For tasks one to three I implemented simple algorithms that cracked the passwords in a very short amount of time. For task 1, used the “product” method of “itertools” to generate all possible strings out of the possible password characters, which in this case were a-z and 1-9. The time it takes to generate these strings increases exponentially with the length of the strings, so I programmed the function to stop when the number of cracked passwords matches the number of input passwords, so no strings are generated unnecessarily. This greatly decreased run time. For the second task, I imported the dictionary using urlopen, split the dictionary into lines (each line was a different password), hashed the passwords, and compared them with the input passwords. For task three, I implemented the same method, but I added the salts to the end of the passwords before hashing them, and then removed them after by removing the last 8 characters of the string.

For task 4, I chose to implement “password mangling” into my assignment. I had to research this topic, as I was not very familiar with it. I had heard of John the Ripper before, as I have an interest in cyber-attacks. John the Ripper is an open-source password cracker and is pretty much the ultimate implementation of password mangling and cracking. It was programmed in C, although scripts to run it in python exist. I, however, only sought to try and crack mangled passwords. Mangled passwords are passwords which swap letters with numbers and symbols. The most common rules (which I used) are:

* Replace ‘a’ with ‘@
* Replace ‘e’ with ‘3’
* Replace ‘i’ with ‘!’
* Replace ‘o’ with ‘0’
* Replace ‘s’ with ‘$’
* Replace ‘t’ with ‘7’

This technique is wrongly assumed to make passwords uncrackable when the technique follows some predictable rules. I am guilty myself of using these replacements, thinking that my password is less guessable. While it does improve upon basic passwords, mangled passwords can be generated from a word list relatively quickly. Thus, mangled passwords are susceptible to a dictionary attack if a mangling function is implemented. This was my goal for this task.

My algorithm was very basic, as I wanted something that would work, and I could not implement anything more ambitious in time. My further attempts did not run properly. My basic algorithm would run through the wordlist (the same dictionary as used in task two and three), hash the passwords, and check them against the inputted passwords. This was the same as my implementation of the dictionary attack. The next step is the additional step which accounted for the possibility of password mangling techniques being used. If there was no match between the password from the wordlist and the passwords inputted (both in hashed form), then the function “passwordMangler” would use the “replace” method to swap letters with numbers and symbols according to the aforementioned mangling rules. In developing this method, I initially converted the password string to a tuple of characters, replacing the characters as instances of them were found. However, this caused issues, as the function was only mangling one instance of a character at a time. In finding a fix, I discovered that the “replace” method was simpler and fixed the issue.

After the password was mangled, it would be encoded, hashed, and checked against the inputted passwords. If no match was found, the algorithm would move onto the next password. It would repeat this for all the passwords in the dictionary. Were the dictionary to be exceptionally long, I would have implemented a feature to stop the algorithm when the list of cracked passwords matches in length to the list of inputted passwords, but password lists like the one I used are typically of manageable length and thus have a good runtime. My algorithm ran in under a second.   
  
The passwords I used as input were chosen to test the functionality of the algorithm. They include various mangled and unmangled passwords. I also included the same password twice, once mangled, once not mangled, to check that the algorithm would crack both. All the passwords inputted were found. An issue with my implementation was that it mangles the entire word, using every rule, at once. For example, from the password “windows”, the mangled password would be “w!nd0ws”. The versions generated from using one rule at a time will not be generated by this algorithm. This means “wind0ws” and w!ndows” will not be tested. While ideally, I would have implemented this feature, I could not get it to run by the deadline.

A problem I did fix, however, was that when the same password was inputted in both mangled and unmangled form, the algorithm would only detect one iteration of the password. I realised that this issue was caused because I was only mangling the password if the original password didn’t match. I removed this if/else statement and both versions of the password were found.

References:

Wikimedia Foundation. (2022, October 2). *John the ripper*. Wikipedia. https://en.wikipedia.org/wiki/John\_the\_Ripper

CODE:

Task 1:

import hashlib, itertools

import string

def brute\_force\_attack(hashed\_passwords):

global counter

passwordCharacters = ['a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z','0','1','2','3','4','5','6','7','8','9']

length = 1

while True:

if counter < noOfPasswords: #making sure that we haven't found all the passwords

for pwd in itertools.product(passwordCharacters, repeat=length): #generate all strings

pwd\_str = "".join(pwd) #convert tuple to string

hashed\_pwd = hashlib.sha512(pwd\_str.encode()).hexdigest() #hash string

if hashed\_pwd in hashed\_passwords: #compare with hashed passwords inputted

crackedPasswords.append(pwd\_str)

counter += 1 #increase counter

break

if not hashed\_passwords:

return

length += 1 #increase length of string

else:

