Wasit Journal of Engineering Sciences

Journal homepage: https://ejuow.uowasit.edu.iq

Analysis attackers' methods with hashing secure password using CSPRNG and PBKDF2

Nada Abdul Aziz Mustafa

Affiliations
Information Technology Section , University of Baghdad, Iraq
Correspondence
Nada Abdul Aziz Mustafa, Information Technology Section, University of Baghdad- College of Languages, Baghdad, Iraq
Email: nada@colang.uobaghada.edu.iq

Received
28-November-2023
Revised
06-December-2023
Accepted
08-February-2024
Doi:
10.31185/ejuow.Vol12.Iss2.502

Abstract
Our use of the Internet does not necessarily mean that our data is secure. Therefore, the need arose to use strong passwords to protect our database from hacking. In order to strengthen passwords and protect the database from hacking, the researcher proposed a system that contains two steps: the first one is storing the username after hashing it by using the hash 256 algorithm which makes it difficult for a hacker to guess the username; the second one is creating a strong passwords to avoid attackers by using one of the following two methods: The first is to add a random salt to the password by using the Cryptographically Secure Pseudorandom Number Generator (CSPRNG) algorithm, then hashing it by using hash 256 and storing it on the website.
The second is to use the Password-Based Key Derivation Function 2 (PBKDF2) algorithm which salts the passwords and extends them (deriving the password) before being hashed and stored on the website, to increase the time and effort of the attacker in guessing them. The results of the two mentioned methods are compared in terms of speed and resistance to attackers, then password attacks are analysed in addition to the calculation of the strength of the password, the time it takes for the user to log in and the time required for the attacker to guess the passwords per second. The Wolfram Alpha is used to calculate the mathematic operations and system implementation was done by using IntelliJ IDEA with java FX.

Keywords: Hash 256, CSPRNG, PBKDF2, Password hashing**,** Dictionary attacks, Brute force attacks, and Rainbow tables attack.

الخلاصة: إن استخدامنا للإنترنت لا يعني بالضرورة أن بياناتنا آمنة. ولذلك ظهرت الحاجة إلى استخدام كلمات مرور قوية لحماية قاعدة بياناتنا من الاختراق. ومن أجل تعزيز كلمات المرور وحماية قاعدة بياناتنا ، اقترح الباحث نظاماً يتضمن خطوتين: الأولى هي تخزين اسم المستخدم بعد تجزئته باستخدام خوارزمية التجزئة 256 مما يجعل من الصعب على المهاجم تخمين اسم المستخدم؛ والثانية هي إنشاء كلمات مرور قوية لتجنب المهاجمين باستخدام إحدى الطريقتين التاليتين: الأولى هي إضافة ملح عشوائي إلى كلمة المرور باستخدام خوارزمية مولد الأرقام العشوائية الآمنة التشفير (CSPRNG)، ثم تجزئتها باستخدام خوارزمية التجزئة 256 وتخزينها على الموقع. والثانية هي استخدام خوارزمية وظيفة اشتقاق المفاتيح المستندة إلى كلمة المرور 2 (PBKDF2) التي تقوم بتمليح كلمات المرور وتوسيعها (اشتقاق كلمة المرور) قبل تجزئتها وتخزينها على موقع الويب، لزيادة وقت المهاجم وجهده في تخمينها. ثم تتم مقارنة نتائج الطريقتين المذكورتين من حيث السرعة ومقاومة المهاجمين، وبعد ذلك يتم تحليل هجمات كلمات المرور بالإضافة إلى حساب قوة كلمة المرور والوقت الذي يستغرقه المستخدم لتسجيل الدخول، والوقت اللازم لتسجيل دخول المهاجم لتخمين كلمات المرور في الثانية الواحدة. وقد تم استخدام Wolfram Alpha لحساب العمليات الرياضية وتنفيذ النظام باستخدام برنامج IntelliJ IDEA مع java FX.

1. INTRODUCTION

There are two important methods for maintaining data security and they have different functions: hashing and encryption. When retrieval of plain text is required, cryptography is the appropriate method, using a key for encryption and decryption [1]. As for hashing, it is one-way and cannot be decrypted, a hacker can try to reverse engineer it [2]. As nothing is safe online, even computers that are not connected to the Internet are not completely secure, which is why passwords are used, not to ensure that attacks will not happen, but to reduce the risk [3].

In previous studies, the passwords are run through a hash algorithm that converts passwords into different letters and symbols. This is not sufficient to provide the necessary protection, especially if the two passwords are identical, which leads to identical hashes. It is then easy to hack passwords through brute force attacks, dictionary attacks, and rainbow attacks [4]. This is why many recent studies have emphasized the use of salting, where

random data is added to passwords before running them through the hash algorithm, which makes them unique and difficult to hack. Using hashing and salting techniques together helps create completely different hashes. Despite the use of a random series of salts, the brute force attack is still effective and therefore there is a need for other techniques to increase security to be more immune to these attacks [5], and this is what we sought in our research .Security can't be guaranteed, however, there are some ways to reduce the risk of attacks, such as adding salt or stretching passwords. The two techniques are considered two tasks to prevent attacks against passwords [6].

