IBIMA Publishing
Communications of International Proceedings
https://ibimapublishing.com/p-articles/44SE/2024/4453124/
Vol. 2024 (24), Article ID 4453124

Security of Credentials: Algorithms And Techniques for Storing Passwords*

Piotr KONTOWICZ

Poznan University of Technology, Faculty of Computing and Telecommunications,
Piotrowo 3, 60-965 Poznań, Poland

Correspondence should be addressed to: Piotr KONTOWICZ, piotr.kontowicz@put.poznan.pl

* Presented at the 44th IBIMA International Conference, 27-28 November 2024 Granada, Spain

Abstract

The developers and administrators face major challenges in assuring the security of the users passwords within the applications as a result of growing breaches and cyber attacks. Improper mechanisms of storage may lead to serious security consequences for password data protection. The article is directed to the presentation of modern approaches and best practices concerning password storing. It discusses the most relevant password storage approaches, considering the risks involved and presenting options and their aftermath. The paper provides a step by step guide on how user passwords should be securely stored.

Keywords: Password security, cryptographic hash functions, password storage

Introduction

One of the major challenges for web application developers is ensuring the secure storage of user passwords. As attacks on web applications become more frequent and lead to an increase in data breaches, protecting user credentials has become a top priority for developers and administrators alike. Neglecting to implement strong password storage mechanisms can have serious consequences, including financial penalties for organizations and damage to their reputation, which is often harder to measure. A data breach can damage customer trust, potentially causing them to stop using the organizations services. Using secure password storage techniques, especially through cryptographic hash functions, can greatly hinder unauthorized access to user credentials. Additionally, when users follow modern security practices such as creating strong and unique passwords these measures become even more effective, further reducing the likelihood of unauthorized access.

Password storage options

When choosing a method to store user passwords, application developers or architects can consider the following approaches:

1. plain text: This method involves storing passwords exactly as entered by users, without any alteration. It is highly insecure and strongly discouraged. Plain text storage allows system administrators to access user credentials, raising the risk of misuse for unauthorized access to other services. Moreover, plain text passwords could be unintentionally exposed through backups or system migration data. If the application is vulnerable to SQL injection[1], attackers may directly retrieve these passwords from the database. Due to these significant risks, storing passwords in plain text is considered unacceptable.

Cite this Article as: Piotr KONTOWICZ, Vol. 2024 (24) “Security Architecture in Java – A pragmatic approach" Communications of International Proceedings, Vol. 2024 (24), Article ID 4453124, https://doi.org/10.5171/2024.4453124

2. encrypted form (ciphertext): Encrypting passwords might seem like an improvement over plain text storage, but this method has inherent vulnerabilities. The main concern is the management of the encryption key, which must also be stored within the system. If an attacker gains access to the database containing encrypted passwords, it is likely they could also locate the encryption key. With this key, the attacker could decrypt the passwords and compromise user accounts. As a result, encrypting passwords is not a secure or advisable practice for password storage.

3. hashing: Cryptographic hash functions are a far superior choice for password storage due to their properties, such as irreversibility and collision resistance. Irreversibility ensures that reconstructing the original password from its hash is computationally infeasible, while collision resistance minimizes the likelihood of different inputs producing the same hash. Hash functions also compress inputs into fixed length outputs, regardless of the input size, ensuring consistency. However, the computational efficiency of hash functions can be a weakness in password storage, as it may facilitate brute force attacks. To address this, additional security mechanisms such as salting and peppering can be applied. Developers can either implement these enhancements themselves or use tested solutions that integrate advanced security features. It is also important to note that hash functions are deterministic, producing consistent outputs for the same inputs with the same algorithm.

4. PBKDF[2] (Password-Based Key Derivation Function): This method is the most secure for password storage. PBKDFs combine cryptographic hash functions with mechanisms that intentionally slow down computations, making brute force attacks significantly more difficult. They also support techniques like salting, which adds unique data to the hashing process, and peppering, which introduces system wide random values, further strengthening security. These features make PBKDFs the most reliable and recommended option for storing passwords securely.

