Descriptive document

Lauri Kangassalo

May 16, 2013

What?

This is an algorithm for cracking MD5-hashes, written in Java. For this it uses a method called rainbow table. I will also be implementing a hashmap in my project.

So what are rainbow tables?

A rainbow table consists of starting points and endpoints of *chains*. A chain contains pairs of plaintext 'passwords' and their corresponding hashes. The whole chain can be calculated from the starting point, using *reduction functions*, which reduce a hash into plaintext.

Basically when cracking a hash we must run the hash through reduction functions and the hashing algorithm, until we come across a plaintext that is an endpoint in some chain in the table.

Rainbow table -method is an adaptation of Martin Hellman's simpler algorithm. The main difference between these two is that rainbow table uses different reduction functions for each *column* of a chain, thus reducing the number of collisions of chains and false alarms.

More on this subject at: `http://en.wikipedia.org/wiki/Rainbow_table` and `http://lasec.epfl.ch/pub/lasec/doc/Oech03.pdf` "Making a Faster Cryptanalyctic Time-Memory Trade-Off" by Philippe Oechslin

Why rainbow tables and MD5?

Rainbow tables are far more efficient in cracking hashes than, let's say, brute force. MD5 was chosen for this project, because it is a fairly light hashing algorithm to crack in terms of processor cycles and memory usage, and is nevertheless widely used, especially in the web.

The Big-O-notation

Obviously, the goal for the hashmap implementation is O(1) for insert, delete and search operations.

The MD5-cracking part of my project is not so easy to analyze. The time required for creating a rainbow table depends on the number of chains per table and chain length the user chooses. However, the rainbow table creation will take O(m*n) time, where m is the number of chains per table, and n is chain length, since a chain represents a row in a table.

How about actually cracking the hashes? Well, the worst case scenario is that we will find no matching endpoint in the rainbow table, and we have to loop through the whole chain for the hash. In this case the time requirement is roughly O(n*n), where n is again the chain length. Of course, we will have to take in account the amount of time it takes to read the whole rainbow table into memory, for most likely the table is so big that it won't even fit in the RAM, and has to be loaded onto a swap partition or equivalent. As we know, loading data from a long-term storage device (such as a hard drive) takes immensely more time than loading it from RAM.