

At the end of the rainbows

You might be puzzled by this title but I'm pretty sure it will reveal its mystery later on. As far as you need to know for now we did not find the pot of gold at the end of the rainbow chains.

The Need 4 Speed

It is always nice to have a little more computing power than you actually need. Normally you would be pleased with your 2 or 3 GHz processor running your machine with ease at about 20% of the CPU time needed. Most of the time it is the RAM or the hard drive slowing down your PC.

But imagine you had a problem and it is (as far as you know) a NP-hard one. So no nice polynomial solution known (to you) that you could apply to solve your problem... But what would be the concrete situation giving you such a problem?

Well it could be the fact that you somehow got hold of a file containing about 10,000 entry sets of usernames and passwords. Imagine further on that these passwords are good for something and hence protected by a hash. That hash could be a SHA-1 hash.

Now this is a good reason for the need of speed on your CPU and also the point where this story begins.

Hopeless? (little technical)

As mentioned above there is no real off-the-shelf-and-ready-in-5-seconds solution for the problem of finding out what the plain text was if you have the SHA-1 hash given.

Some research on the internet however showed an interesting approach to the problem that had been used to crack windows NT LAN man hashes in seconds (after some expensive time of computation that had to be done in advance but only once). The method makes use of an idea that Martin Hellman had about 25 years ago.

It is a method based on creating chains of hashed values that are reproducible. The idea is to hash a value which is the start of the chain, consecutively do that to a certain length and then start with the next chain. The chains are (roughly) built the way that the n-th link of the chain is a hash of the (n-1)-th. The next thing you do after having all these chains computed is starting a new chain based on the hash of the password you want to figure out. Once you find an endpoint of a chain already computed you can (thinking idealistically) reproduce the whole chain and find the password just one step before the last starting point.

From this explanation you can guess that only the start and endpoint of the chains have to be stored (which are still a lot depending on the length of your chains). But this is just a brief and faulty explanation of the basics that are described in this paper about the technique of [RainbowTables](#).

“Hi its me, may I stop by and pick up your PC?”

As you know we are both more or less “computer-guys”. That means that we sometimes have lots of friends (especially when their computers are broken). Well now it was not time for them to call us but for us to call them. The first call was to a friend who we found out was on a holiday and would not come back for a few days. We managed to tell his parents to let us in and take his computer with us to “repair” it.

The second call was to a girl who is actually pretty nice but GIVES A *** ABOUT HER PC. After some

talking she agreed that we could pick up her box too and use it for a while.



CPUs, main boards, and RAM were flying through the room



It took us quite some time to rip all the PCs apart and collect pieces of old hardware to build our brand new computing farm.

This Linux is the Ferrari of the OSs!

As we wanted our systems to be running only the essential services and giving enough processing time the table computation we started out looking for a fast OS that would do. A friend gave us a few CDs containing the fastest Linux on earth. A few hours later we finally were fed up with that distribution not supporting our main boards or network adapters and reached for the good old Win2k disk.



This Linux sucks we use Windows 2000

The tables themselves

The decision of the OS was taken from us by the incompatibility of our hardware. But we of course had to get some software (apart from the OS) that would do the cracking and computation of the rainbow tables. Our plan was to fill up the freshly bought 80GB hard drive to the rim. We planned to distribute the creation of the tables over the 6 computers according to the speed of their CPUs. After all the tables had been created we would just put them on one machine and return all the others to their owners.

The software we used for creating the tables is called [RainbowCrack](#) by [Shuanglei](#).



As the PCs were fed on software we enjoyed some doughnuts.



Does your room look different?

