International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

Password Hashing using Salt Technique

Amol Bhalerao

Keraleeya Samajam (Regd.) Dombivli's Model College, Thakurli (East), Maharashtra, India

ABSTRACT—

We can see that there are so many issues or issues related to security in this world. We send our data from one person to another person. At this point, there are more ways to read our data on the internet. To solve this problem, in this document we use some algorithms with the help of algorithms that have no way to hack or steal data. There are several methods to protect data like Hash, Salt, SHA256, SHA512.

Keywords— Salt technique, cryptography, hash function, SHA.

I. INTRODUCTION

"Web Password Hashing" is a security technique for our web data. With the help of various techniques. There are some algorithms used to secure passwords or to secure data like cryptography, hashing. There are also some buggy algorithms in the market and we should not use these algorithms like MD5 (Merkl Damaged), SHA1 (Secure Hash Algorithm). There are more ways to hack data in this algorithm.To avoid this problem, we use different methods to protect the data, like Hash, Salt, SHA256, SHA512.

II. Hashing

Hashing algorithms are one-way functions. They test a certain amount of data in a fixed-length "fingerprint" that cannot be undone. They also have the property that if the input changes even slightly, the resulting hash will be completely different (see example below). This is great for protecting passwords as we want to store passwords in a way that protects them even if the password file is compromised, but at the same time we need to be able to verify that a user's password is correct.

hash(" hbllo") =	58756879c05c68dfac9866712fad6 a93f8146f337a69afe7dd238f3364946366
hash(" waltz") =	c0e81794384491161f1777c232bc 6bd9ec38f616560b120fda8e90f383853542

For example,

The general example for account registration and verification in a hash-based account system is as follows:

1. The user creates an account.
2. Their password is hashed and stored in the database. At no point is the plain-text (unencrypted) password always written to the hard drive.
3. When the user tries to login, the hash of the password they entered is checked against the hash of their real password (password retrieved from the database).
4. If the hashes match, the user is access. If not, there message is display invalid login.
5. Steps 3 and 4 repeat for everytime when attacker login to site it will display invalid login

Only cryptographic hash functions can be used to implement password hashing. Hash functions like SHA256,SHA512, RipeMD WHIRLPOOL are cryptographic hash functions

III. THE Algorithms

MD5

Firstly designed as a cryptographic hashing algorithm, first published in 1992, MD5 has been shown to have general faults, which make it relatively easy to break. Its 128-bit hash values, which are quite easy to produce, are more commonly used for file verification to make sure that a downloaded file has not been tampered with. It should not be used to secure passwords[3].

SHA-1

Secure Hash Algorithm 1 (SHA-1) is a cryptographic hash algorithm first developed by the US National Security Agency in 1993 and released in 1995. It generates a 160-bit hash value, usually represented as a 40-digit hexadecimal number. From 2005 onwards, SHA-1 was no longer considered secure, since the exponentially increasing division of power and sophisticated methods made it possible to carry out a so-called hash attack and determine the password or the source code without spending millions of resources and time[3].

SHA-2

The successor to SHA-1, Secure Hash Algorithm 2 (SHA-2) is a family of hash functions that produce longer hashes of 224, 256, 384, or 512 bits, written as SHA-224, SHA-256 , SHA-384 or SHA-512. It was first released in 2001, again developed by the NSA, and an effective attack against it has yet to be proven. This means that SHA-2 is generally recommended for secure hashing[3].

SHA-3

It is not a replacement for SHA-2, it was developed by Guido Bertoni, Joan Daemen, Michael Peters and Gilles Van Assche from STMicroelectronics and Radboud University in Nijmegen, Netherlands. It was standardized in 2015.

Bcrypt

As computational power has increased the number of bruteforce guesses a hacker can make for an effective hashing algorithm has increased exponentially.

Bcrypt, which is based on Blowfish encryption and includes a salt, is designed to protect against brute force attacks by intentionally running slower. It features what is called a work factor, which effectively sends your password through a definable number of rounds of renewal before it is encrypted.

By increasing the work factor it takes longer to brute-force the password and match the hash. The theory is that the site owner sets a suitably high-enough work factor to reduce the number of guesses today's computers can make at the password and extend the time from days or weeks to months or years, making it prohibitively time consuming and expensive[3].

PBKDF2

Password-Based Key Derivation Function 2 (PBKDF2), developed by RSA Laboratories, is another key expansion algorithm that makes hashes harder to enforce. It's considered slightly easier to brute force than Bcrypt on a given value since it takes less computer memory to run the algorithm.[3]

Scrypt

Scrypt, like Bcrypt and PBKDF2, is a key-extending algorithm that makes it difficult to brute-force a hash. However, unlike PBKDF2, scrypt is designed to use a large amount of computer memory or force a lot more calculations during execution.

