Hao Moy

CS[4351]

Dr. Longpre

Assignment 1

**Online**

The assignment was coded in Python 3, the attached scripts are documented which should make it easy to find the functionalities textually described below. The most challenging part of this section was to create a script capable of connecting to the provided UTEP webpage and interacting with it. I decided to use the Selenium library for Python as it allows you to connect to a webpage through a browser of choice and interact with the elements within it. Once that was achieved, this was refactored into a function that receives a username and a password as input and enters those arguments into their respective html elements. Afterwards, by using ASCII bounds, I used a nested for loop to compute all the possible combinations of two lowercase letters and the results were submitted to the webpage. It took the algorithm 342 seconds to crack the password. Most of this time was consumed by the lag between submissions.

**Results:**

Username: jonathan34\_-qFq

Password: oe

Time elapsed: 342 seconds

**Random**

The webpage was accessed as described in the Online section. I decided to manually create a dictionary containing all the users as keys and a list containing their respective salt and hashed password as their values. I opted not to create a script to extract this information because there were only 10 users provided, and it would have been more time consuming to automate it. Moreover, I also decided to create an array containing all the valid characters as this already existed in order to compute all the combinations possible of sets of length 1-10.

I created a function that generates an array of argument-specified length. These arrays will serve as placeholders for the creation of sets with all possible combinations of valid characters. Afterwards, I created a recursive function to compute all the combinations of specified characters for a set of specified size. This functions is equivalent to using the itertools product() function from Python. Initially, I used this function to generate the sets, but product() returns a list with all the possible combinations. This is problematic due to the number of possibilities computed per set. The maximum size of a python list is of 536,870,912 elements, which is not enough to house all the possible sets of length 6 and above. I am aware that computing all the possibilities of sets of size 6 and above would have taken several days anyways, but at least through a recursive solution, the problem eventually becomes solvable. Otherwise, by using product() the program would have crashed by trying to populate a list with more than 536,870,912 elements. A match was identified if a generated set (hashed with sha256 and sha1 in that order) was equal to one of the provided hashed passwords. A for loop iterating from 1 to 10 inclusive was used to call the previously mentioned functions.

Five accounts were broken into, nevertheless, I accidentally copied and pasted something from the console, terminating the process before I could retrieve the username and password for the user with a password of length 5. You can look at the log-ins as proof that the account was accessed.

**Results:**

######################################

Currently in combination: 1

User: mary34-qFq

Password: V

Time: 2.3788866996765137

######################################

Currently in combination: 2

User: sofia34-qFq

Password: Au

Time: 2.5660560131073

######################################

Currently in combination: 3

User: linda34-qFq

Password: 9UR

Time: 3.105642557144165

######################################

Currently in combination: 4

User: diego34-qFq

Password: t46U

Time: 54.782586097717285

######################################

Currently in combination: 5

**Dictionary**

The webpage was accessed as described in the Online section. I decided to manually create a dictionary containing all the users as keys and a list containing their respective salt and hashed password as their values. I opted not to create a script to extract this information because there were only 10 users provided, and it would have been more time consuming to automate it. I searched online for the text file described in the assignment handout, downloaded, it, and extracted all its words into an array. Moreover, I reused the same array of valid character I created for the previous section. This array was used to eliminate all invalid characters form the words within the dictionary.

Afterwards, I iterated through the user dictionary and for every user, the word list array was traversed until a match was found. A match was determined when a word from the dictionary was concatenated with the user’s salt, and then hashed with sha256 and then sha1. There is an error in the provided instructions that states that the hashed password was obtained by hashing a word with sha256, concatenating the salt, and then hashing the concatenation with sha1. The salt was added before any hashing was done to the password. It took me an entire afternoon to figure that out, as I assumed it was a mistake in my code.

**Results:**

############

User: james34\_qFq

Password: uncovered

Time: 0.27776074409484863

############

User: linda34\_qFq

Password: blvd

Time: 0.03372836112976074

############

User: sofia34\_qFq

Password: pseudointellectual

Time: 0.16321516036987305

############

User: santiago34\_qFq

Password: earings

Time: 0.06745457649230957

############

User: isabella34\_qFq

Password: apiarists

Time: 0.021823644638061523

############

User: diego34\_qFq

Password: carpenters

Time: 0.034223079681396484

############

User: robert34\_qFq

Password: cull

Time: 0.0540616512298584

############

User: mary34\_qFq

Password: nutshells

Time: 0.14383983612060547

############

User: patricia34\_qFq

Password: telegraphy

Time: 0.21228742599487305

############

User: daniela34\_qFq

Password: huskily

Time: 0.10167884826660156

############

Total time: 1.110073328

**Salting**

Salting is a technique to safeguard passwords when stored. A salt is data that is used as additional input to a one-way hashing function for passwords. Typically, a password and its salt will be concatenated and then hashed, which allows for its storage without plaintext. Cracking these passwords is much more time consuming and difficult for attackers because not only do they have to figure out the hashing function used on the salted password, but the actual salt itself. This is without to mention that access to this information is in most cases very difficult. Moreover, users that use “common” or easily decipherable passwords have an extra layer of protection through salting.

**References:**

Dictionary used: wordsEn.txt

<https://github.com/kvirc/kvirc-scripts/tree/master/BelovedInDistress>

Dictionary Option 2:

This dictionary contains a list of the 100,00 most common English words in order of frequency.

<https://github.com/first20hours/google-10000-english>

Dictionary Option 3:

This dictionary contains a list of the top 1,000,000 most common passwords.

<https://github.com/danielmiessler/SecLists/blob/master/Passwords/Common-Credentials/10-million-password-list-top-1000000.txt>