Password Generation

Derek Moore and Garrett Beard

CSCE 477/877, University of Nebraska—Lincoln

***Abstract*—Having a strong and secure password is crucial to keeping your private information safe in an increasingly digital society. Almost every modern web application requires a particular structure a password must follow to strengthen account security. However, even with these guidelines, a password can still be susceptible to attacks, as many individuals use the same, or variations of their, password across multiple services. Because of this, password generators have become more popular and useful, allowing users to create random, nonpersonal passwords to use on each site. These services and their usefulness are what we focus on in our project.**

I. Introduction

The number of sites users must set up an account to access information is growing. According to Forbes, Google conducted a security survey in 2019, where it found that 66% of internet users stated they use the same password across multiple accounts [1]. If a service's records were ever breached, users reusing passwords could also have their private information exposed on additional, unrelated services. The user would benefit from using different passwords in this way. If these passwords were also random and nonpersonal, compromised data would be harder to crack since a pattern would be hard to find in breaking encrypted passwords. This implication is why password managers and password generators are crucial for a user.

While compromised data protection is not the purpose of this project, we are instead focusing on the generation and analysis of passwords, so that a user can better protect themselves from an attacker. This project is split into two categories; password generation using randomized character and using randomized words, testing a given password using brute force, dictionary attacks using a library of past breached passwords, and words in the English language. While services like this exist, this project sets out to provide additional analysis of the benefits of using different settings our generation offers.

The two attacks used to crack passwords can be categorized as offline attacks. An offline attack relies on having access to the hash of the target password. Getting access to the hash removes the potential of account locking. Many sites/services will lock an account after a certain number of login attempts. A brute force attack would not be feasible under such circumstances. A socially engineered dictionary attack could still be possible, although much less effective.

a. Randomized Passwords

These passwords are generated by combining random characters. These characters include lowercase and uppercase letters, digits, and/or symbols. These are considered the most secure password when generated with a high character count and including the optional uppercase letters, digits, and symbols.

b. Keyword Passwords

These passwords are generated by combining words within the english dictionary. While these passwords are often easier to memorize, they aren't as secure, as they are susceptible to dictionary attacks.

c. Brute Force Attack

The easiest way to attempt to crack a password is to repeatedly make guesses and see if they match. By guessing every possible password, a match is guaranteed to be found. The key issue with brute force attacks is how computationally intensive they are. The total number of possible passwords is given by S^n where S is the size of the character set and n is the length of the password.

d. Dictionary Attack

A dictionary attack is a much more targeted form of a brute force attack. Instead of randomly making guesses, a dictionary of likely passwords is formed. The dictionary can then simply be searched for the password.

II. Implementation

A. Password Generation

a. Randomized Passwords

To generate the password, a user can specify if they would also like it to include uppercase letters, digits, and/or symbols. If none of these are chosen, only randomized lowercase letters will be given. When the user gives these inputs, the program goes through and checks the given input. A list of all characters, digits, and symbols are given as lists. The program then determines which of the sets to add to the default lowercase set to give the entire possible set of characters a password can be generated with. The program then loops through the total amount of characters requested by a user, choosing a random character from the list each time.

b. Keyword Passwords

To generate a keyword password, the program takes advantage of the python package “english-words-py”. This package contains a set of all the words within the english dictionary [3]. The program then gets the desired password length. A user can also make the password more secure by enabling the additional settings offered, which include randomly uppercasing letters within the password, and also replacing letters with symbols that resemble the letter they are replacing.

Once the inputs are given, if the length is not valid given the requested word amount, an error will be displayed and the generation will exit. If it is valid, the program will then begin generation. This is accomplished by choosing a random word within the set. If the user specified random casing or symbol/digit replacement, this will then be performed. The list is then added to a list, and once the desired length is reached, a string of the password is outputted.

B. Password Testing

a. Brute Force Attack

To evaluate how long a brute force attack might take, the user first supplies the password they would like to test. A character set is then built around the characters contained in the password (e.g. digits, lower case, upper case, and symbols). An initial guess the same length at the supplied password is created by repeating the first character of the character set. If the guess does not match the password, then the first character of the guess is replaced by the succeeding character of the character set. When a character in the guess becomes ‘NULL’ a rollover occurs.

In the event of a rollover, a ‘NULL’ occurrence is replaced by the first character of the character set and the succeeding character in the guess is replaced by its succeeding character in the character set. The guess is then rechecked for ‘NULL’ characters. Once the guess contains no ‘NULL’ characters the brute force attack resumes. If the last character of the guess becomes ‘NULL’, it is assumed that the password failed to be cracked.

b. Dictionary Attack

To evaluate how long a dictionary attack might take, the user first supplies the password they would like to test. The SHA-1 password hash is then calculated. HaveIBeenPwned maintains a large database of leaked password hashes [2]. An API call is made to pwnedpasswords.com using the first five digits of the password hash. A set of the top 200 potential matching password hashes is returned. The set is then searched to see if it contains the user’s password hash.

