Security of Passwords

Sirapat Boonkrong*

Abstract

Authentication has become a very important security mechanism. Recently, many attackers have been looking to attack organisations' password databases, since they are the sources which attackers can potentially used to gain unauthorised access to information, network and resources. Therefore, it is important to understand how to securely choose and store passwords in such a way that it can prevent attackers from learning what the users' passwords are. This paper discusses many aspects of password security, starting from a way to generate a secure password to ways to securely store a password. It will be explained that the most common method used to store passwords today, i.e., MD5 or SHA-1 hashing, may not be the best solution. Thus, better solutions in salting passwords and using slower hash functions are introduced.

Keywords: password security, password hashing, salted password, slow hash.

1. Introduction

Information and network security has increasingly become a topic of interest to many security professionals and researchers. It is a well known fact that the aim of security is to achieve confidentiality, integrity and availability [1]. Confidentiality means keeping information or data as a secret. Only those who are authorised should be able to access and read the data. Integrity means that we should have a means to check for the completeness and correctness of data. That is, there should be a mechanism to detect any unauthorised modifications of data that could occur. Availability is the ability to access data, network or resources whenever it is required. In other words, if and when an authorised person would like to have an access to his or her data, network or resources, he or she must be able to at that instant. As mentioned, these three security characteristics are the main goals of security. They are collectively known as the CIA model of security.

The CIA model of security has been around for a long time, stating the required security characteristics. However, as time has progressed, more security goals are required. One of them is authentication. Authentication is now considered a necessity for many applications and systems. Authentication can be described as a method for confirming the identity of a particular entity. Authentication can also be considered as an intrusion prevention system. It is the first line of defence that is used for protecting network and its resources so that only authorised persons can have an access. Authentication is actually a part of a security mechanism called access control, which refers to issues concerning access of system resources. There are four main parts to access control. They are identification, authentication, authorisation and accounting.

Identification is just identifying who each user is. This is, for example, done by issuing a username. Authentication is a process used for determining whether or not users are allowed access. In addition, authentication is a process used for confirming the identity of a user. That is, once a username is entered, the user must prove that this username really belongs to him or her. The next thing that usually comes after authentication is authorisation. Authorisation is a process that deals with restrictions and limitations on access. Authorisation tells what each user is and is not allowed to do or access. In other words, authorisation gives users their access rights to resources. The fourth part of access control is accounting. Accounting keeps records of who has entered the system, what that person has done and when. To put it simply, accounting is a process of keeping a log of the system.

The main focus of this paper will be on the authentication process, concentrating the uses of passwords and their security issues. As mentioned above, authentication is a process of confirming the identity of someone or something. In other words, authentication is proving that "you are really the person you claim to be." One very popular way to carry out authentication is by using a username and a password. Actually, authentication methods can be categorised into many groups, which will be explained in the later sections.

This paper will, therefore, be structured as follows. Section 2 gives an overview of available authentication methods. Section 3 discusses passwords, their security issues and how to choose a password. Ways to store passwords will be discussed in Section 4. Section 5 will conclude the paper.

* *Faculty of Information Technology, King Mongkut's University of Technology North Bangkok.*

2. Authentication Methods

As mentioned briefly in Section 1, there are actually four authentication methods [2] available today. Each of the four methods can be explained as follows.

2.1 Something-you-know method

The first authentication method is called the something-you-know method. It is a method which a user uses what he or she remembers to prove his or her identity. In other words, the user would use his or her knowledge to prove that he or she is really who he or she claims to be. A good example for this authentication method is a password or an ATM pin code.

2.2 Something-you-have method

The second authentication method is called the something-you-have method. This second method of authentication addresses something the user carries in their possession. This includes smart cards and authentication tokens. An authentication token is a small device with a display that presents a six-digit number used to support remote login authentication. A token can be either synchronous or asynchronous. A synchronous token is synchronised with a server. They use the time to generate a number that will be entered during the user login phase. When a user would like to log into a system, he or she enters a username together with the number displayed on the token. This number changes according to the current time. For an RSA SecurID Token [3], the number on the token changes every minute and is also synchronised with the authentication server.

On the other hand, asynchronous tokens use a challenge-and-response system. The authentication server and the token do not need to be synchronised. That is, when a user would like to log in, the server presents the user with a numerical sequence. The user enters this sequence into the token, which produces a response. The user then places this response into the system to gain access.

The main advantage of using an authentication token, either synchronous or asynchronous, is that the user's password changes every time he or she logs in. This is also known as one-time password, which reduces the risk of passwords being guessed by attackers.

