Proceedings of the 2nd International Symposium on Computer, Communication, Control and Automation (ISCCCA-13)

**密码存储中MD5算法的安全性分析**

Mary Cindy Ah Kioon, ZhaoShun Wang and Shubra Deb Das

School of Computer and Communication Engineering

University of Science and Technology (USTB), Beijing

Beijing 100083, China

email: cindyak86@gmail.com , zhswang@sohu.com and shubra\_ustb@yahoo.com

***摘要 - 哈希算法通常用于转换***

***密码进入散列理论上是不可能的***

***解密 本文分析了安全隐患***

***哈希算法MD5密码存储和讨论***

***不同的解决方案，如盐和迭代散列。 我们***

***提出了一种在密码存储中使用MD5的新方法***

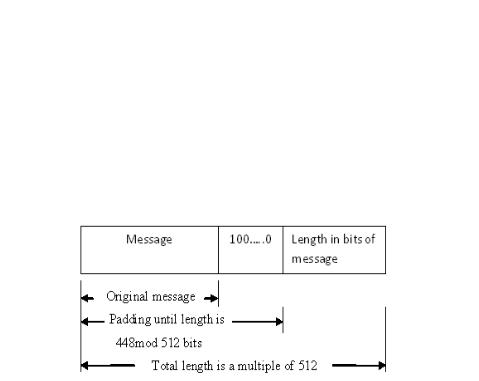
***使用外部信息，计算盐和随机密钥***

***在MD5计算之前加密密码。 我们***

***建议使用按键拉伸进行哈希计算***

***较慢并使用XOR密码来制作最终的哈希值***

***不可能在任何标准的彩虹表中找到。***



MD5（消息摘要算法5）由...设计

Ron Rivest于1991年。MD5处理可变长度

消息转换成128位的固定长度输出。 MD5是

流行散列函数。 它适用于512位的块，

每个进程通过4轮进行，每轮进场

处理16个子块（每个32位）。 512位

消息在处理之前分为16个子块。 如果一个

消息块不高达512位，如图所示填充

图。 可以找到该算法的详细说明

在[1]。

***关键词组件; MD5; 密码存储安全;***

***数据安全; 数据攻击 rainbow表***

I.

INTRODUCTION

With the advent of computer technology, it became more

productive to store information in databases instead of

storing in paper documents. Web applications, needing user

authentication, typically validate the input passwords by

comparing them to the real passwords, which are commonly

stored in the company’s private databases. If the database

and hence these passwords were to become compromised,

the attackers would have unlimited access to these users’

personal. Nowadays, databases use a hash algorithm to

secure the stored passwords but there are still security

breaches. Recently in 2012, Russian hackers released a big

list of cracked passwords from the well-known social

networking sites including LinkedIn. These attacks were

found to be successful due to the use of a weak hashing

algorithm.

Figure 1. Length of message after padding (in bits)

III.

APPLICATION OF MD5 ALGORITHM IN PASSWORD

STORAGE SECURITY

It is highly insecure to store passwords in plaintext in the

database. In order to increase the security of passwords,

MD5 algorithms can be used to hash the original passwords

and the hash values, instead of the plaintext are stored in the

database. During authentication, the input password is also

hashed by MD5 in a similar way, and the result hash value is

compared with the hash value in the database for that

particular user.

II.

HASH FUNCTION

IV.

SECURITY ANALYSIS OF MD5

A hash function is a one-way encryption function that

takes a variable-size input plaintext m and generates a fixed-

size hash output. It is computationally hard to decipher the

hash and any attempt to crack it is practically infeasible. A

“secure” hash function should be able to resist pre-image

attacks and collision attacks. Due to the pigeonhole principle

and birthday paradox, there will be some inputs that will

produce the same hash result. The output length is of fixed

size 128 bits, making a total of 2128 possible output hash

values. This value, as big as it may seem, is not infinite,

hence resulting in collisions.

MD5 algorithm is prone to two main types of attack:

dictionary attacks and rainbow tables.

*A.*

*Dictionary Attacks*

In dictionary attacks, an attacker tries all the possible

passwords in an exhaustive list called a dictionary. The

attacker hashes each password from the dictionary and

performs a binary search on the compromised hashed

passwords. This method can be made much quicker by pre-

computing the hash values of these possible passwords and

storing them in a hash table.

