MANUAL DE USUARIO

V. MEXICANH

ENERO 2013

COMPILED BY MEXICANH TEAM

INDICE

AVISO LEGAL

1. Todos los derechos del programa son exclusivos del autor, Atom.
2. Este programa es para uso personal, no comercial y para propósitos legales en ambientes no comerciales y de negocios. El uso de este software en ambientes gubernamentales, de negocios y corporativos esta estrictamente prohibido sin un consentimiento expresamente escrito por el titular de los derechos del programa.
3. Usted usara este software únicamente con hashes creados por usted.
4. ESTE PROGRAMA SE DISTRIBUYE "TAL CUAL". NINGUNA GARANTIA DE NINGUN TIPO ESTA EXPRESADA O IMPLICADA. USE ESTE SOFTWARE BAJO SU PROPIO RIESGO. EL AUTOR NO SERA RESPONSABLE POR LA PERDIDA DE DATOS , DANHOS, PERDIDA DE BIENES U OTRO TIPO DE PERDIDA POR EL USO O MALUSO DE ESTE SOFTWARE.
5. Si las leyes de tu país no permiten las restricciones en (4) usted tendrá que obtener una licencia individual y escrita por el titular de los derechos del programa para usar este software. Si usted no posee tal licencia, no podrá usar este software.
6. No está permitido que usted distribuya este programa.
7. Usted no puede usar, copiar, emular, clonar, rentar, arrendar, vender, modificar, descompilar, o de otra manera hacer ingeniería inversa a este programa o subconjunto del mismo, excepto al que se provee en el acuerdo de licencia. Cualquier uso no autorizado resultara en una inmediata y automática terminación de esta licencia y podrá terminar en un enjuiciamiento penal o civil.

ACERCA

La familia de software hashcat es un conjunto de herramientas profesionales sin cargo para la comunidad. Hashcat está destinado a ser usado LEGALMENTE como una herramienta para recuperar cadenas de texto plano para una variedad de métodos de cifrado tales como:

- MD5
- SHA1
- MySQL
- phpass, MD5(Wordpress), MD5(phpBB3)
- md5crypt, MD5(Unix), FreeBSD MD5, Cisco-IOS MD5
- MD4
- NTLM
- Domain Cached Credentials, mscash
- SHA256
- descrypt, DES(Unix), Traditional DES
- md5apr1, MD5(APR), Apache MD5
- SHA512
- sha512crypt, SHA512(Unix)
- Domain Cached Credentials2, mscash2
- Cisco-PIX MD5
- WPA/WPA2
- Double MD5
- LM
- Oracle 7-10g, DES(Oracle)
- bcrypt, Blowfish(OpenBSD)
- Joomla
- osCommerce, xt:Commerce
- nsldap, SHA-1(Base64), Netscape LDAP SHA
- nsldaps, SSHA-1(Base64), Netscape LDAP SSHA
- Oracle 11g
- SMF > v1.1
- OSX v10.4, v10.5, v10.6
- MSSQL(2000)
- MSSQL(2005)
- EPiServer 6.x
- OSX v10.7
- vBulletin < v3.8.5
- vBulletin > v3.8.5
- IPB2+, MyBB1.2+

La suite Hashcat está bajo constante desarrollo así que mas algoritmos podrán ser añadidos en el futuro

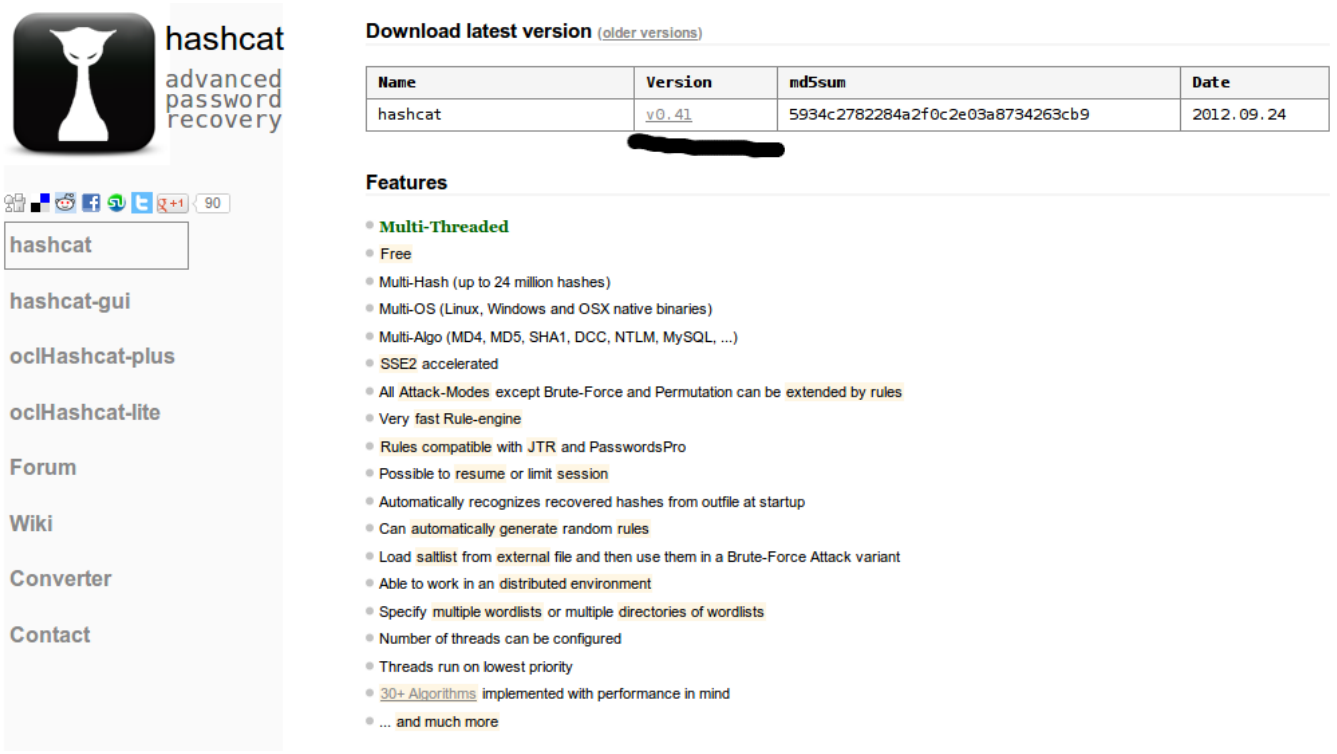
Información adicional en: http://hashcat.net/

1.1 VISTA GENERAL

HASHCAT

La última versión de hashcat la puede obtener a través del sitio web http://hashcat.net/hashcat/

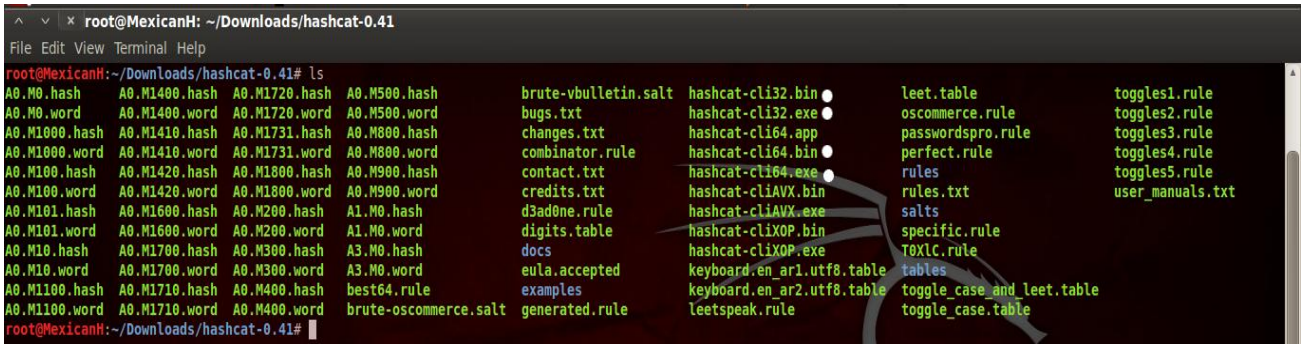
Usted necesitara 7zip para descomprimir el archivo descargado



Hashcat es la herramienta de recuperación de contraseñas más rápida del mundo basada en CPU. No tan rápida como sus contrapartes Oclhashcat-plus,oclhashcat-lite, largas listas pueden ser reducidas a la mitad con un buen diccionario y con un poco de familiaridad con los vectores de ataque. El objetivo de esta sección es presentarle todo lo que tiene que saber para poder para comenzar a correr de manera exitosa hashcat.

Una vez descomprimido habrá una carpeta llamada hashcat-0.41 o hashcat-X.XX donde X representa la versión descargada. Existe también una versión GUI o grafica en Windows, ya que esta versión es mas intuitiva presentaremos poco acerca de ella y nos enfocaremos más a la versión en Linux.

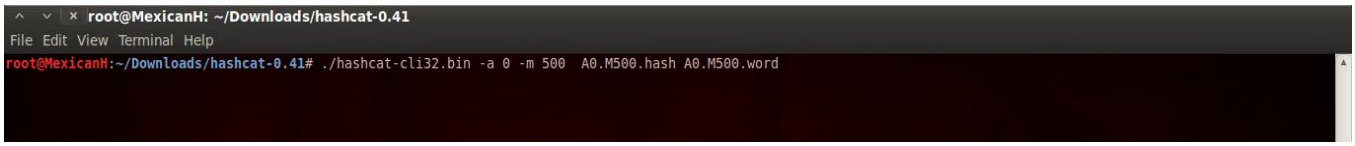
Dentro de la carpeta hashcat-0.41 en nuestro caso veremos una extensa lista de archivos y directorios.



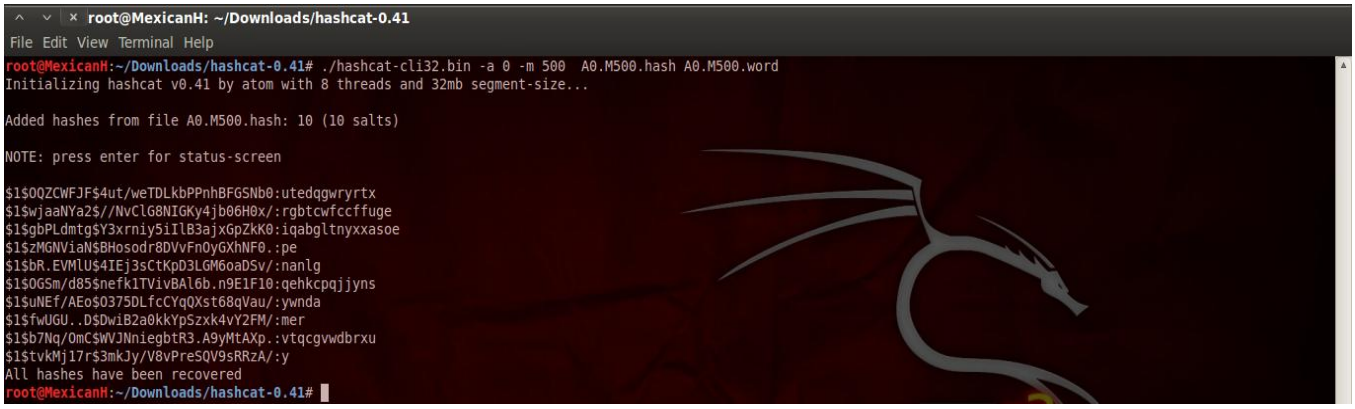
Sus propósitos son los siguientes:

Hashcat-cli(32/64).(bin/exe)- el programa principal. Para sistemas de 32 bits seleccione 32 y para sistemas de 64 seleccione 64. Los usuarios Linux deberán correr el .bin y los usuarios Windows deberán correr el .exe. Los operadores deben ser suministrados de los contrario hashcat no hará nada. En Windows haciendo doble click en el ejecutable solamente abrirá rápidamente y cerrara la línea de comandos DOS. Los usuarios Windows deben correrlo desde la línea de comandos o a través de un .bat

Usando las herramientas suministradas en hashcat hagamos una prueba rápida: D



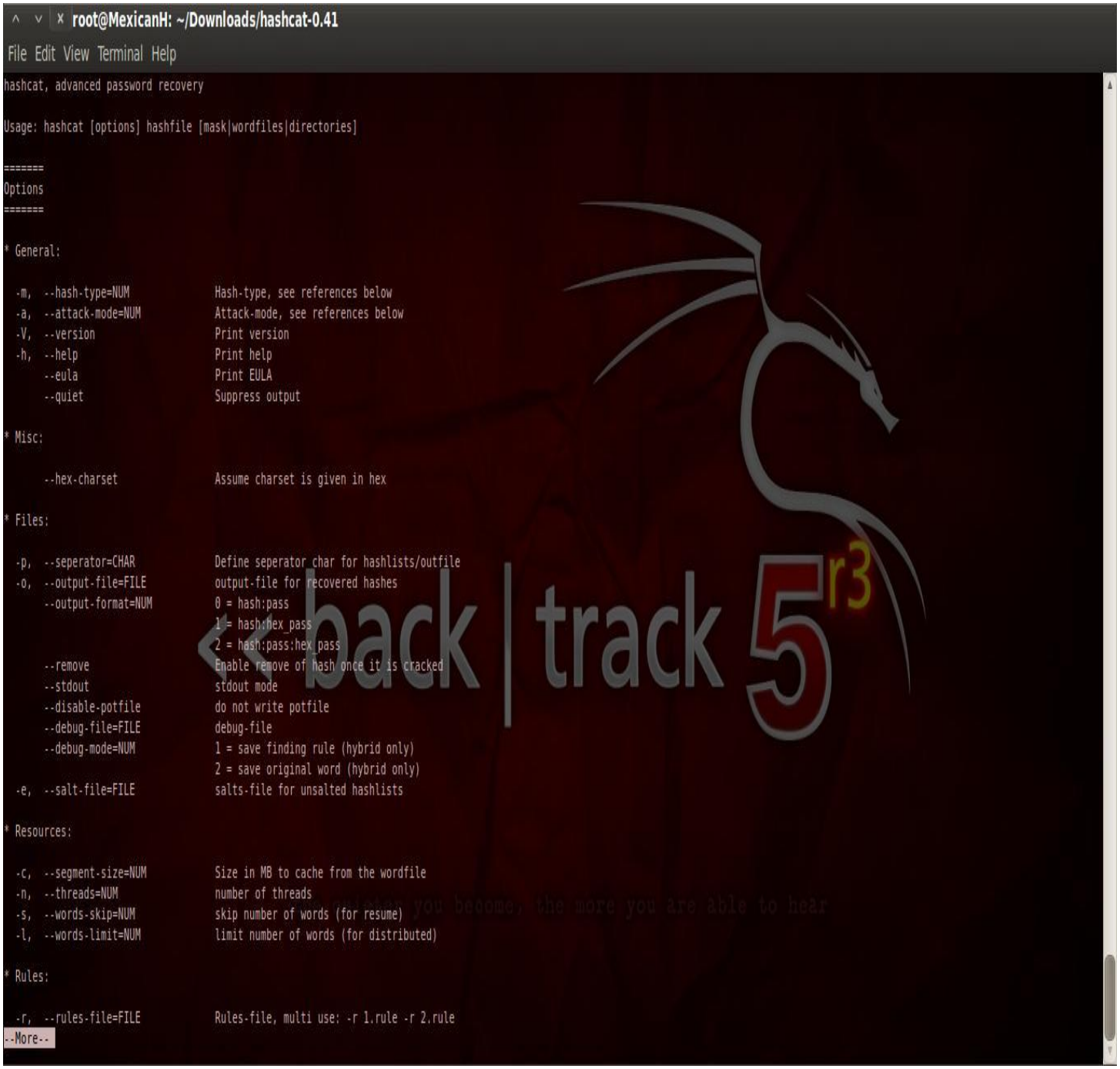
Aquí le hemos dicho a hashcat cargar la lista de hashes A0.M500.hash y usar el diccionario de ejemplos A0.M500.word. Esto debería de darnos un 100% de éxito ya que los hashes UNIX fueron encriptados de las cadenas de texto en el diccionario de ejemplo. Si usted está usando Windows asegúrese de correr hashcat-cli(32/64).exe y omita él "./".



