has_symbols = False
   pool_size = 0
```

```
        if
            character.isuppe
            r():
            has_uppercase =
            Truepool_size +=
            26 break


    for character in passwd:
        if
            character.islowe
            r():
            has_lowercase =
            Truepool_size +=
            26 break


    for character in passwd:
        if
            character.isdigi
            t():has_numbers
            = True pool_size
            += 10 break


    SPECIAL_CHARS = "!”\"#$%&’'()*+,-./:;<=>?@[\]^_`{|}~ "


    if any(character in SPECIAL_CHARS for character in
        passwd):has_symbols = True
        pool_size += 35


    entropy = len(passwd) * math.log(pool_size, 2)


    print(passwd + ": " + str(entropy))
    password_normal_entropy.write(str(entropy) +
    "\n")


for passwd in
    password_leet:
    has_uppercase = False
    has_lowercase = False
    has_numbers = False
    has_symbols = False
    pool_size = 0
```

```
for character in passwd:
    if
        character.isuppe
        r():
        has_uppercase =
        Truepool_size +=
        26 break


for character in passwd:
    if character.islower():
        has_lowercase =
        Truepool_size += 26
        break

for character in passwd:
    if
        character.isdigit
        ():has_numbers =
        True pool_size +=

SPECIAL_CHARS = "!”\"#$%&''()*+,-

if any(character in SPECIAL_CHARS for character in
    passwd):has_symbols = True
    pool_size += 35

entropy = len(passwd) * math.log(pool_size,

print(passwd + ": " + str(entropy))
password_leet_entropy.write(str(entropy) +
```