CSE 539 – Homework 3 **Spring 2015**

Author 1: Mukund Manikarnike (1208597425)

Author 2: Lakshmi Srinivas (1208635554)

Overview

This report explains the following methods used in finding passwords given a few stolen hashes

1. Brute force method
2. Rainbow table

Certain aspects of hashing and password rules used in this assignment are explained below

1. The hashing algorithm used is MD5. The last 32 bits of a hash produced by MD5 are used to produce a 32-bit hash.
2. The rules limiting a password are that
   a. The length would be 4 characters (32-bits) and
   b. The password would take a form that the following regular expression indicates
      - [0-9][a-z][A-Z]

Brute Force

This method of finding passwords is simple and straight-forward. The approach used is that the entire set of passwords possible according to the password limiting rules are generated, hashed and compared with the hashes stolen. If they match the hash provided, it means that the password that produced this hash is the password for the hash whose password is desired. The entire list of passwords obtained for each hash is listed in the table below.

Hash	Password
0x97e75d32	A1B2
0x19fbc7c1	LOVE
0x8f6bb61b	MOVE
0x88df723c	lOvE
0x655ca818	mOvE
0x14928501	1a2b
0x3974cffc	LoVe
0x58712b2b	MoVe
0x7e1d96fd	love
0x8e564270	move

Total Time taken to complete brute-forcing of all 10 password hashes was **110s**

Rainbow Table

This method involves 2 stages which are explained below

1. Rainbow Table generation
2. Rainbow Table Lookup

Rainbow Table generation

A rainbow table is generated as follows

1. Select **‘n’** random words which qualify as possible passwords according to the password limiting rules
2. Apply the hash function on it and the reduce function on the hash that results, in an alternating fashion for **‘p’** times.
3. Store the **‘n’** inputs and outputs of this procedure in an **‘n x 2’** matrix

This matrix will be used in finding the passwords from the stolen hash. Note that this is a precomputed table that will be used once the hashes are stolen and will not be generated at the time of the attack. Before, we proceed to understanding how the look-up is done, a few key points in the generation need to be explained which are

1. What is a reduction function?
2. How does one choose **‘n’** random words?
3. How does one choose **‘p’**, the number of hash chains to be applied?

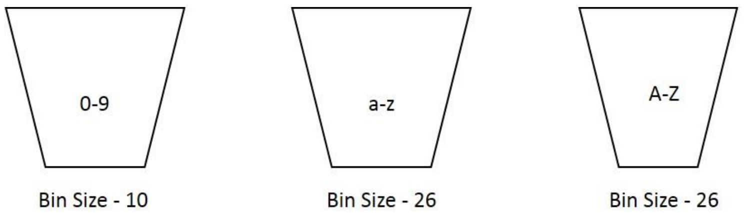
Reduction Function

A reduction function is a function which takes a hash as an input and maps it to a password in the set of possible passwords. The reduction function designed for the purpose of this assignment is explained below.

We are given a hash which is 4 bytes long as shown below.

Hbyte_x	Hbyte_0	Hbyte_1	Hbyte_2	Hbyte_3

A password will have 4 bytes and each of which will have to be in one of the following 3 bins which is a requirement from the password limiting rules.



Byte **‘x’** of a password would be filled with a value from one of these bins. The bin from which to fill in one byte, ‘**x**’ of a password is decided using

bin_no = (Hbyte_x mod 3) where ‘**x**’ is the byte number of the hash

Once the bin is chosen, the character to choose from the bin is decided by using

char_no = (Hbyte_x mod bin_size) where ‘**x**’ is the byte number of the hash

Byte **‘x’** in the password is then set to be equal to the **char_no** chosen from the bin **bin_no**

Analysis of the Reduction Function

Although it is good that a reduction function produces collision-free passwords given a hash, it wouldn’t be possible to produce a completely collision free reduction function which brings us to the focus of a reduction function

The 2 main requirements of a reduction function are that

1. It needs to produce the same output every time that the same hash is given as the input to the function. This is necessary because the rainbow table described above doesn't store the entire hash chain, but only the first and last points in the hash chain. Hence, it becomes necessary for the reduction function to produce the same output every time.
    a. The reduction function described here does ensure this because the output that it produces is a function of each byte of the hash and hence provided with the same hash, it will always produce the same output
2. It needs to produce a fair spread of passwords from the entire set of possible passwords. If there are collisions, some of the hash chains in the rainbow table will merge and hence will result in being not able to obtain certain passwords.
    a. Although there are a few collisions, the algorithm used in the function is fairly random and hence ensures that the passwords produced are fairly collision resistant. It was seen that, out of a 100,000 random hashes passed to the reduction function, there were about 500 collisions which is 0.5%. Hence, the reduction function chosen is good in this aspect too.

