|  |
| --- |
|  |
| Password Hashing |
|  |
|  |
| **Dan Johnson** |
| **4/15/2012** |

|  |
| --- |
| This document contains a brief introduction to hashes, a brief comparison of various hash functions and methods to create more secure hashes. Also, it leads a discussion into how to use hashing technology to protect sensitive data, like passwords, and examples of system breaches where password hashing could have helped protect user data. |

Hackers and other cyber criminals compromise personal and corporate databases daily. Often, this includes extracting personal information and login information including usernames and passwords from the unauthorized database. Losing these passwords forces companies to lock many user accounts until the user resets his or her password. This hampers customer satisfaction and can lead to many customers leaving for a different service. There are many ways to harden a server to prevent database and other system breaches from occurring, but there are also ways to secure data after a breach occurs. One way to protect some user data is through hashing that data.

The best way to describe a hash is that it is a fingerprint of some piece of data. Hashes are typically generated by applying a mathematical algorithm to a piece of input data and generating a fixed length string as its output. Hashes are sometimes referred to as digests or checksums. Hashing functions are used for a variety of tasks, from creating message digests of passwords in a database to creating checksums of large files to ensure that the maintain their integrity after being transferred. (McGlinn, 2005)

All cryptographic hashing functions share several similar properties. The first of these properties is for any given output, it is impossible to determine the original input. It is also infeasible (but not impossible) to determine two inputs that give the same output. It is infeasible to change an input and still receive the same output. The last property is that it is easy to compute a hash for any given input, that is, it does not require large amounts system resources to calculate a hash for a given input. If an algorithm satisfies these requirements, it may be a candidate to be a hashing function. All cryptographic hashing functions undergo extensive testing and scrutiny from the security community and many hashing algorithms come from the results of research by government agencies. (Silva, 2003)

There are several popular hashing functions each with it’s owns merits. Hashing functions are compared based their cryptographic strength. Cryptographic strength of a hashing function can be defined and evaluated in a number of ways, but is generally a combination of the length of the generated hash measured in bits, and the ability of the hashing function to resist cryptanalytic attacks. There are several types of cryptographic attacks including preimage attacks and collision attacks.

There are two types of preimage attacks that are used against cryptographic hash functions. Both of these attacks seek to break the one-way property and are similar in nature. Hoffman and Schneier describe these two types as follows:

Attacks against the "one-way" property:

* A "first-preimage attack" allows an attacker who knows a desired hash value to find a message that results in that value in fewer than 2^L attempts.
* A "second-preimage attack" allows an attacker who has a desired message M1 to find another message M2 that has the same hash value fewer than 2^L attempts.

(L is the number of bits in the output hash.) Both attacks attempt to find a message in less time than it would take a brute force attack to generate. The first-preimage attack seeks to find any message that will result in the given hash. The second-preimage attack attempts to discover another message the will result in the same output hash as another given message. Attacks can find one type of preimage can often times find the other type of preimage. (Schneier & Hoffman, 2005)

A hash collision occurs when two pieces of input data produce the same output hash. This is due to the length of a hash being a fixed amount and the existence of an infinite number of inputs. A collision attack, or birthday attack, uses a property of probability theory called the birthday paradox. The birthday paradox shows that if we select twenty-three people at random, there is a fifty percent chance that two of those people will share the same birthday. By examining this property in detail, cryptographic experts have shown that several hashing functions can be broken in less time than a brute force attack. A collision attack seeks to find two inputs that produce the same output in less than 2^L/2 attempts where L is the number of bits in the output hash. (Schneier & Hoffman, 2005)

There are several popular hashing algorithms. This paper will discuss three of these algorithms in minor detail, highlighting the strength of weakness of each algorithm without diving to deeply into the math behind them. All of the cryptographic hash function discussed are widely popular and have been scrutinized by the cryptographic community in some detail. The first algorithm that will be discussed is the MD5 hashing function.

Ronald Rivest created the MD5 message digest in 1994 when the cryptographic community determined that MD4, its predecessor, insecure. MD5 produces a 16-byte, or 128-bit, hash which is expressed as a 32bit hexadecimal number. MD5 is considered to not be collision resistant and several attacks against MD5 have developed since its inception. Because of this lack of collision resistance, MD5 is not considered a good hashing function for the purpose of creating a one-way hash for security purposes. MD5 is now mostly used to create checksums of large files to ensure data integrity after transfer or download. (Stallings, Brown and Howard, 2008, p.631)

The next function we will discuss is SHA-1. SHA-1 was created by the National Security Agency in the mid 1990’s. SHA stands for Secure Hash Algorithm. SHA-1 is proven to be cryptographically much stronger than MD5 because it is much more collision resistant. SHA-1 creates a 160bit message digest and like MD5 is based (loosely) around the concepts introduced in MD4. SHA-1 is a widely used hashing function. Like MD5, it is used to create checksums of large files but is still cryptographically secure enough to be usable in applications that require collision resistance, like storing hashed passwords in a database (Stallings, Brown and Howard, 2008, p.627-628). In 2005, a group of Chinese researchers discovered mathematical anomalies in the SHA-1 algorithm that could be exploited to discover two inputs that create the same output.

