Informative Overview of Rainbow Hashes

Author: Victor Hornedo Martinez
Advisor: Jeffry Duffany
Polytechnic University of Puerto Rico

Abstract

This article gives an overview for rainbow tables and the results of testing rainbow tables according to the length of the chosen chain. The article presents a password cracking process that contains its own algorithms for reduction functions, changes the length of the chain and generates tables accordingly. These are measured to see the effectivity of the password search in detail. Within the executed tests it was noticed that there is a dependence of rainbow tables size in relation to the password length, the affection of the hash search by the size of the chosen chain and their links to collisions. After completing the testing with different passwords and tables the cause of this arises from the principle of using the reduction function. These results objectively describe the pros and cons of using rainbow tables and finally the article ends talking about what are some effective use cases for this password cracking method.

Introduction

Theoretically all passwords are “crackable” Breaking any encryption system can be done with unlimited time and unlimited computing power, both of which do not exist. Anything less than that unlimited power and time will require chance and good investigative skills. Several methods to break encryption include dictionary attacks, brute-force attacks, and rainbow tables. Knowing that recovering the password requires time, computing power and most of all luck for a dictionary or brute-force attack to find a valid password. Strong passwords increase the likelihood, if not guarantee that it would be harder for attackers to break the encryption of it.

Several styles to break encryption include wordbook attacks, brute-force attacks, and rainbow tables. A dictionary attack tries variations of words in the wordbooks. The speed at which depends upon the computing power of the system being used. Millions of words can be tried each second using a suitable computer system for password breaking. However, dictionary attacks should not be overlooked because of not knowing the password. Although using the highest-grade encryption is easy, quick and effective, a flaw remains with the user in choosing a strong password. The password can make a seemingly impossible to crack file easily done in minutes.

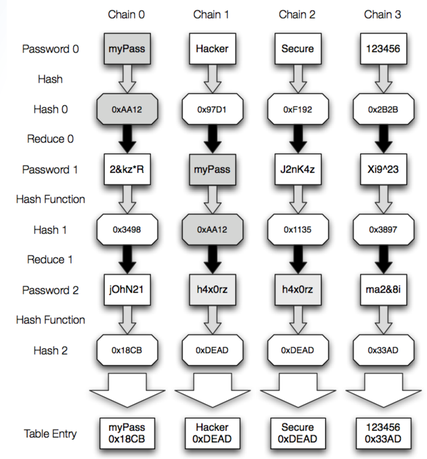
Problem

Rainbow tables are fast and effective at cracking passwords because each password is hashed the same way. Since most people use common passwords or reuse passwords, it makes cracking easy. You solve this issue with password salting. A salt randomizes each hash by adding random data that is unique to each user to their password hash, so even the same password has a unique hash. If someone tried to compare hashes in a rainbow table to those in a database, none of the hashes would match, even if the passwords were the same.

Methodology

The storage techniques to be tested are: MD5 Hashing and SHA-256 Hashing. Although not proven to be secure, a commonly used superset of password hashing is hash chaining. Following the National Institute of Standards and Technology (NIST) guidelines on password strength, “both a weak and robust password will be passed through the stated techniques” [3]. Then, reversal of each of the resulting strings will be attempted using online and offline rainbow tables. The data recorded will be the time taken to reverse the hash or whether the attack was successful.

Figure 1: **Rainbow Hash Flowchart**



By generating hashes of the large collection of accessible strings, a rainbow table attack removes the requirement for this. In the below example we will create two hashes using the words P@55w0rD and thisismypassword (Table 2 & Table 3). Next, we will create a file containing our hashes so we can put them into a cracking program.

Table 2: **MD5 Hashes**

P@55w0rD	B884DBCC2FEDE312066A9B7609A2E3C9
thisismypassword	31435008693CE6976F45DEDC5532E2C1

Table 3: **SHA256 Hashes**

P@55w0rD	7F0897E8D62D3E3641EAFC270D311CBF7 77E67B9DF608571C93056D5AACF3189
thisismypassword	1DA9133AB9DBD11D2937EC8D312E1E25 69857059E73CC72DF92E670928983AB5

In the above example, while the password "thisismypassword" is not unique it serves the purpose of this experiment which is testing a raw simple and complex password using two different hashing algorithms and compare the results.