*A.*

*MD5 algorithm*

*B.*

*Rainbow Tables*

Rainbow tables are made up of hash chains and are more

Published by Atlantis Press, Paris, France.

© the authors, 2013

0706



Proceedings of the 2nd International Symposium on Computer, Communication, Control and Automation (ISCCCA-13)

efficient than hash tables as they optimize the storage

requirements, although the lookup is made slightly slower.

Rainbow tables differ from hash tables in that they are

created using both reduction and hash functions. Reduction

functions convert a hash value to a plaintext. The plaintext is

not the original plaintext from which the hash value was

generated, but another one. By alternating the hash function

with the reduction function, chains of alternating passwords

and hash values are formed. Only the first (chain’s start point)

and last plaintext (chain’s end point) generated are stored in

the table. To decipher a hashed password, we first process

the hashed password through reduction functions until we

find a match to a chain’s end point. We then take that chain’s

corresponding start point and regenerate the hash chain and

find the original plaintext to the hashed password. Rainbow

tables are very easily available online now. There are many

password cracking systems and websites that use rainbow

tables also, for example, OphCrack. Of course, using

rainbow tables do not guarantee a 100% success rate of

cannot be used to decipher the salted hashes. However, two

same passwords will still produce the same hash.

*2)*

*Different random salt for each password and storing*

*the salt within the database itself:* If we use different salts

for each password, two same passwords will have different

hashes. The attacker has to generate different rainbow tables

for each individual user, making it computationally harder

for an attacker to crack the hashes. These salts can be stored

in plaintext in the database. The purpose of the salt is not to

be secret, but to be random enough to defeat the use of

rainbow tables.

*3)*

*Use an existing column value:* An existing column

value like username can be used as salt. This solution is

similar to the second solution discussed above. It defeats the

use of a standard rainbow table, but a hacker might generate

a rainbow table for a specific username, for example, “root”

or “admin”.

*4)*

*Use a variably located calculated salt****:*** The salt

cracking

password systems. However, the bigger the

location can be prefix (salt appended in front of password),

infix (salt appended within the password) or postfix (salt

appended at the end of the password). If the salt’s location is

made random, then cracking the passwords is made harder.

For example, we can set the salt location to be equal to the

password’s length modulo 3. The salt can be calculated by

using a random character sequence (stored in the database)

and using other characters (embedded within the code). For

example, the salt can be made to be a combination of the first

two letters of username, random sequence of characters and

the last 2 letters of username.

character set used for creating the rainbow table and the

longer the hash chain length, the bigger will the rainbow

table be.

V.

COUNTERMEASURES RESEARCH

*A.*

*Information Entropy*

Password strength is usually measured in terms of

information entropy. In simple terms, the higher the

information entropy, the stronger the password and hence the

more difficult it is to crack it. A password of 6 characters

would require only 26 attempts to exhaust all possibilities in

a brute-force attack, while a password with 42 characters

would need 242 attempts. As can be seen, the longer the

password and the larger the character set from which it is

derived, the stronger the password. As best practice and

preliminary requirement, the application should insist that

the user uses a strong password during the registration

process. Strong passwords run less risk of existing in

dictionaries. Common simple passwords like “123456” have

already been banned by Microsoft Hotmail.

*5) Use a variably located calculated salt including*

*information outside the database and the application code:*

The hacker now has to break into the database and the server

containing the application code. He also needs to obtain the

additional information needed to crack the password.

*C.*

*Improvement on MD5 processing*

The following methods can be used to improve the MD5

processing:

*1)*

function is altered, for example using one of the following

functions as shown in (1), (2) and (3):

*Improved hash function:* The hash computation

*B.*

*Salting*

One of the most common reasons to successful password

cracking attacks like the one against LinkedIn was because

they were using unsalted hashes. This makes it much easier

for hackers to crack the system by using rainbow tables,

especially given the fact that many users use very common,

simple passwords and these similar passwords result in

similar hashes. A salt is a secondary piece of information

made of a string of characters which are appended to the

plaintext and then hashed. Salting makes passwords more

resistant to rainbow tables as the salted hashed password will

have higher information entropy and hence much less likely

to exist in pre-computed rainbow tables. Typically, a salt

should be at least 48 bits. Salting can be implemented using

the following ways:

hash = Hash (password + salt)

(1)

(2)

(3)

hash = Hash (Hash (password) + salt)

hash = Hash (password + salt + key)

*2)*

*Iterative hashing*: The password is hashed a number

of times. MD5 is a fast hashing function, that is, it is

computationally fast to calculate. Iterative hashing makes the

calculation slower, hence computationally slower and more

*1)*

*Single salt for all passwords:* Given that the salt is

sufficiently long and complex, a standard rainbow table,

Published by Atlantis Press, Paris, France.