2. Related Work

N. Ogini and N. Ogwara focused on the security of database passwords, where they used the md5 hash algorithm with salt. In their research, they used the username as a salt rather than generating it randomly, which made passwords easier for attackers to guess [7].
B. Diksha., H. Poonam, D. Priyanka they focused on the importance of using hashing with salting in confronting a dictionary attack and stressed that using salt alone, even if it is random, does not prevent a brute force attack [8].
K. Rajeshree , R. Shubhangee, N. Chaitanya and p. Nidhi provided security to the stored data by encrypting it using the AES256 encryption algorithm with pbkdf2, where the username and password were encrypted using a key created from the user's password [9].

3. Salting

The salt is added before or after the password and before the hash to hide the actual password. As a result, hackers face difficulty in discovering the original hashed password because of the salt used. This technique makes the methods of research tables and rainbow tables by attackers ineffective, and salt is used in the form of a random sequence which is different for each user to make the reverse research table attack ineffective [10]. Using the same salt in every hash is considered a major error. It is preferable for the salt to be encrypted and not too short. For example, the probability of a salt consisting of 3 Asci characters is approximately equal to 857,375 possibilities, and this is not a lot in the presence of 1000 gigabyte drives at a cheap price [6]. A salt that is unpredictable and provides a high level of randomness is generated using a secure pseudorandom number generator such as CSPRNG. In order to generate a random salt, the code RNGCryptoServiceProvider ( ) is used, whose sole task is to generate random numbers and these will be stored in a byte array Get Bytes [11].
Examples used for bad segmentation in forms:
md5 (md5 (salt) + md5 (password)), sha1 (str_rot13 (password + salt)).

4. Stretching passwords

It means making the time that it takes to crack the password difficult and costly for attackers by making the password longer before storing it in the database, and it is considered as a good way to thwart attackers [12]. PBKDF2 is one of the common methods used to expand passwords and takes advantage of a slow hash function to make attacks such as dictionary and rainbow table attacks very slow even with a fast Graphical Processing Units (GPU) [13], by repeating the hash several times, the number of iterations generates the time needed to derive the key [14], see Figure 1.

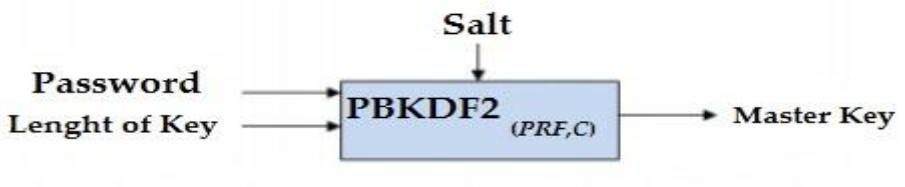

Figure 1 A generic diagram of PBKDF2 [15]

5. Hash function

The hashing algorithm converts a specific sequence into a fixed-length sequence to produce a unique, one-way output. Therefore, the original text cannot be obtained from the output. In order for the hash to be hacked, the attacker must try all possible inputs [16]. The most popular hashing algorithms are MD5, SHA-1, SHA-2, SHA-256, SHA-384 and SHA-512. Hash shortens the data and detects whether the original message has been changed, which is known as reliability, meaning if one bit is changed, we get a different hash. Through this, the user can

know whether one or more letters have been changed, as the hash extracts will be completely different [17]. See example of hashing using hash 256:
Hash256 ("hello") = 2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824

Hash256 ("hbllo") = 58756879c05c68dfac9866712fad6a93f8146f337a69afe7dd238f3364946366

Trying to guess the password is one of the simplest ways to break the hash. The most common methods are dictionary attacks and brute force attacks [18]. Hashing algorithms must be constantly improved to keep pace with the increasingly complex threat and development of devices. Compatibility between the hashing algorithm and text length is necessary, the text length should be less than 2 to the power of 64 for compatibility [19]. As for the steps to implement hash 256, additional bits are added to the message, known as padding bits, where the exact length is 64 bits less than a multiple of 512. While adding, the first bit must be one, and the rest must be filled with zeros. Adding 64 bits of data to make the final plaintext a multiple of 512 is known as the padding length. The default value is needed for eight buffers and we also need to store 64 different keys in an array from K [0] to K [63] which is known as the padding length. The message is divided into blocks of 512 bits each, and the output of each block is an input for the next block [20], see Figure 2:

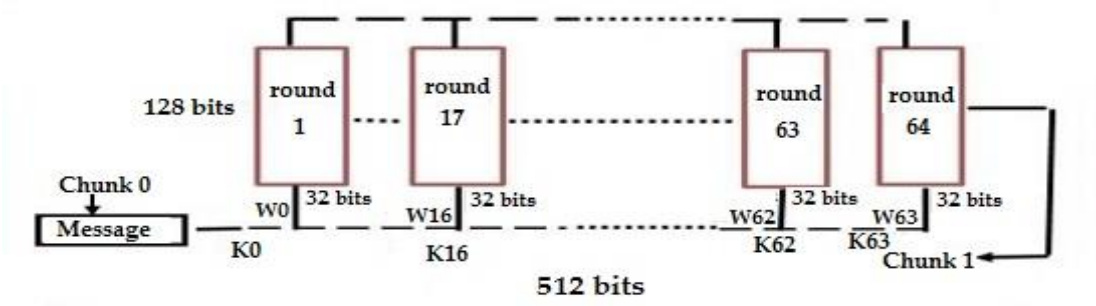

Figure 2 SHA 256 hash algorithm

The value of K[i] is pre-configured, while W[i] is another input that is calculated individually for each block, depending on the number of iterations being processed at the time. The final output of the block will serve as input to the next block with each iteration, and the cycle continues until it reaches the last 512-bit block [21]. The output is the final hash digest and the length of this digest is 256 bits, according to the name of this algorithm [22].

6. Password hashing

Websites store users' passwords after hashing them to achieve security, privacy and reduce the load on the database. The general workflow for account registration and authentication in a hash-based system is done through the following steps [23].
1. Creation of an account by the user, followed by hashing the password and storing it in the database.
2. Upon login, the password hash entered by the user is checked against the password hash stored in the database after which the user is granted access if the hash's matches. If they do not match, the user will be told that they entered invalid login credentials [24]. The properties that must be present in a hashing algorithm in order for it to be effective are: Determinism, when the same input must always produce the same hash output. As it is necessary to detect changes in the input data [25]. The second property is Collision resistance, meaning that it must be mathematically infeasible to find two different inputs that produce the same hash output [26]. This property guarantees data integrity in addition to irreversibility, meaning that determining the original input from the hash output must be mathematically impractical, in addition to Non - Degeneracy which refers to any slight change in the input, produces a significantly different hash. In general, a good hash algorithm must provide a safe and effective way to protect the integrity of the data. The common attacks used to crack password hashes are dictionary attacks and brute force attacks [27]. Guessing the password and hashing the guess is considered the simplest and most common way to hack the hash, which is computationally expensive and least efficient. However, it will eventually always find the password. Examples of these attacks include dictionary attacks that use a dictionary containing hashes for more than 10 to 20 million passwords [28].

7. The proposed system technique

Passwords are the main target for attackers as they are considered the primary key to accessing the data. Hashing passwords to prevent them from being hacked by an unauthorized person before storing them in the database is

considered an insufficient process as it exposes them to attacks such as dictionary attacks and brute force. The attacker can use a rainbow tables attack if they access the data. Adding a different random sequence of salts to each password makes the dictionary attack ineffective as the attacker has to try different salts for each word. A brute force attack is still effective despite using a random string of salts, as a result, another layer of security is needed by making the hash function slow, which is known as key extension or derive the key to be more immune to brute force attacks as the attacker will need more time to test the keys, see table 1.

Table 1 Cracking the hashed password using hash 256 without salt, with salt, using salt with 10 repetitions

password	Hash 256	type	result
Password without salt			
Ali	**862ef3927a7c8acfd3e79aa547800ef285cd9b13a39f5ec2d47554981cb313c8**	**Sha 256**	**Ali**
Password with salt			
Ali#r456vckh@ dewsauigAx@0	**7156e499ac3df1c37d36d5c372dae23ab80ea0b88f8e5e7c219dc20979311456**	**Unknown**	**Not found**
Password with salt & 10 iteration			
Ali#r456vckh@ dewsauigAx@0	**f3805669776303cd2a78fb84231a4ea1e54136961c5747dcd345833d24c8955b**	**Unknown**	**Not found**

In our proposed system, layers of security are achieved to protect the passwords from attack through the following points:

1. Hash the user name using the hash function 256 and store the hash result in the database. This will prevent half of the information from being available to the attacker.
2. Hash passwords using the same hash by applying one of two methods, then storing the hash result in the database. The first method is to generate a random salt (using the CSPRNG function) and add it to the beginning of the password before hashing, followed by storing the salt and the hash result in the database. As for the second method it uses the PBKDF2 algorithm, which generates a random salt of 16 bytes and adds 29000 iterations, then hashes it, stores the salt and the result of the hash in the database, or a salt of 8 bytes can be generated and iterations of 1000, see figures 3, 4, 5.
3. A comparison between the two methods in terms of the time it takes for the user to log in, the password strength, and the time required for the attacker to guess the passwords per second.

