hackinglethani.com

El arte de romper un hash (HashCat) » Hacking Lethani

Lethani

10-13 minutos

En este otro post hablé sobre las contraseñas y las condiciones que deben cumplir para ser consideradas seguras. Sin embargo, si introducimos la contraseña más segura del mundo en un sitio inseguro, un atacante que tenga acceso a la base de datos podrá obtenerla independientemente de lo larga o complicada que sea.

Entonces, surge la pregunta de **cómo guardar las contraseñas de los usuarios de un sitio web de forma segura en una base de datos.** La respuesta es que la forma más segura de hacerlo es no guardarlas. En su lugar, se guarda una referencia a ellas, es decir, un **hash**. Para entender esto, adentrémonos más en el concepto de hash.

Un hash es una **función matemática unidireccional.** Esto quiere decir que **a partir de una contraseña podemos obtener un hash, pero a partir de un hash no podemos obtener una contraseña.**

Además, un hash debe cumplir otras reglas:

- Dos contraseñas distintas no pueden generar el mismo hash (a este fenómeno se le denomina colisión)
- Dos contraseñas muy parecidas deben generar hashes completamente distintos.
- Debe ser determinista: una contraseña siempre genera el mismo hash.

Existen varias funciones distintas para generar hashes, y algunas son más seguras que otras.

Pongamos un ejemplo con la función hash SSHA:

- La contraseña de Juan es “love123” y genera el hash {SSHA}vyQJOrcIff42xrwA+XNJZVFunZEvRpuz
- La contraseña Ana es “love124” y genera el hash {SSHA}WIeFEISf85FIn9RrAMSljMiKsDRseWAE

Volviendo a nuestra problemática, si guardásemos esta contraseña

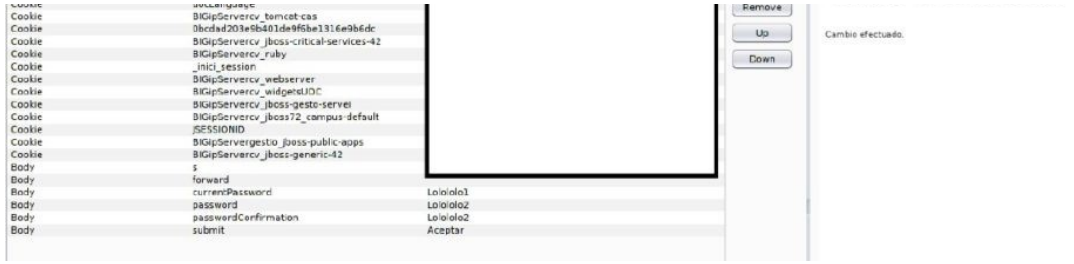
en claro, y un atacante entra en la base de datos, descubriría que la contraseña es “love123”. Pero si no guardamos las contraseñas, no podremos comprobar si Juan es quien dice ser.

Sin embargo, si guardamos {SSHA}vyQJOrclff42xrwA+XNJZVFunZEvRpuz, y un atacante consigue obtener este hash, **seguirá sin saber cual es la contraseña de Juan.** Y para verificar que Juan es quien dice ser, cuando escriba su contraseña, se calculará el hash ssha(“love123”) y **se contrastará este hash resultante con el guardado** en la base de datos: si coinciden, la contraseña es correcta.

A modo de anécdota, hace un tiempo me alarmé porque quise cambiar mi contraseña en la página web del master de ciberseguridad que estoy haciendo, y me apareció el siguiente mensaje:

¿Cómo es posible que me indicase que **la nueva contraseña es similar a la actual**? Si guarda el hash, un solo caracter cambia completamente la salida del hash como hemos visto anteriormente. La única opción posible era que **guardase las contraseñas en claro**.

Pero antes de sacar conclusiones precipitadas, decidí pasar la petición por **BurpSuite**. Y, para gran alivio de todos los estudiantes de esta universidad, pude comprobar que **la validación se hacía en local con Javascript**, puesto que también introduces tu contraseña actual, por lo que simplemente comprobaba que los dos campos del formulario fueran parecidos. Desde Burp podía poner dos contraseñas similares y no tenía problema:



Cómo romper un hash: Hashcat

Hasta aquí la teoría. Ahora vayamos al hacking. Una de las herramientas más utilizadas a la hora de romper un hash es hashcat. Las principales opciones de esta herramienta son:

- **-m:** Tipo de hash a romper. Hashcat puede romper varios tipos de hashes, y dependiendo de cual sea y el formato en el que esté, deberemos escoger un modo u otro.
- **-n:** Número de hilos.
- **-a:** 0 para ataque por diccionario, 1 para combinaciones, 3 para fuerza bruta…
- **-o:** fichero donde guardar los resultados.
- **-O:** optimiza la salida
- **-S:** guarda el progreso en una sesión.

