Digital Guide

(/digitalguide/)

04.10.19 | Seguridad (servidores/seguridad/)

Rainbow tables: qué son y cómo funcionan las tablas arco iris

Con un nombre más acorde a un juego infantil, las tablas arco iris son un poderoso instrumento con el que es posible *crackear* contraseñas. A esta tarea se dedica con ahínco un buen número de personas a ambos lados de la legalidad, unos por el beneficio que aspiran a obtener con ellas, los otros porque su trabajo como expertos consiste en comprobar periódicamente la efectividad de los estándares de seguridad vigentes y las tablas arco iris -traducción del original rainbow tables, también conocidas como tablas rainbow- permiten encontrar claves en unos pocos segundos si se dan ciertas circunstancias.

Aun cuando confiamos en que nuestros lectores no tengan intenciones criminales, es conveniente comprender cómo funcionan estas tablas de búsqueda, pues permiten entender, por un lado, por qué deben usarse contraseñas difíciles y, por el otro, cómo pueden protegerse las cuentas de los usuarios de una página web.

Índice

¿Para qué se utilizan las tablas arco iris?

Hoy las contraseñas ya no se guardan sin cifrar –o eso se espera. Cuando los usuarios de una plataforma fijan una clave de acceso para su cuenta, esta secuencia de caracteres no aparece en texto plano en una base de datos en algún servidor, puesto que no sería seguro: si encontrara la forma de entrar en ella, un hacker lo tendría muy fácil para acceder a todas las cuentas de un determinado usuario.

Para el eCommerce, la banca en línea o los servicios gubernamentales online esto tendría consecuencias fatales. En lugar de ello, los servicios online utilizan diversos mecanismos criptográficos para cifrar las contraseñas de sus usuarios de modo que en las bases de datos solo aparezca un valor hash (valor resumen) de la clave.

Incluso conociendo la función criptográfica que lo ha originado, desde este valor hash no es posible deducir la contraseña, porque no es posible reconstruir el procedimiento a la inversa. Esto lleva a los ciberdelincuentes a recurrir a los **ataques de fuerza bruta**, en los cuales un programa informático intenta "adivinar" la secuencia correcta de caracteres que constituye la contraseña durante tanto tiempo como haga falta.

Este método puede combinarse con los llamados **"diccionarios" de contraseñas**. En estos archivos, que circulan libremente en Internet, pueden encontrarse numerosas contraseñas que bien son muy populares o ya fueron interceptadas en el pasado. Los hackers prueban primero todas las contraseñas del diccionario, lo que les permite ahorrar tiempo, aunque, en función de la complejidad de las contraseñas (longitud y tipo de caracteres), este proceso puede resultar más largo y consumir más recursos de lo esperado.

Consejo

Si utilizas palabras comunes para tus claves facilitas el camino a los atacantes. En nuestro artículo sobre contraseñas seguras (servidores/seguridad/gestor-de-contrasenas-un-vistazo-a-las-mejores-opciones/) conocerás cómo debe ser una buena contraseña.

También disponibles en la Red y también un recurso para descifrar claves secretas, las tablas rainbow van un paso más allá de los diccionarios. Estos ficheros, que pueden llegar a tener un tamaño de varios cientos de gigabytes, contienen un listado de **claves junto con sus valores hash**, pero de forma incompleta: para reducir su tamaño y así su necesidad de espacio en memoria, se crean cadenas de valores a partir de las cuales pueden reconstruirse fácilmente los demás valores. Con estas tablas los valores hash encontrados en un banco de datos pueden ordenarse con sus claves en texto plano.

Este es el funcionamiento de las rainbow tables

Para poder entender cómo funcionan las tablas arco iris conviene echar un vistazo a la **mecánica de los algoritmos de cifrado**, pues esto permite comprender más fácilmente las ventajas de estas listas de búsqueda y el llamado compromiso tiempo-espacio.