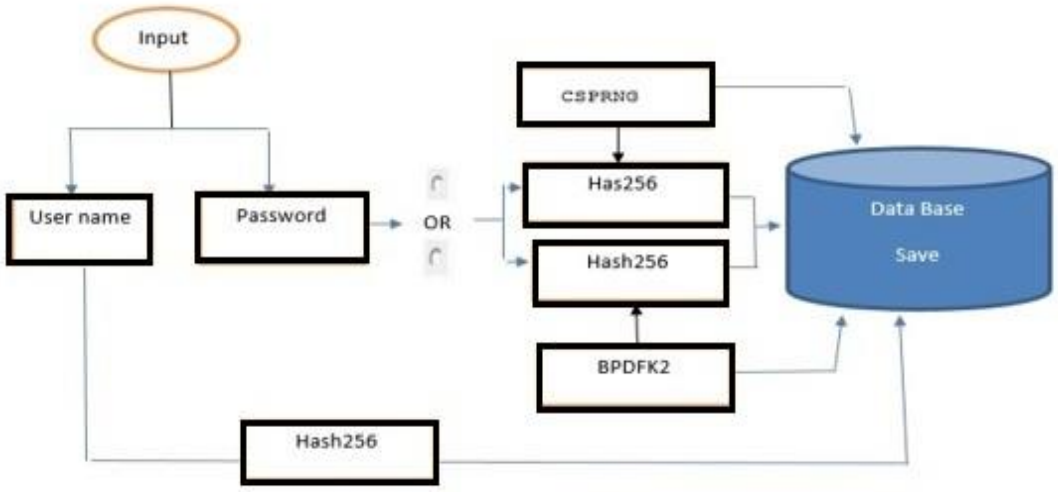

Figure 3 The proposed system for storing the hashed username and password with salt