Este tipo de ataque es un **ataque por máscaras**. Hashcat tiene un

conjunto de reglas por defecto, llamadas máscaras, que indican la forma que tienen las contraseñas que vamos a generar. Sin embargo, lo más interesante es crear tu propia máscara.

Cada posición de la máscara puede estar formado por un caracter o por un conjunto de caracteres. Hashcat define 8 conjuntos de caracteres:

- **?l** : abcdefghijklmnopqrstuvwxyz
- **?u** : ABCDEFGHIJKLMNOPQRSTUVWXYZ
- **?d** : 0123456789
- **?h** : 0123456789abcdef
- **?H** : 0123456789ABCDEF
- **?s** : !”#$%&'()*+,-./:;<=>?@[\]^_`{|}~
- **?a** : ?l?u?d?s
- **?b** : 0x00 – 0xff

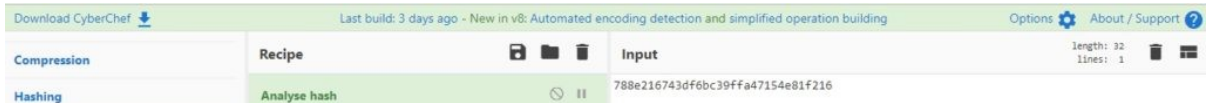
Además, podemos fabricar hasta 4 conjuntos de caracteres propios con la opción **–custom-charset**. Por ejemplo, poniendo **–custom-charset1 ?d?l?u** estaríamos indicando que al poner **?1** ese caracter puede ser un número, una letra minúscula o una letra

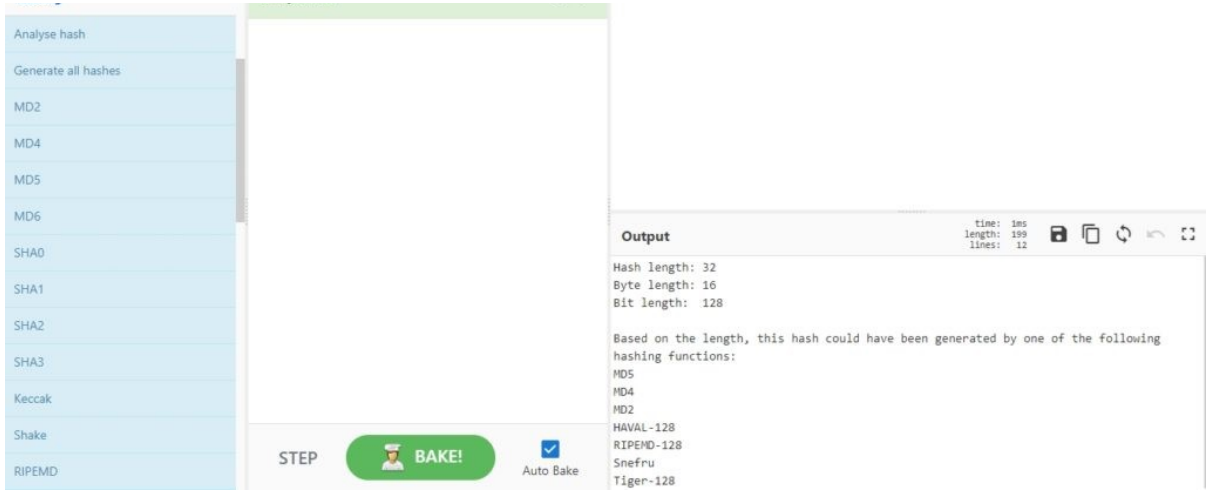
mayúscula.