2.3 Something-you-are method

The third method of authentication deals with a characteristic of a user. This is a process of using body measurements, known as biometrics. Popular biometrics includes fingerprint, palm print, hand geometry, facial recognition, iris pattern and retinal print. Out of all possible biometrics, it has been said that only three are considered truly unique. They are fingerprints, retina of the eye and iris of the eye.

Biometrics work by comparing the users actual characteristic with the stored data in order to determine if the user is really he or she is claiming to be. A problem with this method is that human characteristics can change over time due to normal development, illness or injury. Therefore, fallback or failsafe authentication mechanisms [4] should be created just in case

2.4 Something-you-produce method

The fourth area of authentication is something that a user performs or produces. The two popular methods that fall into this category are signature and voice recognition. With signature recognition, a user signs a digital pad with a stylus that captures the signature. The signature is then saved and compared with a signature on a database for validation.

Voice recognition works in a similar way in that a user speaks a phrase into a microphone. The voice is captured and stored on a database. Later, when authenticating himself or herself, the user speaks the same phrase so that it can be compared with the voice stored on the database.

We now have seen the four authentication methods. The question is which one should we really use? It has been claimed that in order to achieve stronger authentication, hence better security, two authentication methods should be used alongside one another. This is known as two-factor authentication [5]. That is, we could use a smart card together with a password, which would be something you have with something you know. An example is when a user withdraws money from a cashpoint, he or she would have to enter his or her pin code (something-you-know method) as well as inserting a credit card (something-you-have method). Even though two-factor authentication stronger than using just one single method, some researchers [6] still say that it is not enough.

Having said that all four authentication methods exist today, only the first method, specifically passwords, will be considered in this paper

3. Passwords

Password is the most popular authentication method in computing today. That is why we feel that there is a need to discuss passwords in detail.

Before going any further, let us briefly establish here what an ideal password should be. On the whole, it has been suggested that an ideal password should be something that a user can remember, something that a computer can verify, and something that nobody else can guess. This sounds easy, but is difficult to achieve.

The problem with passwords nowadays is that people tend to choose "bad" passwords. These are the passwords that are easy to "crack." What is the solution to this? One solution is to use randomly generated cryptographic keys. This would make the work of cracking a password equivalent to the work of a brute force or exhaustive key search. Does using randomly generated cryptographic keys for passwords sound plausible? Let us compare keys and passwords.

3.1 Key vs Passwords

Suppose an attacker Trudy is confronted with a 64-bit cryptographic key. That means there are possible 2^{64} keys in total, thus on average Trudy must try 2^{63} keys before finding the correct key.

This time, suppose Trudy is confronted with an 8-character password. Each character is 8 bits long, which means there are 256 possible choices for each character. Therefore, the total number of possible passwords is $256^8 = 2^{64}$ passwords. The number appears to be equivalent to the exhaustive key search problem. But . . . is it really?

One issue with passwords is that users do not select passwords at random. The reason is because they have to remember it. For example, users are more likely to choose an 8-character password as password, rather than something random like s9@KOpwA. The implication of this is that clever attackers would make far fewer than 2^{63} guesses before getting the correct password. In other words, the actual number of passwords is far fewer than the number of keys of the same size. The majority of randomly generated passwords are not taken into account since they are not used anyway. We could, therefore, claim that the non-randomness of passwords reduces the amount of work carried out by attackers to crack a password, and is also at the root of many of the most serious problems with passwords.

3.2 Choosing a password

It has been mentioned that weak or bad passwords create problems with security. Examples of bad passwords include Somchai, Doraemon, 08111979 and JohnSmith. The first password sample, Somchai, is just a name of a user. This would be a very easy guess. The second password, Doraemon, would also be easy to guess if anyone knows that the user is a fan of Doraemon. The third is just the user's date of birth, and the fourth contains a first name and a surname of the user. It is clear that all of these examples are weak passwords, because they are not difficult for attackers or anyone to guess.

Authentication can be thought of as the first line of defence of a network or a system. That means security can be said to rest on passwords, an authentication method. Therefore, passwords should be difficult to crack and easy for users to remember. Examples of a better password include sHJiLJ-M50Emim, 876261400154, D0raem0n and MtFbwY. Let us analyse each one in turn to see whether it fits our criteria for good passwords: easy to remember and difficult to guess.