return

hashed\_passwords = ['f14aae6a0e050b74e4b7b9a5b2ef1a60ceccbbca39b132ae3e8bf88d3a946c6d8687f3266fd2b626419d8b67dcf1d8d7c0fe72d4919d9bd05efbd37070cfb41a','e85e639da67767984cebd6347092df661ed79e1ad21e402f8e7de01fdedb5b0f165cbb30a20948f1ba3f94fe33de5d5377e7f6c7bb47d017e6dab6a217d6cc24','4e2589ee5a155a86ac912a5d34755f0e3a7d1f595914373da638c20fecd7256ea1647069a2bb48ac421111a875d7f4294c7236292590302497f84f19e7227d80','afd66cdf7114eae7bd91da3ae49b73b866299ae545a44677d72e09692cdee3b79a022d8dcec99948359e5f8b01b161cd6cfc7bd966c5becf1dff6abd21634f4b']

crackedPasswords = []

counter = 0

noOfPasswords = len(hashed\_passwords)

brute\_force\_attack(hashed\_passwords)

if len(crackedPasswords) != 0:

print("The passwords are: ", crackedPasswords)

else:

print("No passwords found")

TASK 2:

from urllib.request import urlopen, hashlib

passwordDictionary = str(urlopen("https://gist.githubusercontent.com/PeterStaev/e707c22307537faeca7bb0893fdc18b7/raw/6c591618b8c0c46cb7db7a6966754455164cb433/PasswordDictionary.txt").read(), 'utf-8')

hashedPasswords =['31a3423d8f8d93b92baffd753608697ebb695e4fca4610ad7e08d3d0eb7f69d75cb16d61caf7cead0546b9be4e4346c56758e94fc5efe8b437c44ad460628c70','9381163828feb9072d232e02a1ee684a141fa9cddcf81c619e16f1dbbf6818c2edcc7ce2dc053eec3918f05d0946dd5386cbd50f790876449ae589c5b5f82762','a02f6423e725206b0ece283a6d59c85e71c4c5a9788351a24b1ebb18dcd8021ab854409130a3ac941fa35d1334672e36ed312a43462f4c91ca2822dd5762bd2b','834bd9315cb4711f052a5cc25641e947fc2b3ee94c89d90ed37da2d92b0ae0a33f8f7479c2a57a32feabdde1853e10c2573b673552d25b26943aefc3a0d05699','0ae72941b22a8733ca300161619ba9f8314ccf85f4bad1df0dc488fdd15d220b2dba3154dc8c78c577979abd514bf7949ddfece61d37614fbae7819710cae7ab','6768082bcb1ad00f831b4f0653c7e70d9cbc0f60df9f7d16a5f2da0886b3ce92b4cc458fbf03fea094e663cb397a76622de41305debbbb203dbcedff23a10d8a','0f17b11e84964b8df96c36e8aaa68bfa5655d3adf3bf7b4dc162a6aa0f7514f32903b3ceb53d223e74946052c233c466fc0f2cc18c8bf08aa5d0139f58157350','cf4f5338c0f2ccd3b7728d205bc52f0e2f607388ba361839bd6894c6fb8e267beb5b5bfe13b6e8cc5ab04c58b5619968615265141cc6a8a9cd5fd8cc48d837ec','1830a3dfe79e29d30441f8d736e2be7dbc3aa912f11abbffb91810efeef1f60426c31b6d666eadd83bbba2cc650d8f9a6393310b84e2ef02efa9fe161bf8f41d','3b46175f10fdb54c7941eca89cc813ddd8feb611ed3b331093a3948e3ab0c3b141ff6a7920f9a068ab0bf02d7ddaf2a52ef62d8fb3a6719cf25ec6f0061da791']

crackedPasswords = []

def dictionaryAttack(inputPasswords):

for password in passwordDictionary.split('\n'): #split dictionary into lines, and thus passwords

encodedPassword = password.encode('utf-8')

digest = hashlib.sha512(encodedPassword.strip()).hexdigest()

if digest in hashedPasswords:

crackedPasswords.append(password)

dictionaryAttack(hashedPasswords)

if len(crackedPasswords) == 0:

print("No passwords found")

else:

print("The passwords are: ", crackedPasswords)