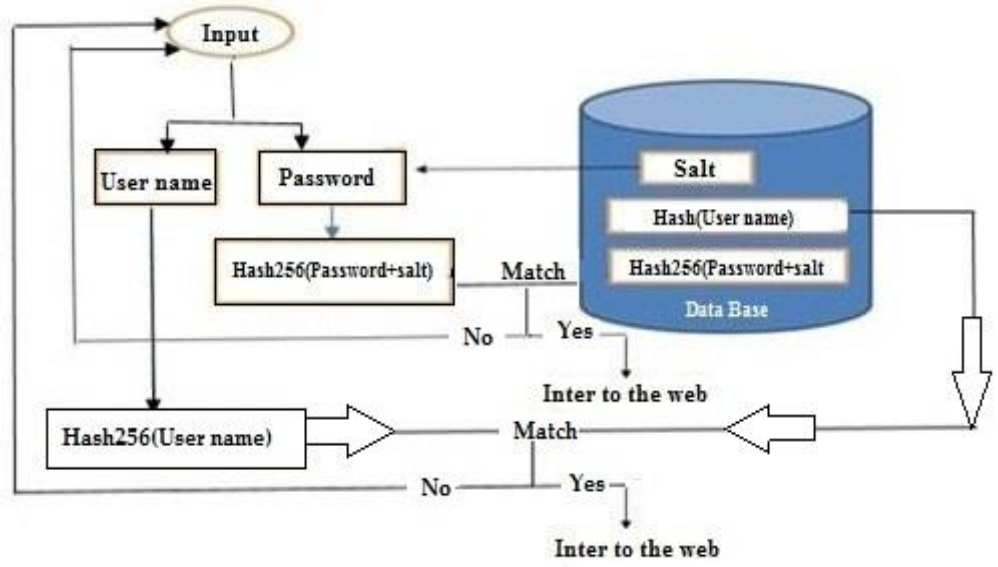

Figure 4 The proposed system for verifying the username and password

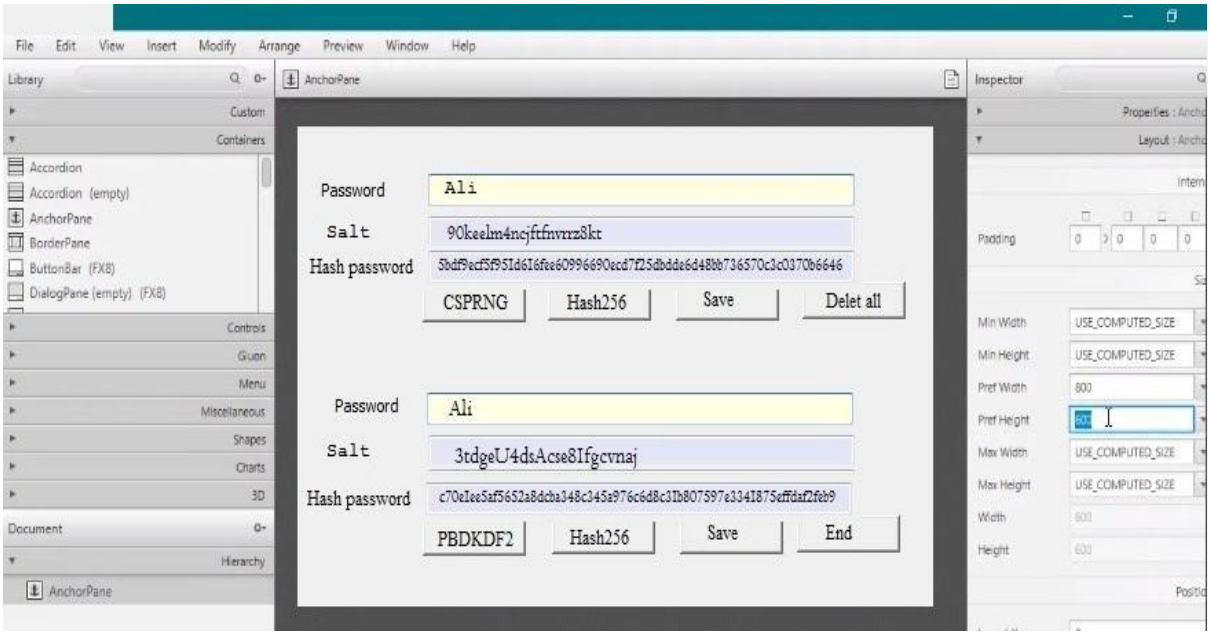

Figure 5 The proposed system using Intellij with JavaFx

The basic codes for proposed system are illustrated as follows:

```
# Insert salt
byte[]salt;
newRNGCryptoserviceprovider().GetBytes(salt=new byte[16]);

# The use of salt and automatic duplication using bpkdf2
Reg_form=registrationForm()
If Reg_form.validate_on_submit():
username=Reg_form.username.data
password=Reg_form.Password.data
pbkdf2_sha256.hash(password)

# Specified number of iterations 1500 and salt size 8 bytes
If Reg_form.validate_on_submit():
username=Reg_form.username.data
password=Reg_form.Password.data
pbkdf2_sha256.using(round=1500,salt_size=8).hash(password)
```

```
# The password hash stored in the database
Reg_form=registrationForm()
If Reg_form.validate_on_submit():
username=Reg_form.username.data
password=Reg_form.Password.data
hashed_password=pbkdf2_sha256.hash(password)
user=user(username=username,password=hashed_pswd)
db.session.add(user)
db.session.commit()
return redirect(url_for('login'))
```

The PBKDF2 algorithm adds salt and repetitions to the password, it will add a salt of 16 bytes and a repetition of 29000 iterations. The greater the number of iterations, the higher resistance we get against dictionary and brute force attacks as they generate a slow key and a slower login time. Increasing the number of iterations from 1,000 to 29,000 is as if a character was added to the password, see Table 2.

Table 2 Attacker's guessing per second

Without repetition	1000 repetitions	10000 repetitions
An attacker can make 50 billion hashes guesses per second.	An attacker can make 50 million hashes guesses per second. The amount of guesses per second has decreased.	An attacker can make 5 million hashes guesses per second. The amount of guesses per second has decreased further.

The table shows that the more repetitions are made, the greater the effort and the more time it takes for the attacker to guess.
After implementing the system, it was found that the time taken to log in for the method of using the random salt using the CSPRNG function is faster than the method of using PBKDF2 by a slight difference when we use 1000 iterations, and the difference in time will be greater the more the number of iterations is increased. As for safety, the second method is better as it requires the attacker to spend time, effort and resources to access the password, see Table 3.

Table 3 Time taken to log in between the CSPRNG method and the PBKDF2 method

The algorithm used	**Password**	**Hash function 256** **hash(salt + password)**	**Time taken to log in**
CSPRNG	Nada	f3424f0a35c32030d8bb8e17f466369f005ef8ea72178c86fca0809731bcb818	≈81.08ms
PBKDF2 (1000 repetitions)	Nada	e3bc5ad8b2763915e1242531d666127dba17f66b4f424c1ebc0f7ec0a67bbd5e	≈85.90ms
PBKDF2 (29000 repetitions)	Nada	fd076b821d2205d753f66bad4612887463b3dad86af5240d60ea4f21704003f2	≈98.04ms

The time it takes for the user to log in and then verify the password and user name cannot be determined as a fixed time, but rather an approximate time to allow access, because the speed depends on the central processing unit, which varies from one calculator to another in addition to the speed of the Internet. When entering the database, the standard encryption format is identified, which specifies dollar signs as segment separators, which begins by defining the hashing algorithm used, which in our system is the SHA256 algorithm with PBKDF2, then the number of repetitions is 29000, and the length of the salt is 16 bytes, and the salt will be different for each saved password, Figure 6 showing access to the database and using the same password, which is Ali, for the second and third user. The result is different hashes due to the use of different random salts with the iterations.

