INFO-F405 – Computer Security
Project 1: Rainbow Tables

Degols Gilles, Delhaye Quentin, Streltsov Anton - Group 11

October 28, 2013

1 Presentation of the project

The goal of the project was to create a software that can find one of the possible passwords corresponding to a stolen fingerprint, using the technique of a rainbow table based attack. The stolen fingerprint has a size of 24 bits and the corresponding password is made of 12 bits. Each password is hashed and reduced four times in a row, using four different reduction functions. At the end of the chain, the initial password and the last fingerprint are stored in the table. The hash is based on the DES encryption algorithm in the ECB mode, using the password as the key and a constant message as the plaintext.

According to the size of the password, there are no more than 2^{12} passwords to create, hash multiple times and store – with their fingerprint – in the rainbow table.

The software can be implemented in Java or C++.

2 Presentation of the work

We found out that the project could be split into three parts, which allows us to divide the work fairly among our group of three people:

- the creation of the rainbow table: the dictionary with the passwords and the fingerprint corresponding to each password;
- the hash of a password;
- the creation of reduction functions and the selection of the best ones.

According to the fact that the stolen fingerprint – or a modified fingerprint, depending on the position in the algorithm – is used to find the same fingerprint in the rainbow table, the search has to be the fastest possible. With this consideration, we decided to use the dichotomic search algorithm on the rainbow table, previously sorted at the end of its creation.

The encryption of the password has been realized with the library Crypto++ as we decided to use C++ as the software language considering it is a better language for manipulating bits.

2.1 Creating the Rainbow Table

Each possible password (2^{12} possibilities) is hashed and then reduced with one of the four reduction functions (`blue`, `green`, `yellow`, `red`). This process is iterated four times on the generated passwords using a different function each time. At the end, the last reduced password is hashed one more time to get the fingerprint and then, the initial password and the final fingerprint are stored in the table.

Once all the passwords and their final fingerprint have been stored, the table is sorted and the duplicate fingerprints are removed. This removal is done by shifting all the unique fingerprints to the beginning of the table. The tail of the table containing only duplicate fingerprints is then ignored. It is to be noted that since there are "only" 4096 possible passwords, we decided to store the table on the heap; for much larger tables, it would be better to write the table in a file and save it on the hard drive disk instead.

2.2 Searching Algorithm

The algorithm used for finding the fingerprint in the sorted table is the dichotomic search. The $O(n^2)$ complexity and the linear speed of convergence are satisfying in our case.

2.3 Sorting Algorithm

The sorting algorithm chosen in our project was the insertion sorting as its average and maximal complexity is $O(n^2)$. The stability of this algorithm is an important element in our choice as the rainbow table is sorted before the deletion of the duplicate fingerprints and their password. As we may encounter many duplicate values depending on the reduction functions used, it is useless to sort the table after each password is added. Other algorithms with a lower complexity – $O(n \log n)$ – have been considered, but the increase in efficiency was not worth the increase in implementation complexity. Although the complexity of the sorting algorithm is an important point, we have to keep in mind that it will be applied only once, during the creation of the rainbow table, so the software is allowed to take more time during this step.

2.4 Reduction functions

The creation of reduction functions was made empirically. The aim during their creation was to have a function returning a 12-bits set based on the maximum number of available bits (24 bits of the fingerprint in this case), obtained by a combination of independent small operations on a bit set (e.g. flip all the bits, or sum two of them to return one).

The selection of the best reduction functions has been made as follows:

- a table with the reduced fingerprint of every password with a specific reduction function is made for the complete dictionary;
- the number of collisions is counted and saved;
- the operation is repeated for every other reduction function that was previously created;
- the four reduction functions that create the lowest number of collisions are chosen, in other words, the ones that produce a number of lines close to one fifth of the 4096 possible passwords.

It is to be considered that there are way too many possible functions to be tested and we only had tested a small portion of them. We kept in mind that they needed to be easy to compute in order to be efficient. As a result, we opted for simple operations on bits such as flip, mirror effect, rotate, `XOR`, partial selection, partial addition...

The following table contains some of the functions that we tested and the corresponding size of the resulting table. The notation used is:

- `Fingerprint mirror(Fingerprint f)` → M
- `Fingerprint rotate(Fingerprint f, int round)` → R(round)
- `Fingerprint flipAll(Fingerprint f)` → F
- `Password keepRight(Fingerprint f)` → KR

- `Password keepLeft(Fingerprint f)` → KL
- `Password hopOne(Fingerprint f)` → H1
- `Password hopTwo(Fingerprint f)` → H2
- `Password sumTwo(Fingerprint f)` → S2

Function	KR	KL	H1	H2	S2	M, KL
Lines in the table	118	673	710	758	724	204
Function	F, KL	R(1), KL	M, F, KL	M, R(15), KL	R(15), H1	R(15), H2
Lines in the table	694	1349	198	702	752	727

Finally, we decided to use the following functions for our table:

- Function 1: R(1), KL
- Function 2: R(2), KL
- Function 3: R(3), KL
- Function 4: R(4), KL

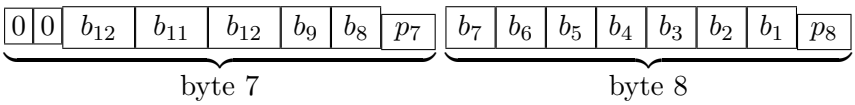
The resulting number of lines in the table was 1298, and among the 4096 possible passwords, 1310 can be found. Although using a combination of the H2 function would have produced a number of lines much closer to one fifth of 4096, the tests showed that using a larger table allowed us to find more passwords.

2.5 DES Encryption

The DES encryption algorithm needs two things to produce its 8-bytes cipher: an 8-bytes message, which in our case is constant `const byte MESSAGE[] = {0x43,0x41,0x43,0x41,0x43,0x41,0x43,0x41};`, and an 8-bytes key, the part we will be working on.

Among those 8 bytes, 8 bits – the last bit of each byte – are parity bits. Their value will not be checked, and as a consequence will not matter.

The key is inflated from the 12-bits password. The 6 first bytes are then simply initialized to zero, while the last two will be organized as follows:



With b_i being the ith bit of the password and p_i the ith parity bit of the key.

The fingerprint will consist of the 24 last bits of the cipher outputted by the DES encryption.

2.6 Finding the Password in the Rainbow Table

The research will consist in four iterations at most. In the begining of each, the current fingerprint will be searched in the table.

If it is found, the corresponding password will go through the different reduction functions (alterned with the encryption algorithm), until the target fingerprint is reached. At this moment, the `found` flag is triggered and the search is over.

However, if the fingerprint is not found in the table, different reduction functions will be applied on it:

- first iteration: 4th function,

- second iteration: 3rd and 4th functions,
- third iteration: 2nd, 3rd and 4th functions,
- fourth iteration: 1rst, 2nd, 3rd and 4th functions.

It is to be noted that our current choice of combination of reduction function does not allow yet a range of fingerprints wide enough to recognize any fingerprint that could be produce by one of the 2^{12} passwords.