Éxito total: D. Todos los hashes recuperados!!! Sencillo no? Pues se pondrá mejor.....

1.2 OPCIONES

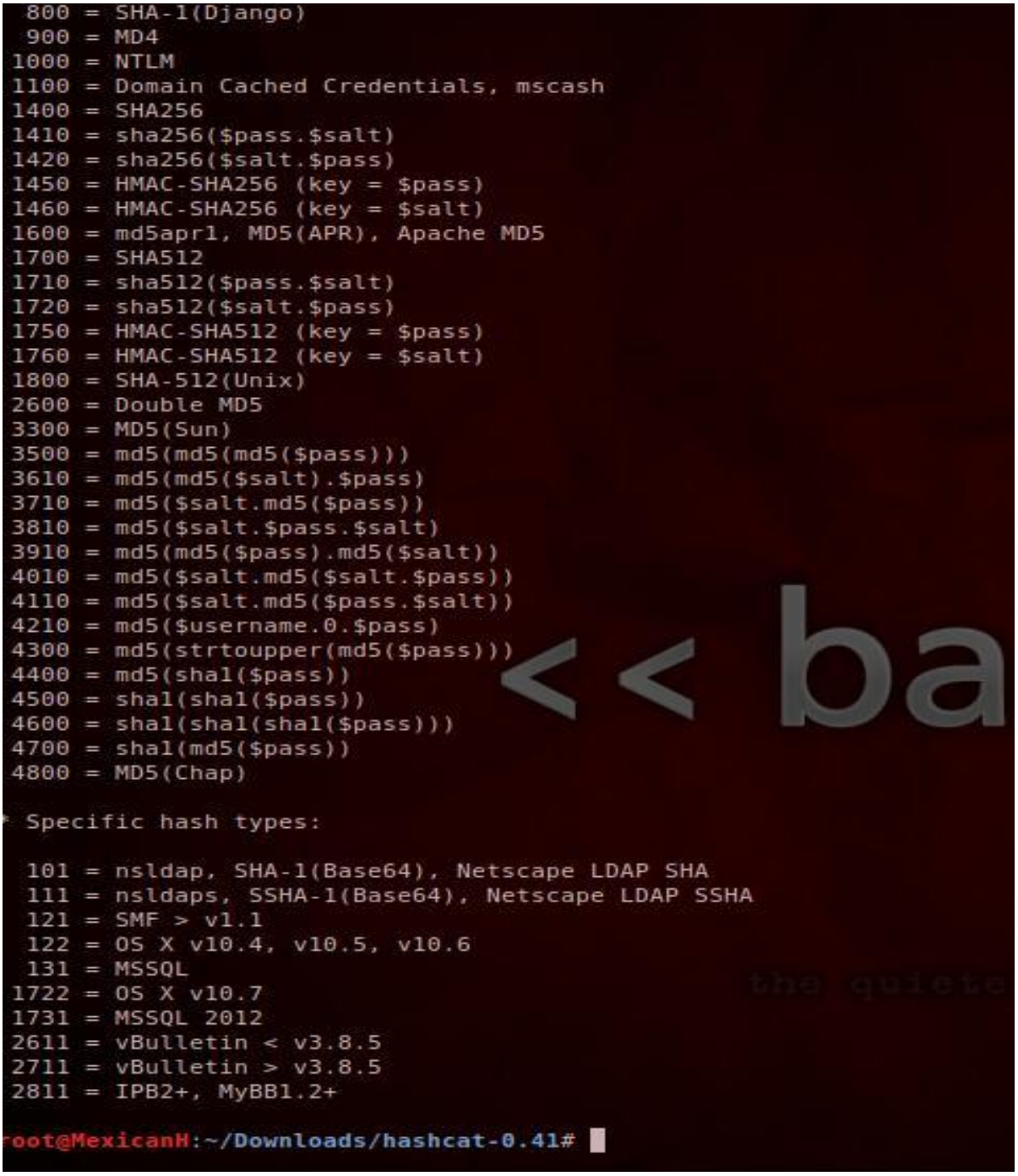
Hashcatcli tiene una gran cantidad de opciones que puede usar para afinar sus ataques.



```
* Rules:

  -r,  --rules-file=FILE            Rules-file, multi use: -r 1.rule -r 2.rule
  -g,  --generate-rules=NUM         Generate NUM random rules
       --generate-rules-func-min=NUM Force NUM functions per random rule min
       --generate-rules-func-max=NUM Force NUM functions per random rule max

* Custom charsets:

  -1,  --custom-charset1=CS         User-defined charsets
  -2,  --custom-charset2=CS         Example:
  -3,  --custom-charset3=CS         --custom-charset1=?dabcdef
  -4,  --custom-charset4=CS         Sets charset ?1 to 0123456789abcdef

* Toggle-Case attack-mode specific:

       --toggle-min=NUM             number of alphas in dictionary minimum
       --toggle-max=NUM             number of alphas in dictionary maximum

* Mask-attack attack-mode specific:

       --pw-min=NUM                 Password-length minimum
       --pw-max=NUM                 Password-length maximum

* Permutation attack-mode specific:

       --perm-min=NUM               Filter words shorter than NUM
       --perm-max=NUM               Filter words larger than NUM

* Table-Lookup attack-mode specific:

  -t,  --table-file=FILE            table file
       --table-min=NUM              number of chars in dictionary minimum
       --table-max=NUM              number of chars in dictionary maximum

==========
References
==========

* Built-in charsets:

   ?l = abcdefghijklmnopqrstuvwxyz
   ?u = ABCDEFGHIJKLMNOPQRSTUVWXYZ
   ?d = 0123456789
   ?s =  !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
   ?a = ?l?u?d?s
--More--
```

```
==========
References
==========

* Built-in charsets:

   ?l = abcdefghijklmnopqrstuvwxyz
   ?u = ABCDEFGHIJKLMNOPQRSTUVWXYZ
   ?d = 0123456789
   ?s =  !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
   ?a = ?l?u?d?s
   ?h = 8 bit characters from 0xc0 - 0xff
   ?D = 8 bit characters from german alphabet
   ?F = 8 bit characters from french alphabet
   ?R = 8 bit characters from russian alphabet

* Attack modes:

    0 = Straight
    1 = Combination
    2 = Toggle-Case
    3 = Brute-force
    4 = Permutation
    5 = Table-Lookup

* Hash types:

    0 = MD5
   10 = md5($pass.$salt)
   20 = md5($salt.$pass)
   50 = HMAC-MD5 (key = $pass)
   60 = HMAC-MD5 (key = $salt)
  100 = SHA1
  110 = sha1($pass.$salt)
  120 = sha1($salt.$pass)
  150 = HMAC-SHA1 (key = $pass)
  160 = HMAC-SHA1 (key = $salt)
  200 = MySQL
  300 = MySQL4.1/MySQL5
  400 = phpass, MD5(Wordpress), MD5(phpBB3)
  500 = md5crypt, MD5(Unix), FreeBSD MD5, Cisco-IOS MD5
  800 = SHA-1(Django)
  900 = MD4
 1000 = NTLM
 1100 = Domain Cached Credentials, mscash
 1400 = SHA256
--More--
```


* Hash types:
0 = MD5
10 = md5($pass.$salt)
20 = md5($salt.$pass)
50 = HMAC-MD5 (key = $pass)
60 = HMAC-MD5 (key = $salt)
100 = SHA1
110 = sha1($pass.$salt)
120 = sha1($salt.$pass)
150 = HMAC-SHA1 (key = $pass)
160 = HMAC-SHA1 (key = $salt)
200 = MySQL
300 = MySQL4.1/MySQL5
400 = phpass, MD5(Wordpress), MD5(phpBB3)
500 = md5crypt, MD5(Unix), FreeBSD MD5, Cisco-IOS MD5
800 = SHA-1(Django)
900 = MD4
1000 = NTLM
1100 = Domain Cached Credentials, mscash
1400 = SHA256
1410 = sha256($pass.$salt)
1420 = sha256($salt.$pass)
1450 = HMAC-SHA256 (key = $pass)
1460 = HMAC-SHA256 (key = $salt)
1600 = md5apr1, MD5(APR), Apache MD5
1700 = SHA512
1710 = sha512($pass.$salt)
1720 = sha512($salt.$pass)
1750 = HMAC-SHA512 (key = $pass)
1760 = HMAC-SHA512 (key = $salt)
1800 = SHA-512(Unix)
2600 = Double MD5
3300 = MD5(Sun)
3500 = md5(md5(md5($pass)))
3610 = md5(md5($salt).$pass)
3710 = md5($salt.md5($pass))
3810 = md5($salt.$pass.$salt)
3910 = md5(md5($pass).md5($salt))
4010 = md5($salt.md5($salt.$pass))
4110 = md5($salt.md5($pass.$salt))
4210 = md5($username.0.$pass)
4300 = md5(strtoupper(md5($pass)))
4400 = md5(sha1($pass))
4500 = sha1(sha1($pass))
4600 = sha1(sha1(sha1($pass)))
--More--
<< back | tra
the quieter you become, the more yo



Si, lo sabemos el número de opciones puede parecer intimidante al inicio, pero esto es solo impacto visual, una vez que usted se acostumbre, serán fáciles de recordar. Muchas de las opciones se explican por sí mismas, así que tómese el tiempo de leer cada una.

Explicaremos las opciones vitales para poder correr hashcat y el resto las explicaremos mas adelante con más detalle.

--hash-type=NUM O –m: El tipo de algoritmo a usar que puede ser MD5, SHA-1 etc.

--hash-type=0 (MD5) O –m 0

--attack-mode=NUM O **-a**: El tipo de ataque a usar en contra de un hash. Usando los diferentes vectores de ataque aumentaran las probabilidades de recuperar la contraseña. Los modos son los siguientes:

- 0 = Straight- simplemente corre todas las palabras del diccionario en contra de la lista de hashes, teniendo un buen diccionario aumentara las probabilidades de recuperar tu hash.
- 1 = Combination – combina las palabras del diccionario dado.
- 2 = Toggle-Case –Cambia todas las letras minúsculas a mayúsculas y viceversa. Dígitos y caracteres especiales son ignorados.
- 3 = Brute-force – Fuerza bruta debe ser usado como último recurso, no es efectivo contra contraseñas largas y puede consumir mucho tiempo.
- 4 = Permutation – Toma las letras de una palabra y las reordena. Ejemplo abc se vuelve abc, acb, bca, bac.
- 5 = Table-Lookup –Rompe una cadena en caracteres individuales y aplica una regla a cada uno que coincida con la regla en la tabla.
  Por defecto el modo 0.

--attack-mode=0 O –a 0 para un simple ataque de diccionario.

--seperator=CHAR O **-p**: Algunas listas de hashes se pueden dar junto con el usuario y su pass encriptado user:33c2a20e201110df4cc723c0a994b4ff en este caso el : es nuestro separador, pueden usarse otros pero por defecto se usa siempre ":".

--output-file O **–o**: especifica donde los hashes rotos serán escritos. Esto debe ser usado si usted planea conservar estos hashes o no desea copiar/pegar desde la terminal. Trabaje inteligente y no más complicado. Por defecto no usado.

--output-file=/user/Desktop/hashes.txt O **–o /user/Desktop/hashes.txt**

--output-format=NUM : NUM puede ser 0, 1 o 2. Generalmente no se necesita pero si en el texto plano hay caracteres en hexadecimal esto necesita ser especificado para prevenir "malos" textos planos. Por defecto Modo: 0.

--output-format=0

--remove : Con esta opción se borrara el hash de la lista una vez que sea crackeado. Esto le ayudara a prevenirle de atacar el mismo hash dos veces. Por defecto no usado.

--stdout : En lugar de tratar de recuperar las contraseñas hashcat simplemente les dará salida en la terminal.

--disable-potfile : Previene que hashcat escriba los hashes rotos en hashcat.pot . Por defecto no usado.

--debug-file=FILE : especifica el archivo en donde la información de depurado será escrita. Por defecto no usado.

--debug-file=/user/Desktop/debug.txt

--debug-mode=NUM : Escribe bien la norma de búsqueda, la palabra original, o la palabra mutada que fue exitosa en contra del hash usado en –debug-file=. Por defecto no usado.

--salt-file=FILE O **–e:** Especifica una lista de "salts" pre-generados para ser usados en una sesión. Esto se usa cuando en un hash con "salt" el "salt" esta ausente. Por defecto no usado.

--segment-size=NUM O **–c** : Especifica la cantidad de memoria en MB que se permitirá en el almacenamiento en cache para el diccionario. Si usted trabaja con una cantidad limitada de memoria deberá ser usado para no interferir con los otros servicios. Con la siguiente opción solo se permitirán 10MB de palabras en cache. Por defecto 32 MB

-c 10

--threads O –n : Para uso en procesadores "multi-threaded". Casi todos los procesadores contienen múltiples núcleos. Si usted tiene un procesador "quad" o de cuatro núcleos entonces, ajuste a –n 4 o –n 6 para uno de seis núcleos. Si usted corre un sistema multi-procesadores, ajuste –n al número de núcleos * numero de procesadores físicos. Por ejemplo un sistema Dual hexacore seria 6 (núcleos) * 2 (procesadores)= 12 (numero de threads) por defecto: 8.

-n 12

--words-skip=NUM O –s : Salta el numero provisto de palabras cuando se resume una sesión detenida. Esto previene de correr palabras contra una lista de hashes de nuevo con lo cual se incrementaría la cantidad de tiempo en un instancia podría tomar. La siguiente opción brincara las primeras 100000 palabras. Por defecto no usado.

-s 100000

--words-limit=NUM O –l : especifica el numero de palabras que serán procesadas. Esto es útil cuando se recupera la misma lista de hashes en diferentes computadoras así la misma computadora no corre palabras que están siendo procesadas por otra. La siguiente opción solo usara las primeras 20000 palabras. Por defecto no usado.

-l 20000

--rules-file=FILE O –r : Especifica el directorio donde se encuentran los archivo de reglas(más adelante se explicara a detalle)

-r /user/Desktop/1.rule

--generate-rules=NUM O –g : Le dice a hashcat que genere un numero de reglas para aplicarse en cada intento, la opción –g 50 le dirá a hashcat crear aleatoriamente 50 reglas sobre la marcha para ser usadas en

esa sesión. Esto puede eliminar la necesidad de largos archivos de reglas, aunque un buen archivo de reglas bien creado puede aumentar las probabilidades de recuperar la contraseña. Por defecto no usado.

-g 50

--generate-rules-func-min/max=NUM : Especifica el numero de funciones que deben ser usadas. Este número puede ser ilimitado pero largas cantidades no se recomiendan. Cuando se usa en conjunto con –g, cualquier regla fuera de este ajuste será ignorada. Por ejemplo –g 50 genera 1 r, 1^f y sa@ todas estas son reglas validas sin embargo 1^f sa@ r $3 será ignorado porque contiene 5 funciones. Por defecto min=1 max=4.

--custom-charset1,2,3,4=CS O -1,-2,-3,-4 : Este es un mapa de caracteres a medida, todos lo derivados de hashcat tienen 4 para crear tu mapa de caracteres a medida. Estos nos sirven en el modo 3 o bruteforce. Si no queremos pasar por todo el mapa de caracteres a-z podemos ajustarlo así -1 abcdef -2 12345 de tal modo al hacer nuestra mascara ?1?1?1?1?2?2 solo abarcara los caracteres indicados y no todo el abecedario y números del 0 al 9.

?1?1?1?1?2?2 – aaaa11- ffff55

--toggle-min/max=NUM : ajustando esta opción le diremos a hashcat esperar un mínimo/máximo de X y Y en el plano. Se puede reducir el tiempo de ejecución al no tomar en cuenta los valores que se encuentran fuera de los requisitos de la contraseña. Por defecto min=1 max=16.

--toggle-min=3 --toggle-max=5

--pw-min/max=NUM : Esto especificara el mínimo y máximo numero de espacio de caracteres cuando se hace el Brute-force o Mass-attack. El siguiente comando intentara todas las combinaciones de caracteres desde 1 a 8 espacios. Solo los espacios comenzando con min y terminando con max serán tomados en cuenta. Por defecto min=1 max=16.