PoC

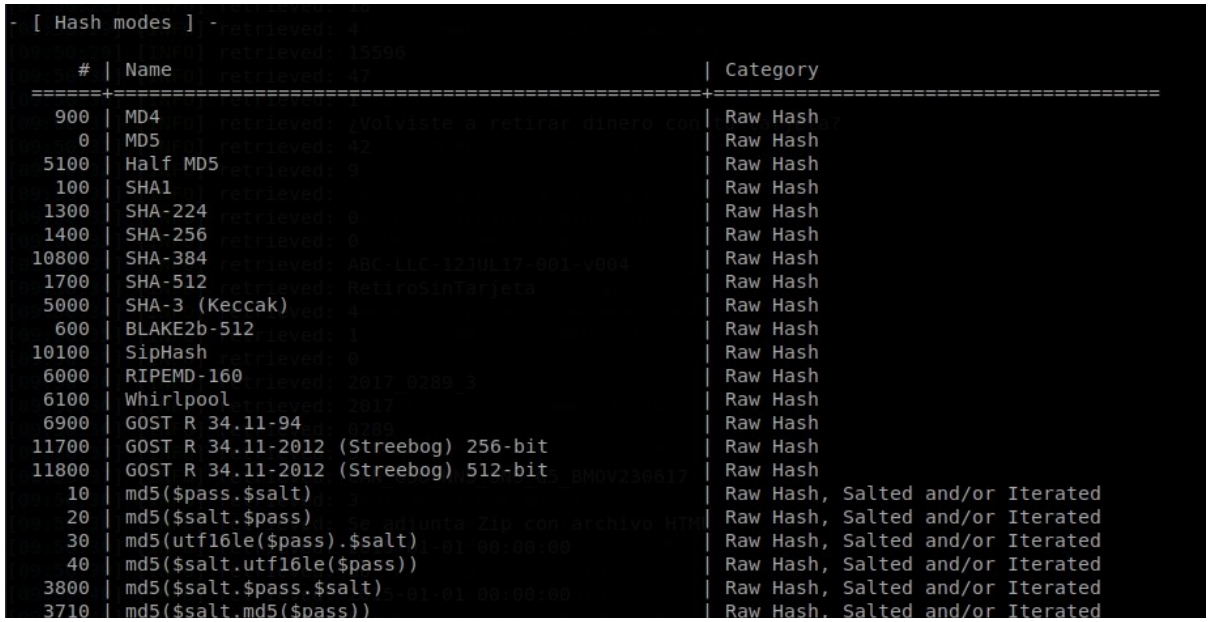
Vamos a realizar una prueba de concepto. Imagina que quieres averiguar la fórmula de la Coca Cola, a la que solo puede acceder el administrador de la empresa. Llevas meses planeando el ataque. Has conseguido que te contraten en la empresa, y has podido entrar en la máquina de un encargado y has obtenido el hash de su contraseña de acceso al servidor central: **1ea9bda0289b5efa8e4a442a87d83d10**. Mediante **ingeniería social**, has utilizando la técnica de **shoulder surfing** mientras el encargado escribía sus credenciales, para descubrir que su contraseña empieza por P y otra letra que no has visto, y el resto es una combinación de 6 números que no has conseguido averiguar, pero que sabes que son números porque utilizó el teclado numérico.

Bien, pues lo primero que debemos hacer es averiguar **de que tipo es el hash** que hemos obtenido. Hay varias formas de hacer esto. Una de ellas es utilizar CyberChef, con la opción de Hashing **“Analyse Hash”:**