TASK 3:

from urllib.request import urlopen, hashlib

passwordDictionary = str(urlopen("https://gist.githubusercontent.com/PeterStaev/e707c22307537faeca7bb0893fdc18b7/raw/6c591618b8c0c46cb7db7a6966754455164cb433/PasswordDictionary.txt").read(), 'utf-8')

hashedPasswordsAndSalts = [('63328352350c9bd9611497d97fef965bda1d94ca15cc47d5053e164f4066f546828eee451cb5edd6f2bba1ea0a82278d0aa76c7003c79082d3a31b8c9bc1f58b','dbc3ab99'),('86ed9024514f1e475378f395556d4d1c2bdb681617157e1d4c7d18fb1b992d0921684263d03dc4506783649ea49bc3c9c7acf020939f1b0daf44adbea6072be6','fa46510a'),('16ac21a470fb5164b69fc9e4c5482e447f04f67227102107ff778ed76577b560f62a586a159ce826780e7749eadd083876b89de3506a95f51521774fff91497e','9e8dc114'),('13ef55f6fdfc540bdedcfafb41d9fe5038a6c52736e5b421ea6caf47ba03025e8d4f83573147bc06f769f8aeba0abd0053ca2348ee2924ffa769e393afb7f8b5','c202aebb'),('9602a9e9531bfb9e386c1565ee733a312bda7fd52b8acd0e51e2a0a13cce0f43551dfb3fe2fc5464d436491a832a23136c48f80b3ea00b7bfb29fedad86fc37a','d831c568'),('799ed233b218c9073e8aa57f3dad50fbf2156b77436f9dd341615e128bb2cb31f2d4c0f7f8367d7cdeacc7f6e46bd53be9f7773204127e14020854d2a63c6c18','86d01e25'),('7586ee7271f8ac620af8c00b60f2f4175529ce355d8f51b270128e8ad868b78af852a50174218a03135b5fc319c20fcdc38aa96cd10c6e974f909433c3e559aa','a3582e40'),('8522d4954fae2a9ad9155025ebc6f2ccd97e540942379fd8f291f1a022e5fa683acd19cb8cde9bd891763a2837a4ceffc5e89d1a99b5c45ea458a60cb7510a73','6f966981'),('6f5ad32136a430850add25317336847005e72a7cfe4e90ce9d86b89d87196ff6566322d11c13675906883c8072a66ebe87226e2bc834ea523adbbc88d2463ab3','894c88a4'),('21a60bdd58abc97b1c3084ea8c89aeaef97d682c543ff6edd540040af20b5db228fbce66fac962bdb2b2492f40dd977a944f1c25bc8243a4061dfeeb02ab721e','4c8f1a45')]

saltedPasswordDictionary = []

crackedPasswords = []

#this is mostly the same as task 2

def dictionaryAttack(inputPasswords):

for password in passwordDictionary.split('\n'):

for pair in hashedPasswordsAndSalts:

saltedPasswordDictionary.append(password + str(pair[1])) #add salt to end of password

for saltedPassword in saltedPasswordDictionary:

saltedEncodedPassword = saltedPassword.encode('utf-8')

digest = hashlib.sha512(saltedEncodedPassword.strip()).hexdigest()

for pair in hashedPasswordsAndSalts:

if digest == pair[0]: #compare with hashed passwords, from the list of tuples inputted

crackedPasswords.append(saltedPassword[:-8]) #remove salt from end of password and add to dictionary

dictionaryAttack(hashedPasswordsAndSalts)

if len(crackedPasswords) == 0:

print("No passwords found")

else:

print("The passwords are: ", crackedPasswords)

TASK 4:

from urllib.request import urlopen, hashlib

passwordDictionary = str(urlopen("https://gist.githubusercontent.com/PeterStaev/e707c22307537faeca7bb0893fdc18b7/raw/6c591618b8c0c46cb7db7a6966754455164cb433/PasswordDictionary.txt").read(), 'utf-8')

passwords = ["W!nd0w$5", "Qu@k3","Behappy", "Windows10"] # mangled and non-mangled passwords to be cracked

hashedPasswords =[]

crackedPasswords = []

for i in passwords:

hashedPasswords.append(hashlib.sha512(i.encode('utf-8')).hexdigest()) #hash the passwords that I entered above

#replace letters with numbers and symbols according to some basic mangling rules

def passwordMangler(password):

password = password.replace('a', '@')

password = password.replace('e', '3')

password = password.replace('i', '!')

password = password.replace('o', '0')

password = password.replace('s', '$')

password = password.replace('t', '7')

return password

def dictionaryAttack(inputPasswords):

for password in passwordDictionary.split('\n'):

encodedPassword = password.encode('utf-8')

digest = hashlib.sha512(encodedPassword.strip()).hexdigest()

if digest in hashedPasswords:

crackedPasswords.append(password)

else: #if the password is not found, try mangling it

mangledPassword = passwordMangler(password)

encodedPassword = mangledPassword.encode('utf-8')

digest = hashlib.sha512(encodedPassword.strip()).hexdigest()

if digest in hashedPasswords:

crackedPasswords.append(mangledPassword)

dictionaryAttack(hashedPasswords)

if len(crackedPasswords) == 0:

print("No passwords found")

else:

print("The passwords are: ", crackedPasswords)