The first password, sHJiLJM50Emim, appears to be random, which makes it very difficult to guess. However, it is not easy to remember. The second password, 876261400154, consists of twelve digits. This seems difficult to guess, but also difficult to remember. It has been documented that well-trained military personnel are only able to memorise up to twelve digits. That means for regular users, it is near impossible to memorise that many random digits. The third password, D0raem0n, looks a good password due to the mixture of letters and numbers. However, this may not be the case, because if anyone knows that the user is a fan of Doraemon, he or she could try to make a guess. The fourth example, MtFbwY, is difficult to guess. It is also very easy to remember, even though the password appears to be random. This fourth example of password is made by a password creation method known as a passphrase [7].

A passphrase is a series of characters derived from a set of words or a sentence. One way to generate a passphrase is that a user thinks of his or her favourite sentence, then takes the first letter of each word and put them together. For example, a user's favourite sentence might be "May the Force be with You". Taking the first letter of each word, the passphrase formed from this sentence would be MtFbwY. Users do not actually have to take the first letter of each word. Any letter can be used, but taking the first letters would be the easiest to remember.

Passphrases are said to be the source of the better passwords that should be used. The following widely published [2] and well known password experiment confirms this claim. The experiment divides people into three groups. Group A select passwords consisting of at least six characters, with at least one non-letter. Group B select passwords based on passphrases. Group C select passwords consisting of eight randomly selected characters. The aim of the experiment is for the experimenters to crack those chosen passwords, and the results are as follows. In Group A, about 30% of passwords are easy to crack, and users find their passwords easy to remember. In Group B, about 10% of the passwords are cracked, and users find their passwords easy to remember. Group C, about 10% of the passwords are cracked, but users find their passwords difficult to remember. From the password experiment, it is clear to see that passphrases provide the best option for passwords. This is because they are difficult to crack, yet easy to remember.

4. Storing Passwords

In the past few months at the time of writing this paper, a number of high profile companies had seen their passwords leaked to the online public even though a lot of efforts had been put into protecting them. Unfortunately, disclosure of password databases is one of the main aims of hackers' community. Therefore, it is important to understand how passwords can be stored, and what each storing method means for the security of passwords. Let us go through and analyse each method in turn.

4.1 Plaintext passwords

The most basic way that a password can be stored is in plaintext. This means that in a password file or a password database, usernames and passwords are stored in a human-readable form. That is, if a password if testpassword, it is also stored in a database as testpassword. When a user enters his or her username and password, the system checks them against the database to see if they match.

This is the worst possible method for storing passwords, in security context. Most reputable systems and Web sites do not store passwords in their plaintext form. This is because if the password database is obtained by an attacker, everybody's password is immediately accessed, known and compromised.

4.2 Encrypted passwords

In order to reduce the risk of passwords being exposed as plaintext, some Web sites have adopted encryption as their solution. Encryption, as a reminder, uses a secret key to transform a plaintext password into a random string of text. This means that if an adversary were to get hold of a password database, he or she would not be able to see what the real passwords are. Only passwords in ciphertext form would be seen. The adversary would need to have the secret key to decrypt them. This does not sound so bad, does it?

The problem with this method is that the secret key is often stored on the same machine or server that passwords are. What happens if the server gets hacked? The hacker would not have to do much work to obtain the secret key, which would allow him or her to decrypt all the passwords. This implies that this method is considered insecure.

4.3 Hashed passwords

Hashing is similar to encryption in the sense that it transforms a password into a random string of letters and numbers. However, as we already know, hashing, such as MD5 [8] and SHA-1 [9] ensures at least two things. Firstly, it is infeasible to do the reverse of a hash. We cannot take the hashed password and run the hashing algorithm backwards to get the original password. Secondly, it is very improbable that two different passwords will produce the same output of random string of letters and numbers.

Hashing is the most widely used method for storing passwords both on a local network and on the Internet. A hash function works by taking a password as its input, and scrambling it to produce a seemingly random result. Two popular hash functions used for storing passwords are MD5 and SHA-1. For example, the MD5 hash of the password “password” is 5f4dcc3b5aa765d61d8327deb882cf99. Using another hash function in SHA-1, the hash value of the same password would be 5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8.

Hash functions are often used in password systems because instead of storing passwords in clear text, only the hash values of the passwords are stored. Since it is almost impossible to reverse the hash, the hashing method seems to be a secure way of keeping passwords. When a user logs into a system, the hash value of the entered password is computed and compared with the hash in the password database. If they match, the password is correct. If not, the user will have to give another attempt. Moreover, since it is said that the hashes are unique, it is very likely that the user will only be able to log in with the correct password.

Unfortunately, over the last months at the time of writing, a number of Internet companies found that their passwords databases had been cracked even though their passwords had been hashed. The reason that this had occurred is because there is a downside to this method. That is, although an adversary cannot reverse a hash value back to its original password, he or she can try many different passwords until one matches the required hash.

