Dictionary Attack

Dictionary attack is method crackers use to penetrate systems using a library of commonly used passwords. This method is best used for offline attacks, as it can take millions of unsuccessful attacks before the answer is found. The dictionary attack was performed using the dictionary from crackstation.net, which includes over 1 million different passwords. The attacker uses each password in the dictionary until one works. This method has one major drawback, in that if the password is not in the dictionary the attacker will never gain entrance. This is the reason most dictionaries used for this attack are extremely large.

For the dictionary attack portion of the project, we attacked twenty different passwords three different times. The top ten passwords are from a compiled list of the most common passwords in use today. The second ten passwords are randomly user generated passwords. The first round was to attempt to find the password without the password being encrypted. The second was when the password was encrypted using SHA-256, and the third was using MD5 to encrypt the password. The passwords and the results are found below.

This project we used three different python scripts in order to search for the password using the dictionary attack. The passwords were encrypted using a known salt, in order to speed up the process. Because the salt has to be of a certain length and has specific restrictions, if an attacker was able to access the hash tables, they would be able to find the answer using the same method as below and attempting every salt.

|  |  |  |  |
| --- | --- | --- | --- |
| Success Table | | | |
|  | Plaintext | SHA-256 | MD5 |
| 123456 | Found | Not Found | Found |
| 123456789 | Not Found | Not Found | Not Found |
| qwerty | Found | Not Found | Found |
| 12345678 | Found | Not Found | Found |
| 111111 | Found | Not Found | Found |
| 1234567890 | Not Found | Not Found | Not Found |
| 1234567 | Found | Not Found | Found |
| Password | Found | Not Found | Found |
| 123123 | Found | Not Found | Found |
| 987654321 | Not Found | Not Found | Not Found |
| pqRtSm | Not Found | Not Found | Not Found |
| 123qwe123Q | Not Found | Not Found | Not Found |
| 0NotAdmin1 | Not Found | Not Found | Not Found |
| solAno | Not Found | Not Found | Not Found |
| zzzAAA | Not Found | Not Found | Not Found |
| 12zzyxyzed | Not Found | Not Found | Not Found |
| ognam234 | Not Found | Not Found | Not Found |
| ocnoycxk | Not Found | Not Found | Not Found |
| dospswra00 | Not Found | Not Found | Not Found |
| xabtohp | Not Found | Not Found | Not Found |

A table showing the password and if it was found using the different encryptions.

As you can see in the table from above, almost all the top ten passwords we made were found in the dictionary. None of the personal passwords we made were found in the dictionary. These findings highlight the problem with the dictionary attack, in that if the word is not in the dictionary, you will never find it. What about the SHA-256 runs? Those runs produced no attacks, with each word taking the maximum amount for time to produce no results. For the purposes of this project, maximum time was at 24 hours. A full list of the runtimes for each are found in the table and graph below.

Graph representing the runtime of the dictionary attacks on the different encryptions. As you can see the SHA-256 took the longest time which is represented with the peak in the middle.

|  |  |  |  |
| --- | --- | --- | --- |
| Runtime for Dictionary Attack | | | |
|  | Plaintext | SHA-256 | MD5 |
| 123456 | 0.0965 | 86400 | 823.4784 |
| 123456789 | 0.1 | 86400 | 1045 |
| qwerty | 0.064121 | 86400 | 393.7979 |
| 12345678 | 0.079994 | 86400 | 824.5169 |
| 111111 | 0.085318 | 86400 | 823.8496 |
| 1234567890 | 0.1 | 86400 | 1045 |
| 1234567 | 0.0834648 | 86400 | 850.6463 |
| Password | 0.069825 | 86400 | 628.3607 |
| 123123 | 0.1 | 86400 | 1045 |
| 987654321 | 0.1 | 86400 | 1045 |
| pqRtSm | 0.1 | 86400 | 1045 |
| 123qwe123Q | 0.1 | 86400 | 1045 |
| 0NotAdmin1 | 0.1 | 86400 | 1045 |
| solAno | 0.1 | 86400 | 1045 |
| zzzAAA | 0.1 | 86400 | 1045 |
| 12zzyxyzed | 0.1 | 86400 | 1045 |
| ognam234 | 0.1 | 86400 | 1045 |
| ocnoycxk | 0.1 | 86400 | 1045 |
| dospswra00 | 0.1 | 86400 | 1045 |
| xabtohp | 0.1 | 86400 | 1045 |

This table shows the run time for the cracking of each password using the specified encryption. .1, 86400, and 1045 all represent the maximum time for their respective encryption runs.

Results

As you can see in the table and graph, the runtime for SHA-256 took much longer than the other runtimes, in fact it is the only test to time out at the 24-hour mark. The other algorithms have a maximum time of the amount of time needed to search through the entire library of passwords. The fact that the entire run couldn’t go through one iteration in the library for SHA-256 is very surprising. MD5 does perform 30 less operations while performing its encryption, but it is surprising the 30 steps amount to such a large time difference.

If given more time, I would like to attempt to crack a larger variety of encryptions. I would also like to try to implement a new program that is better than the built-in Kali password cracker. To implement either of these would take upwards of a year to do properly, but would be an interesting and rewarding challenge.