Tecnología de cifrado

Desde que se comenzaron a aplicar funciones hash (servidores/seguridad/funcion-hash/) criptográficas en el cifrado, los algoritmos no han dejado de evolucionar y los estándares que hace diez años se consideraban inexpugnables hoy son vistos como graves vulneraciones de la seguridad. Todos, no obstante, tienen algo en común, y es que **el contenido que debe cifrarse se somete a diversos**

algoritmos hasta que finalmente se genera un valor hash. Este valor resumen es normalmente una cifra hexadecimal con una longitud fija, que no depende de la del contenido inicial. Al final del proceso siempre resulta, por ejemplo, un valor hash de 128 bits.

Tres son los aspectos decisivos en el cifrado:

1. La misma entrada genera siempre el mismo valor hash: solo así este valor puede funcionar como **suma de verificación** (del inglés "checksum"). ¿Es la clave introducida idéntica a la que hay en la base de datos? El sistema solo autoriza el acceso cuando ambos valores hash coinciden.
2. Un valor hash debería ser siempre único: dos entradas diferentes no deberían generar el mismo valor resumen, pues solo siendo únicos pueden garantizar que se ingresa la clave correcta. Como el número de valores hash posibles está limitado, pero el de posibles entradas no, es imposible descartar este tipo de coincidencias, llamadas **colisiones** en este contexto. Las funciones hash modernas y los valores hash con una longitud suficiente intentan mantener este riesgo a raya.
3. Los valores resumen no son reversibles, es decir, que a partir del valor resumen no es posible deducir el contenido original (la clave). Por ello **tampoco es posible descifrar valores resumen**, como se sostiene a veces de forma algo imprecisa. Solo pueden reconstruirse.
4. Las funciones hash deben ser muy **complejas**, pero no en exceso: si un algoritmo trabaja demasiado rápido, facilita el trabajo a los atacantes y ya no puede garantizar la seguridad. Pero la transformación tampoco debe ser excesivamente compleja porque en definitiva ha de llevarse a la práctica.

📌 Hecho

Los valores hash no solo se aplican al cifrado de claves de acceso. También se utilizan como sumas de verificación para programas. En este caso, los algoritmos generan un valor resumen a partir del código fuente que permite comprobar, por ejemplo, que la versión del programa que se ha descargado es idéntica al original y no un software malicioso.

Funciones de reducción

Los valores hash contenidos en las tablas rainbow no se crean con ocasión de un ataque, sino con anterioridad, de modo que los hackers puedan hacerse con ellas y utilizarlas para encontrar claves de acceso. Pero como estos ficheros son muy grandes, se aplica una función de reducción que permite ahorrar memoria. Esta función **convierte al valor hash** en un texto plano –no devuelve al valor original del valor hash, esto es, a la contraseña original, pues esto no es posible, sino que genera un texto completamente nuevo.

A partir de este texto se crea otro valor resumen nuevo, un proceso que en una tabla arco iris no sucede una sola vez, sino muchas, de forma que se genera **una cadena**. En la tabla definitiva, no obstante, solo aparecen la primera clave y el último valor resumen de la cadena. Con esta información y utilizando las mismas funciones de reducción es posible averiguar todos los demás valores. El valor hash que se quiere *romper* se reduce y se resume una y otra vez siguiendo las mismas reglas,

comprobando cada resultado con los valores que figuran en la tabla para encontrar su clave correspondiente.

El desafío a la hora de crear una tabla reside en el hecho que la palabra inicial que representa el comienzo de una cadena no puede figurar como texto plano en otra cadena precedente.

Con este método puede reducirse en gran medida el tamaño de estas tablas, aun teniendo todavía un volumen de varios cientos de gigabytes.

Compromiso espacio-tiempo

Se habla de una situación de compromiso espacio-tiempo o tiempo-memoria cuando la memoria se reduce a costa de una ejecución más lenta o a la inversa, el tiempo de ejecución se reduce incrementando el uso de la memoria. Un ataque de fuerza bruta necesita muy poco espacio, porque los cálculos criptográficos comienzan siempre de cero en cada ataque. En cambio, una tabla que contiene miles de millones de contraseñas junto con sus valores hash ocupa mucha memoria, aunque puede descifrar muy rápido. Las rainbow tables representan un **acuerdo intermedio,** pues aunque también lleva a cabo cálculos en tiempo real, lo hace a pequeña escala, de modo que, en comparación con las tablas completas, reduce claramente las necesidades de memoria.