Results and Discussion

The results for both SHA256 and MD5 hashes using the online method of cracking we can start to see its drawbacks. While not needing the user to download tables to run, the process comes with the drawback of slower times. The reason for this is “because it searches for all available tables it can find while proceeding with the validation of the hash existence” [4]. If it does not exist, it will make a new entry of the hash in its database. After this step it keeps going threw the process until it can find the result. The result of the two passwords is displayed in the table below.

Table 4: MD5 Password Crack Time Using Crackstation

P@55w0rD	Not found
thisismypassword	0.7 seconds

Table 5: **SHA256 Password Crack Time Crackstation**

P@55w0rD	Not found
thisismypassword	0.8 seconds

The results for both SHA256 and MD5 hashes using the online method of cracking we can start to see its drawbacks. The reason for this is “because it searches for all available tables it can find while proceeding with the validation of the hash existence” [4]. If it does not exist, it will make a new entry of the hash in its database. After this step it keeps going threw the process until it can find the result. The result of the two passwords is displayed in the table below. As we can see the hash for “thisismypassword” both in the MD5 and SHA256 variants were the easy passwords variable used for the experiment.

Table 6: MD5 Password Crack Time Using Ophcrack

P@55w0rD	414 seconds
thisismypassword	34 seconds

Table 7: **SHA256 Password Crack Time Using Ophcrack**

P@55w0rD	419 seconds
thisismypassword	39 seconds

As we can see from the Table 6 & Table 7, it takes considerably longer to crack these hashes. It takes a bit more time than the online method because we don’t have the luxury of just comparing the testing hash to public hashes available. The smallest rainbow table available is the basic alphanumeric one, and even it is 388 megabytes. That's the default table you get with the Ophcrack bootable ISO. Even that small-ish table is remarkably effective. It wasn’t expected that this rainbow table would not work on the passwords with non-alphanumeric characters (%&^$#@!*) because the table doesn't contain those characters The table that found the result for both variants had size of 207 gigabiytes in total.

Conclusions and Future Work

Unlike other techniques, huge storage is needed for Rainbow Table Attacks and sadly the decreasing price per Mbyte for storage solutions nowadays doesn’t help our safety. To avoid being a victim of Rainbow Table Attack, it is strongly advised to perform frequent password changes. Password security it’s all about just following recommended best practices and trying to at least keep up with cybercriminals. However, as the popularity of less secure hashing algorithms fell, and as password salting became a more common practice, rainbow tables have fallen out of common use.

As a plus in future testing we can try salting it such as "thisismypassword" + “s41Ty” to make it more unique. It is highly unlikely to be a password that would show up in a rainbow table already. This means that hackers would need to do all the costly the computation themselves. Adding a salt to the hashing process is a great way to force the hash to be more unique, complex, and increase its security without giving extra requirements to a user. The salt is usually stored with the password string in the database. Adding salt can help to mitigate password attacks, like rainbow tables, because it will be encrypting the user's password with a random string that wouldn't be naturally included in any rainbow table. You can also add pepper to extra secure your data from this sort of attack. The difference between salt and pepper is that “pepper is a site-wide static value that is kept a secret and not stored in the database”[3].

References

[1] Shavers, B., & Bair, J. “Cryptography and Encryption. In Hiding Behind the Keyboard”, (2016) (pp. 133–151). Elsevier.

[2] Information Security Stack Exchange. [Online]. https://security.stackexchange.com/questions/92865.

[3] Rainbow table attacks and cryptanalytic defenses. (2022, February 26). [Online]. https://www.esecurityplanet.com/threats/rainbow-table-attack/

[4] CrackStation. (2019, June 5). Secure Salted Password Hashing - How to do it Properly. [Online]. https://crackstation.net/hashing-security.htm

[5] International Journal on Advances in Software, vol. 4 no 3 & 4, year 2011. IARIA Conferences. [Online]. http://www.iariajournals.org/software/

Acknowledgements

I would like to acknowledge Dr. Jeffrey Duffany for initially guiding me through this work. I would also like to thank other members in the Computer Science & Engineering department which have given me ideas and encouragement to continue with this research.