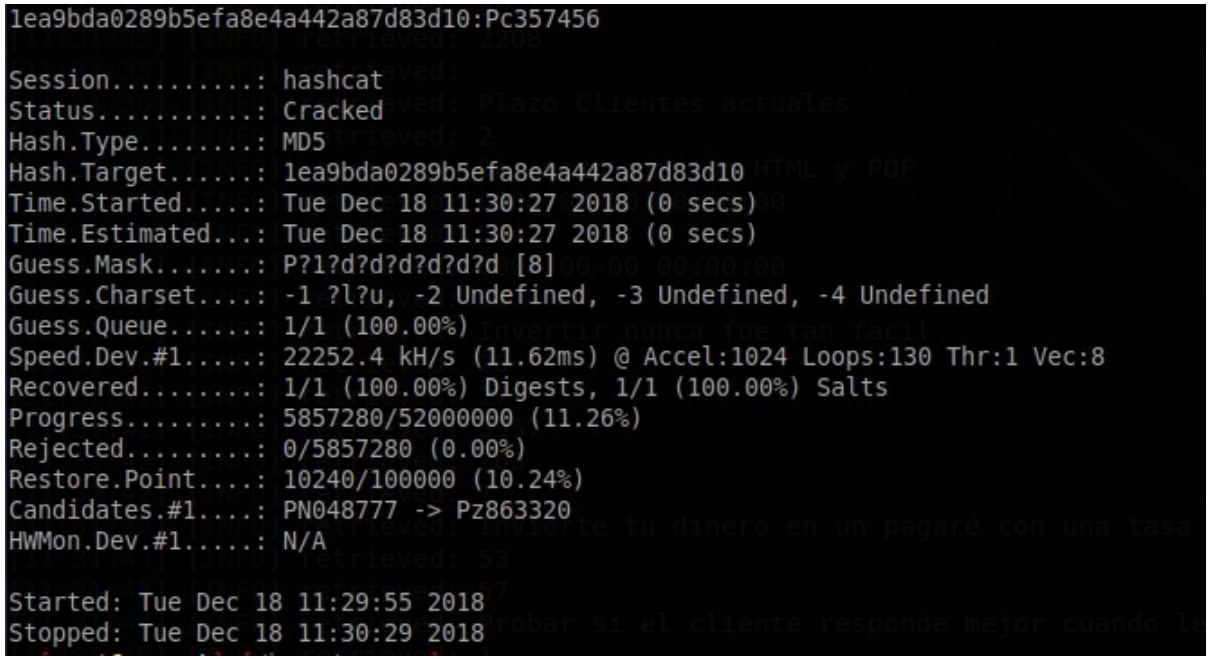




Parece que el hash es del tipo **MD5**. Para indicarselo a hashcat, hacemos un **hashcat –help** y vemos los distintos modos de ejecución que hay. En este caso, el **modo 0** es el que corresponde a MD5, pues tenemos el hash tal cual, sin salt:



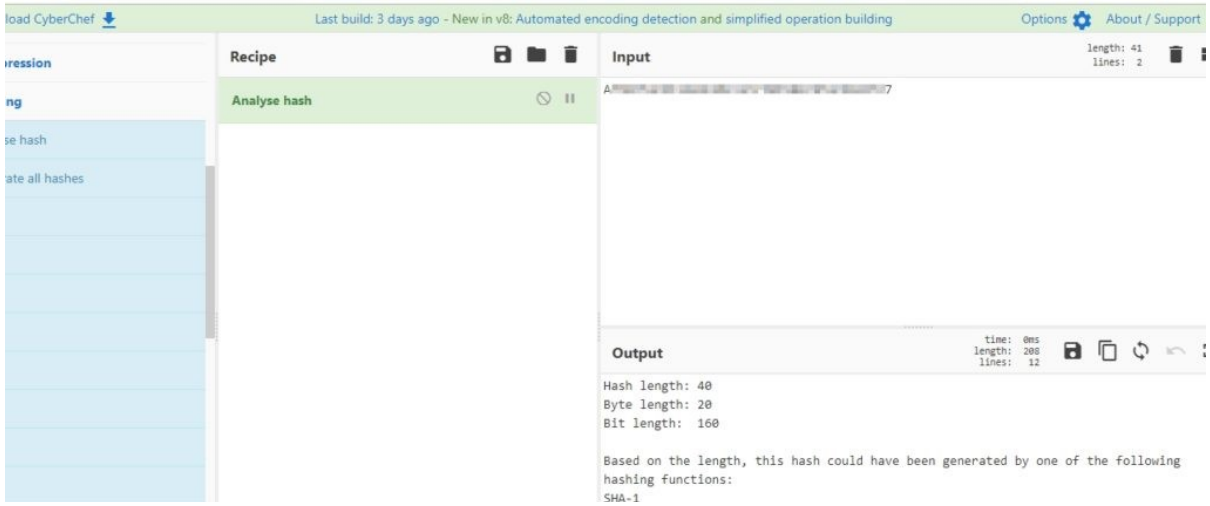
Por tanto el comando de hashcat sería **hashcat -m 0 1ea9bda0289b5efa8e4a442a87d83d10 -a 3 –custom-charset1 ?l?u P?1?d?d?d?d?d?d -O** (modo MD5, el hash, tipo de ataque fuerza bruta, ?1 es minusculas o mayusculas, máscara: primero una P, después una mayuscula o una minuscula, y después 6 numeros, optimizado). Además, en caso de no tener Intel OpenCL deberemos ejecutar el comando con **–force.** Lo ejecutamos, y en unos segundos obtenemos la contraseña:

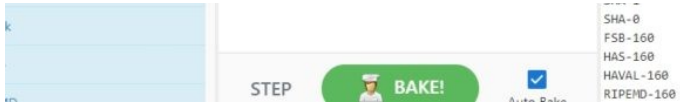


La contraseña es **Pc357456**. Ahora ya podremos acceder al servidor central.

