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?

Rainbow table is an improved version of Martin Hellman's algorithm for the same purpose. Basically it tries to crack hashes by creating *chains* (or rows in the table) that contain seemingly randomly generated possible passwords. From these chains only the starting point and the endpoint are saved to the so called rainbow table.

In the heart of this algorithm is a reducing function, which reduces hashes in to possible passwords. This function is then used to find a corresponding endpoint for the hash being cracked.

The rainbow table -method differs from Hellman's original algorithm so that every reducing function is different depending of the 'column' of the chain. This reduces the number of collisions of chains and thus reduces 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 willl find no matching hash in the rainbow table, and we have to loop through the whole chain for the hash. In this case the time requirement is O(m), where m 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.