© the authors, 2013

0707

Proceedings of the 2nd International Symposium on Computer, Communication, Control and Automation (ISCCCA-13)

difficult to crack. The number of iterations can typically be

made to be equal to 1000.

*3)*

*Key stretching*: This makes a password more resistant

to pre-computation attacks by making the attack workload

bigger. Iterative hashing is used, where a weak key is fed

into the hash algorithm and the output results in a stronger

key. There are 3 key stretching methods depending on the

input used for the iterative hashing:

*a) Simple Key stretching:* Only the key is hashed

iteratively, as in (4). No salt is involved.

Figure 2. Generate complex password through columnar transposition

Fourthly, an additional random information string of 128

bits is generated for each user and stored in an external file,

e.g. in a flash drive.

key = Hash (key)

(4)

*b) Password Key stretching:* The password along with

the key are both used in the loop.

*c) Salted Key Stretching*: The key, password and salt

are used in the loop. This method is the best of the three key

stretching methods.

*Transform the password before hashing*: Before

calculating the MD5 hash for the password, the latter is

transformed using a simple cipher method.

Finally, the password is hashed using a formula based on

key stretching. The hashing process is similar to a cipher

block chaining method, where the output of one round is

used in the input of the other round, as shown in Fig. 3. By

calculating the XOR result of the hash value at one round

with the one at the previous round, the resulting hashed value

is made impossible to find in any standard rainbow table.

The final hashed password is then stored in the database.

The system authenticates a user by calculating the hash value

(the random key is retrieved from the database for use),

which is then compared to the stored hashed password. An

example of how the hashed passwords will appear in the

database is shown in Fig. 4.

*4)*

*5)*

*Chaining method and XOR(Exclusive OR) cipher:*  If

iterative hashing is used, the hash output from the current

iteration is used in the input for the next iteration. We use a

simple XOR cipher to compute the final hash by “XORing”

the hash output value from all iterations. A simple XOR

cipher is typically of the form shown in (5). If the key is

finalHash = HV0 ^ HV1 ^ ……^ HVN ,

made random enough,

impossible to crack.

the

ciphertext

will

be almost

HV0 = Hash (CpxPassword, additionalInfo);

HV1 = Hash (CpxPassword, HV0, salt);

HVN= Hash (CpxPassword, HVN-1, salt);

N is the number of iterations and ^ is XOR.

HV: Hash Value and

plaintext XOR key = ciphertext

(5)

*D.*

*Example of an improved MD5 processing*

We will now demonstrate how we can hash passwords in

databases using an improved version of MD5. There are five

main steps involved.

CpxPassword: Complex Password

Figure 3. Example of hashed passwords in a database using the improved

MD5

First, a random key string of random length is first

generated. Its character set is {0-9, a-z, A-Z}.This random

key string is used to generate the complex password and is

also stored in the database for later use during password

authentication.

Secondly, the password is transformed into a complex

password through columnar transposition cipher. Assuming

that the random key is “YDCiA” and the password is

“crazyrichard”, the password is first converted into a matrix

of 5 columns (same as length of random key) and the blank

cells are alternately filled with “\*” and “@”, as shown in

Fig.2. Using columnar transposition cipher, the complex

password generated is “ya\*ac\*ridcrrzh@”.

Thirdly, the salt is calculated by finding the XOR value

of the random key string with the complex password, row by

row. In our example, the salt is " %i\b\"s6\*\r\"".

Figure 4. Example of hashed passwords in a database using the improved

MD5

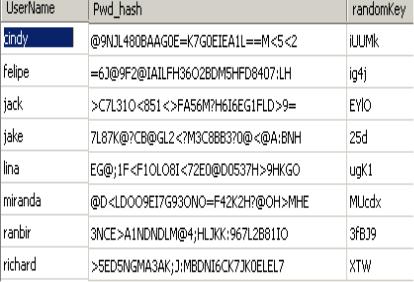
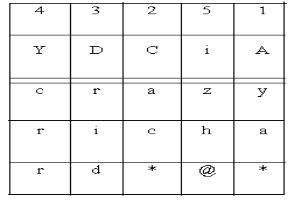
The overall algorithm can be summarized in Fig. 5. The

initialization vector used here is the additional information,

Published by Atlantis Press, Paris, France.