ID	Username	password
1	Wafa	test
2	Ali	$pbkdf2-Sha256$29000$9s3jfImxXqsngjCm10rnvA$30a308de5da03dacb63dcf134714e90c593efeaaae11b3447cbfd6a50d716b90
3	Ali	$pbkdf2-sha256$29000$7j6j8IwlbsgkKIsyerysmv$084b5b862f891d93c70f0f00a82eaed260a91060fcc15b090103bc2b9958f05e

Figure 6 Database used, the same password for the second and third user with a different hash result

To calculate the strength of the password, the number of possible characters is multiplied by 1. For example, if the password contains letters and numbers, then the number of letters in the English language is 26 and the numbers are 10 from 0 to 9, so the total = 36 letters and the password consists of 6 letters, so the result of the strength of the password = 36^6, that is, 2176782336 billion possible passwords. In conclusion, whenever the password is a combination of uppercase and lowercase letters, symbols and numbers, it will create huge possibilities which the attacker must pass through, this requires a high cost and a lot of effort. Wolfram Alpha was used to calculate the mathematical operations. Calculating the strength of the password is done using the following equation [29].

Sample space of a password=B = C ^ N.
B - The total number of possible passwords
C- Number of characters when the pool of (lowercase a-z=26, uppercase A-Z=26, numbers=10 and symbols=33)
N- Number of characters our password has, see table 4

Table 4 Calculating the strength of different passwords

C	password	N	(Password strength B)
26(lowercase)	hedfskui	8	208827064576
95 (33+26+26+10)	*@#$vl4B	8	6634204312890625
26(uppercase)	ANBVIJKGFDRESCX	15	1677259342285725925376
95 (33+26+26+10)	$rRtd348j0bv%fx	15	463291230159753366058349609375

In conclusion, the larger the value of B, the more difficult it is to guess the password and thus the more difficult it is to crack. As for the time, it is related to the speed of the server during password verification, and therefore the appropriate number of repetitions must be set. To calculate the time required for an attacker to guess passwords per second, which is equal to the strength of the password divided by 50 million per second, is done by applying the following equation, see Table 5.

((B / 50 million) seconds = ((36 ^ 6) / 50 million) seconds= 0.7256 minutes

B represents the strength of the password from the previous example

Table 5 Time required to increase cost

B	The time required for the attacker to guess =((B/50 million seconds))				
	minutes	hours	days	months	years
208827064576	69.61	1 hour 9 minutes 36.54 seconds			
6634204312890625		36857	1536	50.49	
4632912301597533660583496093 75					2.936×10^14 average Gregorian years

According to the table, to reduce the attacker's guesses per second, the cost value is increased, which is achieved by increasing the password strength.

8. Analysing password attacks and how to avoid them

An attacker can run a brute force attack or a dictionary attack as long as he can use the hash stored in the website. If the database is breached without being detected, then it's a big problem. The reason for this is that the hacker can easily make the website infect users with malware. Password hacking can be divided into two categories: Online attacks which are attempted by penetrating the password login page directly from the server, this is very difficult as it is limited by the speed of the network and can be easily detected. As for password attacks without an Internet connection, they save time for the attacker, when using password cracking tools, the attacker can break the encryption of passwords after taking their hash, in addition to hacking the password using a variety of software techniques and stealing user data by sending viruses that destroy the memory. The process of replacing letters with numbers and symbols in passwords is considered as a dangerous process as it is easy to detect, for example, 3 for E, 4 for A, and @ for a, and therefore it should be avoided. The attacker aims to identify the minimum length and complexity of the password, for example symbols and upper and lower case letters, as it aids in guessing. Information is collected from social media or deceptive conversations, these information help in guessing the password. Advanced password cracking tools often use a dictionary and mix numbers and symbols to imitate a real password with complex requirements. To mitigate a dictionary attack, it is necessary to limit the number of incorrect attempts to enter so that the user's account is automatically locked when the error is repeated for a certain number of attempts, in addition to enforcing strict password rules. Password – such as requiring users to create strong passwords.