--pw-min =1 –pw-max=8

--table-file=FILE : Especifica que tabla usar con –a 5. Las tablas pueden ser ubicadas en la carpeta Tables o puede crear unas por sí mismo. Por defecto no usado.

--table-min/max=NUM: Cualquier palabra fuera del rango especificado sera ignorada. Por defecto 10.

--perm-min/max=NUM : Cualquier palabra fuera del rango definido sera ignorada. Por defecto min=2 y max=10.

1.3 VECTORES DE ATAQUE

Straight. (Directo).Este ataque simplemente corre una lista de palabras y prueba todas las cadenas contra cada hash. Este es un vector extremadamente efectivo para una primera pasada, asumiendo que usted tiene un buen diccionario como el famoso Rockyou :D

Combination. Junta dos palabras del diccionario dado e intenta con ellas. Puede ser útil en contra de contraseñas largas, la gente trata de añadir complejidad a sus contraseñas escribiéndolas dos veces o personas que usan su primer y segundo apellido. Por ejemplo atom usa su apellido derp para hacer una contraseña, si atom y derp existen en el diccionario, atomderp será usado y también derpatom.

Toggle-case. Simplemente cambia las letras minúsculas por mayúsculas y viceversa. PasS se volvería pASs.

Brute-force. Extremadamente util en GPU, Fuerza bruta con CPU puede tardar para siempre. Este vector tratara cada combinación **--pw-min/max.** Esencialmente trata cada combinación por incrementos hasta que encuentra el texto plano requerido. Altamente poco inteligente y usado como último recurso, excepto en maquinas con un alto poder en GPU.

Permutation. Este vector reordena todas las letras suministradas en el diccionario que coinciden en min/max. Por ejemplo min=1 y max=3 tomara las palabras con una longitud de 1 a 3 y las reacomodara en todas las posiciones posibles. Puede ser efectivo contra generadores aleatorios de contraseñas. por defecto min=2 y max=10.

Table-lookup. Esta función a cambiado un poco con respecto a su antecesor hashcat 0.40 pero aun así es un altamente avanzado vector de ataque que rompe una cadena en caracteres individuales y aplica una regla definida en **table-file=FILE** a cada uno. Por ejemplo password es roto en cada carácter: p a s s w o r d. Entonces hashcat mira en la tabla por las reglas que deben ser aplicadas en cada carácter. En este caso nuestra tabla tendría

a=a

a=A

p=p

p=P

o=o

o=O

o=0 (cero)

Ahora cada caracter que coincida será cambiado y probado. Así que por cada a, a y A será probada, por cada p, p y P será probado y por cada o, o, O y 0 será probado. Para aquellos que están familiarizado con las mascaras sería algo así -1 pP -2 aA -3 oO0 ?1?2ssw?3rd. Luce algo extraño quizás para usted pero o se preocupe esto será explicado en oclhashcat donde es altamente usado.

1.4 EJEMPLOS

Para los siguientes ejemplos usaremos una lista tomada de http://forum.md5decrypter.co.uk/

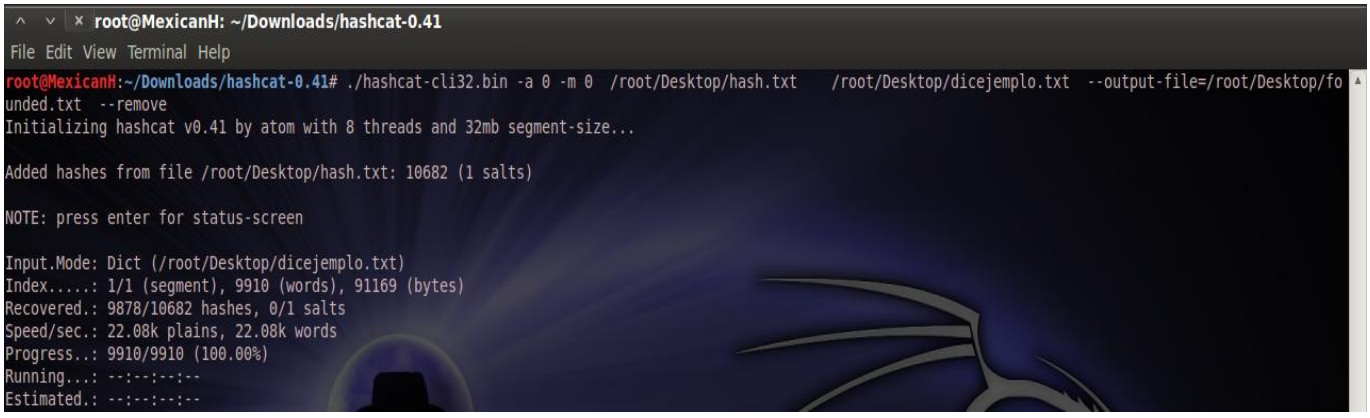
STRAIGHT.

Supongamos que tenemos una lista de aproximadamente unos 10000 hashes D: parece algo imposible. Pero nuestro amigo hashcat nos ayudara en esta difícil misión :D

El comando en la terminal fue ./hashcat-cli32.bin -a 0 -m 0 /root/Desktop/hash.txt /root/Desktop/dicejemplo.txt --output-file=/root/Desktop/founded.txt --remove

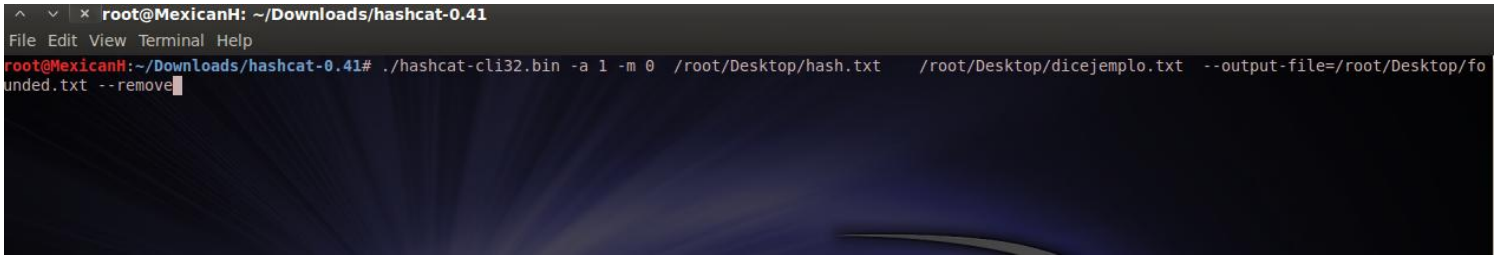
Lo que le dijimos a hashcat fue que usara el modo –a 0 de ataque directo con diccionario con el tipo de algoritmo –m 0 que corresponde a MD5 usando la lista de hashes y el diccionario de ejemplo en los directorios especificados y reescribiendo los hashes recuperados en el archivo indicado. Ahora veamos el resultado.

Un éxito casi total :D el ataque directo recupero 9878 hashes. Esto depende totalmente del diccionario si la palabra no se encuentra el hash no podrá ser recuperado. Como vemos en el diccionario solo hay 9910 palabras y teníamos 10682 hashes. Como haremos para recuperar el resto? Con la demás opciones podremos mutar las palabras y ver si eso nos trae algún resultado.



COMBINATION

Probemos ahora con la lista restante el ataque de –a 1 o de combinación, como antes explicamos si hay contraseñas que coincidan o que sean palabras duplicadas, las encontrara :D



En general lo único que cambio en el comando fue –a 1 lo demás quedo intacto, como ves manejar hashcat no resulta difícil después de todo, el comando escrito queda de la sig. Manera : ./hashcat-cli32.bin -a 1 -m 0 /root/Desktop/hash.txt /root/Desktop/dicejemplo.txt --output-file=/root/Desktop/founded.txt --remove

Ejecutaremos de nuevo hashcat con los nuevos ajustes y veremos qué resultados tendremos

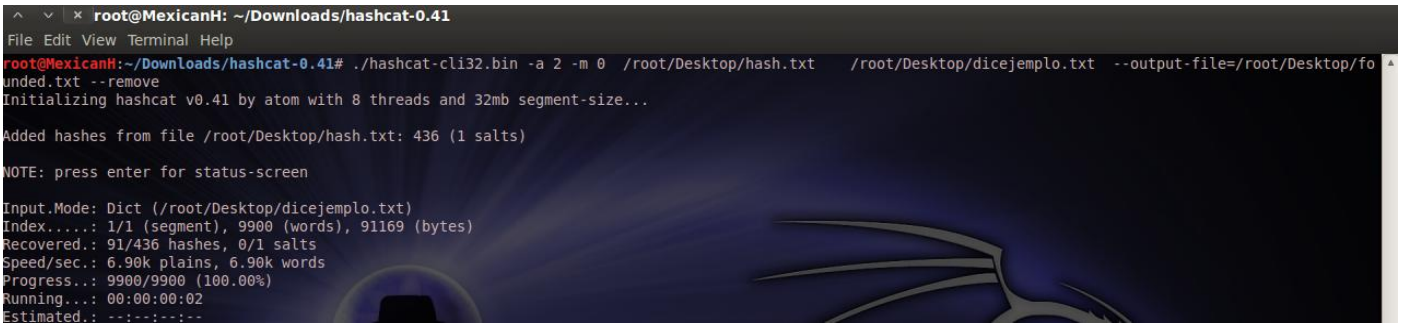
Como podemos ver recuperamos 368 hashes mas y eso con un mismo diccionario, pero aun nos quedan más hashes por recuperar, intentemos otro tipo de ataque con el mismo diccionario y esperemos tener algo de suerte.

TOGGLE-CASE

Usemos ahora el modo –a 2 en el cual las minúsculas son cambiada a mayúsculas y viceversa los parámetros son iguales solo cambiaremos el modo de ataque.

El comando es básicamente el mismo solo con –a 2 de diferencia: ./hashcat-cli32.bin -a 2 -m 0 /root/Desktop/hash.txt /root/Desktop/dicejemplo.txt --output-file=/root/Desktop/founded.txt --remove

Ejecutamos de nuevo hashcat y vemos que nos traerá a la salida.

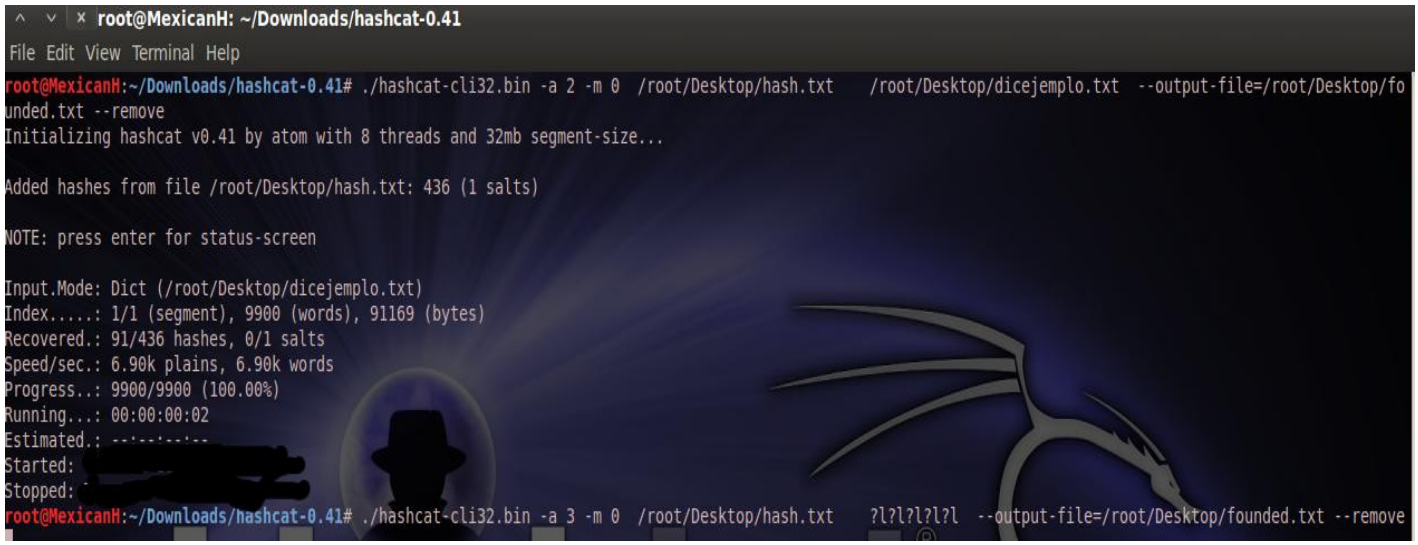


Ahora podemos observar que recuperamos cada vez menos contraseñas, en total 91, esto es algo común, las contraseñas seguras son las que quedan a lo último, o en el peor de los casos, no pueden ser recuperadas XD. Algo más que podemos diferenciar es la cantidad de palabras en el diccionario como no especificamos un máximo **--toggle-max** hashcat tomo por defecto 16, entonces las palabras mayores de 16 no fueron tomadas en cuenta, en este caso fueron 10 palabras excluidas.

Aun quedaron contraseñas sin recuperar, intentaremos con los vectores de ataque restantes.

BRUTE-FORCE

Como se menciono antes la fuerza bruta en cpu es lo último a lo que debemos recurrir a menos que sepamos que la lista tiene un patrón, este puede ser una mayúscula al inicio, tres o dos números al final etc. En este caso no sabemos nada acerca de la lista y probaremos con algo sencillo ya que largas cadenas pueden tomar horas.



Podemos apreciar la diferencia de que ahora no estamos usando el diccionario sino una “mascara” que emulara al diccionario e intentara en un espacio de 5 todas las letras del abecedario. De otra manera la máscara ?l?l?l?l?l hará combinaciones desde aaaaa hasta zzzzz.

El comando en la terminal fue ./hashcat-cli32.bin -a 3 -m 0 /root/Desktop/hash.txt ?l?l?l?l?l --output-file=/root/Desktop/founded.txt –remove

Como resultado solo recuperamos dos contraseñas no fue mucho, ya que si la cadena de texto es menos de 5 o más 5 no podrá ser recuperada. Es una de las limitantes de usar fuerza bruta. Podríamos aumentar la complejidad añadiendo caracteres a medida como -1 ?d?l?s para probar todas las combinaciones de minúsculas, mayúsculas y números pero eso podría tardar una eternidad usando CPU.