The last hash function discussed in this paper is SHA-256. SHA-256 is one hashing function in the SHA-2 family. The SHA-2 family derives from the SHA-1 algorithm. Though the SHA-2 is based on SHA-1, the mathematical exploits used to break SHA-1 have not been proven to have propagated to the SHA-2 family. Like SHA-1, SHA-256 is the creation of the National Security Agency with a publication date of 2001. SHA-256 creates a two-hundred-fifty-six bit output, giving it its name. It is considered to be much stronger than SHA-1 because of its increased output size. Though it is more secure, it is not as popular as SHA-1. (Stallings, Brown and Howard, 2008, p628)

The following chart displays the amount of time it takes to perform the MD5, SHA-1, and SHA-256 hashing algorithms on a set of files of various sizes. It is shown that though SHA-1 and SHA-256 are mathematically more secure, this does not impact the average runtime by a significant amount even at files of over 700MB. It should also be noted that the time to create the hash for each algorithm is quite low, indicating that it is computationally easy to create a hash for any given input. Any of these algorithms should be sufficiently fast enough to be used in a database for password hashing. It can also be concluded that because the amount of CPU time needed to complete the task, it will not have a great impact on the overall performance of our system. The code used to create this chart of values is located in Appendix A.

|  |  |  |  |
| --- | --- | --- | --- |
| Time Analysis of Hash Functions | | | |
|  | 9MB | 77MB | 700MB |
| md5 | 0.036 | 3.074 | 9.507 |
| sha1 | 0.184 | 0.435 | 10.075 |
| sha256 | 0.138 | 3.022 | 11.021 |

SHA-1 and SHA-256 are both strong candidates to be used in databases. This paper will focus on SHA-1 implementation because of its popularity. MD5 should not be used to hash passwords before of how quickly an attack can compromise the integrity of the hash.

Using and comparing hashed passwords is done in a similar manner as typical password based authentication except with a small amount of overhead. Basically, it is a comparison of the digital signatures of each password, or the hash of each password. Now, instead of the attack seeing each user’s password in plaintext, they see a list of hash values. How does this affect their ability to determine the original password? A common method for determining original input is through a brute force attack. How effective is this form of attack? Brute force attacks are considered to be infeasible because of the number of iterations that must occur to find a single match, and even when a match is found, there is no guarantee that it is the correct password. However, it is possible to determine the original user’s password in this manner given enough time by trying each match. (McGlinn, 2005)

Is there a way to improve this in some manner? Yes. There are several strategies for creating even more secure hashes. One method is to “salt” the input with some unique value. A salt value will ensure that two users who share the same password will have a password hash that is unique to each user. One method would be to simply concatenate the username and password then hash the associated combination. The format for such a combination would be: sha1(username + password). While this is more secure, a more accepted and cryptographically better approach is to combine the hash of the username with the hash of the password then hash that newly created hash value. The format for this combination can be expressed in the following way: sha1(sha1(username) + sha1(password)) (Ullrich, 2011). The following table shows an example of this strategy in action with two users that share a similar password. The code used to create this chart is available in Appendix B.

|  |  |  |
| --- | --- | --- |
| Username | Alfred | Nigel |
| Password | Penguin | Penguin |
| sha1(password) | s3\"►↑╣\☺?≤≡t╜₧àPΦ╘Ä | s3\"►↑╣\☺?≤≡t╜₧àPΦ╘Ä |
| sha1(username + password) | ΣQ▐╦éHBÄ{<7ré◄┐i}\_►H | 4èT@é ↓¢\e→VÇJ╫▄↓╟ |
| sha1(sha1(username) + sha1(password) ) | ad┴ë²ä◄╡É∟ºY¥├xƒ██ | ¥\_₧ß╧-(╢%òé┬∩┘ ╣iT≤ò |

Implementing such a system in a database is not as difficult as it sounds. In James McGlinn’s article entitled *Password Hashing*, he discusses a brief example in PHP of performing this method of authentication. Below the example provided in the article for storing the password in the database:

<?php

/\* Store user details \*/

$passwordHash = sha1($\_POST['password']);

$sql = 'INSERT INTO user (username,passwordHash) VALUES (?,?)';

$result = $db->query($sql, array($\_POST['username'], $passwordHash));

?>

From this we see that it is simple to create the hash digest in PHP and store it inside the database. McGlinn goes further to explain how to authenticate users in PHP now that the passwords are stored in an encrypted form. The user’s password is hashed upon attempting to login and the calculated hash is compared with the hash stored in the database. (McGlinn, 2005)

Considering the security gains of password hashing and the low overhead and relative ease of implementing such a system, not every company takes the action to hash user’s passwords. One such company is Microsoft. Earlier this year, the Microsoft Store in India was compromised by a group of hackers. In addition to defacing the Microsoft Store website, the hacker’s were able to gain access to the system’s database. What the hacker’s discovered is a database where user’s information including usernames, email and phone numbers, was stored in plaintext. The hackers now could gain access to each of these user’s accounts without any more effort required. Had they hashed the passwords for these users, the hackers would have had a more difficult time running rampant throughout the system (Gallagher, 2012). It later came to light that hackers also compromised credit card billing information and setup a hotline for users whose identity was stolen because of this breach (Agarwal, 2012).

