Password Cracking and Time-Memory Trade Off

Author: Jason R. Davis
Site: TMTO[dot]ORG
Date: March 13th, 2005

Table of Contents:

Intention

This paper has a single intention: Define “TMTO” and its perfections/flaws, applications, and significance in the future.

Introduction

Every time I go on line, I usually am up to no good. My intentions are often never hostile, but I do take part in the shady business of password cracking. Meaning, I actively use unorthodox methodology, that I know for a fact the FBI frowns down upon, to obtain hashes. Once obtained I usually spend a few hours cracking these hashes via good old fashion brute forcing. Now, brute forcing is the most reliable method of password cracking in existence today. Theoretically, there is no password you cannot break, but the catch to all this, is time. You can spend 5 seconds or 10,000 years brute forcing a password depending on its weaknesses or strengths in this reference. The driving force behind brute forcing is the ability to crack short passwords in minutes or hours instead of years. The other wonderful thing about brute forcing is the "dictionary" aspect. If a password is say, 10 characters long, it will take quite a while - I'm not going to compute the time - to crack, except if it's a dictionary word. I can simply run a word list through my brute forcing utility and for lack of a better word, "shazam", I break the password.

180 Degrees

Now after 5-10 years of refining, brute forcing has proved its success. Meanwhile the world of encryption has grown aware of its weaknesses. Hashing algorithms have been revamped to overcome certain problems, such as collisions, to ensure their existence and usability. A result of this effort made by the world of encryption has decreased the usefulness of brute forcing. To make it worse, even the corporations have caught on, enforcing password policies and expirations. So now, by the time I crack a password, it’s most likely been changed already. This makes brute forcing look extremely unappetizing, to say the least. Now I ask myself, why spend 3-4 hours of cumulative effort to break in, if the password I get is only good for 2 weeks? Hence, the 180 degrees, as the world of brute forcing has begun its inevitable turn from success to worthless existence.

Preconception of TMTO

A few years back, a talented individual with great foresight, thought about the scenario we are in today. A time when processing power would be excessive and abundant, not to mention storage space even more so. Then with even more foresight, predicts the strengths of algorithms available in this day and age. After much thought, this individual generated the theory of "Time-Memory Trade Off". Which when developed was not feasible, and was simply that, a theory. But the principal is the core of what will drive the password cracking of the entire world into the future as brute forcing becomes a favorite past time.

The actual principal can be summed up as follows (not verbatim): Instead of generating all possibilities every time; generate and store them one time. Once generated and stored, recalling a specific value takes seconds regardless of the original password's length and/or complexity.

<u>This is the utopia of password cracking!</u>

Reverse Lookup Database

I'm not going to explain the difference between encryption and hashing, as this paper only applies to algorithms that return an exact value each time. The diversity of encryption (elliptic, symmetric, asymmetric, etc...) has resulted in a gray area between the two. Simply keep this in mind: If I process the letter "a" through an algorithm and get the same result every time, then this concept applies; regardless of whether it is an encryption or hashing algorithm.

I like to call the end result - a finished product - of "Time-Memory Trade Off", where all possibilities have been generated and stored a "Reverse Lookup" database rather than a "TMTO" database. Essentially, that is what it becomes. Once all values have been computed and stored, you supply a hashed or encrypted version of some text and the table returns the original text. Liken this concept to DNS, you supply a FQDN and it returns the associated IP address.

Still Out of Reach

A completed Reverse Lookup database is still out of reach, even today. In my own research I've run into one simple problem: storage shortage. I have been focusing on MD5 as my algorithm of choice, so for my sake I'll use MD5 as the algorithm of example from now on.

I have processed all possibilities of a-z (lowercase) to the length of 7 for the MD5 algorithm. That results in 26^7 possibilities. Now, a-z to the length of 7 is not that impressive. But it in fact is, considering the database is just over 170GB before indexing and optimizations. This database size also includes a special technique of removing the filename from the file, originally brought to light by a colleague, NullAck. So without that technique you can add 26^7*2 bytes to 170GB, making it even more intimidating and large. Now on such a small scale, 170GB is not so small, as I try to move up to 8 length or 26^8. A rough database size can be calculated as follows: if 26^7 = ~170GB then 26^8 or 26*170GB = ~4.5TB. For each length I add to the database you can take the current size and multiply it by the number of characters utilized (a-z or 26 in my case). This exponential database size increase puts a damper on the physical feasibility of TMTO. This has resulted in many headaches as I develop and scale my database to longer lengths or add characters to the running character set.

Rainbow Tables: Salvation or Poor Substitute?

Under extreme scrutiny of storage limitations, the concept of rainbow tables was developed. Instead of computing all values and storing them, Rainbow Tables compute statistically common combinations based on randomly generated returns of functions. This has proved very effective in reducing the storage requirements of a TMTO database. The statistics are sound, but the efficiency of a Rainbow Table isn't even close to that of a true TMTO database. I believe the math behind Rainbow Tables is the stepping stone that needs to be taken in order to reach the end goal of storing all possibilities.

I am not bashing Rainbow Tables; they are effective in most cases. In my research I've cracked 94.4% of a predetermined set of MD5 hashes I ran through Rainbow Tables generated using RainbowCrack. Which is 94.4% efficient, and for password cracking better than average, if not above average in all cases. Now, I've noticed that certain passwords are cracked in different time-frames also; some faster than others. Also, it matters which character set you utilize when generating the tables.

MD5 Reverse Lookup

I've been developing an MD5 Reverse Lookup database as stated above. Any password hashed with MD5 utilizing the characters a-z, 1-7 length is cracked 100% of the time. Not to mention in average of 30 milliseconds.

In a comparison I built the same database using the same character set and length with RainbowCrack. I found 94.4% of the 1000 hashes I tried to crack were indeed broken. That's a mere 5.6% disadvantage to my database. This gives praise to Rainbow Tables for the time being. Keyword being "time", as character sets grow, RainbowCrack falls greatly short of perfect, but proves its effectiveness better than brute forcing.

To make the MD5 Reverse Lookup more effective I developed a method of inserting word lists into the database. This results in something extremely similar, if not identical, to a dictionary brute force. Only needing to run the word list through that database once, meaning a 1,000,000 entry word list once inserted will never have to be inserted again nor will the process have to be run again. Those values are stored along with all the others and can be recalled in the same time frame. This makes the database more efficient.

Let’s say I have a MD5 Reverse Lookup database, character set a-z, with 1-8 length. This will crack a lot of passwords, but anything with a single capital letter or number cannot be broken. This is when I insert my word list, which can contain numbers, caps, and special characters. This boost performance when cracking complex passwords, that can only be broken within a reasonable time frame through dictionary based brute forcing. Of course, this does not remedy the problem. I still can't break passwords 100% of the time, nor do I have the storage to make a database that can, YET.

Conclusion

With the concept of TMTO being partially implemented on a small scale now, we must look at what it can become. 10 years ago, individuals saw the future of brute forcing and helped it grow to what it is now. We must do the same with TMTO, because with its huge success on such as small scale, imagine the final result of many peoples, including my own effort to prove that TMTO can work and should be the defacto standard when it comes to password cracking of the future.

Jason R. Davis
TMTO[dot]ORG