PERMUTATION

Este vector lo que hará ser reacomodar las palabras en nuestro diccionario, si tenemos por ejemplo ABC lo convertirá en:

ABC

ACB

BAC

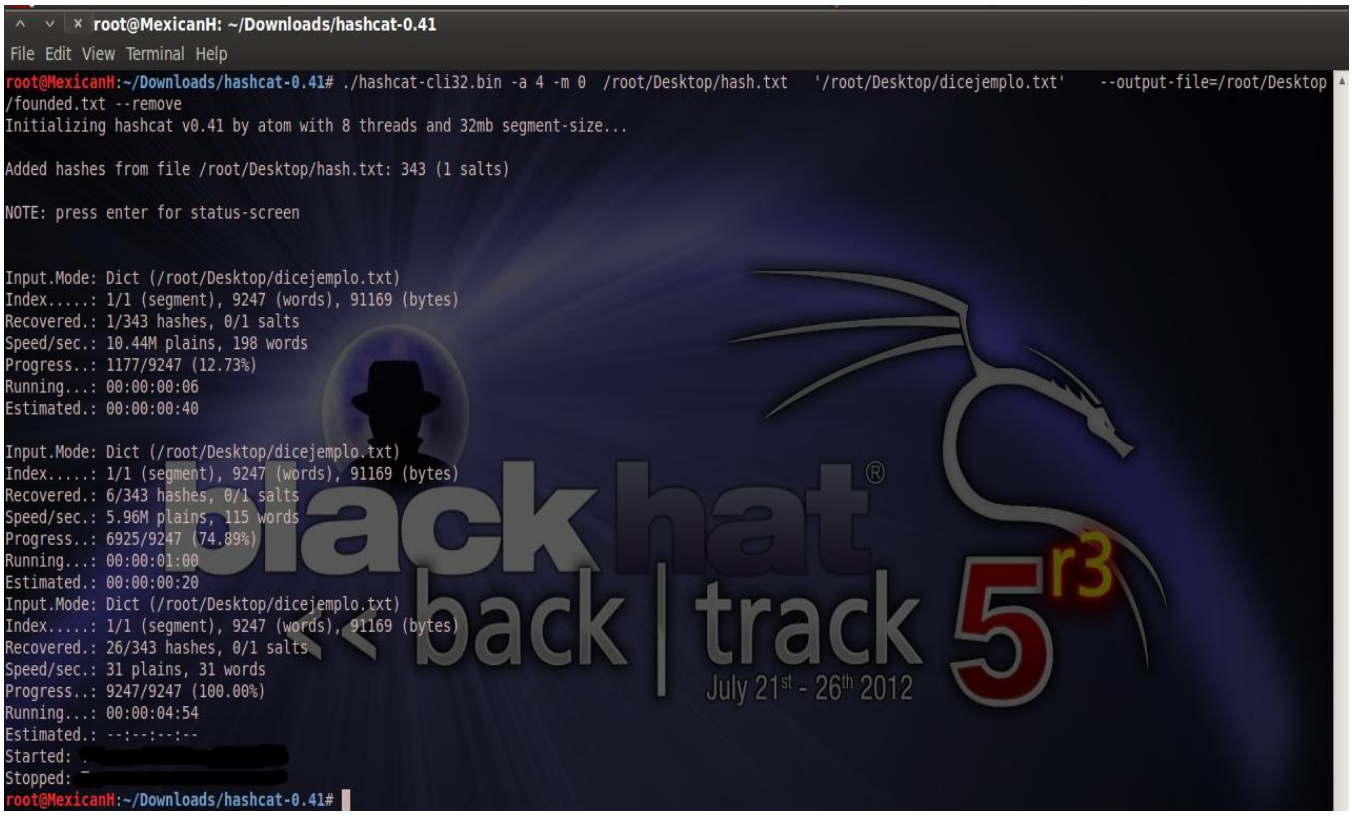
BCA

CAB

CBA

Regresamos a nuestro primer diccionario y usamos el ataque –a 4. El comando escrito fue ./hashcat-cli32.bin -a 4 -m 0 /root/Desktop/hash.txt '/root/Desktop/dicejemplo.txt' --output-file=/root/Desktop/founded.txt –remove

Veamos el resultado más abajo.



Comparado con los demás ataques que realizamos este tomo un poco más de tiempo, pero recuperamos mas hashes que usando el ataque de fuerza bruta, en total 26. Vemos también que la cantidad de palabras usadas fue de 9247 menos que la cantidad total ya que no especificamos el largo máximo, por defecto hashcat toma la cantidad de max=10.

TABLE-LOOKUP

Probemos el ultimo de nuestros vectores de ataque, para este ataque usamos el modo –a 5 y añadimos la opción **--table-file=**'/root/Desktop/tablas.table' donde se encuentran nuestras reglas para la tabla.

a=a

a=A

a=4

@=a

@=@

o=o

o=O

o=0

i=1

i=i

s=S

s=5

s=s

Usamos algo sencillo ya que esto en ocasiones puede llegar a tomar un par horas, usted puede configurar sus reglas de la tabla como más le guste, en este caso usamos algo que la gente usa para escribir sus contraseñas. Sustituir letras por números, mayúsculas y caracteres especiales.

El comando escrito fue ./hashcat-cli32.bin -a 5 -m 0 /root/Desktop/hash.txt '/root/Desktop/dicejemplo.txt' --output-file=/root/Desktop/founded.txt --remove --table-file=/root/Desktop/tablas.table --table-min=1 --table-max=20

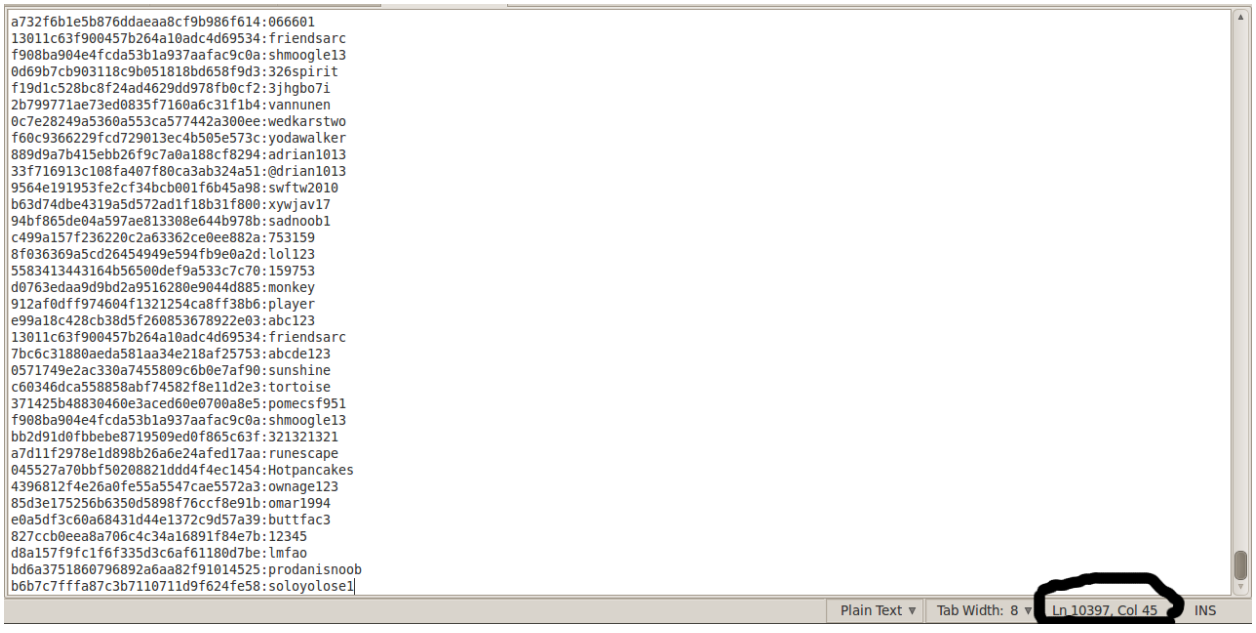
Añadimos las opciones **--table-min=1 --table-max=20** para aumentar nuestras probabilidades, ya que no sabemos el largo de todas las palabras en nuestro diccionario y hashcat toma por defecto max=10

Ejecutemos y veamos.

Como podemos ver recuperamos una buena cantidad, en total 32 más. Pero observamos que las contraseñas bien pensadas y seguras son las restantes, podrían ser rotas usando otro diccionario y aplicando los vectores anteriores o usando reglas pero eso lo veremos en oclhashcat. En la cantidad de palabras vemos que siguen siendo menor a la cantidad original del diccionario, por distintas razones, hay palabras de más de 30 caracteres, o algunas no encajaron con las reglas que escribimos en la tabla.

En total quedaron 285 hashes sin romper, pero de una cantidad de 10682 a 285 es una buena cantidad :D

Demos una mirada a nuestro archivo de contraseñas recuperadas.



Si hacemos matemáticas 10682-285=10397.

1.5 ESPECIFICOS HASHCAT

Hascat es especialmente bueno para reducir largar listas de hashes usando métodos rápidos como el ataque de diccionario (–a 0) O el toggle-case(–a 2) y el resto. Aunque es la herramienta de recuperación basada en CPU mas rápida del mundo, a hashcat le tomaría años romper un hash con una larga cadena de texto usando Brute-force (-a 3). Aqui es donde oclhashcat entra en juego. Los usuarios con una tarjeta grafica con posibilidades GPGPU (General-Purpose Computing on Graphics Processing Units) pueden además usas hashcat.

2.1 OCLHASHCAT-PLUS(CUDAHASHCAT-PLUS)

La ultima versión de hashcat está disponible para su descarga desde http://hashcat.net/oclhashcat-plus/ . 7zip es un requerido para descomprimir el archivo. Oclhashcat es la herramienta de recuperación de contraseñas basada en GPU más rápida del mundo, codeada por Atom para ambas plataformas GUN/Linux y Windows de 32 y 64 bits. Este es un gran avance desde las soluciones basadas en CPU debido a la gran cantidad de dato que se pueden procesar usando las plataformas de CUDA (NVIDIA), Stream (AMD) y OpenCL.

Lo que tomaría horas en un CPU ahora puede tomar minutos en un GPU y aquí tenemos como hacerlo :D

El proceso de instalación es el mismo, si tienen dudas dirígete de nuevo a la sección 1.1

Unas ves descomprimidas veremos unos archivos y unas carpetas, daremos una explicación rápida y después iremos de lleno a oclhashcat.

Una nota rápida: Atom ha divido oclhashcat en 2 ejecutables, el primero es oclhashcat para usuarios con tarjetas AMD(ATI). Esto requerirá que tengan los drivers y el SDK instalado. Aquí no veremos cómo instalar los drivers daremos hecho que ya lo tienen instalados. Para instalarlos les daremos los links de referencia:

http://www.backtrack-linux.org/wiki/index.php/Install_OpenCL

http://ati.amd.com

Los otros dos principales ejecutables son cudahascat que es para lo usuarios con NVIDIA. Para NVIDIA solo necesitamos el forceware driver.

http://www.backtrack-linux.org/wiki/index.php/CUDA_On_BackTrack

Por simplicidad nos referiremos a ambos ejecutables como oclhashcat y los ejemplos serán corridos todos usando oclhashcat. Para los usuarios Nvidia, tienen que correr cudahashcat o tendrán un error. Tampoco hay diferencia entre los ejemplos corridos en oclhashcat y cuda hashcat así que no tienen de que preocuparse.

oclHashcat(32/64).(bin/exe)- es el ejecutable principal

oclExample.(sh/cmd)- es un archivo de ejemplo para actuar como un inicio rápido para oclhashcat, será usado como ejemplo, pero después de leer un poco mas no lo necesitara.

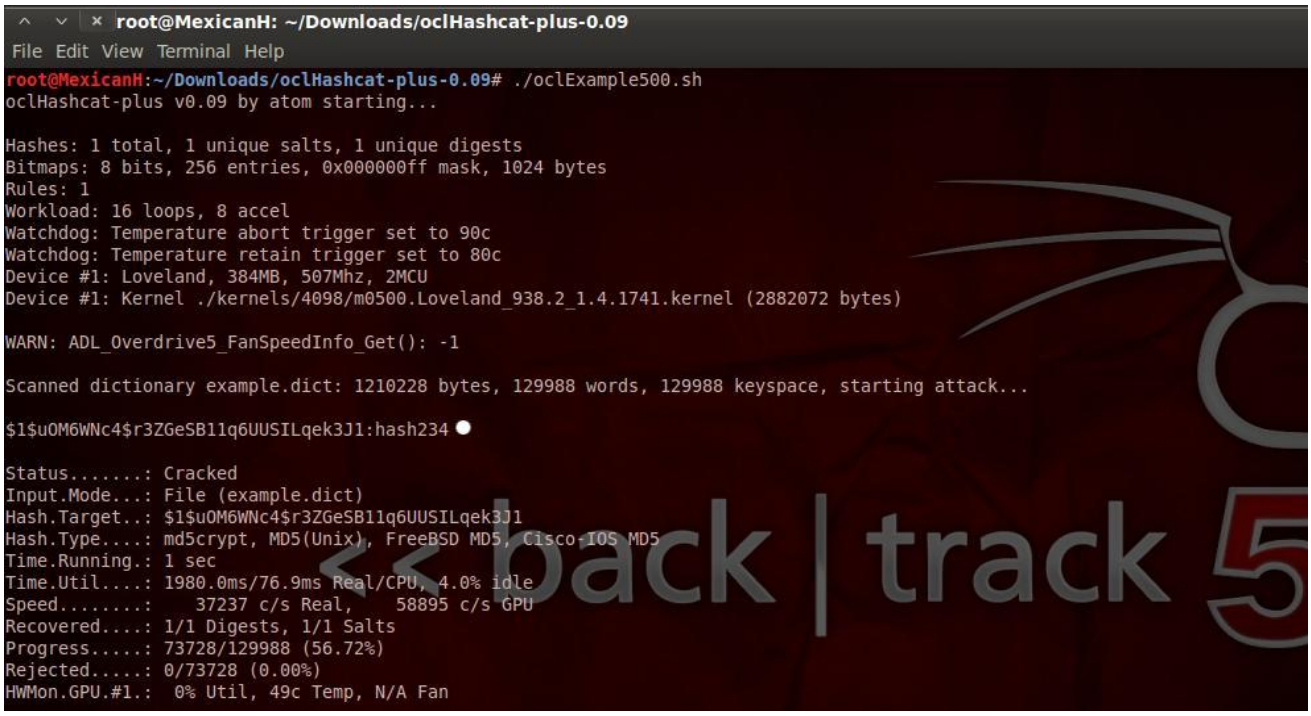
example.dict – un diccionario de ejemplo de hashcat

example.hash – son las cadenas de texto encriptadas de el diccionario de ejemplo

doc/ - son los documentos que pertenecen a oclhashcat

kernels/ - es el directorio donde los kernel del hardware son almacenados. Esto no debe ser tocado a menos que seas un miembro autorizado de la comunidad hashcat, de otro modo, puedes dañar el ejecutable.

Usando los ejemplos suministrados corramos rápidamente hashcat.

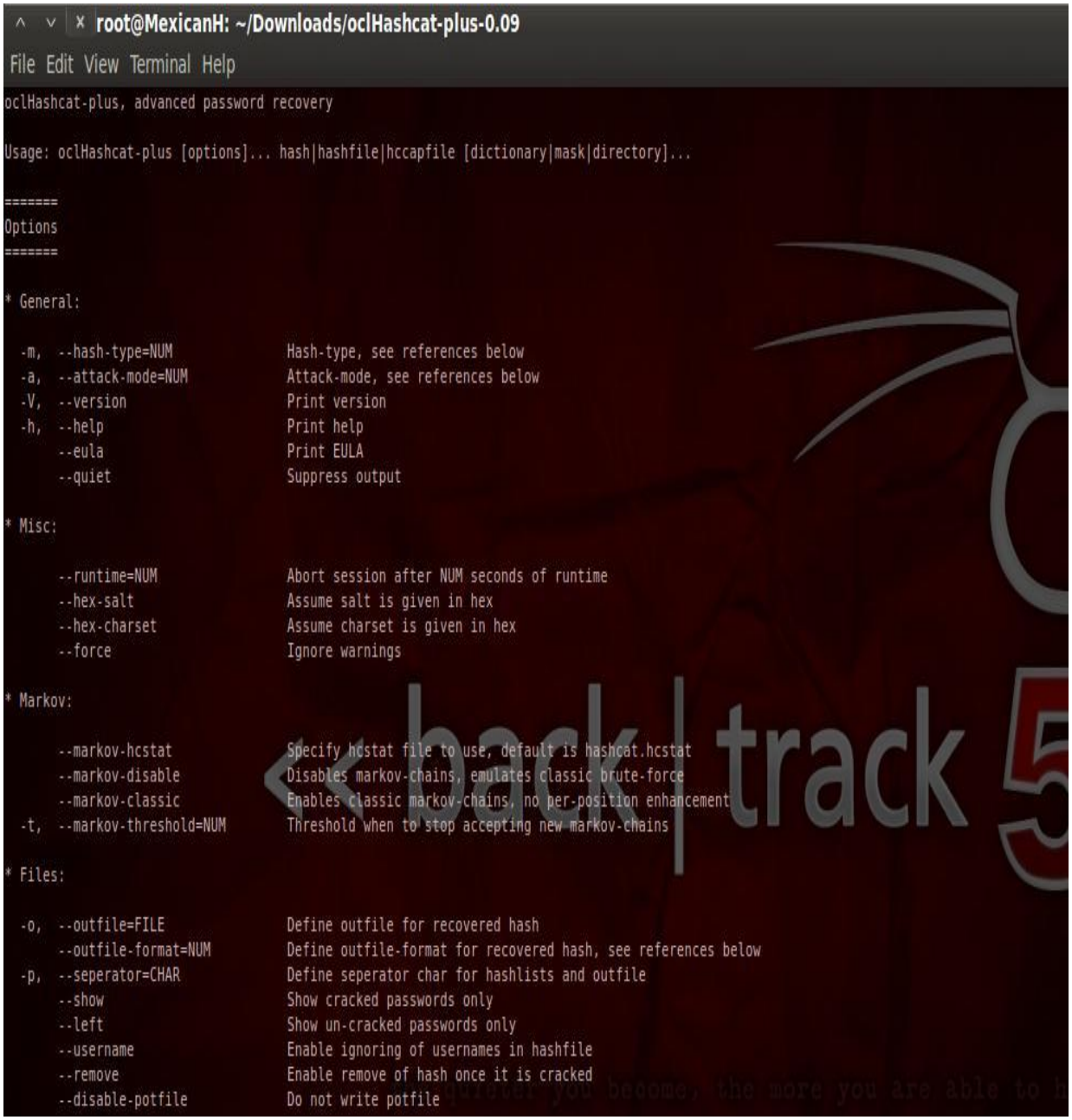


Como vemos, tenemos muchos datos a la salida, lo importante aquí esta marcado con el punto blanco que es el hash recuperado, en este caso es solo uno, pero se imaginan una lista de unos 1000 y apareciendo todos en la terminal? Sería algo fastidioso, por eso en los ejemplos, preferimos no mostrarlos a la salida.