Hemos conseguido acceder al servidor central, lo que nos da acceso a una web de administración. Sin embargo, el encargado no tiene suficientes privilegios como para acceder al archivo que contiene la fórmula de la Coca Cola. Así que has explorado el sitio web un rato y has descubierto que mediante un ataque de inyección SQL puedes obtener toda la base de datos. En la tabla de usuarios, has encontrado la contraseña del administrador, pero de nuevo está **hasheada**.

Cuando compruebas de nuevo en CyberChef qué tipo de hash es, descubres que se trata de **SHA1** (he decidido ofuscar el hash dado que se trata de una contraseña real):





Esto son palabras mayores, ya no se trata de un simple MD5. Vamos a tener que personalizar más el comando de hashcat. Vamos a suponer que el administrador habrá utilizado una **clave de entre 4 y 8 caracteres**, que es lo más común. Por tanto, tenemos que añadir la opción **–increment**, para que vaya aumentando la longitud de las combinaciones, y la opción **–increment-min=4,** para que la longitud mínima sea 4. La máscara será **?a?a?a?a?a?a?a?a**, puesto que entendemos que la clave de un administrador contendrá mayúsculas, minúsculas, números y símbolos (aunque lamentablemente para la seguridad de los sistemas, en muchas ocasiones no es así). Como es SHA1, utilizaremos el **tipo 100**, con el modo de fuerza bruta. De momento, el comando nos quedaría así:

hashcat –increment –increment-min=4 -m 100 -a 3 hash.txt ?a?a?a?a?a?a?a?a

Añadamos algunos parámetros más que nos pueden ser de utilidad. Con **-w** le indicamos lo exhaustivo que puede ser utilizando recursos del ordenador o servidor donde lo vayamos a ejecutar, siendo 1 el modo más bajo, y 4 el más alto, que puede

provocar que el escritorio no responda. Lo pondremos a 3, nivel alto. Con **-D** indicamos el tipo de dispositivo que utilizará para crackear las contraseñas. Si disponemos de un dispositivo con GPU, obtendremos mejor rendimiento, por lo que podemos indicarle las opciones 1 y 2, para que utilice tanto CPU como GPU. Con **-o** le indicamos el archivo de salida donde debe guardar la password crackeada.

Por último, dado que es normal que un proceso de hashcat tarde horas o incluso días, vamos a t**omar algunas precauciones para salvaguardarnos de perder todo el progreso** en caso de que el proceso muera, o la conexión al servidor se pierda, o se apague el ordenador.

