Weekly reports

Lauri Kangassalo

May 23, 2013

Week 1

During this week I was mostly studying the basic principles of hash cracking and rainbow tables. I have never done anything like this before, but I think I now handle the basic principles of how rainbow tables work.

It is still unclear to me how to optimize my code for cracking hashes, since I don't know how exactly JVM handles memory allocations and such. For example I'm uncertain wether I should use byte arrays in my code, or convert those arrays into long values. I'm sure that as my code progresses I will find an answer to these problems.

I started coding the very heart of the algorithm, the reduction function. It seems to be working correctly, but as stated before, I am uncertain whether it's efficient enough. I also managed to code a class for generating rainbow tables and saving them into a file.

Next I'm going to proceed to the actual hash cracking part.

Week 2

I started this week by coding the class that cracks the hashes *(MD5Crack)*, and got it almost done. However, I discovered that I can't proceed in this part of my program unless I create my own implementation of hashmap/hashset first. Another option would be to create a new class for byte arrays, and I'm worried that it could affect performance, since byte arrays are in such a great role in my software. Besides, I feel that it would be kind of an overkill to create such a class, since all I would need it for would be an equals and a hashcode method.

Since I found myself in a temporary dead-end, for not being able to actually test the password cracking, I decided to code some secondary parts of the program. Firstly, the passwords are no longer fixed in length, for example the user could now choose to include password of lengths 3 to 9 to the rainbow table. This feature is still somewhat buggy, as the program doesn't read the passwords from the rainbow table file correctly. I will get back to this problem as soon as I can get more pressing problems in the code in order. Secondly, I created a simple user interface for the program, to aid in performance testing. Also, all printing is now handled in classes *UIHelper* and *UI*.

Performance has been tested by hand from the very start of this project. Sadly, I haven't been able to test the hash cracking part, since it doesn't work yet. However, in my experience the table creator *TableCreator* class's performance is sufficient. I have been creating rainbow tables of various sizes, but haven't yet tried to run my program on large enough parameters, since creating a really working rainbow table will take a lot of time and disk space. I will get back to this topic as soon as I can rely on the reading of rainbow table files, so that I don't waste my time creating a table file I can't open in the future.

During this week I also started rereading the materials from Data Structures -course regarding hash tables. I implemented a crude version of a hash table (with insufficient testing), but realized that I cannot make an array large enough / the whole file can't be read to memory at once, since the table files can be tens or even hundreds of gigabytes in size. I have to figure out what I'm going to load to memory and what to keep on the disk, and how to access the data on the disk. I also countered some major performance issues when testing my hash table implementation. This time I used linked lists to deal with collisions, next I'm going to try open addressing - since I won't need to create new object for each value then - and see how it affects performance.