Hashing Basics

Transcribed on July 7, 2025 at 2:33 PM by Minutes AI

Speaker 1 (00:01)

Bienvenidos a esta nueva sesión del Módulo dos.

Vamos a trabajar el concepto de hashing.

En esta ocasión lo que vamos a trabajar durante los próximos minutos es el concepto de hashing, los tipos de algoritmos que tenemos en la parte de hashing y luego dónde aplicamos o dónde podemos aplicar realmente en el mundo real los temas de hashing.

El hashing podemos decir que es un procedimiento criptográfico el cual nos va a permitir obtener un resumen de una información.

¿Qué queremos decir con un resumen de la información?

Al final un resumen de la información puede ser un conjunto de caracteres que luego definiremos que representan.

Por ejemplo, van a representar el resumen de un la información o de un texto o de un contenido, un fichero, un binario, lo que sea.

De forma que si yo tengo un texto o dos textos diferentes y yo aplico un hash sobre ellos, voy a tener un conjunto de caracteres que lo van a identificar de forma unívoca y gracias a ese conjunto de caracteres voy a poder saber si esa frase es manipulada en algún momento o no, por lo cual tiene que ver con la integridad de la información que estamos manejando.

Este resumen lo llamamos así resumen cuando tenemos el hash.

Este resumen es fácil de conseguir gracias a una función matemática en el cual dado una información, un fichero o lo que sea, le aplicamos una función matemática y obtenemos el resumen volver atrás.

Es decir, de ese resumen, que luego veremos un ejemplo, de ese resumen hacia atrás realmente va a ser complejo, va a ser computacionalmente muy complejo de forma que no vamos a poder realizarlo de forma sencilla.

Es decir, en otras palabras, el hashing es el proceso criptográfico por el cual nosotros vamos a conseguir pasar de texto bytes a un subconjunto de datos.

Vamos a decir que es un conjunto de caracteres hexadecimales.

Puede ser que tengamos hashes de 40 bits, puede ser que tengamos hashes de 256 bit, dependerá luego de la robustez del algoritmo que estemos utilizando.

Pero al final estamos obteniendo un conjunto de caracteres hexadecimales que representan pero no saben decirnos nada de lo que están representando en resumen que están formando esa información.

Lo que estamos haciendo al final es una especie de firma o hacer un resumen, lo que denominamos hashing volver hacia atrás de la firma o del resumen.

Es algo competencialmente muy costoso, por lo cual no vamos a poder hacerlo en un tiempo razonable y aquí es donde entraría el concepto de cracking y todo esto.

Aquí radica la importancia con un hash, es decir, en un sentido desde el texto original al resumen es fácil computacionalmente, pero volver hacia atrás es muy complejo computacionalmente.

Ahí está un poco la robustez de los hases.

Vamos a ver un pequeño ejemplo.

Imaginemos que tenemos un algoritmo de hashing, por ejemplo, recibimos una entrada a través de una función matemática.

La entrada ya como comentaba antes, puede ser un fichero, puede ser un texto o lo que sea.

El algoritmo hace el resumen de los datos de entrada, ahí veis un ejemplo, f de data y veis el baffd, realmente ahí nos da un poco igual, fijaos que son datos, caracteres hexadecimales, no hay ningún caractereci, hay unos puntos suspensivos.

Ese hash puede ser de 40 bit, puede ser de 128 bits, puede ser lo que sea, se obtiene un valor y si algún dato de esos es modificado del fichero, de la fuente de entrada o lo que sea, o que sea un bit, ese hash que se genera de esa modificación ya no tiene nada que ver con el hash que hemos obtenido antes y podemos detectar fácilmente que ha habido una manipulación.

Afecta directamente a la integridad de la información.

Los algoritmos de hashing son herramientas muy buenas para poder detectar manipulaciones en la información que estamos consumiendo.

Un ejemplo muy clásico es cuando en ciertas instrucciones de Linux nos dice aquí tienes la ISO con la versión x de esta distribución y el hash del fichero es este.

Es decir, cuando tú te descargues el binario o la ISO, la ova o lo que sea, tienes que obtener este inmohar y de esa forma vas a saber que estás ante los mismos datos que has descargado o que tú esperabas descargar, mejor dicho.

Bien, bueno, aquí tenemos un pequeño ejemplo.