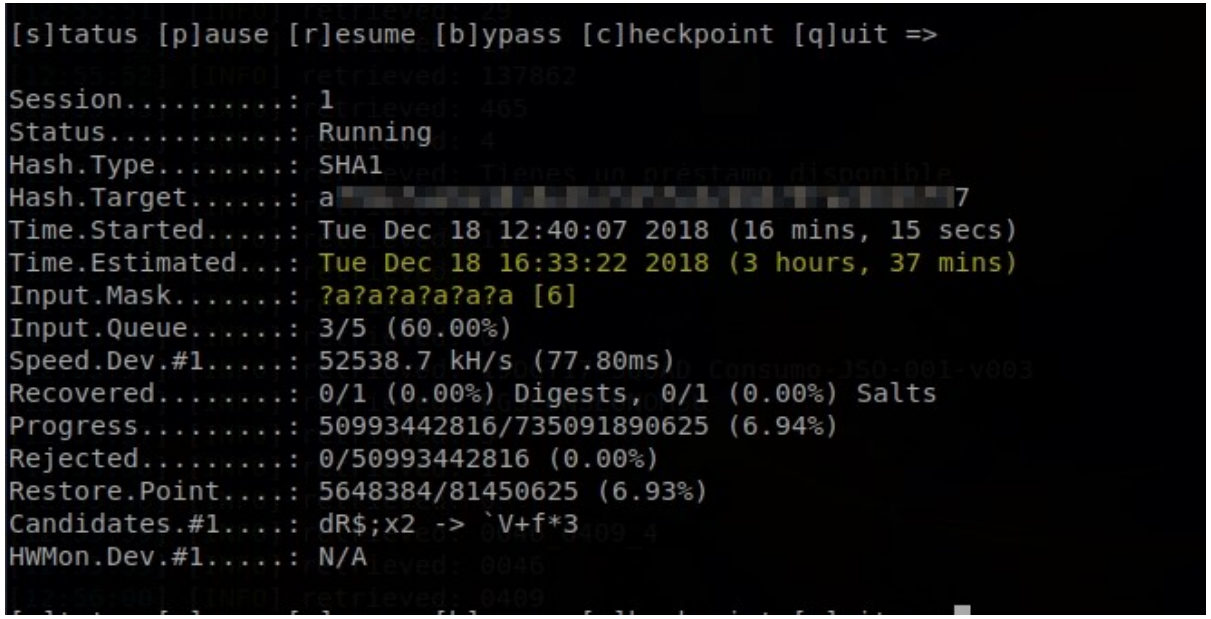
Para ello, ejecutaremos también el parámetro **–session 1,** que nos permite retomar una sesión que se ha abortado guardándola con el nombre “1”. Además, emplearé el comando **nohup**, que nos asegura que el comando que se escriba a continuación se seguirá ejecutando, aunque se bloquee el equipo, y lo ejecutaremos en background poniendo un & al final. Añadiremos el comando **–status** de hashcat, para que automáticamente vaya indicando el estado.

Y si por cualquier motivo debemos abortar el proceso, después podremos recuperar el progreso mediante el comando **hashcat –status 1 –restore.**

Finalmente, el comando quedaría así:

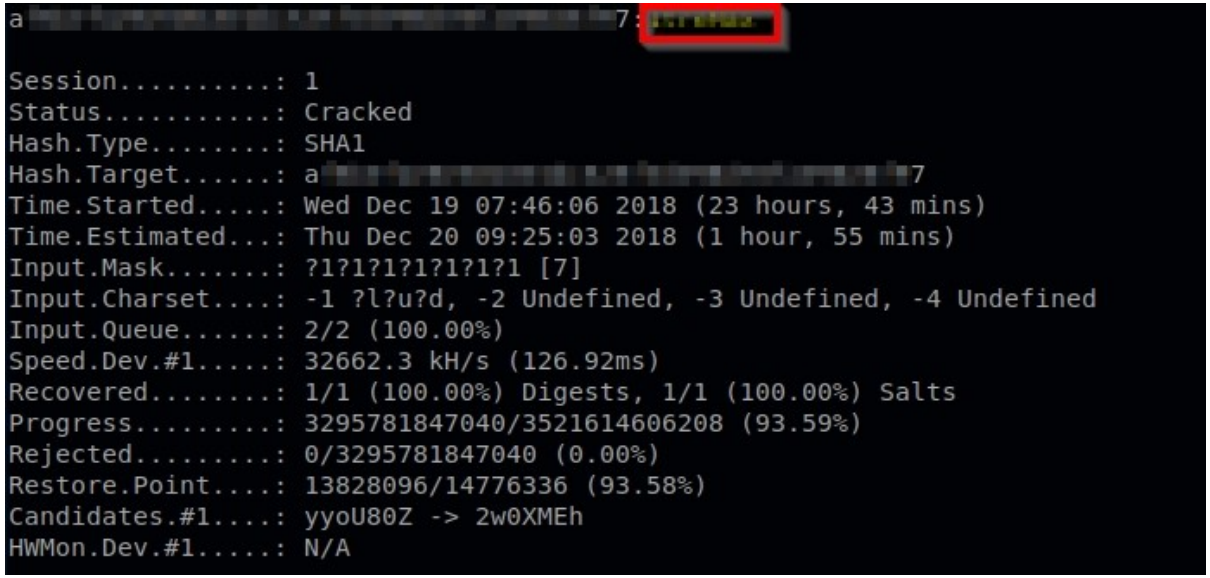
nohup hashcat –increment –increment-min=4 -m 100 -a 3 -w 3 -D 1,2 –session 1 -o dick hash.txt ?a?a?a?a?a?a?a?a –force –status &

El comando **nohup** descarta la entrada y añade la salida del comando que se ejecute a **'nohup.out'.** Por tanto, podemos hacer un **tail -f nohup.out** y nos irá mostrando cómo va el crackeo, con la seguridad de que pase lo que pase se va a seguir ejecutando.



