Yuliia STEPANENKO[1], Valeriia SOLODOVNIK[2]

Scientific supervisors: Andriy FESENKO[3], Larysa MURYTENKO[4]

DOI: https://doi.org/10.53052/9788366249868.24

BEZPIECZNE PRZECHOWYWANIE HASŁA Z FUNKCJĄ KRYPTOGRAFICZNEGO HASH

Streszczenie: Przeprowadzana jest analiza metod autoryzacji użytkowników w systemie. Analizowany jest algorytm haszowania hasła. Przeprowadzane jest porównanie popularnych algorytmów haszujących.

Słowa kluczowe: hasło, algorytm haszowania, uwierzytelnianie, łamanie haseł, funkcja deterministyczna, iteracje, sól.

SECURE PASSWORD STORAGE WITH CRYPTOGRAPHIC HASH FUNCTION

Summary: The analysis of methods of user authorization in the system is carried out. The password hashing algorithm is analyzed. Comparison of popular hashing algorithms is carried out.

Keywords: password, hash algorithm, authentication, password cracking, deterministic function, iterations, salt.

1. Formulation of the problem

Passwords play a critical role in our whole life. It's the most common security method to authenticate or verify a user's online identity [1]. They provide a powerful guard

[1] Taras Shevchenko National University of Kyiv, Faculty of Informaition Technology, Cyber Security and Information Protection: juliastepanenko2900@gmail.com
[2] Taras Shevchenko National University of Kyiv, Faculty of Informaition Technology, Cyber Security and Information Protection: valeriamin6@gmail.com
[3] Taras Shevchenko National University of Kyiv, Faculty of Informaition Technology, Cyber Security and Information Protection: aafesenko88@gmail.com
[4] Taras Shevchenko National University of Kyiv, Faculty of Informaition Technology, Cyber Security and Information Protection: myrutenko.lara@gmail.com

against unauthorized access to systems and data, and are ubiquitously used in various online activities such as communication, learning and, of course, shopping and banking. User authentication via password relies on the *something you know* authentication factor, i.e., you know some secret that no one else does. Although two other authentication factors *something you have* (e.g., hardware token) and *something you are* (e.g., fingerprint) have not gained a wide acceptance on the Internet, above all because of their high cost but limited flexibility [2]. On the other side, passwords are very simple, inexpensive, easy to implement, and convenient to use for users as well as developers. Consequently, they occupy the key position in online user authentication, and this situation will not change in the foreseeable future due to the above reasons.

Unfortunately, passwords suffer from two seemingly intractable problems: password cracking and password theft. Password cracking (also known as password guessing) is an attack in which an adversary attempts to guess the users' password. For security reasons, it makes sense to store passwords in hashed form. This guards against the possibility that someone who gains unauthorized access to the database can retrieve the passwords of every user in the system [3].

Hashing performs a one-way transformation on a password, turning the password into another String, called the hashed password. «One-way» means that it is practically impossible to go the other way – to turn the hashed password back into the original password. There are several mathematically complex hashing algorithms that fulfill these needs. By default, the Personalization module uses the MD5 algorithm to perform a one-way hash of the password value and to store it in hashed form. The hashed password value is not encrypted before it is stored in the database. When a member attempts to log in, the Personalization module takes the supplied password, performs a similar one-way hash and compares it to the database value. If the passwords match, then login is successful. This is how it works. As stated by OWASP, hash functions used in cryptography have the following key properties:

- it's easy and practical to compute the hash, but difficult or impossible to re-generate the original input if only the hash value is known;
- it's difficult to create an initial input that would match a specific desired output.

Thus, in contrast to encryption, hashing is a one-way mechanism. The data that is hashed cannot be practically «unhashed» (Fig. 1).

There ia another property that makes hash functions suitable for password storage. This is the fact that they are deterministic. A deterministic function is a function that given the same input always produces the same output. This is vital for authentication, since we need to have the guarantee that a given password will always produce the same hash; otherwise, it would be impossible to consistently verify user credentials with this technique.

There are numerous functions used for password hashing including: MD5, SHA1, SHA256 - SHA512, PBKDF2, BCRYPT, SCRYPT and Argon2.