Main differences between hash functions and PBKDF

Hash functions like SHA-256[3], SHA-512, and SHA-3[4] are designed to produce fixed length hashes from input data. These functions are commonly used in areas such as digital signatures and verifying data integrity. While they were originally developed for tasks like testing file integrity, their design emphasizes computational speed, making them highly efficient for these purposes. However, this efficiency, coupled with their single pass operation and the absence of built in features like salting or peppering, makes them less suitable for securely storing passwords. Their vulnerability to brute force attacks is further exacerbated by advancements in hardware, such as GPUs, which enable highly parallelized computations.

On the other hand, Password-Based Key Derivation Functions (PBKDFs) are explicitly designed for secure password storage. These functions include iterative processing and provide built in support for salting and peppering, which significantly enhance their resilience against brute force attacks compared to general purpose hash functions. This improved security does come with a tradeoff, as the iterative nature of PBKDFs intentionally increases computational demands, thereby slowing the hashing process to effectively deter attackers.

Brute force and dictionary attacks on passwords

Brute force[5] and dictionary attacks are among the most common techniques for compromising hash functions. Brute force attacks involve systematically generating all possible combinations of characters from a specified set and hashing each combination until a match with the target password is found. Although this method is guaranteed to succeed eventually, it is highly time consuming and demands significant computational power. However, its efficiency increases when the target password is short or lacks complexity, such as when it does not incorporate a variety of character groups.

Dictionary attacks[6], on the other hand, rely on predefined lists of potential passwords, known as dictionaries. These dictionaries often contain passwords exposed in data breaches or generated from language specific word lists. The attack hashes each word from the dictionary and compares the resulting hash with the target value. A match indicates the password has been successfully identified. Although faster than brute force attacks, dictionary attacks are not always successful; their effectiveness depends on the quality and comprehensiveness of the dictionary.

Hashcat[7] is one of the most widely used tools for executing brute force and dictionary attacks. This cross platform software supports both CPU and GPU based processing, offering flexibility in implementing these

attacks. Performance tests on a system with 64 GB of RAM, an AMD Ryzen 9 5950X processor, and an NVIDIA GeForce RTX 3060 GPU have been conducted. The results are summarized in Table 1.

These findings emphasize the necessity of employing dedicated password storage solutions to mitigate password recovery attempts following a data breach. Such solutions reduce the hashing throughput for potential passwords, significantly increasing the computational workload on attackers. For example, computing a specific number of PBKDF2-SHA512 hashes using the Python passlib library takes approximately 167,180 times longer than computing the same number of SHA-256 hashes. This dramatic increase in computational time enhances password security by raising the cost and complexity of attacks. In the comparison provided, the heightened computational expense discourages attackers from pursuing such methods, making the attack far less viable.

Table 1. Computing performance on the presented hardware platform

Algorithm	Performance (number of digests calculated per second)
SHA2-256	3307.5 MH/s
SHA2-512	979.5 MH/s
SHA3-256	729.6 MH/s
SHA3-512	733.2 MH/s
Django (PBKDF2-SHA256)	127.6 kH/s
Python passlib PBKDF2-SHA256 *	44339 H/s
Python passlib PBKDF2-SHA512 *	19784 H/s

Techniques for reinforcing password storage

Password storage functions often include additional mechanisms, such as salting and peppering, to enhance the security of stored passwords. These techniques are designed to strengthen the resistance of hashed passwords against brute force attacks and to counteract the effectiveness of rainbow tables[8]. Here is how these mechanisms work:

1. salting: Salting involves adding a randomly generated string, known as a salt, to the password before hashing it. Each user is assigned a unique salt, which is stored in the database alongside the hash. This ensures that even if two users have the same password, their hashes will be different. By introducing this variability, salting significantly complicates attack attempts and makes precomputed rainbow tables ineffective.
2. peppering: Peppering adds another layer of security by appending a secret, fixed string, called a pepper, to the password before hashing. Unlike salts, peppers are not stored in the database and remain known only to the system or application administrator. This secrecy further complicates efforts to reverse engineer the password. To maximize its effectiveness, the pepper should be random, sufficiently long, and securely stored outside the database.

Both techniques work together to enhance the overall robustness of password storage systems, providing stronger protection against unauthorized access attempts.