The attack is carried out through trial and error in a brute force attack, whereby a group of numbers and symbols are entered. The method used is a software method for experimentation. This method is considered excellent when the password is short. Therefore, it is preferable that the password is at least 8 characters long. This method is used from the attack as a last resort as it is not effective. The use of salt prevents lookup and rainbow table attacks from hacking large sets of hashes. Due to the presence of advanced graphics cards (GPU) and hardware dedicated to calculating billions of hashes per second. Dictionary and brute force attacks on each hash remain effective, and for this reason, Key extension technology is used. When a website is hacked, it is necessary to determine how to hack and eliminate the security vulnerability. Users must be informed by email notifications. The strength of the password can be shown to the user to determine the security level desired. It is preferable that the minimum length

of the password be 13 characters, contain 3 symbols and 4 numbers. The most important thing is not to force users to change the password every now and then as it leads to inconvenience and thus choosing weak passwords.

9. Conclusions

The user name may be predictable. It may be the surname, the first letter of the name, the name of one of the children, or the email address. Thus, half of the information necessary to log in is available to the attacker, and only the password is needed. To prevent the user name information from being obtained by the attacker in this system, the user name was hashed using a hash of 256 and stored. A common mistake is to use the same salt in every hash, the reason for that is: if two users have the same password, they will still have the same hash. The attacker can still use a reverse lookup table attack to run a dictionary attack on every hash at the same time. This is why in this system, a new random salt was created every time the user creates an account or changes his password, Figure 3. After calculating the strength of password B, we conclude that the longer the key is, contains symbols, letters, and numbers, the greater the strength of the password, see Table 4. In this system the attacker's guesses per second were reduced in our system by increasing the cost value, which was obtained from increasing the strength of the password, see Table 5. We conclude that using the BPDFK2 method is better than the CSPRNG method due to the presence of repetitions, the greater the number of repetitions, the greater the time and effort for guessing by the attacker Table 1 and 2. The system application has proven that the hashing algorithm must be fast enough to handle the required volume of data in a reasonable period of time, that is, achieving security by balancing speed and security (complexity). If the log in time is long due to the increase in the number of repetitions, this leads to fatigue and inconvenience for the user, and thus increases the likelihood of entering a weak password by the user.

References

[1] P. Gauravram, "Cryptographic Hash Functions: Cryptanalysis, Design and Applications". Ph.D. thesis, Brisbane, Australia: Faculty of Information Technology, Queensland University of Technology, 2003.

[2] N.A.A. Mustafa, "An Improved Method for Hiding Text in Image Using Header Image" Waist Journal of Computer and Mathematic Science, Vol. 1, No. 4, pp.134 -148, 2022.

[3] S. Baibhav, J. Elizabeth" Safe Vault: A Password Manager", International Journal of Engineering Applied Sciences and Technology, Vol. 6, Issue 12, ISSN No. 2455-2143, Pages 93-97, Published Online April 2022 in IJEAST.

[4] P. Priyanshu, G. Parul, M. Ankit, K. Sameera, C. Ashutosh, "Brute Force, Dictionary and Rainbow Table Attack On Hashed Passwords", International Journal of Creative Research Thoughts ,IJCRT 2021 IJCRT Volume 9, Issue 4 April 2021, ISSN: 2320-2882, 2021.

[5] R. Urvesh , "An Experimental Evaluation on the Dependency Between One-Way Hash Functions and Salt", Conference: 2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT) At: Kharagpur, India,2020.

[6] L. ChangHee , L. Heejo , "A Password Stretching Method Using User Specific Salts", Conference: Proceedings of the 16th International Conference on World Wide Web, WWW 2007, Banff, Alberta, Canada, May 8-12, May 2007. DOI: 10.1145/1242572.1242772.

[7] N. Ogini, N. Ogwara, "Securing Database Passwords Using A Combination Of Hashing And Salting Techniques", IPASJ International Journal of Computer Science (IIJCS),Vol. 2, No. 8, pp. 52-58, pp. 52-58, 2014.

[8] B. Diksha., H. Poonam, D. Priyanka, "Overview Of Web Password Hashing Using Salt Technique", International Research Journal of Engineering and Technology (IRJET), e-ISSN: 2395-0056, p-ISSN: 2395-0072, Volume: 04 Issue: 11 | Nov -2017.

[9] K. Rajeshree , R. Shubhangee , N. Chaitanya , P. Nidhi, "An Effective Mechanism For Securing And Managing Password Using AES-256 Encryption & PBKDF2", International Journal of Electrical Engineering and Technology (IJEET), Volume 12, Issue 5, pp. 1-7, May 2021.

[10] N. Merhav, A. Cohen. “Universal Randomized Guessing With Application To Asynchronous Decentralized Brute–Force Attacks”, IEEE Transactions on Information Theory, 66(1):114–129, 2020.