Mecánica de las tablas arco iris

La situación inicial es esta: dado un valor hash, se pretende conocer la clave de acceso que lo originó. En un primer paso, **se busca este valor en la lista**. Si se encuentra al comienzo o al final de una cadena, entonces es fácil encontrar la contraseña, porque solo habrá que reconstruir los pasos que recorrió la cadena hasta obtener el resultado que se busca. ¿Qué ocurre entonces cuando el valor resumen no se encuentra en la tabla?

En este caso, se comienza reduciendo el valor con la misma función con la cual se creó la cadena. El resultado se somete a su vez a la función resumen, proceso que se denomina *hashear* y que se repite tantas veces como sean necesarias hasta que se encuentre al valor resumen en algún punto final. Con todo, esto no significa que se haya encontrado la clave, aunque sí **se ha encontrado la cadena** en la cual se esconde el valor hash. Entonces se comienza en el punto inicial de la cadena y se lleva a cabo de nuevo esta alternancia de reducción y *hashing* hasta que se logra encontrar el valor hash que se busca y con él a la clave en texto plano.

Por qué se llaman tablas arco iris

Llegados a este punto, quizá lo más normal sea preguntarse qué tiene que ver el arco iris con estas tablas de descifrado de claves. La razón es muy sencilla: en la práctica **no se utiliza solo una función de reducción**, sino una diferente en cada paso, porque así se consigue un mejor resultado y se evita que

se repitan valores hash en la tabla, si bien con un inconveniente: encontrar en la cadena combinaciones de valores hash y claves es más complejo.

Para lograrlo se han de seguir las reducciones por orden. Si se parte de la suposición de que se ha creado una cadena con las reducciones R_1, R_2, R_3, la búsqueda comenzaría con la función R_3. Si esta no arroja resultados, se pasa a R_2 y luego otra vez a R_3, etc. **Si se marcan con colores**, esta alternancia de funciones puede dar lugar a un colorido arco iris, dando así nombre a estas tablas de búsqueda de claves.

Las rainbow tables en la práctica

La mejor forma de entender la mecánica de las tablas arco iris es siguiendo un ejemplo paso a paso. Para poder hacerlo no utilizaremos ninguna de las funciones hash conocidas para asegurar claves, puesto que son demasiado complejas para fines didácticos. Utilizaremos en cambio una función mucho más sencilla, el **hash de multiplicación**:

$$h(k) = \lfloor m * (kA \bmod 1) \rfloor.$$

(fileadmin/DigitalGuide/Screenshots_2018/rainbow-tables-multiplikative-methode-beispiel.png)

En esta ecuación, *k* es la clave de acceso, *m* es un multiplicador (que en nuestro ejemplo es *2.000*). Para *A* se suele utilizar la proporción áurea 0,618. (φ). *Mod* (modulo) extrae el resto de una división que en esta función tiene lugar por 1. La función de parte entera redondea el resultado a un número entero en caso que sea necesario. El resultado, *h(k)* es el valor hash del valor de entrada *k*.

! Nota

Puedes probar esta función en Excel utilizando la función MULTIPLO.INFERIOR para redondear y la función RESIDUO para mod: =MULTIPLO.INFERIOR(RESIDUO(A1*0,618;1)*2000;1)

Consideremos como claves probables una secuencia de dos cifras, lo que resulta en un tramo del 00 al 99. Esto mantiene a la tabla en un marco razonable y evitamos tener que convertir las letras en cifras previamente.

Aplicando el hash de multiplicación a la contraseña *78* resulta:

$$h(k) = \lfloor 2000 * (78 * 0{,}618 \bmod 1) \rfloor = \lfloor 2000 * (48{,}204 \bmod 1) \rfloor = \lfloor 2000 * 0{,}204 \rfloor = \lfloor 408 \rfloor.$$

(fileadmin/DigitalGuide/Screenshots_2018/rainbow-tables-beispiel-berechnung.png)

Como nuestro valor hash está compuesto de 4 cifras lo completamos con un 0: *0408*.