Practical Recommendations

Using functions specifically designed for password storage is highly recommended in modern systems. Many modern day web application development tools natively support these functions. The following are the most commonly recommended options:

1. Argon2id: Currently considered the best choice for password storage, Argon2id was the first place winner of the 2015 Password Hashing Competition[9]. It allows fine tuning of key parameters, such as execution time, memory usage, and the number of processing threads. Its reliance on memory intensive operations makes it highly resistant to GPU based attacks, ensuring robust password security.
2. scrypt: For environments where Argon2id is not supported, the scrypt algorithm is a strong alternative. Like Argon2id, it permits customization of parameters, including execution time, memory requirements, and thread count. Its design also resists GPU based attacks, making it a secure option for password hashing.
3. bcrypt: Bcrypt is one of the most widely used algorithms for password storage. It is particularly suitable for systems requiring compliance with the FIPS-140 standard. A key feature of bcrypt is its configurable cost parameter, which sets the number of iterations to balance security and performance. While it is

secure and easy to implement, newer algorithms offer greater flexibility. One limitation of bcrypt is its input length restriction, typically capped at 72 bytes. Ignoring this limit can result in security vulnerabilities[10].

4. PBKDF2: Similar to bcrypt, PBKDF2 offers implementations compatible with the FIPS-140 standard, making it a suitable choice for systems requiring such compliance. However, PBKDF2 is less effective against hardware based attacks compared to newer algorithms, as it does not utilize memory intensive operations.

All the algorithms mentioned above are considered secure and offer adequate protection for password storage. However, whenever possible, it is advisable to prioritize newer algorithms like Argon2id or scrypt for their enhanced security features. For systems requiring specific compliance with standards, older, well established algorithms such as bcrypt or PBKDF2 may be preferred to fulfil those requirements.

Conclusion

The security of stored passwords can be significantly enhanced through the application of salting and peppering techniques. These measures effectively complicate brute force attacks, dictionary attacks, and the exploitation of rainbow tables. Furthermore, algorithms specifically designed for password storage should be employed to ensure optimal protection. The combined use of these techniques and dedicated algorithms substantially increases the computational cost of attacks, often rendering them economically unviable for malicious actors.

Acknowledgements

This research was funded by the Polish Ministry of Science and Higher Education (No. 0313/SBAD/1310).

Bibliography

- A. Sadeghian, M. Zamani, and S. M. Abdullah, “A Taxonomy of SQL Injection Attacks,” in *2013 International Conference on Informatics and Creative Multimedia*, Sep. 2013, pp. 269–273. doi: 10.1109/ICICM.2013.53.
- “PBKDF2,” *Wikipedia*. May 30, 2024. Accessed: Nov. 24, 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=PBKDF2&oldid=1226427534
- “SHA-2,” *Wikipedia*. Nov. 23, 2024. Accessed: Nov. 24, 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=SHA-2&oldid=1259180206
- “SHA-3,” *Wikipedia*. Nov. 20, 2024. Accessed: Nov. 24, 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=SHA-3&oldid=1258489076
- “Brute-force attack,” *Wikipedia*. Nov. 21, 2024. Accessed: Nov. 24, 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Brute-force_attack&oldid=1258773260
- “Dictionary attack,” *Wikipedia*. Nov. 06, 2024. Accessed: Nov. 24, 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Dictionary_attack&oldid=1255837237
- “hashcat - advanced password recovery.” Accessed: Nov. 24, 2024. [Online]. Available: https://hashcat.net/hashcat/
- H. Kumar *et al.*, “Rainbow table to crack password using MD5 hashing algorithm,” in *2013 IEEE Conference on Information & Communication Technologies*, Apr. 2013, pp. 433–439. doi: 10.1109/CICT.2013.6558135.
- “Password Hashing Competition.” Accessed: Nov. 24, 2024. [Online]. Available: https://www.password-hashing.net/
- “Okta AD/LDAP Delegated Authentication - Username Above 52 Characters Security Advisory.” Accessed: Nov. 23, 2024. [Online]. Available: https://trust.okta.com/security-advisories/okta-ad-ldap-delegated-authentication-username/