C. Where to find

All of our code relating to the testing and generation of passwords can be found within the github repository [5].

III. Results

A. Main Menu

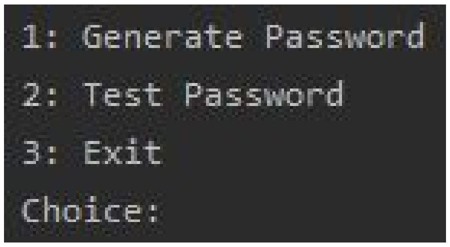


Figure 1: Screenshot of the main menu

B. Generation Menu

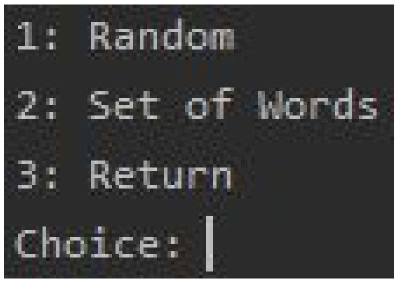


Figure 2: Screenshot of the password generation menu. This menu allows the user to choose between the two generation methods

C. Testing Menu

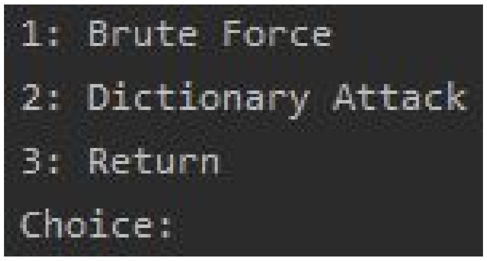
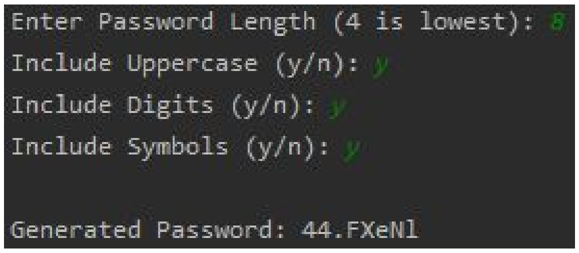


Figure 3: Screenshot of the password testing menu. This menu allows the user two choose between two different attack methods

D. Generated Password Using Random Setting



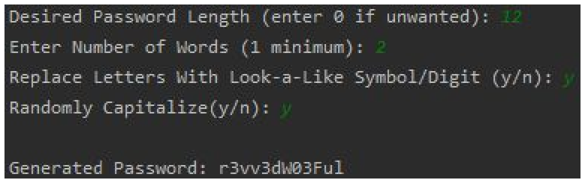
```
Enter Password Length (4 is lowest): 8
Include Uppercase (y/n): y
Include Digits (y/n): y
Include Symbols (y/n): y

Generated Password: 44.FXeNl
```

Figure 4: Screenshot of the options given to a user during the randomized character password generator

Settings Chosen (10 character length given) (symbols, digits, uppercase)			Generated Password
n	n	n	fotryddiel
n	n	y	jETvZWPbfP
n	y	n	e3cjvrembr
n	y	y	Zp6sG5mqPj
y	n	n	.e(.zp()$r
y	n	y	*:ByS.m_hd
y	y	n	7>cb*==$h%
y	y	y	tMch$2LkNT

Figure 5: Table of Passwords generated using the different options a user has.

E. Generated Password Using Random Setting



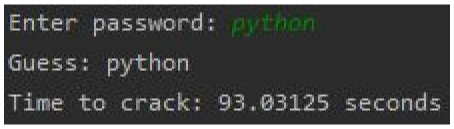
```
Desired Password Length (enter 0 if unwanted): 12
Enter Number of Words (1 minimum): 2
Replace Letters With Look-a-Like Symbol/Digit (y/n): y
Randomly Capitalize(y/n): y

Generated Password: r3vv3dW03Ful
```

Figure 6: Screenshot of the options given to a user during the keyword password generator

Settings Chosen (10 character length given) (uppercase, symbols/digits, # of words)			Generated Password
n	n	1	handmaiden
n	y	1	1nt3r$t1c3
y	n	1	propoRTIon
y	y	1	@MB@$$@d0R
n	n	2	ceaseheron
n	y	2	5t@rrwh3@t
y	n	2	forcEfunGi
y	y	2	5@L3mh31n3

Figure 7: Table of Passwords generated using the different options a user has during the keyword password generations.

