WHAT IS HASHING?

Introduction

The security triad (also known as the CIA triad) defines the overarching principles of information security. One of the triad's primary principles is assuring the integrity of data. In other words, how can we be confident that the data has not been changed? The goal of this principle is to provide assurance that the data has not been modified, tampered with, or corrupted in any way.

The method most often used to verify data integrity is called hashing. At a high level, hashing takes a message digest (sometimes referred to as a fingerprint) of the data when it is sent and another when it is received. The two fingerprints are compared, and the results will highlight whether the data has been modified in any way. Sounds simple enough, right? But how does the hashing process actually work?

What is hashing?

Before dissecting the hashing process, let's first look at the different parts of the process. There are three key components involved: input, hash function, and hash value.

Input

The input is the data itself. This data may take any form such as a text file, a Microsoft PowerPoint presentation, an MP3 music file, a JPG picture, an email, or any other type of data. Since the input data can be of many different types, the size will be variable – anywhere from a few bytes up to several hundred megabytes or more.

Hash function

The hash function is the algorithm used to generate the fingerprint. This is the mathematical process that takes the input data and generates the fingerprint. The mathematics used by the algorithm are outside of the scope of this paper.

Hash value

The hash value is the output of the hash function. As you have probably guessed, the hash value is the resultant fingerprint of the input data. While the hash value is a binary number, it is typically displayed as a hexadecimal value.

Variable input, fixed output

An important concept to understand is that the hash value is a fixed size regardless of the input data size. The size of the output depends on the algorithm used. For example, the MD5 hash function will generate a 128-bit hash value while SHA-1 will be 160 bits. To illustrate this, let's look at an example.

The MD5 hash value for the opening paragraph of this document is:
fb84fe5514eebe360ec434bc326c70d2

The MD5 hash for Ernest Hemingway's novel The Old Man and the Sea is:
e6200a8a14a76ce2e19bac3f48d2f036

You'll notice that both hash values are the same length (128 bits in this example). Yet, The Old Man and the Sea is considerably longer than the opening paragraph of this document (about 27,000 words vs 131).

Properties of a hash algorithm

In order to be considered viable, there are four goals that a hash algorithm needs to meet:

1. Running the same hash function on the same input data must yield the same hash value.
2. Small changes to input data should result in large changes to the hash value.
3. Each resultant hash value for different input data should be unique.
4. The hashing process must be one way (i.e. it can't be reversed).

Repeatable results

This goal is straight forward. When the same input data is run through the same hash function, it must generate the same hash value each and every time. If the MD5 hash algorithm is used with the opening paragraph of this document six months from now, it must provide an identical hash value to what's shown above. Simple enough, but vitally important to when determining data integrity. More on this later.

Minor change in input yields major change to output

Another aspect of a good hash function has to do with how changes to the input data affect the resulting hash value. A very minor change with the input data should yield a drastic change to the hash value. We can illustrate this by again using the first paragraph of this document. As you will recall, the MD5 hash for the first paragraph is:

fb84fe5514eebe360ec434bc326c70d2.

By simply changing the first letter of the paragraph to lower case, results in the hash value of:

bfe4ad909063ac25f8a4abbc6cf44c22

As you can see, the hash value is drastically different after the simple change to the input data.

Each hash value should be unique

The goal here is that no two different sets of input data should result in the same hash value (referred to as a collision if it were to happen). As you may be able to tell, it is not outside the realm of possibility that a collision could occur

since any length of input data always provides a fixed length hash value. As a result, the number of possible hash values is finite. It is, however, very unlikely that it would occur.

The MD5 hash algorithm we used earlier results in a 128-bit hash value. This is one of the smaller hash sizes available, yet there are more than 3.4 x 10^38 possible hash values. So, while possible, it is extremely unlikely that a second input could be found that generates the same hash value.

Process must be one way

An important goal of hashing is that the process must be one-way. In other words, it cannot be reversed so the original input data cannot be derived given the hash value and the hash function that was used. Most modern hashing algorithms in use today comply with this goal. The mathematics used to ensure this are outside the scope of this paper but suffice it to say that this is the case.

Common hash algorithms

There are many different hash algorithms available for use today. There are so many, in fact, that it isn't feasible to review them all here. As a result, we'll limit the review to a few of the most common.

