Akshat Tyagi (N11436815)

Assignment 1-2: : Password Cracking Tool using Python

I. Technique Employed -

The password dump consisted of password leaks from three different sources - Yahoo, LinkedIn, and Formspring. Each of these sources used a different method of storing the password. For all three projects, the cracked passwords have been stored in a file named 'crackedpasswords.txt'. To crack the LinkedIn and FormSpring password dumps, a wordlist was used to carry out a dictionary attack. In this attack, a string from a wordlist is converted into a hashe. The hash is then compared with the hashed passwords stored in the password dump. A match would indicate that the string from the wordlist was the original password. Information regarding the wordlist used has been illustrated below -

Name	**Source**	**Word Count**	**Format**
hk_hlm_founds.txt	http://home.btconnect.com/md5decrypter/hlm/	171 Million words **Actual extent used due to size restrictions:** LinkedIn Cracker: 100k words FormSpring Cracker: 2.5 Million words	.txt

The word list contained a lot of duplicate data. This was removed using the uniq command of Linux as follows:
cat hk_hlm_founds.txt | uniq > hk_hlm_founds_noduplicate.txt

The technique used to crack each of the dumps has been described below.

1. **Yahoo** : The Yahoo password dump consisted of some database information, followed by strings with information such as ID, Username, and the actual password. The file was first read in a variable 'file_lines'. It was then iterated from the first to the last line in the part of the file where the raw strings were stored. Using the split() function of Python, the string was split into 3 constituents using ':' as a delimiter. The part of the string containing the password was appended to the raw string and written in the 'crackedpasswords.txt' file.

2. **LinkedIn**: The LinkedIn password dump consisted of SHA1 hashes of the original user passwords. To crack these hashes, a dictionary attack was used. The hashes in the file SHA1.txt were read and stored in a set. Now, the wordlist was iterated. At each iteration, it was checked if the SHA1 hash of the word is present in the set. If yes, a match was found and the word was written in crackedpasswords.txt, along with the original hash. In order to hash the words, Python's 'hashlib' library was used, which could use both SHA1 and SHA256 to hash passwords.

3. **FormSpring:** The FormSpring dump consisted of SHA256 hashes. The hashing technique used was specifically developed to protect against dictionary attacks by adding a salt value to the original string and then hashing it. However, it was observed that there was a flaw with the way the hashes were computed by the company. The hashes used consist of salt values between 00 and 99. This vulnerability was exploited to crack the passwords. The hashes in SHA256.txt were stored in a set. The words in the wordlist were then hashed by adding a salt value between 00 and 99 to the original text. Each of the 100 hashes formed for one word were then searched for in the set and if there was a hit, the corresponding word was written to the crackedpasswords.txt file along with the hash. When I used the original 171

million strong wordlist, there were over a thousand hits. However, after shortening the list to 2 million due to size restrictions (File size was over 2 GB), there were only 175 hits.

II. Alternate Techniques Considered -

Although the implemented approach did yield the commensurate result, two alternate approaches were considered for the project. These approaches have been illustrated below -

1. Instead of using a wordlist of common words and strings, a brute force based approach was considered, where a wordlist would be developed using brute force by forming as many combinations as possible using a particular character set and a predetermined string length. This approach wasn't used because the resulting wordlist would be too big in size and would not be as exhaustive as a wordlist of common phrases. The limited computing power offered by a PC would also prove to be a hindrance.
2. For the LinkedIn and FormSpring password dumps, instead of calculating a hash for a word in a wordlist and comparing it with the hashes stored in a set, an alternate approach was tried where the hash value from the password dump was used as a comparison tool by iterating the hash over the hashes of the words in the wordlists. Such an approach would be too time consuming and was thus not used.

III. Password Storage Technique and Corresponding Difficulty -

Each of the 3 password dumps used a different storage technique to store the password. These techniques have been discussed below and each of the techniques have been rated for the difficulty faced in cracking them -

Password Dump	**Storage Technique**	**Difficulty Rating**
Yahoo (password.file)	The passwords were stored unencrypted as a part of a string, along with the email address.	**3** (lowest) **Reason**: There was no effort made to hide the password. It only took a few string operations to create the file displaying the desired output.
LinkedIn (SHA1.txt)	The passwords were stored as SHA1 hashes. This hashed password was 160. It was observed that most of the hashed values had their first 5 characters replaced with 0.	**2** **Reason**: The password cracking process involved the use of dictionary attack. It was certainly tougher than cracking the Yahoo passwords. Also, it took a while to figure out the fact that the first 5 characters of the password was replaced with 0s. This didn't allow any matches in the initial stages.
FormSpring (formspring.txt)	The passwords were stored using SHA256 encryption. A salt was added to the original password before the hashing process. The generated hash was 256 bits long.	**1** (highest) **Reason**: Each password could have 100 different salt values. This made the cracking process slightly harder.