The cost is low for valid users who only need to encrypt a password to check if it matches a stored value. But for someone trying hundreds of thousands of passwords, the cost or time is prohibitive.

There are also some algorithms that are failing in the market and we don't use them, like MD5 (Merkl Corrupted), SHA1 (Secure Hash Algorithm). If we use this algorithm, there is a possibility that we will take over our personal data available on the Internet or in the browser.

To avoid this problem we use here salt technique. Salt technique is use for secure data. Some people are use so easy passwords like date of birth, mobile number or may be personal information. It is easy to attacker broken this password so avoid this problem we use salt technique[3].

Example :-

When we create an account in Facebook, when we enter the ID and password salt technique, also add your own password to make it harder for the attacker who stole the passwords, so the Attackers cannot access the password and our data is protected by the Salt technique. Using the Salt technique, we can prevent SQL injection attacks, dictionary attacks, and birthday attacks.

IV. Methods of creating salt steps :-

1. Get Password
2. Generate the salt using a trusted function or method.
3. Append the salt to the original password.
4. Generate the salt hash password using appropriate hash function.
5. Store the salt and the salt hash in the database.

Example :- How the password is getting stored

The table is having 2 username and password combinations.

Username	Password
User1	Password123
User2	Password456

The value of the salt is 64 bit that is 8 bytes. The hashed value will be the has of the salt value appended to the plaintext password. Both the hashed value and salt value are getting stored.

Username	Saltvalue	String to be hashed	Hashed value = SHA256(password+salt value)
User1	E1F1532E5765	Password123+salt value	72AE25495A7891C40622D4F493
User2	84C03D0340DF	Password456+ salt value	B4B660AB670867E9C732F7DE8

V. THE WRONG WAY: SHORT SALT AND SALT REUSE

The most common salt implementation errors are reusing identical salt in multiple hashes or employing a salt that's too short.

1. Salt Reuse

A common mistake is to use the same salt in each hash. The salt is permanently coded in the program or is randomly generated once. This is unsuccessful because if two users have the same password, they still have the same hash. An attacker can still use a reverse lookup table attack to perform a dictionary attack on each hash at once. They just have to salt each guessed password before encrypting it.

If the salt is hard-coded in a popular product, lookup tables and rainbow tables can be created for that salt to make it easier to crack the hashes generated by the product. Every time a user creates an account or changes their password, a new random salt should be generated.[2]

2. Short Salt

If the salt is too short, an attacker could create a lookup table for each possible salt. For example, if the salt is only three ASCII characters, there are only 95 x 95 x 95 = 857,375 possible salts. That might sound like a lot, but if each lookup table only has 1MB of most common passwords, that's only 837GB total, which isn't much considering 1,000GB hard drives can be had for less than $1,000 today.

For the same reason, the username should not be used as an extension. Usernames can be unique to a single service, but they are predictable and are often reused for accounts in other services. An attacker can create lookup tables for common usernames and use them to crack salted hashes by username. To prevent an attacker from creating a lookup table for every possible salt, the salt must be long. A good rule of thumb is to use a salt that is the same size as the output of the hash function.

For example, the output of SHA256 is 256 bits (32 bytes), so the salt must be at least 32 random bytes[2].

VI. CONCLUSION

Salt gives security. Salt technique prevents attackers or is useful to avoid dictionary attacks or birthday attacks. Salt's hashed password prevents the attacker by: The attacker must now recalculate their entire dictionary for each and every account they are trying to crack. But salt can only help against ready-made dictionaries, if an intruder gains access to our system and executes a brute force attack, then Salt does not offer the necessary security.

References

[1]. A cryptography application using salt hash technique, by Pritesh .N. Patel, Jigishak Patel.

[2]. Salted Password Hashing –https://www.codeproject.com/Articles/704865/salted-password-Hashing-Doing-it-Right

[3]. Passwords and hacking: the jargon of hashing, salting and SHA-2 explained https://www.theguardian.com/technology/2016/dec/15/passwords-hacking-hashing-salting-sha-2

[4]. Stronger Password Authentication using Browser Extensions by Blake Ross, Collin Jackson, Nick miyake, Dan Boneh, Jonh C Mitchell

[5]. Search Security http://searchsecurity.techtarget.com/definition/salt, Retrived 15th Oct, 2011