International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 04 Issue: 11 | Nov -2017 www.irjet.net p-ISSN: 2395-0072

Overview Of Web Password Hashing Using Salt Technique

Diksha.S.Borde[1], Poonam.A.Hebare[2], Priyanka.D.Dhanedhar[3]

[1,2,3]*Dept. of Master of Computer Application, MGM's Jnec college, Maharashtra, India*

---***---

Abstract *We can see in this world there are so many problem or issues are occurred about security. We send our data one person to another person that time there are more possibilities to lick our data on internet. For solve this problem we use in this paper some algorithms with the help of algorithm there are no chance to hack data or stole data. There are various method for secure data like hash, salt, SHA256, SHA512.*

Key Words: **Salt technique, cryptography, hash function, SHA, web security.**

1.INTRODUCTION

"Web Password Hashing" it is a process of secure our web data. With the help of various techniques. There are some algorithm are used to secure password or secure data like cryptography , hashing. There are also some algorithms which is fail in market and we should not use this algorithms like MD5(Merkl Damaged),SHA1(Secure Hash Algorithm).In this algorithm there are more possibilities hack data. To avoid this problem we use various method for secure data like hash, salt, SHA256, SHA512.

1.1 What is password hashing? 1

Hash algorithms are one way functions. They try some amount of data into a fixed-length "fingerprint" that cannot be reversed. They also have the property that if the input changes by even a little bit, the resulting hash is completely different (see the example below). This is great for protecting passwords, because we want to store passwords in a form that protects them even if the password file itself is compromised, but at the same time, we need to be able to verify that a user's password is correct[2].

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** may be used to implement password hashing. Hash functions like SHA256, SHA512, RipeMD, and WHIRLPOOL are cryptographic hash functions.

If storing password in a plain text or is compromised through easy encryption method then there are possibilities of decrypting of password[2].

2. The Algorithms 2

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 cryptographic hashing algorithm firstly design by the US National Security Agency in 1993 and published in 1995.It generates 160-bit hash value that is typically rendered as a 40-digit hexadecimal number. As of 2005, SHA-1 was considered as no longer secure as the exponential increase in dividing power and sophisticated methods meant that it was possible to perform a so-called attack on the hash and produce the source password or text without spending millions on computing resource 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 hash values with 224, 256, 384 or 512 bits, written as SHA-224, SHA-256, SHA-384 or SHA-512.

It was first published in 2001, designed by again by the NSA, and an effective attack has yet to be demonstrated against it. That means SHA-2 is generally recommended for secure hashing[3].

SHA-3, while not a replacement for SHA-2, was developed not by the NSA but by Guido Bertoni, Joan Daemen, Michaël Peeters and Gilles Van Assche from STMicroelectronics and Radboud University in Nijmegen, Netherlands. It was standardised in 2015[3].

Bcrypt

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

Bcrypt, which is based on the Blowfish cipher and includes a salt, is designed to protect against brute-force attacks by intentionally being slower to operate. It has a so-called work factor that effectively puts your password through a definable number of rounds of extension before being hashed.

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 algorithm for key extension that makes hashes more difficult to brute force. It is considered slightly easier to brute force than Bcrypt at a certain value because it requires less computer memory to run the algorithm[3].

Scrypt

Scrypt like Bcrypt and PBKDF2 is an algorithm that extends keys and makes it harder to brute-force attack a hash. Unlike PBKDF2, however, scrypt is designed to use either a large amount of computer memory or force many more calculations as it runs.

For valid users having to only hash one password to check if it matches a stored value, the cost is small. But for someone attempting to try 100,000s of passwords it makes cost of doing so much higher or take prohibitively long.

There are also some algorithms which is fail in market and we are not use this algorithms like MD5 (Merkl Damaged), SHA1(Secure Hash Algorithm). If we used this algorithm there are some chance to taken our personal data which is available on web or 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:

If we created account on facebook when we entered the id and password salt technique also add their own password to make it more difficult for attacker which stole the passwords that's why the attacker cant access the password and our data is protected by salt technique.
With the help of salt technique we can avoid SQL injection attack, dictionary attack, birthday attack.

2.1 Methods of creating salt steps:

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

Example: How Password Store

The first table has two username and password combinations.

Username	Password
User1	Password123
User2	Password456

The salt value is 8 bytes(64 bit) long. The hashed value is the hash of the salt value appended to the plaintext password. Both the salt value and hashed value are stored.

Username	**Saltvalue**	**String to be hashed**	**Hashed value=SHA256(password+salt value)**
User1	E1F1532E5765	Password 123+ salt value	72AE25495A7891C40622D4F493
User2	84C03D0340DF	Password 456+ salt value	B4B660AB670867E9C732F7DE8

3. The WRONG Way: Short Salt & Salt Reuse

The most common salt implementation errors are reusing the same salt in multiple hashes, or using a salt that is too short.

1) Salt Reuse

A common mistake is to use the same salt in each hash. Either the salt is hard-coded into the program, or is generated randomly once. This is unsuccessful because if two users have the same password, they'll still have the same hash. An attacker can still use a reverse lookup table attack to run a dictionary attack on every hash at the same time. They just have to apply the salt to each password guess before they hash it. If the salt is hard-coded into a popular product, lookup tables and rainbow tables can be built for that salt, to make it easier to crack hashes generated by the product.

A new random salt must be generated each time a user creates an account or changes their password[2].

2) Short Salt

If the salt is too short, an attacker can build a lookup table for every possible salt. For example, if the salt is only three ASCII characters, there are only 95x95x95 = 857,375 possible salts. That may seem like a lot, but if each lookup table contains only 1MB of the most common passwords, collectively they will be only 837GB, which is not a lot considering 1000GB hard drives can be bought for under $100 today.

For the same reason, the username shouldn't be used as a salt. Usernames may be unique to a single service, but they are predictable and often reused for accounts on other services. An attacker can build lookup tables for common usernames and use them to crack username-salted hashes. To make it impossible for an attacker to create 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 should be at least 32 random bytes[2].

3. CONCLUSIONS

Salt provide security.
Salt technique prevent the attackers or helpful for avoid the dictionary attack, birthday attack.
Salt hash password prevents the attacker by mean of - the attacker will now have to recalculate their entire dictionary for every individual account they're attempting to crack. But salt can only help against prebuilt dictionaries, if intruder gets access to our system and if uses brute force attack, than salt will not provide must security.

REFERENCES

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

[2] Salted Password Hashing- http://www.codeproject.com/Articales/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

BIOGRAPHIES

Diksha Sukhdeo Borde
MCA TY
MGM' JNEC Aurangabad

Poonam Ashok Hebare
MCA TY
MGM' JNEC Aurangabad

Priyanka Dilip Dhanedhar
MCA TY
MGM' JNEC Aurangabad