2.1 OPCIONES

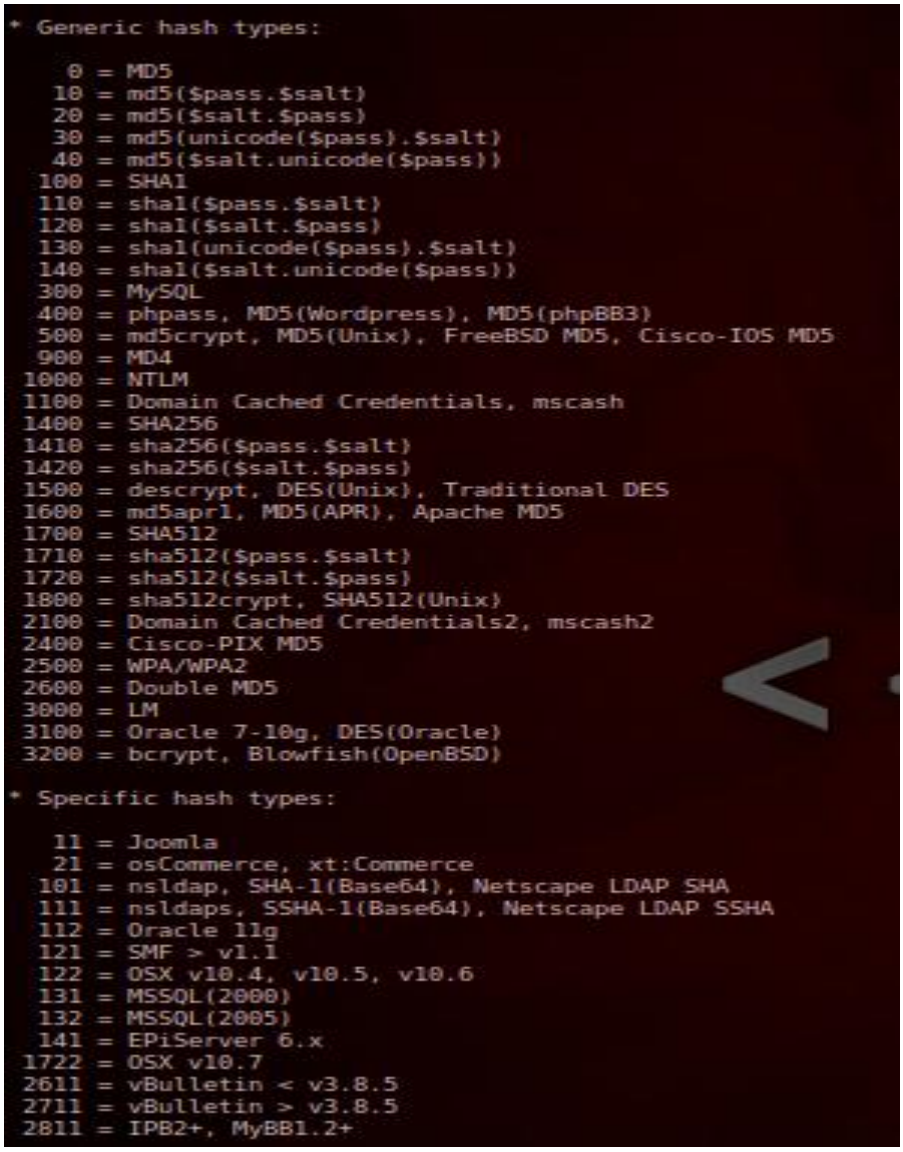
Hascat viene con un gran repertorio de opciones veamos que tiene :D

Para eso en la terminal tecleamos ./oclHashcat-plus32.bin --help



```
* Resources:

  -c,  --segment-size=NUM            Size in MB to cache from the wordfile
       --cpu-affinity=STR            Locks to CPU devices, seperate with comma
       --gpu-async                   Use non-blocking async calls (NV only)
  -d,  --gpu-devices=STR             Devices to use, separate with comma
  -n,  --gpu-accel=NUM               Workload tuning: 1, 8, 40, 80, 160
       --gpu-loops=NUM               Workload fine-tuning: 8 - 1024
       --gpu-temp-disable            Disable temperature and fanspeed readings and triggers
       --gpu-temp-abort=NUM          Abort session if GPU temperature reaches NUM degrees celsius
       --gpu-temp-retain=NUM         Try to retain GPU temperature at NUM degrees celsius (AMD only)

* Rules:

  -j,  --rule-left=RULE              Single rule applied to each word from left dict
  -k,  --rule-right=RULE             Single rule applied to each word from right dict
  -r,  --rules-file=FILE             Rules-file, multi use: -r 1.rule -r 2.rule
  -g,  --generate-rules=NUM          Generate NUM random rules
       --generate-rules-func-min=NUM Force NUM functions per random rule min
       --generate-rules-func-max=NUM Force NUM functions per random rule max

* Custom charsets:

  -1,  --custom-charset1=CS          User-defined charsets
  -2,  --custom-charset2=CS          Example:
  -3,  --custom-charset3=CS          --custom-charset1=?dabcdef
  -4,  --custom-charset4=CS          Sets charset ?1 to 0123456789abcdef

* Increment:

  -i,  --increment                   Enable increment mode
       --increment-min=NUM           Start incrementing at NUM
       --increment-max=NUM           Stop incrementing at NUM
```

```
==========
References
==========

* Outfile Formats:

    1 = hash[:salt]
    2 = plain
    3 = hash[:salt]:plain
    4 = hex_plain
    5 = hash[:salt]:hex_plain
    6 = plain:hex_plain
    7 = hash[:salt]:plain:hex_plain

* Built-in charsets:

   ?l = abcdefghijklmnopqrstuvwxyz
   ?u = ABCDEFGHIJKLMNOPQRSTUVWXYZ
   ?d = 0123456789
   ?a = ?l?u?d?s
   ?s =  !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
   ?h = 8 bit characters from 0xc0 - 0xff
   ?D = 8 bit characters from german alphabet
   ?F = 8 bit characters from french alphabet
   ?R = 8 bit characters from russian alphabet

* Attack modes:

    0 = Straight
    1 = Combination
    3 = Brute-force
    6 = Hybrid dict + mask
    7 = Hybrid mask + dict
```



En las imágenes podemos ver todas las opciones. Oclhashcat funciona un poco diferente a hashcat, oclhashcat usa dos mascaras, izquierda y derecha, lo mismo sucede en el ataque de combinación usa dos diccionarios diferentes, uno derecho y uno izquierdo.

Hay ciertas opciones que son idénticas en hashcat y oclhashcat así que no las repetiremos todas.

./oclHashcat64.bin hashlist dictizquierdo dictderecho

Es algo confuso quizás pero no te preocupes más adelante en los vectores de ataque lo explicaremos mas a detalle.

Ahora pasemos a las opciones que tenemos disponibles.

--quiet : suprime la salida hacia el “STDOUT” es decir, en tu terminal de Linux, no serás floodeado con texto, hashes recuperados y errores de programa. Por defecto no usado.

--remove : Igual que en hashcat, una vez que el hash es recuperado, es borrado de la lista de hashes, así previniendo que oclhashcat lo ataque de nuevo. Por defecto no usado.

--output-file=FILE O –o : Especifica donde los hashes serán escritos. Esto debe ser usado si usted planea preservar los hashes, o no desea copiarlos y pegarlos desde la terminal. Por defecto no usado.

--output-file=/user/Desktop/founded.txt O **–o /user/Desktop/founded.txt**

--output-format=NUM : NUM puede ser 0, 1 o 2. Normalmente no se necesita pero si un texto plano contiene caracteres en hexadecimal esto necesita ser especificado para evitar textos planos erróneos. Por defecto modo 0.

--output-format=0

--salt-file=FILE O **–e :** Suministra los "salt" para los hashes que lo tienen ausente. útil para los hashes vbull o xtcommerce . Por defecto no usado.

--gpu-devices=STR O **–d** : Esto es usado para especificar el número de GPU (en orden) para usar, si usted tiene una instalación multi-GPU. Esto es útil si usted trabaja con ventanas, como Windows, Gnome o KDE. Debido a la gran cantidad de datos que se procesan, el escritorio tendrá algo de lag. Usted puede omitir su primer GPU para asegurar una buena operación de su escritorio mientras recupera los hashes. Por defecto todas las tarjetas son usadas.

-d 2,3,4

--gpu-accel=NUM O **–n**: Define el ajuste de la carga de trabajo. Mientras más alto es el valor la tarjeta grafica trabajara más. Valores altos pueden ser útiles en ataques de fuerza bruta mientras que valores bajos pueden ser útiles con ataques de diccionario. No hay un valor que se pueda decir que es el correcto. Pruebe diferentes valores y vea cual se ajusta mejor a sus necesidades. Por defecto 80.

-n 400

--runtime=NUM : este operador, solo le da un límite de tiempo de ejecución a hashcat. Por defecto no usado.

--runtime=60 (60 segundos)

--hex-salt : Asume que el "salt" esta dado en hexadecimal. Por defecto no usado.

--hex-charset : Asume que el mapa de caracteres esta dado en hexadecimal. Por defecto no usado

--username: Ignora los nombres de usuario en las listas de hashes, si tenemos user:hash, para evitar la molestia de editar largas listas, usamos este operador, para ahorrarnos tiempo. Por defecto no usado.

--force: Ignora las advertencias que arroje hashcat, en ciertos algoritmos como sha-1 te pedirá usar oclhashcat-lite si es un solo hash. Con este operador ignorara la advertencia. Por defecto no usado.

--gpu-async: Es ayuda a los Gpus de baja Gama, Solo para Nvidia.

--cpu-affinity=STR : Especifica el número de unidades cpu que se usaran, si su procesador de multi-nucleo. Por defecto no usado.

--cpu-affinity=2,3,4

--gpu-loops=NUM: esto es usado para ajustar mas la carga de trabajo. Mas específicamente, el numero de palabras por unidad de trabajo. Provee más trabajo a la máscara de la izquierda. Por defecto 128.

--gpu-loops=1024

--gpu-temp-disable : Deshabilita las lecturas de los disparadores temperatura y el ventilador. Úselo bajo su propio riesgo. Por defecto no usado.

--gpu-temp-abort=NUM: Establece la temperatura del GPU con la que oclhashcat abortara la sesión para evitar daños al GPU. Por defecto 90 grados Celsius.

--gpu-temp-abort=90

--gpu-temp-retain=NUM : establece la temperatura que oclhashcat tratara de mantener para evitar un sobre calentamiento del GPU. Por defecto 80 grados Celsius.

--gpu-temp-retain=80

--rule-left=RULE O **–j** and **–rule-right=RULE** O **–k:** Son reglas que se aplican a los diccionarios de la derecha o la izquierda cuando se usa el ataque de combinación y/o hibrido. La teoría puede ser un poco confusa, pueden visitar el siguiente sitio http://ob-security.info/?p=31%20. Y mas adelante en los vectores de ataque explicaremos un poco más. Por defecto no usado.

--increment O **–i** : incrementa la máscara con +1 en cada posición. Por ejemplo ?l?l?l?l?l al inicio correrá

?l

?l?l

?l?l?l

?l?l?l?l

?l?l?l?l?l

Por defecto no usado.

--increment-min=NUM : comienza a incrementar posiciones en la posición dada si tenemos una máscara de ?l?l?l?l?l?l?l y le decimos a oclhashcat que comience en 3.

?l?l?l

?l?l?l?l

?l?l?l?l?l

?l?l?l?l?l?l

?l?l?l?l?l?l?l

Esto se hace para ahorrar tiempo, si uno ya sabe que la contraseña no es de 3 caracteres. Se puede usar cualquier mascara o carácter a medida que uno especifique. Por defecto no usado

--increment-max=NUM : Lo mismo que el anterior solo que este le dará el limite a oclhashcat de detenerse en la posición que le especifiquemos. Por defecto no usado.

increment-max=12

Los operadores Markov son una nueva mejora a la versión 0.9 de oclhashcat, la teoría es algo confusa trataremos de explicarla

El ataque markov es un ataque parecido al de fuerza bruta pero basado en estadísticas, en lugar de especificar una máscara o un mapa de caracteres, especificamos un archivo.

Una vez que el archivo es creado en un paso previo este contiene información estadística, que lleva a cabo un análisis automatizado de un diccionario dado.

Para hacer este análisis usamos una herramienta nueva llamada "hcstatgen" la cual es parte de las nuevas utilerías hashcat “hashcat-utils package”.

http://www.hashcat.net/files/hashcat-utils-0.9-32.7z

http://www.hashcat.net/files/hashcat-utils-0.9-64.7z

La utilería hcstatgen genera un archivo .hcstat, el cual es procesado con otra de las nuevas utilerías "statprocessor" que genera las palabras basadas en un orden estadístico del archivo .hcstat .

En versiones anteriores esto no estaba en las funciones de oclhashcat en la versión 0.9 viene integrado, y se usa automáticamente al usar cualquier mascara tales como ?d?d?lu . el archivo por defecto es hashcat.hcstat el cual fue hecho usando la lista rockyou.txt

Como funciona?

Trataremos de explicarlo un poco mas a fondo.

En el ataque Brute-force o Mass-attack nosotros podemos especificar un limite del “keyspace” estableciendo un mapa de caracteres menor para reducir el tiempo de ataque. En el ataque markov tenemos algo parecido llamado "threshold". Todo lo que se tiene que hacer es especificar un número. Mientras mas alto el número más alto será el "threshold" para sumar otro enlace entre dos caracteres en la tabla de dos niveles en la que se basa el ataque markov.

No es tan necesario saberlo todo a fondo según las notas de Atom, solo tenemos que recordar que mientras más alto el valor mas alto será el “keyspace” así que el ataque será mas profundo y tardara mas. Si usamos un --markov-threshold=0 estaremos usando ataque puro de fuerza bruta pero con cadenas markov.

Más adelante les explicaremos como crear el .hcstat ahora les explicaremos los operadores markov

Markov:

--markov-hcstat: especifica el directorio donde se encuentra el archivo .hcstat . Por defecto es hashcat.hcstat

--markov-hcstat /user/Desktop/markov.hcstat

--markov-disable : Deshabilita el uso de cadenas markov y hacemos un ataque puro de fuerza bruta

--markov-classic: habilita el uso clásico de cadenas markov, no hay un realce por cada posición, según notas de Atom, el modo classic es un 29 % menos eficiente. Úselo según le convenga.

--markov-threshold=NUM O –t: especificamos cuando oclhashcat deja de aceptar más cadenas markov.

Pongámoslo así

Un threshold de 3 y un largo de 6 caracteres = 3^6=729

Un threshold de 10 y un largo de 8 caracteres = 10^8=100000000

A menos que especifiquemos lo contrario con –t 0 o –t 10, al usar una máscara cómo ?l?l?l?l?l hashcat intentara con todo el keyspace, puesto de otra manera (# de caracteres)^(largo).