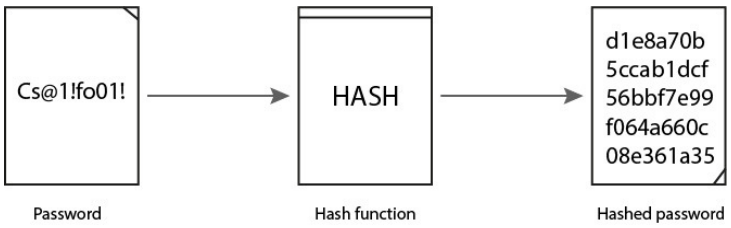

Figure 1. Hashing algorithm flow example – one-way

A hashing scheme takes as an input a plaintext password and transforms it into a hash value considering three parameters: hash function, iterations and salt. More specifically, the core parameter of a hashing scheme is the employed hash function, such as MD5 [3]. The iterations parameter is optional and specifies the number of consecutive executions of the employed hash function to compute the hash value. For example, if a hashing scheme uses the MD5 hash function and the number of iterations is 100, then it will conduct 100 consecutive executions of MD5 to compute the password hash. The number of iterations can be adjusted so that the computation of the hash value takes an arbitrarily large amount of computing time (also known as key stretching) [4]. In this way, iterations are used to slow down password guessing attacks. Regarding the last parameter, the salt is a cryptographically-strong random value that is combined with the value to be hashed. The salt is known to the system and is unique for each password stored. Consistently mixing the salt with the password each time the password is hashed creates a unique hash value even if others use the same password [5]. Using random 6 salts, rainbow tables become ineffective. That is, an attacker won't know in advance what the salt value is and therefore he or she cannot pre-compute a rainbow table.

2. Conclusion

Various types of cryptographic systems exist that have different strengths and weaknesses. Typically, they are divided into two classes; those that are strong, but slow to run and those that are quick, but less secure.

The service should store a cryptographically strong hash of passwords that cannot be reversed. Hashing is the process of performing a one-way transformation of any string of bytes into a different and deterministic new string of characters. There are many examples of hashes that are appropriate for use to store passwords, such as PBKDF2, Argon2, Scrypt, or Bcrypt.

According to Jeff Atwood, "hashes, when used for security, need to be slow." A cryptographic hash function used for password hashing needs to be slow to compute because a rapidly computed algorithm could make brute-force attacks more feasible, especially with the rapidly evolving power of modern hardware. We can achieve this by making the hash calculation slow by using a lot of internal iterations or by making the calculation memory intensive. It's important to achieve a good balance of speed and usability for hashing functions. A well-intended user won't have a noticeable performance impact when trying a single valid login.

Cryptography is a constantly changing field. However, experience has shown that there are some algorithms to avoid when storing passwords. Password guessing

attacks greatly benefit from multiple processing cores, especially for hashing schemes that can be executed in parallel. MD5, SHA1, SHA256, SHA512 hash functions can be executed in parallel on multi-processor systems, fact that increases significantly the efficiency of password guessing attacks. MD5 and SHA-1 are common hashing algorithms used today but it's a necessary to avoid these algorithms because these technologies are deprecated. There are many examples of hashes that are appropriate for use when storing passwords, such as PBKDF2, Argon2, Scrypt, or Bcrypt.

REFERENCES

1. HERLEY C., VAN OORSCHOT P., PATRICK A. S.: Passwords: If we're so smart, why are we still using them? In Proceedings of the Financial Cryptography and Data Security Conference (2009).
2. STRAHS B., YUE C., WANG H.: Secure Passwords Through Enhanced Hashing, 1–3.
3. NTANTOGIAN C., MALLIAROS S., XENAKIS C.: Evaluation of Password Hashing Schemes in Open Source Web Platforms, 5-7.
4. BELLARE M., CANETTI R., KRAWCZYK H.: Keying hash functions for message authentication. In Proceedings of Crypto'96 (1996), 1–15
5. MADDOX I., MOSCHETTO K.: Modern password security for system designers. What to consider when building a password-based authentication system, 3-16.