Mildred Suriel

Final

Information Security

Professor Termini

August 2, 2020

Data Encryption, Hashing, and Salting

Data security has been a largely discussed topic pretty much since language was first created as a form of transferring data. Ensuring that data can be delivered to the intended recipient without risking the data being compromised along the way has been a largely addressed issue over the years. The solution to this problem is the concept known as data encryption, which ensures that only trusted agents can understand the contents of the secured data. With the evolution of computers, many encryption algorithms are now easier to break, and so additional techniques to make the algorithms harder to compromise, such as hashing and salting, have also been developed.

During the Roman era while Julius Caesar was in power, he developed what is known today as the Caesar Cipher. His cipher represents one of the earliest forms of data encryption, which created a system that intended to make interpreting the message possible only by the intended recipient. The concept behind the Caesar Cipher is to shift all letters of a message by the same amount, which will result in a message that is not able to be read without reversing the cipher. For example, the traditional Caesar Cipher had a 3-letter shift, which means that all ‘A’s will be written as ‘D’s, and so on (with the end of the alphabet wrapping back to the beginning when incrementing). This system allowed for him to transfer written letters with other people, without risk of the message being intercepted along the way.

Basic encryption methods like the Caesar Cipher relied on the fact that, at the time, nobody understood the principles behind the cipher to be able to reverse it. In today’s age, data encryption is a largely discussed topic, and so it is much more likely that someone could figure out this system today. With the evolution of computers, the ability to quickly process large amounts of data has also caused our ability to reverse a basic encryption method to occur in a matter of seconds with proper programming, causing the need for more advanced encryption methods to be developed.

There have been many advanced encryption algorithms developed over the years that are still used within computer systems today. Some of the most used ones include RSA, AES, and 3DES. These algorithms rely on the use of prime numbers or multiple rounds of encryption, which make breaking the encryption difficult or impossible to brute force with the processing power in a computer today, given the amount of time that would be needed to try all potential solutions. These encryption algorithms can also be viewed as communication encryptions, as they ensure a secure method of transferring data from one user to another. The next issue is the fact that these encryption algorithms still expect some form of passwords to be stored locally for decryption. So, aside from just secure communication channels, we also need secure ways to encrypt user passwords and encryption algorithm passwords so that the secure communication channels cannot be compromised.

Computer passwords are the most popular form of data that requires advanced methods of encryption and validating. A computer password is used so that a system can validate the end user, and so there is always some form of password database that is maintained to ensure the password matches the system’s expectations. The issue here though is that any password database should not contain passwords in plain text such that they risk user accounts being compromised if the database is compromised. The common solution to this issue is the use of data hashing, which ensures that the passwords are never saved in plain text.

Data hashing is the concept of taking a string of data of any length and converting it into an output of any pre-determined length. The hash can then be used to verify the integrity of input data compared to the previous result. Hashes can be used to verify file downloads or other data transfers are not corrupted and are typically used for password database protection. Two of the most popular hashing algorithms used today are MD5 and SHA. These methods of hashing rely on taking a string of data, appending data to make it a specific length (if it is not already an expected length), and then performing logical functions (ORing, ANDing, and XORing) on the data to generate a unique hash for that set of data. In data hashes, it is possible that two different inputs result in the same output hash value (depending on the complexity of the hashing protocol), which is known as a collision. The MD5 algorithm is designed such that a collision should only occur is approximately 1\*10^-45, ensuring that the likelihood of a collision is unrealistically low to worry about.

Without context, the process of hashing might not immediately explain how it can be used for better protecting managed password databases. However, the benefit of hashing is that a password database can take the original password, generate a unique hash for the password, and store the password hash within its database. When the user then provides its password, the database can perform the hash function on the input password and verify that it matches the stored hash, ensuring that the password is verified without ever having to store the password in plain text. As a security specialist, one should never configure or program a database such that confidential account information is ever stored in plain text, and hashing should always be considered instead.

Due to the nature of hashes, they are meant to be a one-way function where the input cannot be determined from the output hash, and so the original data can never be derived from the hash. This method ensures that the original data is never compromised when performing validation. However, hashes can be exploited by using what is known as a hash table or rainbow table. These tables test a large amount of input combinations and store what their associated hash values are. Because of hash tables, a hacker can create a list of common passwords and search for any matching password hashes in a compromised database, gaining knowledge of the user’s password. Due to the large growth in hash table size as the number of tested passwords increases and the significantly large number of password combinations, hash tables are typically limited to testing only the most common shorter passwords. Due to the possibility of hash tables, its even more important that longer passwords are used.

Another method of protecting passwords and hashes is to perform salting. Salting is a concept where a password is prepended or appended with a psuedo-randomly generated string to increase the length of the password before being hashed. This salting ensures that if someone were to use a hash table, the hash generated due to the salting being added to the password does not match any hash in the has table. Salting for each password should be a unique string that is pseudo-randomly generated such that when cryptographically analyzed, it is difficult or impossible to find a pattern that can be exploited. This ensures that the salting cannot be predicted, making hash table attacks ineffective. Also, the benefit of generating a unique salt for each password ensures that two users with the same password do not have the same hash, making them completely unique. If a user updates their password to a previously used password, this salting will also ensure that the previous hash for the same password is not repeated.

This paper briefly discussed the importance of data security and how encryption, hashing, and salting can all contribute to making data more secured. Whenever a confidential password database is being managed, the user should always store password hashes instead of plain text hashes. Hashing can be used for verifying the integrity of any data and is most commonly used for password database protection. Salting is a strong tool that can be used with hashing to generate unique hashes for all passwords and ensure passwords cannot be targeted by hash tables.

Sources:

1. <https://www.comparitech.com/blog/information-security/encryption-types-explained/>
2. <https://www.comparitech.com/blog/information-security/encryption-hashing-salting/>
3. <https://medium.com/@weblab_tech/encrypted-client-server-communication-protection-of-privacy-and-integrity-with-aes-and-rsa-in-c7b180fe614e#:~:text=AES%20is%20a%20symmetric%20encryption%20algorithm.&text=AES%20key%20length%20is%20128,only%20with%20a%20private%20key.>
4. <https://www.avira.com/en/blog/md5-the-broken-algorithm#:~:text=(32%20bytes).-,The%20probability%20of%20just%20two%20hashes%20accidentally%20colliding%20is%20approximately,for%20MD5%20is%20terribly%20low.>
5. <https://auth0.com/blog/adding-salt-to-hashing-a-better-way-to-store-passwords/>