También hay una página donde podemos obtener hashes fácilmente en m, pero incluso hay herramientas en los sistemas linux tenemos en el propio macOS también tenéis por ejemplo md sum, sa sum, tenéis diferentes herramientas, diferentes binarios que van a generar los hashes.

Y bueno, ahí tenéis un ejemplo.

Fijaros que estamos haseando la palabra high wall con m, ahí veis el tamaño del hash, fijaros el resumen, un a siete, seis, seis, siete, b, af, d, f, ese es el resumen, lo que llamamos resumen.

Luego lo mismo, la palabra high wall la estamos haciendo con sha, fijaos que el hash es más largo, el conjunto de caracteres hexadecimales es más largo.

También hemos hecho con sa, veis que es más largo y con sa 512 que es muy largo.

Cuanto más largo es el hash o el conjunto de caracteres del hash, pues significa que hay mayor robustez porque hay menos probabilidad de que haya una colisión.

¿Qué es una colisión?

Dos palabras o dos ficheros que generan el mismo hash.

Eso puede ocurrir porque al final si tenemos, imaginaos, 40 caracteres para representar todo hashing, lógicamente en algún momento va a haber una colisión, va a haber algo que son dos cosas diferentes que generen el mismo hash, eso es una colisión, eso es un problema.

Cuanto mayor es el número de caracteres que yo tengo para utilizar para generar mi hash, lógicamente mayor espacio de caracteres, mayor charset, por lo cual menos colisiones vamos a encontrar.

¿Cuando alguien es capaz de tener la posibilidad de generar una colisión, lo que está consiguiendo es romper el algoritmo, por qué?

Porque al final tiene una forma de poder generar colisiones, entonces la robustez del algoritmo y la integridad quedaría lógicamente comprometida.

Bien, bueno, aquí otro ejemplo que queríamos enseñaros es, bueno, tenemos un fichero llamado text, como veis ahí estamos aplicando el binario msum, nos está dando el hash de ese fichero, el fichero contiene texto, hemos generado el hash como hemos visto, modificamos el valor del fichero y generamos de nuevo un hash.

Fijaros que hemos pasado, vamos a pasar mejor dicho, tenemos un high wall, ahí tenemos los hases en ms, a continuación modificamos el high wall en vez de con una exclamación, con una interrogación, y fijaros como simplemente ese cambio, la exclamación por interrogación, genera que ya el fichero es distinto, con lo cual el hash md y el hash sa son diferentes, en cambio el hasamas es radical, no es que cambie una posición, no, no, es

Es importante a la hora de aplicar robustez y de intentar no conseguir predecir un hash en una variación.

Bien, este es un buen ejemplo, si volvemos hacia atrás a cambiar el interrogante por la exclamación, obtendríamos el mismo hash tanto en M como en SA que en la primera slide que hemos puesto.

Es decir, vamos ahora pasar a los tipos de algoritmo.

Tenemos aquí los tipos de algoritmos, tenemos algoritmos, hemos visto ya en los ejemplos como m a 512, al final por ejemplo el md pues produce salidas de 16 bytes hexadecimales, son todos, actualmente el md no es considerado un algoritmo seguro por las colisiones.

Luego tenemos el sa que produce salidas de 20 bytes también conjunto de datos en hexadecimal también, no es un algoritmo seguro, aunque usando algoritmo de soporte o secundario pues tendría una validez.

Si yo utilizo sa, porque además del sa quiero dar un sa, está bien, es un algoritmo de soporte en ese caso, una segunda opinión digamos, pero tampoco es seguro a día de hoy, pero también por lo mismo tema de colisiones, etc.

Y luego fijaros que hablamos del sa 512, produce salidas de 64 bytes, también en sa decimal, es un algoritmo robusto, actualmente es un algoritmo robusto y habéis visto que produce una salida muy larga, 64 bytes al final es una salida muy larga, pero es un algoritmo que de momento es robusto y muy utilizable, se puede realizar un saque.

Entonces soporte un SA al final como soporte está bien claro, si tú utilizas dos algoritmos ya estás haciendo que esto sea mucho más seguro.

Explico por qué.

Si tenemos un m, por ejemplo, 1 SA 512, la probabilidad de que la colisión del mismo fichero que un atacante está utilizando para esos dos algoritmos es prácticamente imposible, es prácticamente ínfima.

Es decir, conseguir esa manipulación para que la misma entrada genere el mismo hash que esperamos en dos tipos de algoritmos diferentes es algo muy, muy poco probable.