Another company found to be storing passwords in plaintext is Sony. In 2011, hackers were able to compromise Sony Pictures website via a SQL injection attack. The attackers, part of an organization known as LulzSec, were able to still over one million usernames, passwords, addresses, email address, etc. in a single attack. Like Microsoft, they discovered that none of the information they gained access to was encrypted in any way. This is was the second such problem for Sony, who’s Playstation Network was also compromised. In this attack, over one-hundred million usernames and passwords were stolen. In addition to the user data lost, Sony also faced a class action lawsuit to this attack from disgruntled users. (Schwartz, 2011)

Password hashing is an effective means to protect user data. In addition to protecting passwords, the same strategy could and should be applied to other types on information. Also, there are some types of information stored in database that should be stored in different ways. One piece of information that should be protected is credit card information. Instead of storing this information using a one-way hash, this information should be stored using some other form of encryption, such as symmetric key encryption. By storing this information using symmetric key encrypt, the data can be derived from the encrypted output at a later time (Stallings, Brown & Howard, 2008, p42). Companies should still use extreme caution when setting up such as system as credit card information is extremely valuable to criminals.

Hardening systems from intrusion is only part of data protection. Hashing algorithms provide a solution to several problems of storing confidential information, including passwords. Hackers have victimized millions of users because administrators did not take the necessary steps to protect user data. Given the low overhead of hash based authentication, implementing such a solution is critical to a complete security solution.

Appendix A

Note: Due to constraints from the timeit library, values must be hardcoded into the application before it is launched. To use this in another setting, the highlight items need to be changed to the hash algorithm you wish to use and the file you wish to use it on.

\_\_author\_\_ = 'Dan Johnson'

from hashlib import md5, sha1, sha256

from timeit import Timer

#This function creates a hash of a given input file given a hashing function

def createHash(fileName, hashFunc=sha1()):

f = open(fileName, 'rb')

while True:

data = f.read(102400)

if len(data) == 0:

break

hashFunc.update(data)

#print hashFunc.digest

f.close()

if \_\_name\_\_ == '\_\_main\_\_':

t = Timer("createHash('pycharm-2.5.exe')",

"from \_\_main\_\_ import createHash")

print t.timeit(1)

Appendix B

\_\_author\_\_ = 'Dan Johnson'

from hashlib import sha1

def createHashes(username, password):

print 'Username: ' + username + ' Password: ' + password

print 'sha1(password) ' + sha1(password).digest()

print 'sha1(username + password) ' + sha1(username + password).digest()

print 'sha1(sha1(username) + sha1(password) ' + sha1(

sha1(username).digest() + sha1(password).digest()).digest()

if \_\_name\_\_ == '\_\_main\_\_':

createHashes('Nigel', 'penguin')

createHashes('Alfred', 'penguin')

References

Agarwal, A. (2012, February 27). *Not just email addresses, credit card numbers also stolen from microsoft india store*. Retrieved from http://www.labnol.org/india/microsoft-india-store-hacked/20891/

Gallagher, S. (2012, February 14). *Microsoft's store site in india defaced; hackers find plain text passwords*. Retrieved from http://arstechnica.com/business/news/2012/02/microsofts-store-site-in-india-defaced-hackers-find-plain-text-passwords.ars

McGlinn, J. (2005, March 20). *Password hashing*. Retrieved from

http://phpsec.org/articles/2005/password-hashing.html

Silva, J. (2003). An overview of cryptographic hash functions and their uses. Retrieved from http://www.sans.org/reading\_room/whitepapers/vpns/overview-cryptographic-hash-functions\_879

Schneier, B., & Hoffman, P. (2005). attacks on cryptographic hashes in internet protocols. Retrieved from http://www.ietf.org/rfc/rfc4270.txt

Schwartz, M. (2011, June 3). *Sony hacked again, 1 million passwords exposed*. Retrieved from http://www.informationweek.com/news/security/attacks/229900111

Stallings, W., Brown, L., & Howard, M. (2008). *Computer security, principles and practice*. Upper Sadddle River, NJ: Pearson Education.

Ullrich, J. (2011, June 28). *Hashing passwords*. Retrieved from http://www.dshield.org/diary.html?storyid=11110