[11] P.Thottempudi, T.Thottempudi, K. Bhushan, N. Rani, “Generation of Cryptographically Secured Pseudo Random Numbers Using Fpga”, International Journal of Electronics and Communication Engineering & Technology (IJECET), ISSN 0976 –6464(Print), ISSN 0976 – 6472(Online), Volume 5, Issue 2, pp. 21-29 IAEME, February 2014.

[12] O. Yoshimura, K. Arai, H. Okazaki, Y.Futa, “Formalization of Security Requirements And Attack Models For Cryptographic Hash Functions In Pro verif”, Proceedings of the 2019 International Conference on Security and Management, SAM 2019, Las Vegas, Nevada, USA, July 28 - August 1, 2019.

[13] N. Ali, B. Al Farawn, H.Rjeib, “Adding Salt to Hashing: A Better Way to Store Passwords”, In Telkomnika Telecommunication, Computing, Electronics and Control, volume 18, 2020.

[14] A. Iuorio , A. Visconti, “Understanding Optimizations and Mea-Suring Performances of PBKDF2”, in 2nd International Conference on Wireless Intelligent and Distributed Environment for Communication,WIDECOM 2019, Milan, Italy, February 11-13, 2019, vol. 27 of Lecture Notes on Data Engineering and Communications Technologies, pp. 101–114, Springer, 2019.

[15] T. Meltem, B. Elaine, B. William, C.Lily, “Recommendation for Password-Based Key Derivation”, NIST Special Publication 800-132, U.S. Department of Commerce, National Institute of Standards and Technology, December 2010.

[16] B. Rompay, “Analysis and Design of Cryptographic Hash Functions, MAC Algorithms and Block Ciphers”, Ph.D. thesis, Leuven, Belgium: Electrical Engineering Department, Katholieke Universiteit, 2004.

[17] S.Rajeev, G. Geetha, “Cryptographic Hash Functions: A Review”, International Journal of Computer Science Issues, ISSN (Online): 1694-0814. Vol 9. 461, 2012.

[18] Z. Zhao, Z. Dong, Y. Wang, “Security Analysis of A Password-Based Authentication Protocol”, Proposed to IEEE 1363, Theoretical Computer Science, Vol. 352, No. 1, pp. 280–287, 2006.

[19] R. Rivest, A. Benjamin, B.Daniel V, (et al), “The MD6 Hash Function”, Computer Science and Artificial Intelligence Laboratory, October 24, 2008.
https://www.researchgate.net/publication/228524420_The_MD6_hash_function.

[20] R. Roshdy, M. Fouad, M. Aboul-Dahab, “Design and Implementation a New Security Hash Algorithm Based On Md5 and Sha-256”, International Journal of Engineering Sciences & Emerging Technologies, ISSN: 2231 – 6604, August 2013.

[21] C.Kumar, C. Suyambulingom, “Cryptographic of High Security Hash Functions”. International Journal of Engineering Research & Technology (IJERT), ISSN: 2278-0181, Vol. 1 Issue 3, 2012.

[22] H.Handschub, H.Gilbert, “Evaluation Report Security Level of Cryptography – SHA-256”, Technical Report, Issy-les-Moulineaux, January 2002.

[23] H. Choi, S. C. Seo, “Optimization of PBKDF2-HMAC-SHA256 and PBKDF2-HMAC-LSH256 in CPU Environments,” In Information Security Applications - 21st International Conference, WISA 2020, Jeju Island, South Korea, August 26-28, 2020, vol. 12583 of Lecture Notes in Computer Science, pp. 321–333, Springer, 2020.

[24] T. Thangavel, A. Krishna, “Efficient Secured Hash Based Password Authentication in Multiple Websites”, International Journal on Computer Science and Engineering, Vol. 02, No. 05, 2010, 1846-1851.

[25] K. Magnitude, R. Katti, “A Hash-Based Strong Password Authentication Protocol with User Anonymity”, International Journal of Network Security, Vol.2, No.3, PP.205–209, May 2006.

[26] X. Wang, D. Feng, X. Lai, H. Yu, “Collisions for Hash Functions MD4, MD5, HAVAL-128, and RIPEMD”, Jinan 250100, China: The School of Mathematics and System Science, Shandong University, 2004.

[27] L. Bosnjak, J. Sres, B. Brumen,” Brute-Force and Dictionary Attack on Hashed Real-World Passwords”, Conference: 2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics, DOI: 10.23919/MIPRO.2018.8400211, May 2018.

[28] K. Chanda, “Password Security: An Analysis of Password Strengths and Vulnerabilities”, I. J. Computer Network and Information Security, pp. 23-30, 2016.

[29] D. Matteo, M. Pietro, Y. Roudier, “Password Strength: An Empirical Analysis”, 2229, Route des Crêtes Sophia Antipolis, France, January 2010.
https://www.researchgate.net/publication/312444298_Password_strength_An_empirical_analysis.