Así que bueno, ahí también está un poco la seguridad o la robustez de todo esto tipo de algoritmo que encontraremos.

Generalmente utilizaremos IMAX, por supuesto, pero utilizaremos generalmente estos a modo de algoritmos de resumen.

¿Y ahora llegamos a la parte final donde queríamos hablar también un poco del hashing, de dónde utilizarlo, etc.

Qué provoca todo lo que hemos estado viendo?

Pues provoca que el hashing sea utilizado para verificar que un dato es íntegro.

Si me mandas un fichero y yo sé porque tú me lo estás mandando, oye, este fichero cuando lo abras, o este binario cuando lo abras, antes de abrirlo, cuando lo analices, tiene que darte este hash.

Cuando lo descargues, por ejemplo, como ocurre en ciertas páginas web, cuando te descargues este binario, tiene que darte este hash.

Entonces nos permite verificar que un dato es íntegro, que un fichero no ha sido modificado.

Cuando tú es íntegro.

También, por ejemplo, nos permite saber si un texto ha sido manipulado en algún momento, un mensaje ha sido manipulado, aunque sea un byte como hemos visto antes, aunque sea un bit como hemos visto antes, eso hemos cambiado la exclamación por interrogación y hemos visto que eso radicalmente cambia los hases, con lo cual permite detectar que un dato es íntegro, que un fichero no se ha modificado, que por ejemplo el texto de un email no ha sido modificado en ningún momento, que eso es muy interesante, está alineado con las firmas, cuando firmamos algo estamos ahí haciéndole una firma que en el momento que se modifique cualquier dato, cualquier carácter de ese email, la firma ya no corresponde, ya no salía igual, ese hash ya no saldría igual.

Y nos va a permitir también verificar en general la integridad de la información en diferentes aspectos, diferentes ámbitos.

Y además, aunque no está puesto aquí, también hay que decir que los hashes permiten que las contraseñas puedan ser almacenadas de forma más segura en bases de datos, por ejemplo, utilizando diferentes algoritmos de hashing.

De forma que tú me das una contraseña, yo lo haseo y yo compruebo contra mi fichero, mi base de datos, que lo que tú me has introducido ya haseado, pues corresponde con la contraseña que está almacenada.

Es decir, pensemos en un sistema operativo.

Un sistema operativo recibe un usuario, contraseña, inicia sesión, coge la contraseña, la hasea, en este caso no utiliza m, pero sí puede utilizar, por ejemplo windows lm, ntlm, lo hasea y comprueba el hash que haya almacenado en la Xampp, el fichero de Security Account Manager.

Ese fichero en Windows almacena esa información.

Entonces, si ese hash es igual al que está almacenado, significa que la contraseña que has introducido tú es correcta.

¿Podría haber una colisión?

Podría darse el caso, pero es poco probable.

Y eso funciona así.

Entonces nos permite, es un mecanismo que permite almacenar ciertos elementos sensibles que no necesitamos recuperar, no necesitamos que sean reversibles.

Por ejemplo, la contraseña, tú la introduces, se hasea y yo compruebo si esos hashes son iguales, el que tú has generado con el que yo tengo almacenado.

De forma que entonces en ese caso puedo decir si son iguales, la contraseña ha sido correcta, si no, pues no.

Entonces, bueno, pues esto es importante también verlo.

Y bueno, como conclusiones, hemos estudiado el tema del hashing, hemos podido definir que es un procedimiento criptológico o criptográfico que permite crear un resumen de la información.

Ese resumen al final es lo que llamamos hash y ese hash tiene diferentes longitudes, desde has de de 16 bytes, 20 bytes, 40 bytes, hasta 64 bytes, por ejemplo, sa que hemos visto diferentes longitudes.

Y estos hases lo que nos permiten son muchísimas operaciones, los puedes utilizar en diversos campos, desde saber si lo que me está llegando ha sido manipulado o no, saber si lo estoy descargando en lo que espero o no, o incluso poder comprobar que las contraseñas que estoy introduciendo son correctas y que el proveedor no tiene por qué conocer la contraseña, simplemente tiene almacenado lo que es el hash y no hace falta lo otro.

Bueno, pues esto es un poco todo lo que hemos visto en esta sesión del módulo dos.

Nos vemos en la siguiente sesión.