To go into a little bit more detail, the problem with simple hash functions is that it is possible for hackers to pre-compute all hashes of all possible passwords, and stored them in databases. These databases of pre-computed hash values are known as rainbow tables [10]. If a password database leaks, with the help of rainbow tables (which is essentially a list of billions of different hashes and their matching passwords), hackers can just look up the hashes in the rainbow table. To see this, just try typing 5f4dcc3b5aa765d61d8327deb882cf99 into Google. It will quickly be seen that it is the MD5 hash for “password”.

If hashes are not found in rainbow tables, it means that they are the hashes for long and complex passwords that have not been pre-computed. This is one reason why picking a long and complex password is a good idea, since hackers will not have had it already pre-computed.

4.4 Salted passwords

A way to ease the concerns of rainbow tables is with something called salt. Salting a hash [11] means adding a random string of letters and numbers, called a salt, to the beginning or end of a password before hashing it. In other words, instead of just hashing a password H(password), we compute H(password; salt). In this method, a different salt is used for each password. Even if the salts are stored on the same server as passwords, it will still be very difficult to find salted hash values in the rainbow tables.

Let us look at an example of the salting method. Suppose a password, “password”, is chosen by a user, together with a random salt value “fh90$$PA28”. Therefore, instead of computing the hash value of “password”, H(password), H(passwordfh90$$PA28) is calculated to obtain the hash value 29e20a65f0d0336463b0391174ac74b3 for MD5 or the hash value b6ac9606d892f5af75ec9ceb39435a1e5f567dcf for SHA-1. These values could be put into Google and no results would be found.

Since each user has a different salt applied to his or her password, rainbow tables are practically useless. This is because it is not possible to pre-compute the hashes of all possible passwords with all possible salt values.

Unfortunately, it has to be said that password cracking techniques benefit from two things: the speed of hash functions and Moore’s Law. First of all, originally hash functions, MD5 or SHA-1, were not designed to protect passwords. As we have learned, they were design for checking the integrity of messages by detecting modifications. For this reason, hash functions were designed to be very fast.

Secondly, Moore’s Law states that processor speeds and overall processing power will double every two years. This means that as computers have increased in speed and with the speed of hash functions, it has made it possible for rainbow tables to start attacking passwords, even when salted. In other words, hackers can now compute hash values of millions of passwords per second, applying different salts with each password. This, in turn, makes the rainbow tables larger. Hence, hacking with rainbow tables has become more possible.

The solution to this problem is to use a hash function that is slow. If the hash function itself is slow, the process of “enlarging” rainbow tables will also be slowed down.

4.5 Slow hashes

Fortunately, there are hashes that are a lot less speedy than MD5 and SHA-1, the property stated above. Experts are now pointing to slower hashes as a better option for storing passwords. Hash functions like MD5 and SHA-1 are fast, as mentioned. That is, if we type in a password, it will return the hash values quite quickly. In brute force attack, time is a very important factor. By using a slower hash function, brute force attacks will take much longer, since each password takes more time to compute.

One such hash function is known as bcrypt. bcrypt [12] is a hash function based on an encryption algorithm called Blowfish. bcrypt is just like a normal hash function with an additional parameter. bcrypt takes a password, a random salt value as well as a cost as its arguments. The cost tells the hash function how hard to work in computing the hash value. Hence, it determines how long the calculation will take. The idea of bcrypt is actually quite simple. bcrypt makes sure that it always takes the same amount of time to hash the same password, regardless of how powerful the hardware used to compute the hash is. To re-emphasise, bcrypt allows users to specify the work factor used to generate the hash. It means that if somebody changed the value of the work factor, the hash value of the password would be different.

Another and newer algorithm is known as scrypt. scrypt [13] is actually a password-based key derivation function. scrypt was purposefully designed to be very computational expensive and intensive, so that it takes a long time to compute. The main idea of scrypt is that it has a very large memory requirement that comes from large pseudorandom bit strings generated as a part of the algorithm. These large strings, although not generated at the same time, require the use of a lot of memory space. It appears that scrypt is the algorithm that concerns with the memory usage. There is a well known computing tradeoff namely: more memory is needed if spped is to be increased, or reducing the memory space at the cost of the speed of the computation. This is the reason why scrypt is another suitable way to store passwords on any system.

It has been stated in [14] that on modern hardware, the cost of cracking a password that uses scrypt to hash is approximately 100 billion times more than the cost of cracking the same password using a normal hash function. Moreover, when running scrypt, users are able to specify such parameters as the maximum amount of time or the maximum amount of memory used to execute scrypt algorithm. scrypt is available for download at [14].

