WIKIPEDIA

Salt (cryptography)

In cryptography, a **salt** is random data that is used as an additional input to a one-way function that "hashes" data, a password or passphrase. Salts are closely related to the concept of nonce. The primary function of salts is to defend against dictionary attacks or against its hashed equivalent, a pre-computed rainbow table attack.[1]

Salts are used to safeguard passwords in storage. Historically a password was stored in plaintext on a system, but over time additional safeguards developed to protect a user's password against being read from the system. A salt is one of those methods.

A new salt is randomly generated for each password. In a typical setting, the salt and the password (or its version after Key stretching) are concatenated and processed with a cryptographic hash function, and the resulting output (but not the original password) is stored with the salt in a database. Hashing allows for later authentication without keeping and therefore risking the plaintext password in the event that the authentication data store is compromised.

Since salts do not have to be memorized by humans they can make the size of the rainbow table required for a successful attack prohibitively large without placing a burden on the users. Since salts are different in each case, they also protect commonly used passwords, or those who use the same password on several sites, by making all salted hash instances for the same password different from each other.

Cryptographic salts are broadly used in many modern computer systems, from Unix system credentials to Internet security.

Contents

Unix implementations

1970s-1980s

Earlier versions of Unix used a password file `/etc/passwd` to store the hashes of salted passwords (passwords prefixed with two-character random salts). In these older versions of Unix, the salt was also stored in the passwd file (as cleartext) together with the hash of the salted password. The password file was publicly readable for all users of the system. This was necessary so that user-privileged software tools could find user names and other information. The security of passwords is therefore protected only by the one-way functions (enciphering or hashing) used for the purpose. Early Unix implementations limited passwords to 8 characters and used a 12-bit salt, which allowed for 4,096 possible salt values. This was an appropriate balance for 1970s computational and storage costs.[2]

1980s-

The shadow password system is used to limit access to hashes and salt. The salt is 8 characters, the hash is 86 characters, and the password length is unlimited.

Example usage

Here is an incomplete example of a salt value for storing passwords. This first table has two username and password combinations. The password is not stored.

Username	Password
`user1`	`password123`
`user2`	`password123`

The salt value is generated at random and can be any length, in this case 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	Salt value	String to be hashed	Hashed value = SHA256 (Password + Salt value)
`user1`	`E1F53135E559C253`	`password123+E1F53135E559C253`	`72AE25495A7981C40622D49F9A52E4F1565C90F048F59027BD9C8C8900D5C3D8`
`user2`	`84B03D034B409D4E`	`password123+84B03D034B409D4E`	`B4B6603ABC670967E99C7E7F1389E40CD16E78AD38EB1468EC2AA1E62B8BED3A`

As the table above illustrates, different salt values will create completely different hashed values, even when the plaintext passwords are exactly the same. Additionally, dictionary attacks are mitigated to a degree as an attacker cannot practically precompute the hashes. However, a salt cannot protect against common or easily guessed passwords.

A complete password storage scheme would typically include a salt and pepper.[3]

Common mistakes

Public salt or salt reuse

A public salt is when a programmer uses the same salt for every hashed password.

While this will make current rainbow tables useless, if the salt is hard coded into a popular product that salt can be extracted and a new rainbow table can be generated using that salt.

Using a single salt also means that every user who inputs the same password will have the same hash. This makes it easier to attack multiple users by cracking only one hash.

When an attacker executes a dictionary attack to break a single password, the public hash offers no security as the attacker knows the password and the salt, the only two inputs to the hash function.[3]

Short salt

If a salt is too short, it will be easy for an attacker to create a rainbow table consisting of every possible salt appended to every likely password. Using a long salt ensures that a rainbow table for a database would be prohibitively large.[3]

Web application implementations

It is common for a web application to store in a database the hash value of a user's password. Without a salt, a successful SQL injection attack may yield easily crackable passwords. Because many users re-use passwords for multiple sites, the use of a salt is an important component of overall web application security.[4] Some additional references for using a salt to secure password hashes in specific languages (PHP, .NET, etc.) can be found in the external links section below.

Benefits

To understand the difference between cracking a single password and a set of them, consider a single password file that contains hundreds of usernames and hashed passwords. Without a salt, an attacker could compute hash(attempt[0]), and then check whether that hash appears anywhere in the file. The likelihood of a match, i.e. cracking one of the passwords with that attempt, increases with the number of passwords in the file. If salts are present, then the attacker would have to compute hash(salt[a], attempt[0]), compare against entry A, then hash(salt[b], attempt[0]), compare against entry B, and so on. This defeats "reusing" hashes in attempts to crack multiple passwords.