Clave	Valor hash
cero	cero
01	1236
02	0472
03	1708
...	...
78	0408
...	...
99	0364

En una tabla rainbow para esta función hash se aplican ahora **funciones de reducción**. Una forma muy sencilla de reducir un valor hash es utilizando solo sus dos últimas cifras. Si tomamos la clave *78* y su correspondiente valor hash *0408,* su reducción es *08*. A partir de aquí se vuelve a recomponer el valor hash con ayuda de la función multiplicativa.

La frecuencia de las repeticiones es libre, pero cuantas más repeticiones, menos memoria necesita la tabla –si bien se incrementa el tiempo de ejecución. En nuestro ejemplo aplicaremos la reducción tres veces:

c1	h1	c2	h2	c3	h3	c4	h4
cero	cero	cero	cero	cero	cero	cero	cero
01	1236	36	0496	96	0656	56	1215
02	0472	72	0992	92	1712	12	0832
03	1708	08	1888	88	0768	68	0048
04	0944	44	0384	84	1824	24	1664
05	0180	80	0879	79	1644	44	0384

Esta tabla representa la cadena completa con todos los resultados de las funciones hash y de reducción. El objetivo de una tabla arco iris es reducir su tamaño, así que la rainbow table final solo contiene el **valor inicial (c1) y el final (h4)**; el resto de valores puede deducirse de ellos:

c1	h4
cero	cero
01	1215
02	0832
03	0048

c1	h4
04	1664
05	0384
06	1260
07	0656
09	0944
10	0607
11	0539
13	0607
14	1824
17	0272
18	0651
19	1104
20	1664
21	0204
22	1552
25	0944
26	1215
27	0832
29	1664
30	0384
31	1260
33	0272
34	0944
37	0992
38	0656
39	1824
40	1440
41	0159
42	0272
43	0651
45	1824
46	0204
47	cero
49	0384
50	cero
53	0048
54	1664
55	0384

c1	h4
57	0656
58	1328
59	0651
61	0539
62	0992
63	0656
65	1440
66	cero
69	1104
70	1664
71	0204
73	1712
74	0384
77	0832
78	0048
81	1260
82	1712
83	0272
85	0428
86	1484
89	1824
90	0384
93	0700
94	1552
95	1824
97	1552
98	1036
99	0384

! Nota

En este ejemplo el tamaño de las tablas arco iris se ha reducido muy poco respecto a la tabla original: 140 frente a las 200 entradas iniciales. Esto se debe a que, con la finalidad de mostrar la mecánica de las tablas rainbow en la praxis, se ha escogido una tabla ya de entrada de tamaño moderado, las funciones utilizadas no son muy complejas y se han llevado a cabo pocas reducciones.

En esta tabla vemos que ya no están todos los valores hash. Si el valor hash conocido fuera *1888*, lo buscaríamos en la tabla en vano, puesto que este valor ahora **se esconde en una cadena**. En consecuencia, se le aplica la reducción, dando como resultado *88*. Este valor tampoco está en la tabla, así que se calcula de nuevo su valor hash (*0768*) y su reducción (*68*). El valor hash que le corresponde (*0048*) se encuentra en la tercera fila, si bien la clave en la misma fila (*03*) tampoco pertenece directamente al valor hash, puesto que solo es el comienzo de la cadena. Pero sí ofrece un punto de partida para el siguiente cálculo: a partir de *03* se calcula el valor hash *1708*. Lo reducimos a *08* y reconstruimos de nuevo el valor hash: *1888*, que es el valor hash que buscábamos. En consecuencia, la contraseña que corresponde a *1888* es *08*.

Cómo puedes defenderte de las tablas arco iris

Ahora que hemos visto cómo los atacantes pueden acceder a cuentas de usuario utilizando rainbow tables, queda clara la necesidad de tomar medidas de defensa. Tanto los usuarios como los responsables de páginas web tienen a mano algunas estrategias con las que impedir este tipo de ataques o al menos dificultarlos.