MD5

Message Digest v5 (MD5) is likely one of the more recognizable hash functions in use today. As has been mentioned previously, MD5 provides a 128-bit hash value. It is important to note that, despite its continued use, MD5 has been broken and is no longer considered cryptographically secure.

SHA-1

The Secure Hash Algorithm v1 (SHA-1) provides a 160-bit hash value. SHA-1 was designed by the National Security Agency (NSA) is still considered cryptographically secure, but the increased computational power of computers in use today make it more likely that it will be broken. As a result, in 2010 it was no longer recommended for use.

SHA-2

The Secure Hash Algorithm v2 (SHA-2) is actually a family of hash algorithms. Also developed by the NSA, SHA-2 uses a different mathematical process than SHA-1 and has several variants that produce different size hash values. The variants are SHA-224, SHA-256, SHA-384, and SHA-512. These algorithms produce 224, 256, 384, and 512-bit hash values respectively.

SHA-3

The Secure Hash Algorithm v3 (SHA-3) is an algorithm (formerly known as Keccak) that was chosen after a competition among non-NSA developers. SHA-3 provides variants that produce hash values of the same length as SHA-2. Although SHA-3 corrects some of the weaknesses found in SHA-2, it is not in common use yet currently.

HMAC

Hash-based Message Authentication Code (HMAC), as the name implies, uses hashing but with a twist. HMAC can use any cryptographic hash function as its base but also appends a secret key to the input data. The hash function is then referred to as HMAC-X (where X is the hash function used – such as HMAC-SHA224). As a result, HMAC is a dual-purpose function. It serves as both a hash function as well as a message authentication method.

Hashing and data integrity

So, now that we've got all the background information, how is hashing used? Let's look at a couple use case examples.

Verifying data integrity

As was mentioned in the opening paragraph, maintaining data integrity is one of the key concepts for information security and hashing is the mechanism used to verify it. Here's a simple look at how that process works. Our old InfoSec friends Alice and Bob will help us to illustrate this example.

Alice needs to send a spreadsheet to Bob with some important financial information. Both she and Bob would like some assurance that the information Alice sends is what Bob receives. It's possible that a person with malicious intent could intercept and modify the information before Bob receives it, or it's possible that the data is unintentionally corrupted in transit. Either way, Alice and Bob would like to be confident that the information remains intact.

To accomplish this, Alice needs to get a fingerprint for the spreadsheet. To generate it she'll select a hash function (let's say SHA-256) and run the algorithm on the spreadsheet. The output of the hash function will be the fingerprint (hash value). Alice can then send both the spreadsheet and the fingerprint to Bob and identify the hash function she used.

When Bob receives the information, he can then generate his own fingerprint with the spreadsheet that he received and the appropriate hash function. He will then compare the two hash values, if they are the same, then the he can be confident that it is the original data that Alice sent. If not, then the data has been modified in some way.

Integrity and authentication

An alternative is to use HMAC to provide data integrity as well as authenticate the sender. Let's now take a look at how that works. Again, Alice and Bob will help us out by building on the example used in the previous section.

For this, Alice will need to take the additional step of agreeing on a pre-shared secret key (PSK) with Bob. This will be a pass phrase that only she and Bob will know. From there she will follow the same process as in the previous example. The only difference is that the PSK will be sent separately so any malicious actors won't have all information necessary to recreate the process. Now, when Bob receives the information, he will generate his own fingerprint but will need to include the PSK. If the two fingerprints match, he can not only be confident that the data remains intact but also be sure that Alice is the sender.

Conclusion

As we can see, hashing plays an important role in information security. When considering hashing, there are three important concepts to remember:

1. Hashing plays a key role in assuring data integrity. Since maintaining data integrity is one of the three primary pillars of the security triad, hashing is vitally important to information security.
2. Even minor changes to the input data must result in major changes to the hash value output. While this in and of itself is a small thing, it makes data integrity losses easily identifiable.
3. The hashing process must be one way. In other words, the original input data cannot be determined from the hash value. This is important because user passwords are often stored as the hash value. If the process were reversible, then the passwords could be derived, stolen, and used to access the systems they were supposed to protect.