Salts also combat the use of hash tables and rainbow tables for cracking passwords.[5] A hash table is a large list of pre-computed hashes for commonly used passwords. For a password file without salts, an attacker can go through each entry and look up the hashed password in the hash table or rainbow table. If the look-up is considerably faster than the hash function (which it often is), this will considerably speed up cracking the file. However, if the password file is salted, then the hash table or rainbow table would have to contain "salt . password" pre-hashed. If the salt is long enough and sufficiently random, this is very unlikely. Unsalted passwords chosen by humans tend to be vulnerable to dictionary attacks since they have to be both short and meaningful enough to be memorized. Even a small dictionary (or its hashed equivalent, a hash table) has a significant chance of cracking the most commonly used passwords. Since salts do not have to be memorized by humans they can make the size of the rainbow table required for a successful attack prohibitively large without placing a burden on the users.

More technically, salts protect against hash tables and rainbow tables as they, in effect, extend the length and potentially the complexity of the password. If the rainbow tables do not have passwords matching the length (e.g. an 8-byte password, and 2-byte salt, is effectively a 10-byte password) and complexity (non-alphanumeric salt increases the complexity of strictly alphanumeric passwords) of the salted password, then the password will not be found. If found, one will have to remove the salt from the password before it can be used.

Additional benefits

The modern shadow password system, in which password hashes and other security data are stored in a non-public file, somewhat mitigates these concerns. However, they remain relevant in multi-server installations which use centralized password management systems to push passwords or password hashes to multiple systems. In such installations, the root account on each individual system may be treated as less trusted than the administrators of the centralized password system, so it remains worthwhile to ensure that the security of the password hashing algorithm, including the generation of unique salt values, is adequate.

Salts also make dictionary attacks and brute-force attacks for cracking large numbers of passwords much slower (but not in the case of cracking just one password). Without salts, an attacker who is cracking many passwords at the same time only needs to hash each password guess once, and compare it to all the hashes. However, with salts, each password will likely have a different salt; so each guess would have to be hashed separately and compared for each salt, which is considerably slower than comparing the same single hash to every passworc

Another (lesser) benefit of a salt is as follows: two users might choose the same string as their password, or the same user might choose to use the same password on two machines. Without a salt, this password would be stored as the same hash string in the password file. This would disclose the fact that the two accounts have the same password, allowing anyone who knows one of the account's passwords to access the other account. By salting the passwords with two random characters, even if two accounts use the same password, no one can discover this just by reading hashes.

Recent cyber attacks using salt cryptography

After gaining access to servers downloading users information from those servers and showing the data online has become common.

- Adobe customer data breach[6] where the user took the information and kept the data online using salt cryptography

See also

- Password cracking

- Cryptographic nonce
- Initialization vector
- Padding
- "Spice" in the Hasty Pudding cipher
- Rainbow tables
- Pepper (cryptography)

References

1. "Passwords Matter" (http://bugcharmer.blogspot.com/2012/06/passwords-matter.html). Retrieved 2016-12-09.
2. "How Unix Implements Passwords [Book]" (https://www.safaribooksonline.com/library/view/practical-unix-and/0596003234/ch04s03.html).
3. "Secure Salted Password Hashing - How to do it Properly" (https://crackstation.net/hashing-security.htm#salt).
4. "ISC Diary – Hashing Passwords" (http://www.dshield.org/diary.html?storyid=11110). Dshield.org. Retrieved 2011-10-15.
5. http://kestas.kuliukas.com/RainbowTables/
6. "Adobe customer data breached – login and credit card data probably stolen, all passwords reset" (https://nakedsecurity.sophos.com/2013/10/04/adobe-owns-up-to-getting-pwned-login-and-credit-card-data-probably-stolen-all-passwords-reset/). *Naked Security*. 2013-10-04. Retrieved 2017-05-18.

External links

- Morris, Robert; Thompson, Ken (1978-04-03). "Password Security: A Case History." Murray Hill, NJ, USA: Bell Laboratories. Archived from the original on 2013-08-21.
- Wille, Christoph (2004-01-05). "Storing Passwords - done right!".
- OWASP Cryptographic Cheat Sheet
- how to encrypt user passwords

Retrieved from "https://en.wikipedia.org/w/index.php?title=Salt_(cryptography)&oldid=821644119"