Qué puedes hacer como usuario

Como regla de oro, sobra decir que las claves han de ser suficientemente largas y contener mayúsculas, minúsculas, cifras y caracteres especiales. Como descifrar una clave así es difícil, seguir esta regla te defiende de los ataques de fuerza bruta y de tablas arco iris, ya que con la **longitud de la clave** aumenta exponencialmente el tamaño de la tabla. Tampoco se recomienda utilizar palabras, sino solamente secuencias aleatorias, lo que impide los ataques de diccionario. Cualquier gestor de contraseñas (servidores/seguridad/gestor-de-contrasenas-un-vistazo-a-las-mejores-opciones/) puede ayudarte a crear claves seguras.

Tampoco conviene utilizar las claves más de una vez. No importa qué tipo de ataque se utilice: una vez que alguien ha logrado allanar una base de datos, descifrar las contraseñas y acceder a datos personales, es muy sencillo probar si con la misma contraseña se puede acceder a otras cuentas.

Las armas del administrador

Los gestores de los servidores también tienen recursos para proteger a sus usuarios. Antes que nada se ha de impedir con valores hash el acceso no autorizado a las bases de datos. Naturalmente, es más fácil decirlo que conseguirlo, como prueban las numerosas intrusiones a los servidores de las grandes compañías. Por esto, el valor hash es de protección obligatoria y para ello es fundamental **evitar a toda costa los algoritmos obsoletos**.

Tanto **MD5** como **SHA-1** vienen considerándose inseguros hace ya un tiempo y sus tablas arco iris se encuentran fácilmente en Internet. Con ellas pueden encontrarse claves *hasheadas* con gran facilidad.

Por eso es deber del administrador web mantenerse al día respecto a la existencia de algoritmos nuevos o a la eficacia de la función hash que se ha utilizado. **SHA-2** y su variante más conocida, **SHA-256**, siguen siendo válidas, si bien entretanto ya ha aparecido **SHA-3**, que promete aún más seguridad por más tiempo.

Para dificultar el descifrado con tablas arco iris se utiliza el denominado valor **salt** ("sal" o "semilla"), una secuencia aleatoria que crea el sistema para la clave que define un usuario al abrir una cuenta. Este valor se acopla a la clave y ambos se someten a la función hash, creando así un valor diferente que el que generaría la contraseña sin salt.

Salt y el valor hash se guardan juntos en la base de datos, lo que puede dar lugar a cierta confusión por la facilidad con que un atacante podría obtener de una vez nombre de usuario, valor hash y salt si consiguiera acceder a ella. Y, de hecho, este método realmente no protege contra los ataques de fuerza bruta y de diccionario, pero **sí ante las tablas de colores**, puesto que estas se crearon antes que la base de datos y de forma independiente a ella. Como su creador no los conocía, los salts no pueden figurar en ellas.

Otra ventaja de salt es que los usuarios no tienen que recordarla, por lo que puede ser todo lo larga o caótica que sea posible. Esto ocasiona un valor hash complejo que dificulta trabajar con él. El valor salt también impide que una contraseña genere siempre el mismo valor hash aunque la utilicen dos personas diferentes. Del mismo modo, si un usuario utiliza la **misma clave para varios servicios**, salt evita que se guarde siempre el mismo valor hash en diferentes páginas.

Además de salt también está **pepper** ("pimienta"). Este valor participa dificultando el ataque de fuerza bruta o de diccionario. También constituye una secuencia aleatoria que, sobre todo si se acompaña de salt, entra en el valor hash junto con la clave. A diferencia del salt, **pepper** no se almacena junto con los otros datos de acceso en la base de datos, sino separado y en un lugar más seguro.

A menudo se utiliza como pepper una cadena de caracteres fija para todas las claves de la plataforma, de modo que no protege si varios usuarios poseen la misma clave, porque utilizan entonces el mismo pepper, lo que de nuevo conduce a un valor hash idéntico. En consecuencia, los administradores deberían decantarse por una **combinación de salt y pepper**.

04.10.19 | Seguridad (servidores/seguridad/)

Cifrado (tags/cifrado/) Seguridad (tags/seguridad/)

Artículos similares