© the authors, 2013

0708



Proceedings of the 2nd International Symposium on Computer, Communication, Control and Automation (ISCCCA-13)

which is a random string of 128 bits. Each user has a

different initialization vector value.

initialization vectors and salt are different for each user. For

two users with the same passwords, the final hash value will

be completely different.

VII.

CONCLUSION

Password storage security is one important aspect of data

security as most systems nowadays require an authentication

method using passwords. Hashing algorithms such as MD5

are commonly used for encrypting plaintext passwords into

strings that theoretically cannot be deciphered by hackers

due to their one-way encryption feature. However, with time,

attacks became possible through the use of dictionary tables

and rainbow tables. In this paper, we discussed different

methods to thwart these attacks: (1) the use of a strong

password to reduce the probability of it existing in a

dictionary, (2) using salts, (3) key stretching and iteration

hashing to make the MD5 computation slower, (4) chaining

method, where the output of one iteration is used in the input

of the next iteration and the use of a different initialization

vector for each password, (5) encrypting the password before

hashing and (6) XOR cipher to make the final hash value

impossible to find in any rainbow table. An implementation

of the proposed approach is carried out using C# as

programming language and Microsoft SQL Server as

database. With our proposed approach, the attacker will now

have to hack into the database, the server containing the

application code as well as the external file.

Figure 5. Improved MD5 processing (IV: Initialization Vector)

VI.

PERFORMANCE ANALYSIS

First of all, an attack using a standard rainbow table

would fail because the hashed password stored in the

database is not of hexadecimal form and hence would not

exist in any standard rainbow table. We tried a few online

MD5 decryption tools like http://www.md5decrypter.co.uk/

and downloaded software tools like Cain and Abel. But,

since all of them use rainbow tables where the MD5 hashes

are in hex form and our stored hash value is in ASCII, the

attacks would already fail from the beginning, as shown in

Fig. 6.

Also, by XORing the output hash values from each

iteration makes it almost impossible to find out the original

hash output at the first round. Generally, if we XOR the

plaintext with a key to calculate the cipher text, we can get

back the plaintext simply by using XORing the cipher text

with the key. However, if the key is random, then it is almost

impossible to get back the plaintext. In our example, the key

is totally random as we are hashing intermediate hash output

values and hence given the final hash value, it is impossible

to decipher it**.**

ACKNOWLEDGMENT

The work reported in this paper was supported by

the Beijing Natural Science Foundation of China (Grant

No. 4112037).

The authors would like to thank and acknowledge the

support and assistance of relatives, friends and colleagues.

REFERENCES

[1]

[2]

Rivest, R. The MD5 message-digest algorithm. RFC 1321, 37 (April

1992).

Zhang Shaolan, Xing Guobo, Yang Yixian, Improvement and

Security Analysis on MD5 [J]. Computer Application, 2009, vol.

29(4):947-949.

[3]

[4]

Xiaoling Zheng, JiDong Jin, Research for the Application and Safety

of MD5 Algorithm in Password Authentication, 9th International

Conference on Fuzzy Systems and Knowledge Discovery, 2012.

H. Mirvaziri, Kasmiran Jumari, Mahamod Ismail, Z. Mohd Hanapi, A

new Hash Function Based on Combination of Existing Digest

Algorithms , The 5th Student Conference on Research and

Development – SCOReD 2007, 11-12 December 2007, Malaysia.

[5]

Md. Didarul Alam Chawdhury, and A.H.M. Ashfak Habib, Security

Enhancement of MD5 Hashed Passwords by using the Unused Bits of

TCP Header, Proceedings of 11th International Conference on

Computer and Information Technology (ICCIT 2008) 25-27

December, 2008, Khulna, Bangladesh

Figure 6. Invalid character in the MD5 Hash in Cain and Abel

Also, even if two users have two same passwords, the

random key used to encrypt the passwords will be different,

resulting in different complex passwords. Furthermore, the

Published by Atlantis Press, Paris, France.

© the authors, 2013

0709