--attack-mode=NUM O **–a:** El tipo de ataque a usar en contra de un hash. Usando los diferentes vectores de ataque aumentaran las probabilidades de recuperar la contraseña. Los modos son los siguientes:

- 0 = Straight- simplemente corre todas las palabras del diccionario en contra de la lista de hashes, teniendo un buen diccionario aumentara las probabilidades de recuperar tu hash.
- 1 = Combination – combina las palabras de los diccionarios dados, recuerda que Oclhashcat usa un diccionario izquierdo y uno derecho.
- 3 = Brute-force – Fuerza bruta o también conocido como Mask Attack. A diferencia de hashcatcli que es usado como un paso desesperado, con oclhashcat puede ser realizado en minutos u horas con la ayuda del GPU, y usando el ataque markov.
- 6 = Hybrid dict + mask – es un ataque hibrido, fácil especificamos en la izquierda un diccionario y en la derecha una mascara como si fuésemos a usar un mass attack. Por ejemplo /user/Desktop/diccionario.txt ?d?d?d lo que hará Oclhashcat será añadir la mascara en la parte del frente de la palabra del diccionario. Si tenemos en el diccionario “aaaaa” la palabra seria “aaaaa000 hasta aaaaa999 “
- 7 = Hybrid mask + dict – lo mismo descrito anteriormente pero al revés, tendríamos de “000aaaaa hasta 999aaaaa”.
  Nota: para los modos 0, 1, 6, 7 el uso de reglas puede ser usado, lo explicaremos más adelante.

2.2 VECTORES DE ATAQUE

Bueno si, al inicio dijimos que no usaríamos Windows pero no queremos discriminar a nadie, XD. Así que también presentaremos ejemplos en esta plataforma, para los usuarios Linux esto no presentara ningún problema, escribir los comandos en la terminal de Windows es prácticamente los mismo que escribirlo en la terminal Linux :D

Straight. (Directo).Este ataque simplemente corre una lista de palabras y prueba todas las cadenas contra cada hash. Este vector ya lo hemos estudiado en la versión cpu de hascat. La versión GPU tiene una manera particular de cargar los diccionarios, cada palabra del diccionario será puesta en un buffer especial por el largo de la cadena de caracteres, es decir el largo de cada palabra tiene un buffer único, las palabras de 8 letras serán almacenadas en el buffer numero 8 y así sucesivamente, pero como sucede normalmente los diccionarios no están ordenados, así que para optimizar la carga del diccionario te sugerimos, si usas Linux usar http://hashcat.net/wiki/doku.php?id=hashcat_utils la herramienta splitlen junto con el comando "sort"

Sort –u dicc.txt > diccordenado.txt

Si usas Windows usa http://home.btconnect.com/md5decrypter/unix-utils.zip

Esta es una de las razones por la que es difícil en Oclhashcat restaurar la sesión como su contraparte basada en CPU.

Combination. Junta dos palabras del diccionario dado e intenta con ellas. Puede ser útil en contra de contraseñas largas, la gente trata de añadir complejidad a sus contraseñas escribiéndolas dos veces o personas que usan su primer y segundo apellido. Por ejemplo Atom usa su apellido derp para hacer una contraseña, si Atom y derp existen en el diccionario, atomderp será usado y también derpatom. Lo especial de este ataque en oclhascat es que podemos aplicar reglas a los diccionarios de izquierda y derecha.

Brute force (Mask attack)- con este ataque probamos todas las combinaciones de un espacio dado pero más especifico, igual que en un ataque de fuerza bruta. La razón de esto es para no apegarse al tradicional ataque de fuerza bruta es para reducir los candidatos de contraseñas. Convirtiéndolo en un ataque más especifico y eficiente. Por cada posición tenemos que configurar un marcador, si nuestra contraseña a crackear tiene 8 espacios usaremos 8 marcadores.

- ?l = abcdefghijklmnopqrstuvwxyz
- ?u = ABCDEFGHIJKLMNOPQRSTUVWXYZ
- ?d = 0123456789
- ?s = !"#$%&'()*+,-./:;⇔?@[\]^_`{|}~
- ?h = 8 bit characters from 0xc0 - 0xff
- ?D = 8 bit characters from german alphabet
- ?F = 8 bit characters from french alphabet
- ?R = 8 bit characters from russian alphabet

Si queremos solo letras minúsculas usaríamos ?l?l?l?l?l?l?l?l así nuestra mascara probara desde" aaaaaaaa" hasta "zzzzzzzz"

Pero y si queremos solo numero y letras? Qué tal si el pass es geyepn123 ? pues fácil, en Oclhashcat tenemos los caracteres a medida entonces configuraríamos así:

--custom-charset1=?l?d O de manera análoga **-1 ?l?d**

Nuestra mascara seria ?1?1?1?1?1?1?d?d?d

Ahora calculemos el tiempo, digamos que es un algoritmo rápido como el MD5 y nuestro GPU tiene una velocidad con el MD5 de 100 millones de contraseñas por segundo o 100M/s

$36*36*36*36*36*36*10*10*10= 36^6 *10^3 =2176782336000/100000000=21767.82$ seg.

21767.82/60=362.79 minutos

362.79/60= 6.04 horas

Parecerá mucho tiempo, pero comparado con CPU esto podría tomar años. Hay GPU más poderosos donde podemos llegar hasta a 1000M/s y con multi-gpu unos 8000M/s

Hybrid. " dict + mask" y "mask + dict" básicamente es solo un ataque de combinación, de un lado tienes un diccionario y del otro simplemente un ataque de fuerza bruta, en otras palabras los caracteres de fuerza bruta son usados como prefijo o sufijo en cada una de las palabras del diccionario, es por eso que se llama Hibrido. Como alternativa se puede usar el modo 3 o mask attack o el ataque basado en reglas.

2.3 RULE-BASED ATTACK

El ataque basado en reglas es uno de los ataques más complicados que existen en la familia hashcat. La razón de esto es muy simple, el ataque basado en reglas es como un lenguaje de programación, diseñado para generación de candidatos de contraseñas. Tiene funciones para modificar, cortar o extender palabras y tiene operadores condicionales etc. Esto lo hace un ataque más flexible, seguro y eficiente.

El motor de las reglas fue hecho paraqué pueda ser compatible con otros programas de recuperación de contraseñas como John the ripper y Passwordspro, que no veremos en este manual.

Lo más importante al escribir reglas es sabiendo que es lo que queremos escribir, esto significa que tendría que analizar docenas de contraseñas en texto plano para poder definir un patrón en ellas. Por ejemplo sabemos que comúnmente las personas añaden un digito a sus contraseñas, con esto ya tenemos dos parámetros.

*queremos añadir algo

* El valor que queremos añadir es un digito

Las siguiente funciones son 100 % compatibles con John the ripper y passwordspro.

Name	Function	Description	Example Rule	Input Word	Output Word	Note
Nothing	:	do nothing	:	p@ssW0rd	p@ssW0rd	
Lowercase	l	Lowercase all letters	l	p@ssW0rd	p@ssw0rd	
Uppercase	u	Uppercase all letters	u	p@ssW0rd	P@SSW0RD	
Capitalize	c	Capitalize the first letter	c	p@ssW0rd	P@ssW0rd	
Invert Capitalize	C	Lowercase first found character, uppercase the rest	C	p@ssW0rd	p@SSW0RD	
Toggle Case	t	Toggle the case of all characters in word.	t	p@ssW0rd	P@SSw0RD	
Toggle @	TN	Toggle the case of characters at position N	T3	p@ssW0rd	p@sSW0rd	
Reverse	r	Reverse the entire word	r	p@ssW0rd	dr0Wss@p	
Duplicate	d	Duplicate entire word	d	p@ssW0rd	p@ssW0rdp@ssW0rd	
Reflect	f	Duplicate word reversed	f	p@ssW0rd	p@ssW0rddr0Wss@p	
Rotate Left	{	Rotates the word left.	{	p@ssW0rd	@ssW0rdp	
Rotate Right	}	Rotates the word right	}	p@ssW0rd	dp@ssW0r	
Append Character	$	Append character to end	$1	p@ssW0rd	p@ssW0rd1	
Prepend Character	^	Prepend character to front	^1	p@ssW0rd	1p@ssW0rd	
Truncate left	[	Deletes first character	[	p@ssW0rd	@ssW0rd	
Trucate right	]	Deletes last character	]	p@ssW0rd	p@assW0r	
Delete @ N	DN	Deletes character at position N	D3	p@ssW0rd	p@sW0rd	*
Delete range	xNM	Deletes M characters, starting at position N	x02	p@ssW0rd	ssW0rd	*
Insert @ N	iNX	Inserts character X at position N	i4!	p@ssW0rd	p@ss!W0rd	*
Overwrite @ N	oNX	Overwrites character at postion N with X	o3$	p@ssW0rd	p@s$W0rd	*
Truncate @ N	'N	Truncate word at position N	'6	p@ssW0rd	p@ssW0	
Replace	sXY	Replace all instances of X with Y	ss$	p@ssW0rd	p@$$W0rd	
Purge	@X	Purge all instances of X	@s	p@ssW0rd	p@W0rd	+
Duplicate first N	z	Duplicates first character N times	z2	p@ssW0rd	ppp@ssW0rd	
Duplicate last N	Z	Duplicates last character N times	Z2	p@ssW0rd	p@ssW0rddd	

Duplicate all	q	Duplicate every character	q	p@ssW0rd	pp@@ssssWW00rrdd	
Duplicate word	pN	Duplicate entire word N times	p3	Pass	PassPassPass	

* indica que N inicia en 0. Para posiciones de caracteres diferentes que 0-9 use A-Z (A=11)

+ indica que esta regla es implementada en hashcat únicamente.

Las siguientes funciones no están disponibles en John the ripper y Passwordspro

Name	Function	Description	Example Rule	Input Word	Output Word	Note
Swap front	k	Swaps first two characters	k	p@ssW0rd	@pssW0rd	
Swap back	K	Swaps last two characters	K	p@ssW0rd	p@ssW0dr	
Swap @ N	*XY	Swaps character X with Y	*34	p@ssW0rd	p@sWs0rd	*
Bitwise shift left	LN	Bitwise shift left character @ N	L2	p@ssW0rd	p@æsW0rd	*
Bitwise shift right	RN	Bitwise shift right character @ N	R2	p@ssW0rd	p@9sW0rd	*
Ascii increment	+N	Increment character @ N by 1 ascii value	2	p@ssW0rd	p@tsW0rd	*
Ascii decrement	-N	Decrement character @ N by 1 ascii value	-2	p@ssW0rd	p?ssW0rd	*
Replace N + 1	.N	Replaces character @ N with value at @ N plus 1	.1	p@ssW0rd	psssW0rd	*
Replace N - 1	,N	Replaces character @ N with value at @ N minus 1	0,1	p@ssW0rd	ppssW0rd	*
Duplicate block front	yN	Duplicates first N characters	y2	p@ssW0rd	p@p@ssW0rd	*
Duplicate block back	YN	Duplicates last N characters	Y2	p@ssW0rd	p@ssW0rdrd	*
Title	E	Upper case the first letter and every letter after a space	E	p@ssW0rd w0rld	P@ssw0rd W0rld	

* indica que N inicia en 0. Para posiciones de caracteres diferentes que 0-9 use A-Z (A=11)

Bueno y dirá, quizás, wtfck? Que son esas tablas? Pues fácil son los comandos que usaremos en las reglas, usaremos lo que está en la segunda columna donde dice “Function”, no nos molestaremos en traducir estas tablas a español, así que si no sabe ingles pues use google para traducir XD.

Estas reglas las podemos guardar en un archivo .txt o si quiere cambiarle la extensión a .rule para no confundirse no hay problema, esto no afecta en nada.

Supongamos que ya creamos nuestro archivo donde guardaremos nuestras reglas

Como mencionamos antes si queremos añadir una cifra a nuestras palabras del diccionario en el archivo escribiremos

$1

Así que si tenemos en nuestro diccionario:

putoepn

A la palabra le será añadida la cifra 1 y tendremos

putoepn1

Si lo que queremos es añadir todos los números del 0 al 9 para eso tenemos el ataque hibrido.

Si queremos hacer más complejo el ataque podemos añadir

:

r

ru

Con eso le dijimos a hashcat que primero no haga nada, solo comparar la palabra, en la segunda línea le dijimos que invierta la palabra y en la tercera línea le dijimos que invierta la palabra y que además la palabra la pase toda a mayúsculas.

Otra característica especial de hashcat es que con hashcat puedes generar reglar aleatorias sobre la marcha en una sesión. Esto es de mucha utilidad cuando estas totalmente sin ideas.

--generate-rules=NUM – con esto le decimos a hashcat que genere el numero de reglas indicadas a aplicar en cada intento.

--generate-rules-func-min=NUM
--generate-rules-func-max=NUM
Estos operadores especifican el número de funciones que deben ser usadas. Si ponemos como mínimo 1 y máximo 3 el resto de las funciones serán ignoradas. Por ejemplo digamos que hashcat genero las funciones

:

]]]]

ru

lo que pasara entonces es que la segunda función ‘]]]]’ será ignorada.

Esto es de manera general, si quieres saber mas entra a la web del autor

http://hashcat.net/wiki/doku.php?id=rule_based_attack

2.4 EJEMPLOS

Para los siguientes ejemplos usaremos otra lista tomada de http://forum.md5decrypter.co.uk/

STRAIGHT.

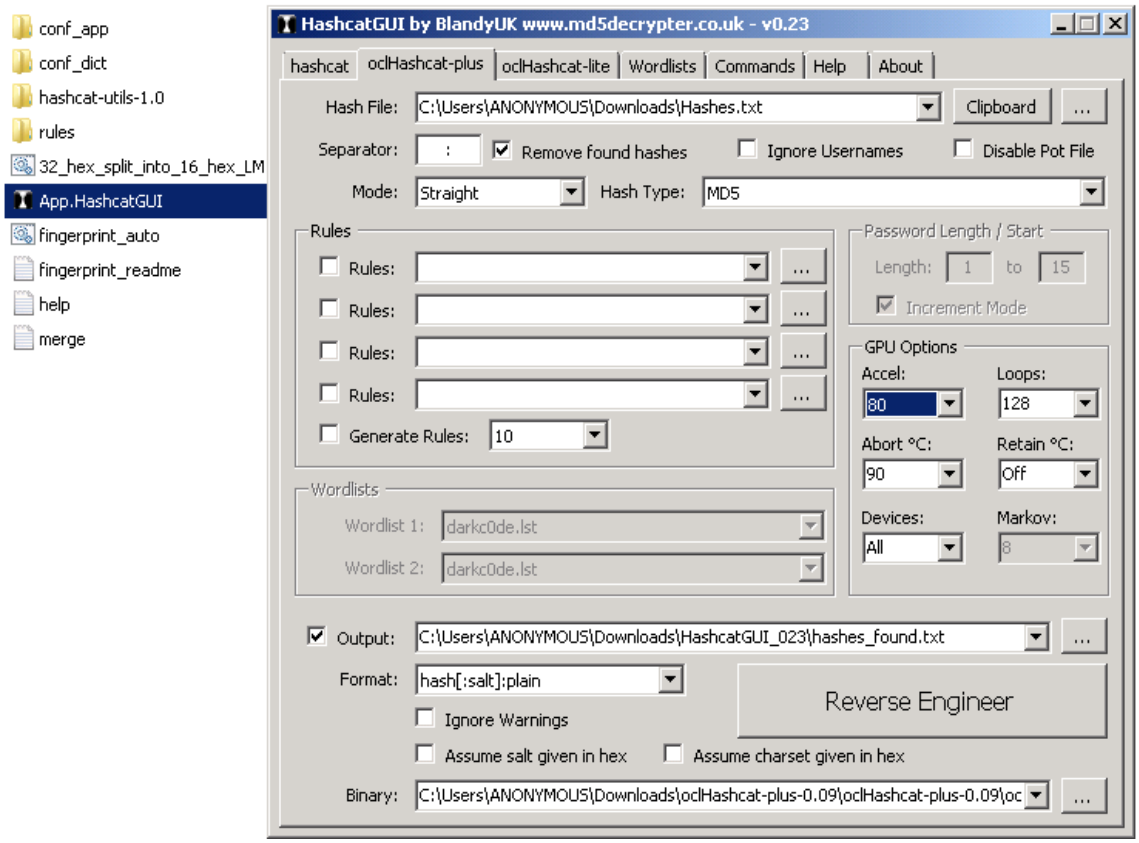
Supongamos que tenemos una lista de aproximadamente unos 37000 hashes D: mucho más extensa que la que vimos en la versión CPU, a diferencia de la primera lista, esta ya la hemos roto anteriormente, lo que haremos será reproducirles de nuevo el proceso de cómo lo hicimos y les mostraremos lo especial de esta lista además les demostraremos el poder del GPU.