Even though scrypt's RFC document is still in the draft version, there has been an attempt to compare the cost of cracking a password (in one year) that is stored by using the methods explained in this paper. The author of [15] explained that the results are estimated costs for building a password-cracking machine based on a 130nm semiconductor process. The results are taken from [15] and are shown in Table 1 as follows.

Table 1 *Estimated Cost of Hardware to Crack One Password in One Year.*

Technique/ Password Size	6 Lower Case Letters	8 Lower Case Letters	8 Characters	10 Characters
MD5	< $1	< $1	< $1	$1.1K
Salted MD5	< $1	< $1	$130	$1.1M
bcrypt	< $1	$4	$130K	$1.2B
scrypt	< $1	$150	$4.8M	$43B

A main advantage of bcrypt and scrypt is that over time, the cost can be increased to keep pace with faster and faster computers, and keep passwords safe by making the hash function slower and slower.

5. Conclusion

Computer networks, specifically the Internet, have provided us with many benefits. Unfortunately, with those benefits, come threats that could cause many damages to our assets. One way to prevent those threats is to use authentication. Authentication is an intrusion prevention mechanism used to confirm an identity of an entity. Many methods of authentication are presented here in this paper. One has been a focus, in particular. That method is the use of passwords.

Passwords are the most commonly used authentication mechanism due to convenience. This paper has discussed how to choose a password that is easy to remember, but difficult to guess. A password by the means of passphrase is considered the best way to match those criteria.

Ways to store passwords were the next to be discussed. There are many methods that we can use to store passwords. They include storing them as plaintext, encrypting them, hashing them, hashing them with a salt value and hashing them using slow hash algorithms. It has been explained that some of them are more secure than the others. In other words, applying slower hash functions appears to be a better method, at least in the near future. However, just like any other aspects of security, methods for storing passwords will have to be reviewed from time to time. In addition, human factor plays an important part. Users can help by increasing the complexity of their password to ensure a longer and more complex password.

6. References

[1] Martin Abadi, "Security Protocols and Specifications," *In Foundations of Software Science and Computation Structures: Second International Conference, FOSSACS'99*, Vol. 1578, pp. 1-13, Springer-Verlag, Berlin Germany, 1999.

[2] Mark Stamp, *Information Security: Principles and Practice*, Second Edition, John Wiley & Sons Inc., New Jersey, 2011.

[3] Benjamin Halpert, "Mobile Device Security," *In Proceedings of the 1st annual conference on Information security curriculum development*, *InfoSecCD'04*, pp. 99-101, ACM, 2004.

[4] Ariel Rabkin, "Personal knowledge questions for fallback authentication: security questions in the era of Facebook," *In Proceedings of the 4th symposium on Usable privacy and security*, *SOUPS'08*, pp. 13-23, ACM, 2008.

[5] David Coffin, "Two-Factor Authentication," *Expert Oracle and Java Security*, pp. 177-208, Apress, Springer Link, 2011.

[6] Bruce Schneier, "Two-Factor Authentication: too little, too late," *Communications of the ACM*, Vol 48, No. 4, pp. 27, April, 2005.

[7] Sigmund N. Porter, "A password extension for improved human factor," Computer & Security, Vol.1, Iss. 1, pp. 54-56, January, 1982.

[8] Ron Rivest, "The MD5 Message Digest Algorithm," RFC1321, Internet Engineering Task Force, April, 1992.

[9] Federal Information Processing Standards Publication 180-1, "Secure Hash Standard," the National Institute of Standards and Technology, April, 1995.

[10] Antony G. Robertiello and Kiran A. Bandla, "Attacks on MD5 Hashed Passwords," Technical Report, George Mason University, USA, 2005.

[11] Robert Morris and Ken Thompson, "Password security: a case history," *Communications of the ACM*, Vol. 22, Iss.11, pp. 594-597, November, 1979.

[12] Niels Provos and David Mazières, "A Future-Adaptable Password Scheme," *In Proceedings of 1999 USENIX Annual Technical Conference*, pp. 81-92, 1999.

[13] C. Percival and S. Josefsson, "The scrypt Password-Based Key Derivation Function," Internet Draft, Internet Engineering Task Force, September, 2012.

[14] Scrypt: Technical Details, Available online at http://www.tarsnap.com/scrypt.html: Retrieved on 26th December, 2012.

[15] Colin Percival, "scrypt: A new key derivative function," *The Technical BSD Conference*, *BSDCan 2009*, May, 2009.