Preparing for the long run

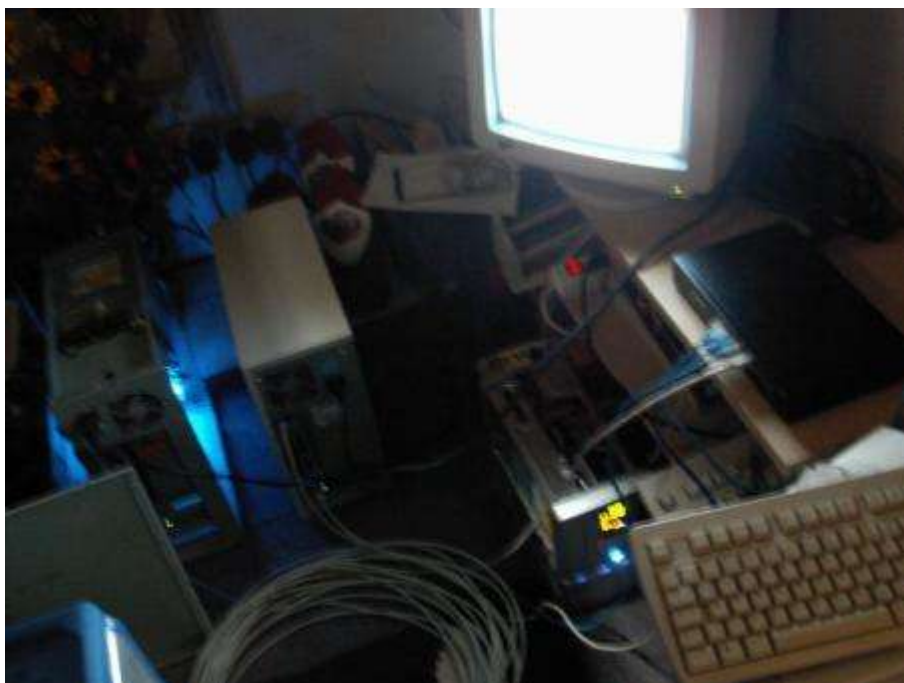
After all the PCs were set up right and ready to run we moved them into another room where we could open the window to let the heat out. Another positive effect of a separate room was that we could close the door to reduce the cooling noise so that we could talk without having to scream in the other room.



Our little farm set up and ready to go.

The only thing we could run in trouble with was that Tim's parents discovered the use of up to 8 PCs at one time in their house. They would probably not like the side effects on their monthly electricity bill.

The other thing we had to think about was obviously the heat in the room and the opened window which was another hazard in case of rain. It would have been a tricky situation telling our friends that the next time they would see their PCs would be the afterlife.



Have a good night honeys.

There is always an other way ...

After some days we noticed that it would probably take years or decades to complete the scheduled

set of tables. At that cost we had to rethink the project.

The advantages of RainbowTables was obvious. There is no faster way known to us finding the plain text for a given hash. When we would be finished decrypting all our passwords we could go and offer password decryption for rent over the internet (using our tables). There are probably a few people looking for that kind of service.

But the disadvantages were even more clear. The computation of the tables would be a few times the computation of a full dictionary (have you tried that for SHA-1 ?).

Now we got to the point where we did some thinking and came up with the idea to parse the file for all password hashes and compare them (some people actually used the same passwords which reduced our attack).

The first (stupid) approach would be to crack the passwords one by one and each requiring an exponential amount of computation. The method we finally implemented was about as stupid as the first but reduced the linear factor of the cost from 10,000 to ~13. The idea was using the standard brute force approach of computing a password, hashing it and comparing it to the real hash. But this time we would use the sorted list of the password hashes and do a binary search on it looking for the freshly computed hash. As you can see this already makes us about 750 times faster.

But this still sucks since for a reduced char set containing only 64 characters (e.g. capitals, lowers and digits) and a max password length of only 16 characters we have to do about $2^{(6*16)}$ hashes and our (luckily rather fast) lookups.

Don't use short passwords

The thing that helped us out a lot were short passwords that we discovered pretty early during our brute force search. So don't use short passwords and you always go better by putting in some non-standard characters (that would change the 6 in the exponent to an 8 and effectively being a factor of 5,000,000,000 in the time needed for a similar search).

In a few days we were able to crack nearly 4,000 passwords on a single machine. After that we switched of the other boxes and returned them to their owners who probably now even more consider us as freaks.