Primero que nada lanzaremos nuestro hashcat-GUI descargado previamente de:

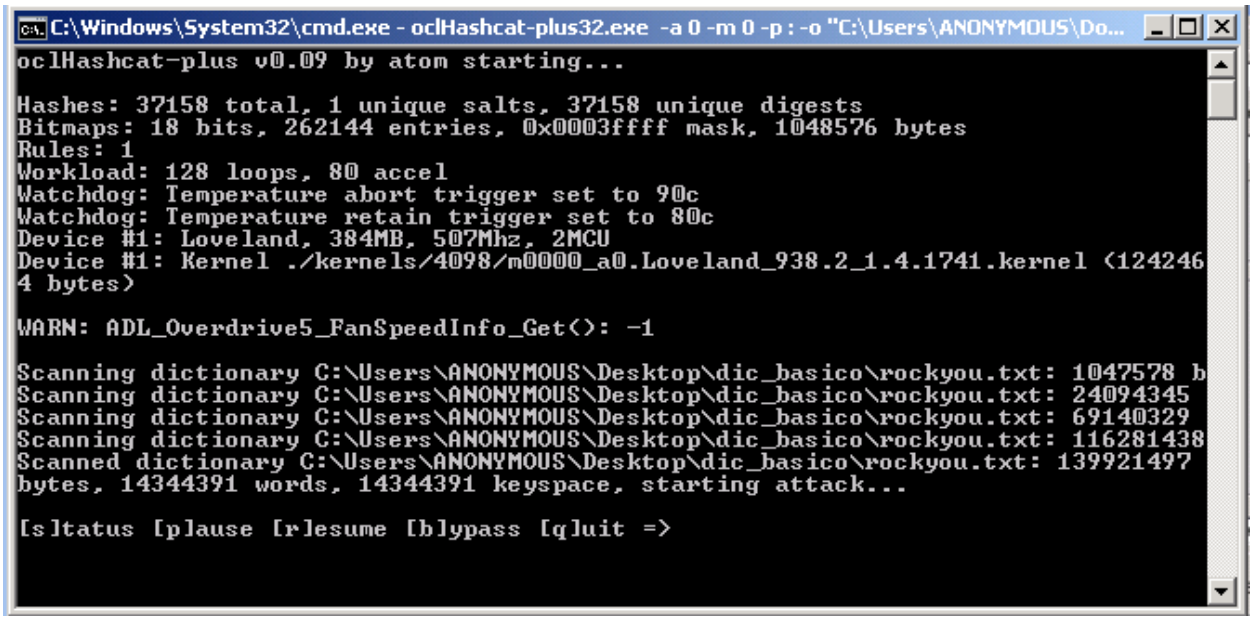
http://www.md5decrypter.co.uk/downloads.aspx

home.btconnect.com/md5decrypter/HashcatGUI_023.zip

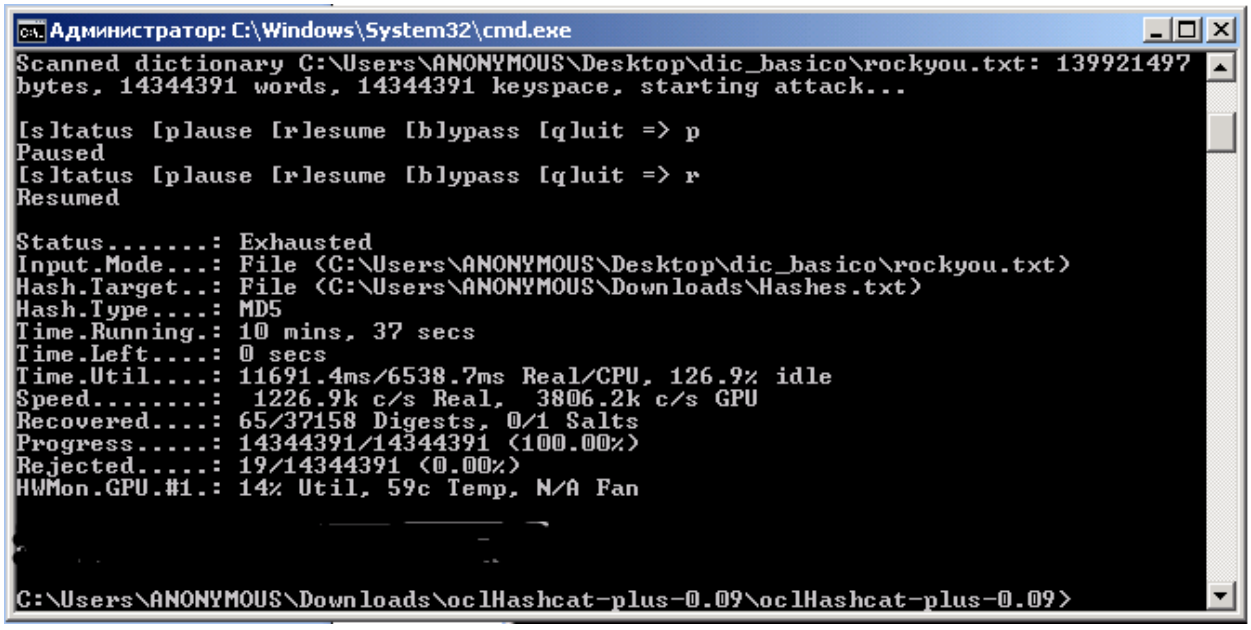
Lo que tendremos será esta imagen



Como podemos ver, la interfaz es bastante sencilla, en la pestaña donde dice wordlists ahí agregaremos los directorios de nuestros diccionarios. Intentaremos con una lista básica como todos sabemos Rockyou.txt. para lanzar hashcat GUI daremos un click en “Reverse Engineer”.



Aquí podemos observar cómo cargo nuestro diccionario Rockyou.txt y vemos que la cantidad exacta de hashes son 37158. Tenemos varias opciones si queremos ver cómo va el proceso de recuperación presionaremos “s”. Realmente no hay mucho que decir.

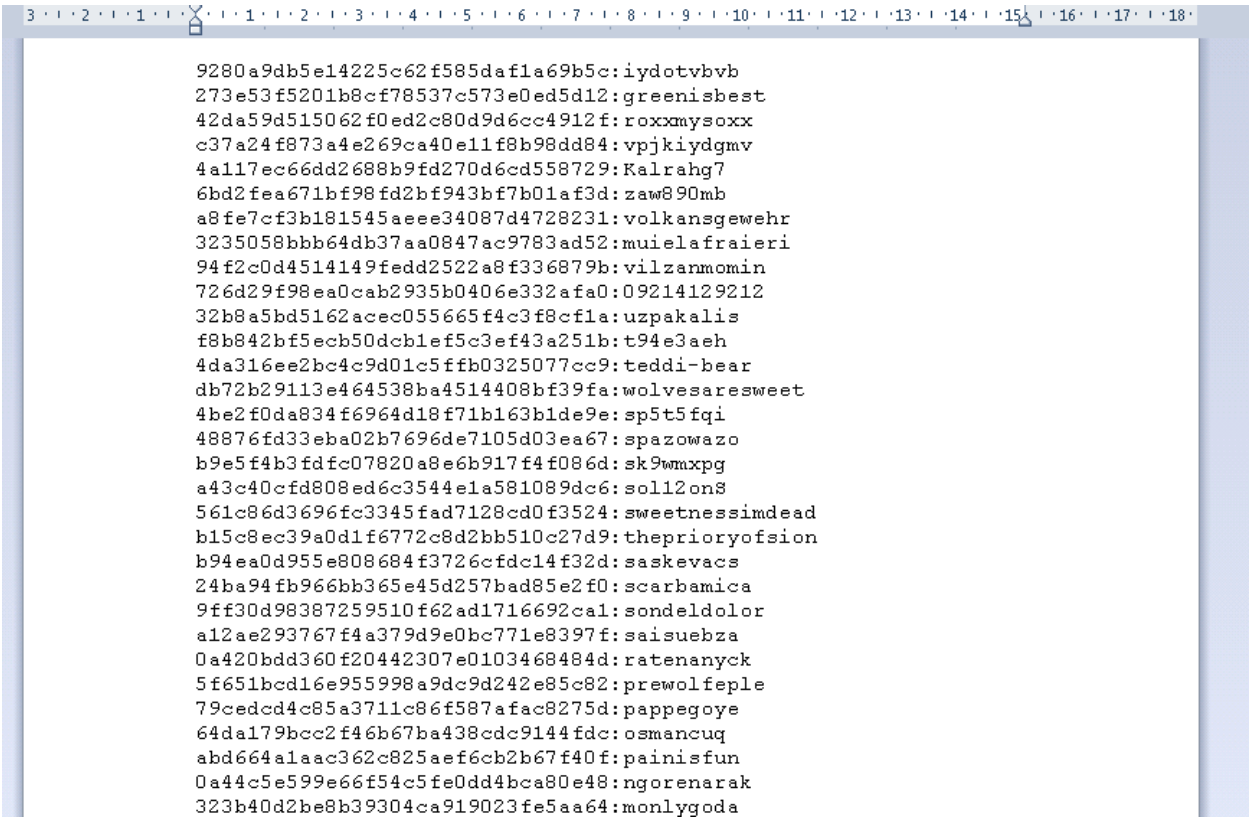


A la salida vemos que solo recuperamos unos pocos hashes, la lista contiene aprox. 14 millones de palabras, lo más probable es que la lista no cuente con ningún tipo de patrón o sean contraseñas creadas aleatoriamente.

Si fuésemos un usuario Linux teclearíamos el siguiente comando en nuestra terminal:

./oclhashcat-plus32.bin -a 0 -m 0 -o "/user/Desktop/hashes_found.txt" --outfile-format=3 -n 80 --remove --gpu-loops=128 "/user/Desktop/Hashes.txt" "/user/Desktop/rockyou.txt"

Demos una Mirada a nuestro archivo de hashes recuperados.



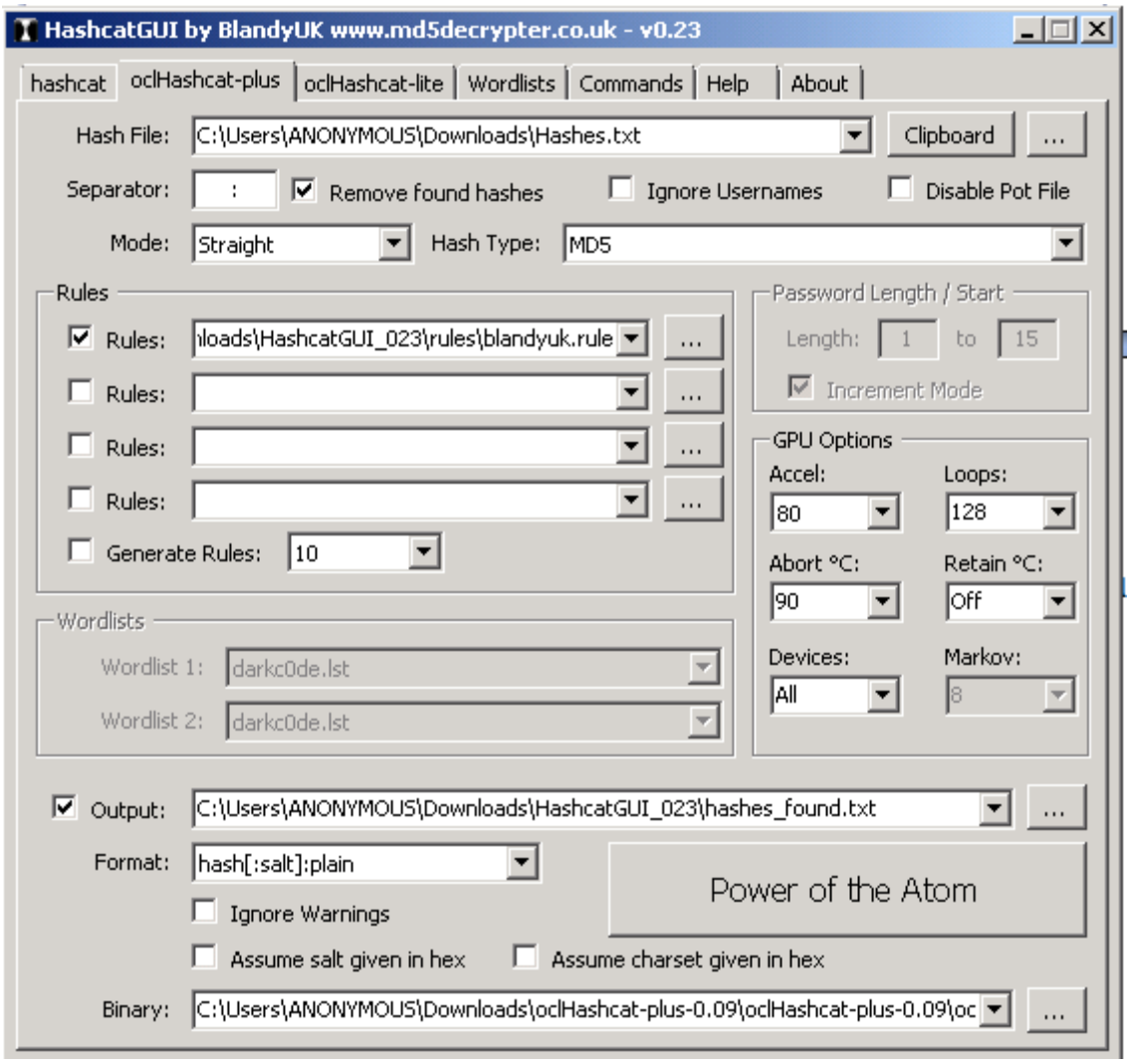
```
9280a9db5e14225c62f585daf1a69b5c:iydotvbvb
273e53f5201b8cf78537c573e0ed5d12:greenisbest
42da59d515062f0ed2c80d9d6cc4912f:roxxmysoxx
c37a24f873a4e269ca40e11f8b98dd84:vpjkiydgmv
4a117ec66dd2688b9fd270d6cd558729:Kalrahg7
6bd2fea671bf98fd2bf943bf7b01af3d:zaw890mb
a8fe7cf3b181545aeee34087d4728231:volkansgewehr
3235058bbb64db37aa0847ac9783ad52:muielafraieri
94f2c0d4514149fedd2522a8f336879b:vilzanmomin
726d29f98ea0cab2935b0406e332afa0:09214129212
32b8a5bd5162acec055665f4c3f8cf1a:uzpakalis
f8b842bf5ecb50dcb1ef5c3ef43a251b:t94e3aeh
4da316ee2bc4c9d01c5ffb0325077cc9:teddi-bear
db72b29113e464538ba4514408bf39fa:wolvesaresweet
4be2f0da834f6964d18f71b163b1de9e:sp5t5fqi
48876fd33eba02b7696de7105d03ea67:spazowazo
b9e5f4b3fdfc07820a8e6b917f4f086d:sk9wmxpg
a43c40cfd808ed6c3544e1a581089dc6:soll2on8
561c86d3696fc3345fad7128cd0f3524:sweetnessimdead
b15c8ec39a0d1f6772c8d2bb510c27d9:theprioryofsion
b94ea0d955e808684f3726cfdc14f32d:saskevacs
24ba94fb966bb365e45d257bad85e2f0:scarbamica
9ff30d98387259510f62ad1716692ca1:sondeldolor
a12ae293767f4a379d9e0bc771e8397f:saisuebza
0a420bdd360f20442307e0103468484d:ratenanyck
5f651bcd16e955998a9dc9d242e85c82:prewolfeple
79cedcd4c85a3711c86f587afac8275d:pappegoye
64da179bcc2f46b67ba438cdc9144fdc:osmancuq
abd664a1aac362c825aef6cb2b67f40f:painisfun
0a44c5e599e66f54c5fe0dd4bca80e48:ngorenarak
323b40d2be8b39304ca919023fe5aa64:monlygoda
```

Nada especial, muchas son puro texto plano, otras con números etc. Intentemos ahora con el mismo diccionario pero agregando reglas. Usaremos reglas predefinidas que vinieron en el archivo comprimido de hashcat.

```
:
r
d
f
q
ru
du
fu
qu
rl
dl
fl
ql

$1 ss$
^1 ss$
T0$1 ss$
T0^1 ss$

$# ss$
^# ss$
T0$# ss$
T0^# ss$
```

Este es un ejemplo de cómo debe de lucir el archivo donde se encuentran nuestras reglas.