Lo más interesante del **status** son las dos lineas que he resaltado en amarillo. **Input.Mask** nos indica que está probando contraseñas de **longitud 6**. Esto significa que ha probado todas las combinaciones de las contraseñas de longitud 4 y 5 y no lo ha encontrado. **Time.Estimated** indica que calcula que quedan unas **3 horas y media para terminar con las contraseñas de longitud 6** y empezar con las de longitud 7.

Con un poco de paciencia y un servidor potente, finalmente puedes **crackear un hash SHA1 con hashcat:**

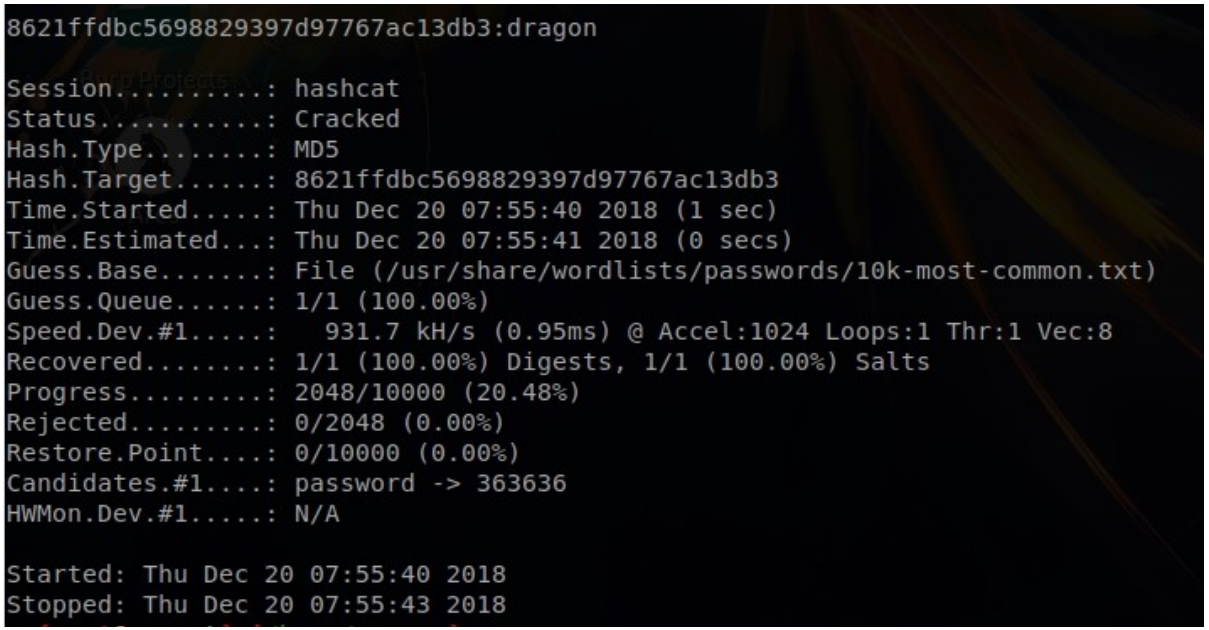


¡Por fin podremos hallar la formula de la Coca Cola!

Además de los ataques de fuerza bruta, hashcat ofrece la opción de realizar **ataques por diccionario**. Este tipo de ataques consisten en introducir una lista de palabras e ir realizando el hash correspondiente de esas palabras para ver si en algun caso coincide con el hash que queremos crackear. Si coincide, entonces esa palabra es la que estamos buscando.

Para ello, solo tenemos que utilizar el **modo -a 0** e indicarle la ruta del diccionario. Veamos un ejemplo sencillo con un MD5:

hashcat -m 0 -a 0 8621ffdbc5698829397d97767ac13db3 /usr/share/wordlists/passwords/10k-most-common.txt –force



El problema de los ataques por diccionario es que, si la palabra no está en el diccionario, el intento de crackeo fracasará, y si el diccionario es demasiado extenso, el ataque tardará mucho tiempo.

Existen otras herramientas para realizar fuerza bruta y ataques de diccionario como puedan ser hydra o John the Ripper. En futuras entradas realizaré una comparación entre estas herramientas, mostrando similitudes y diferencias.

Lethani.