F. Cracked Password



```
Enter password: python
Guess: python
Time to crack: 93.03125 seconds
```

Figure 8: Screenshot of the password testing results

IV. Analysis & Comparison

Generated Password	Brute Force Attack	Dictionary Attack
qfns	0.109 s	Secure
ghCB	1.422 s	Secure
Vk2H	4.016 s	Secure
mJ1|	13.141 s	Secure
hgqmn	2.5 s	Secure
ObEbN	139.953 s	Secure
f1FNL	323.266 s	Secure
yV=7*	1235.969 s	Secure
ouoizz	126.578 s	Secure
nDFdTV	>1800 s	Secure
N9aTTT	>1800 s	Secure
tFZ!2?	>1800 s	Secure
kimcal	75.0 s	0.016 s
fatevane	>1800 s	0.031 s
incomparable	>1800 s	0.031 s
mikehunt123	>1800 s	0.031 s

Figure 9: Table of different passwords and the amount of time taken to crack it using brute force and dictionary attacks

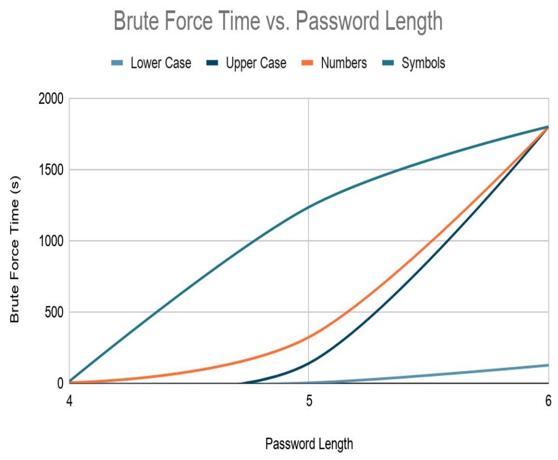


Figure 10: This graph visualizes the effects that password attributes have on the time it takes to complete a brute force attack. Note that the graph shows most ending at 1800s as this was the cutoff, these lines will go further. Each group contains the previous (e.g. Lower Case ⊂ Upper Case)

V. Conclusion

We addressed the security of passwords, including the problem of reusing passwords across accounts, length of password, and use of symbols and casing to increase security. We found that a password is most secure when it uses as many different characters as possible and is a decent length. An easy way to achieve a strong password is by stringing together random words. This enables the user to remember a longer password that is resistant to dictionary attacks. Any password longer than 12 characters with at least one uppercase character will remain immune to brute force attacks for the foreseeable future [4]. Along with this, we also created a way to test password security by running a brute force or dictionary attack, and a way for a user to generate passwords through randomized characters or keywords. This implementation and analysis will help educate internet users on how to better secure information across multiple accounts.

VI. Appendix

All of the code was written in python. The code can be found within the github repository [5]. To run, clone the repository by running “git clone” followed by the link given from github. If cloning does not work, ensure that github is properly installed on the machine [8]. Ensure that python is installed on the machine wanting to run the application. To install python, go to [6]. Once installed, install pip, a package manager for python. To install pip, go to [7]. Then, run “pip install english-words”. Then, run “pip install requests”. Then run “python main.py”. The application will then run, and it can be navigated using the command line menu.

VII. References

[1] Winder, Davey. “Google Confirms Password Security Update For 1 Billion Users.” *Forbes*, Forbes Magazine, 2 Oct. 2019, www.forbes.com/sites/daveywinder/2019/10/02/google-confirms-password-security-update-for-1-billion-users/?sh=4ed78afe2bfc.

[2] Hunt, Troy. “Pwned Password List.”

[3] Wiens, Matt. “English-Words.” *PyPI*, 20 Aug. 2018, pypi.org/project/english-words/. Python Library

[4] Hivesystems. “Time It Takes a Hacker to Brute Force Your Password.” *Reddit*, 8 Sept. 2020, i.redd.it/5g3ayy7pwxl51.jpg.

[5] gbeard2. “gbeard2/Pwd.” *GitHub*, 22 Nov. 2020, github.com/gbeard2/pwd.

[6] “Download Python.” *Python.org*, 22 Nov. 2020, www.python.org/downloads/.

[7] “Pip Installation.” *Installation - Pip 20.2.4 Documentation*, 22 Nov. 2020, pip.pypa.io/en/stable/installing/.

[8] “1.5 Getting Started - Installing Git.” *Git*, git-scm.com/book/en/v2/Getting-Started-Installing-Git.