



Bruteforce.py

By Malik Eddy

```
"""
Password brute-force algorithm
Python 3
"""

import string
from itertools import product
from time import time
from numpy import loadtxt

easy_dB = {'admin': 'admin'}
```

- Using the 'numpy' library
 - Good w/ large data
 - provides a high-performance multidimensional array object, and tools for working with
- Database data structure is a dict.
 - Key = username
 - Value = password

show_all_usr():

- Will show all valid users in database
- Users IDed by key

```
def show_all_usr():  
    print('\n')  
    print('          USERS          ')  
    for key in easy_db.keys():  
        print(key)  
    print('\n')  
    menu()
```

• add_usr():

- Will add a user to the database
 - If username != already exists
 - If password == 5 chars
- easy_db[usr]= pwd
 - Key -> user
 - Value user -> pwd

```
def add_usr():  
    usr = input("\nEnter username: ")  
    if (usr in easy_db.keys()):  
        print('\nUser already exists\n')  
    else:  
        pwd = input("Enter password (5 characters only): ")  
        if (len(pwd) == 5):  
            easy_db[usr] = pwd  
            print('\nUSER CREATED\n')  
            menu()  
        else:  
            print('\nInvalid password\nUSER NOT CREATED\n')  
            menu()
```

Permutations

- A Permutation is an ordered Combination
- For this program
 - Password = 5 chars → _ _ _ _ _
 - #of letters in alphabet == $26 * 2$ (to include uppercase)
 - #of digits in base 10 decimal number == 10 (0-9)
 - #of possibilities for 1st char = $26 * 2 + 10 == 62$
 - #of possibilities for 5 char password → $62 * 62 * 62 * 62 * 62$
 - 625 possible combinations (916,132,832 combinations)
 - IT TAKES A LOT OF TIME + MEMORY TO DO EVERY COMBINATION
 - A 'dictionary attack' helps to reduce time

The BruteForce

```
def bruteforce(username, max_nchar=5): #change max_nchar to accept larger pass|

    password = easy_dB[username]

    print('\n1) Comparing with most common passwords / first names')
    common_pass = loadtxt('probable-v2-top12000.txt', dtype=str)
    common_names = loadtxt('middle-names.txt', dtype=str)
    cp = [c for c in common_pass if c == password]
    cn = [c for c in common_names if c == password]
    cnl = [c.lower() for c in common_names if c.lower() == password]

    if len(cp) == 1:
        print('\n=====+[+]SUCCESS[+]=====', '\nUSERNAME: ', username,
              '\tPASSWORD:', cp)
        return cp
    if len(cn) == 1:
        print('\n=====+[+]SUCCESS[+]=====', '\nUSERNAME: ', username,
              '\tPASSWORD:', cn)
        return cn

    if len(cnl) == 1:
        print('\n=====+[+]SUCCESS[+]=====', '\nUSERNAME: ', username,
              '\tPASSWORD:', cnl)
        return cnl
```

- Bruteforce(username, max_nchar=5):
 - Will take in a username and
 - Try every possible combination until passwords match
 - 1st part is a dictionary attack not bruteforce
 - Used to save time



2 Decent Sized Text Files

- probable-v2-top12000.txt
 - < 1000 entries
- middle-names.txt
 - < 9800 entries
- Total entries ~ 10k entries

The Hard Work

- 2) Digits Cartesian Product
 - Tries all possible number combinations
- 3) ASCII Lowercase + Digits
 - Tries all possible lowercase + digit combinations
- 4) ASCII Uppercase/Lowercase + Digits
 - Tries all possible uppercase + lowercase + digit combinations

```
print('\n2) Digits Cartesian Product')
for l in range(max_nchar, max_nchar + 1):
    generator = product(string.digits, repeat=int(l))
    print("\t...%d digit" % l)
    p = product_loop(password, generator, username)
    if p is not False:
        return p
```

```
print('\n3) ASCII Lowercase + Digits')
for l in range(max_nchar, max_nchar + 1):
    print("\t...%d char" % l)
    generator = product(string.ascii_lowercase + string.digits, repeat=int(l))
    p = product_loop(password, generator, username)
    if p is not False:
        return p
```

```
print('\n4) ASCII Uppercase/Lowercase + Digits')
# If it fails, we start brute-forcing the 'hard' way
# Same as possible_char = string.printable[:-5]
all_char = string.ascii_uppercase + string.digits + string.ascii_lowercase

for l in range(max_nchar, max_nchar + 1):
    print("\t...%d char" % l)
    generator = product(all_char, repeat=int(l))
    p = product_loop(password, generator, username)
    if p is not False:
```