Choosing n and p

Choosing the appropriate **'n'** and **'p'** is important because it will result in a space-time complexity trade-off. The approach used in choosing these parameters also depends on the reduction function chosen.

Choosing 'n'

The value **'n'** would be based on the space complexity that the attacker would be okay to use up. The approach used in deciding 'n' for this assignment was to calculate the amount of local memory in MB that would be used and hence.

Given the password limiting rules, the size of the entire set of possible passwords is 14,776,336.

In order to create an **n x p** matrix of hash-chains, the number mentioned above was factorized in various ways. The suitable factorization scheme that would be good on time and space complexity was decided as **59582 x 248**.

Hence **'n'** was chosen as 59582. Since the entire matrix isn't stored in this approach, the rainbow table size comes up to **465KB** given that each word is of length 4 bytes and the rainbow table is an n x 2 matrix. From the entire set of 14,776,336 passwords, every 248th password was chosen as the initial set of 59582 passwords.

Choosing 'p'

From the above factorization scheme, **'p'** could have been set to 248 if the reduction function were completely collision resistant. In order to account for the fact that there could be collisions, **'p'** was set to **4 x 248** which is **992.**

Statistics of the rainbow table generated are summarized in the table below

RB Table – Number of rows	59582
RB Table – Number of Hash-chains	992
Size of the Rainbow Table	465KB

The time taken to generate the rainbow table was **220s**

Rainbow Table Look-up

The look-up procedure is a simple procedure as described below

1. Given a hash for which the password needs to be found, a reduction function is applied on it and then a hash function on the reduced output in an alternating fashion.
2. After each time a reduction function is applied, the output is compared with the 2nd column in the rainbow table. If a match is found, it is possible that if the hash-chain that was produced while generating the rainbow table is reproduced using the starting point of the matching row stored in the rainbow table, the hash-chain could contain the hash for which the password is desired. It is also possible that the output of a reduction function could match one of the values in the 2nd column of the rainbow table, but the hash-chain doesn't contain the required hash. In such cases, the hash chain is continued and the same process is repeated until **'p'** times.
3. If a match is found, the password in the hash-chain before the hash that matched is the password for the hash whose password is desired. If no match is found, then the rainbow table is unsuccessful in obtaining the password.

The lookup implemented for this assignment was able to find passwords for a few hashes and not for some, the summary of which is described below.

Hash	Password	Time Taken (s)
0x97e75d32	A1B2	1
0x19fbc7c1	*Not Found*	3
0x8f6bb61b	*Not Found*	3
0x88df723c	lOvE	1
0x655ca818	*Not Found*	3
0x14928501	1a2b	1
0x3974cffc	LoVe	2
0x58712b2b	*Not Found*	3
0x7e1d96fd	love	2
0x8e564270	move	1

Total time taken to complete the procedure was **21s**

Note – One might observe that there is a difference between sum of times taken for each password to be found and the total time taken. This could be attributed to delay caused by prints in the program.

Summary

The rainbow table was able to obtain only a subset of the passwords which could be attributed to collisions in the reduction function, the length of the chain chosen and the choice of the initial set of words out of the entire possible set of words to form the dictionary.

The password length chosen for this project was only 4 bytes long and hence brute-forcing took much lesser time than pre-computing the rainbow table itself. But, given a precomputed rainbow table, finding passwords using a rainbow table is much faster.

If the passwords were to be longer, the positive aspects of a rainbow table would be much more evident. If the reduction function and the length of the hash-chain are fine-tuned to produce a better rainbow table, this method would supersede the brute-force method used to find passwords.