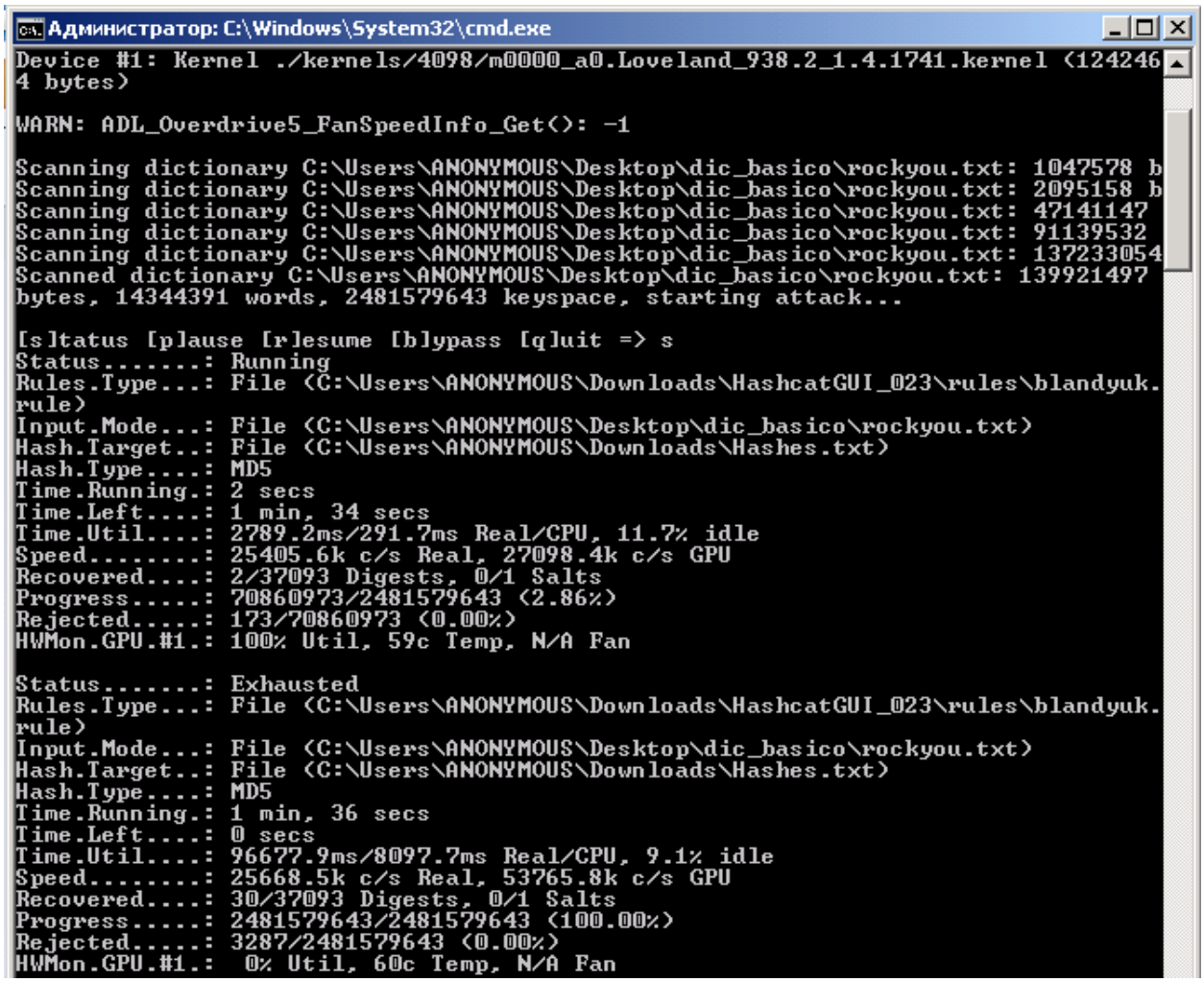
Lo único que tenemos que hacer es agregar el archivo, marcando el cuadro donde dice Rules y seleccionando el archivo de reglas, en este caso usamos el archivo blandyuk.rule que se encuentra en la carpeta rules.



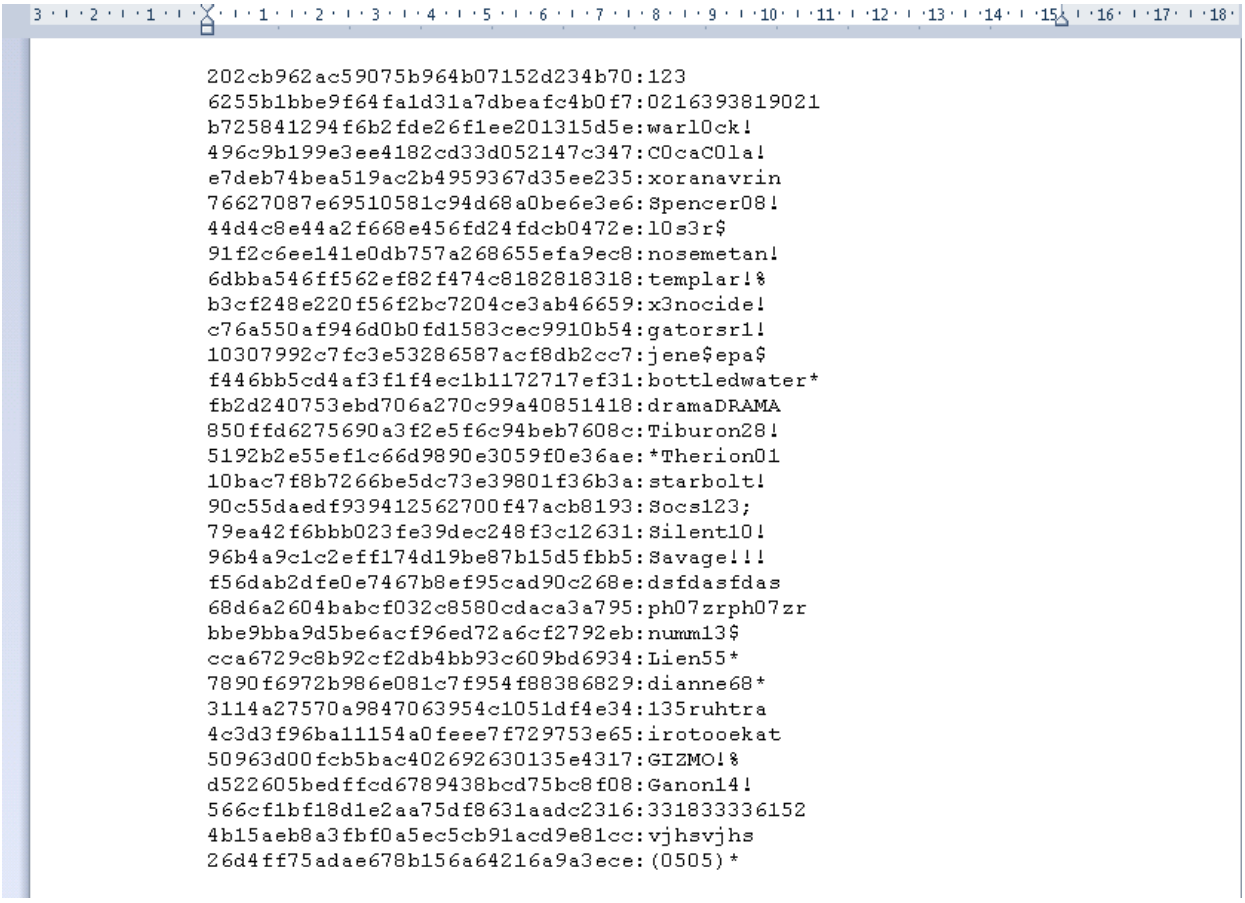
Para un usuario Linux el comando seria:

./oclhashcat-plus32.bin -a 0 -m 0 -o "/user/Desktop/hashes_found.txt" --outfile-format=3 -n 80 --remove --gpu-loops=128 -r "/rules/blandyuk.rule" "/user/Desktop/Hashes.txt" "/user/Desktop/rockyou.txt"

Ahora vemos si tenemos algo de suerte.



Nuestro proceso tomo no más de dos minutos y recuperamos en total 30 hashes, pero esto aun sigue siendo poco. Las reglas son efectivas pero si queremos reducir esta lista en el menor tiempo posible tendremos que usar otro método, mientras tanto veamos nuestro archivo de hashes recuperados.



```
202cb962ac59075b964b07152d234b70:123
6255b1bbe9f64fa1d31a7dbeafc4b0f7:0216393819021
b725841294f6b2fde26f1ee201315d5e:warl0ck!
496c9b199e3ee4182cd33d052147c347:C0caC0la!
e7deb74bea519ac2b4959367d35ee235:xoranavrin
76627087e69510581c94d68a0be6e3e6:Spencer08!
44d4c8e44a2f668e456fd24fdcb0472e:l0s3r$
91f2c6ee141e0db757a268655efa9ec8:nosemetan!
6dbba546ff562ef82f474c8182818318:templar!%
b3cf248e220f56f2bc7204ce3ab46659:x3nocide!
c76a550af946d0b0fd1583cec9910b54:gatorsr1!
10307992c7fc3e53286587acf8db2cc7:jene$epa$
f446bb5cd4af3f1f4ec1b1172717ef31:bottledwater*
fb2d240753ebd706a270c99a40851418:dramaDRAMA
850ffd6275690a3f2e5f6c94beb7608c:Tiburon28!
5192b2e55ef1c66d9890e3059f0e36ae:*Therion01
10bac7f8b7266be5dc73e39801f36b3a:starbolt!
90c55daedf939412562700f47acb8193:Socs123;
79ea42f6bbb023fe39dec248f3c12631:Silent10!
96b4a9c1c2eff174d19be87b15d5fbb5:Savage!!!
f56dab2dfe0e7467b8ef95cad90c268e:dsfdasfdas
68d6a2604babcf032c8580cdaca3a795:ph07zrph07zr
bbe9bba9d5be6acf96ed72a6cf2792eb:numm13$
cca6729c8b92cf2db4bb93c609bd6934:Lien55*
7890f6972b986e081c7f954f88386829:dianne68*
3114a27570a9847063954c1051df4e34:135ruhtra
4c3d3f96ba11154a0feee7f729753e65:irotooekat
50963d00fcb5bac402692630135e4317:GIZMO!%
d522605bedffcd6789438bcd75bc8f08:Ganon14!
566cf1bf18d1e2aa75df8631aadc2316:331833336152
4b15aeb8a3fbf0a5ec5cb91acd9e81cc:vjhsvjhs
26d4ff75adae678b156a64216a9a3ece:(0505)*
```

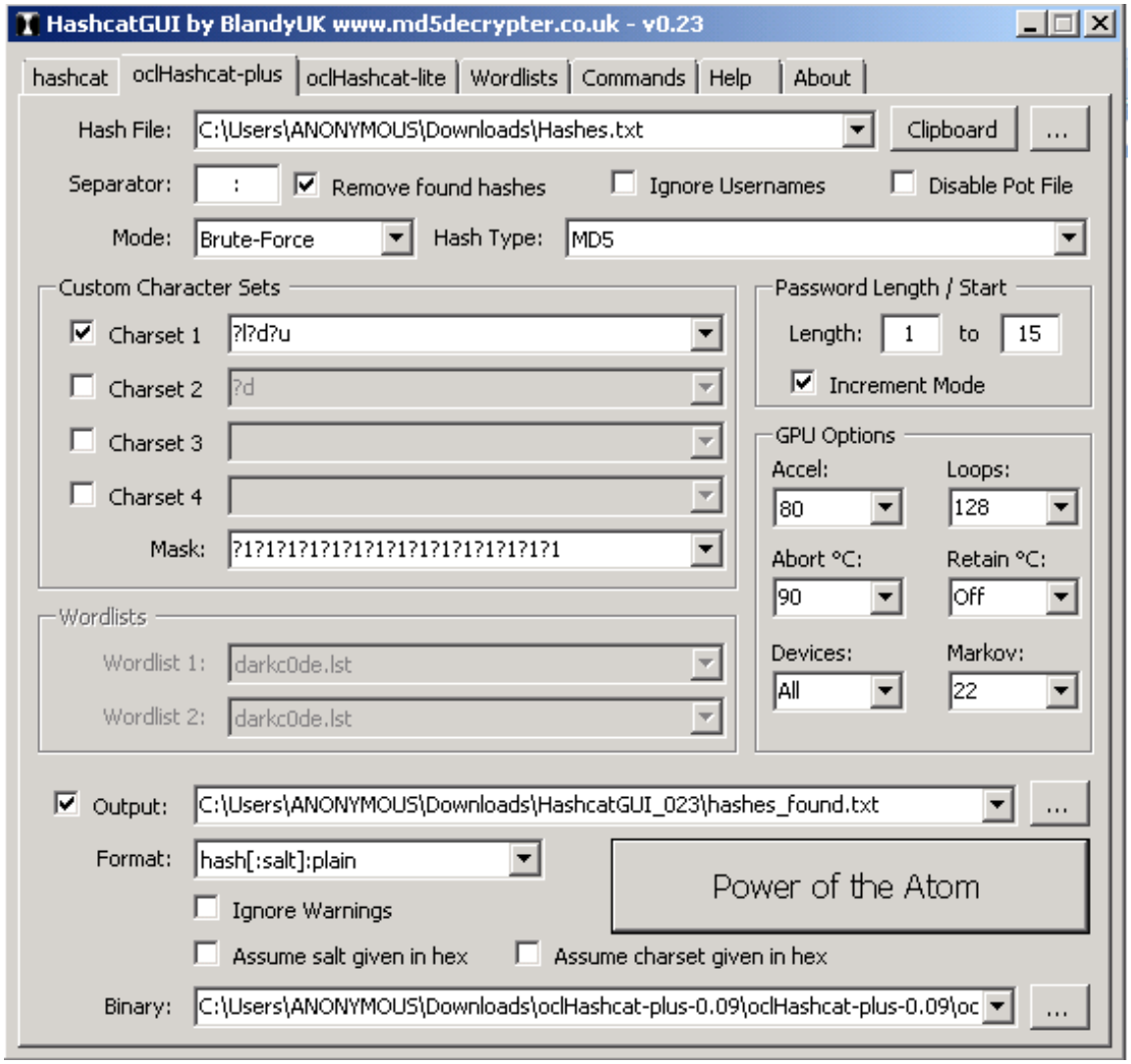
Vemos que también hay caracteres especiales como asteriscos y símbolos de admiración. Teniendo la ventaja del GPU podemos reducir esta lista rápidamente usando el modo –a 3 o Mask attack usando markov chains.

Para un usuario Linux el comando seria:

./oclhashcat-plus32.bin -a 3 -m 0 -o "/user/Desktop/hashes_found.txt" --outfile-format=3 -n 80 -t 22 --remove --gpu-loops=128 -1 ?l?d?u -i --increment-min=1 --increment-max=15 "/user/Desktop/Hashes.txt" ?1?1?1?1?1?1?1?1?1?1?1?1?1?1?1

no olvide que al usar markov reducimos o aumentamos el numero de combinaciones pero sin reducir el keyspace, si usamos markov 22 y en el keyspace tenemos ?1?1?1 seria 22^3 =10648 combinaciones, como tenemos el incremento habilitado luego tendríamos 22^4 y así sucesivamente.

Con lo que vera ya será suficiente para que usted pueda crear sus propios métodos, mascaras, con las reglas quizás tarde un poco en entenderlas pero no son nada complicadas, vea lo archivos predefinidos en la carpeta rules y tendrá una idea. Les dejaremos el code para los modos –a 6 y –a 7.



Aquí ya ajustamos nuestro Oclhashcat para iniciar el ataque, ahora damos click en Power of the Atom y veremos qué suerte nos trae.

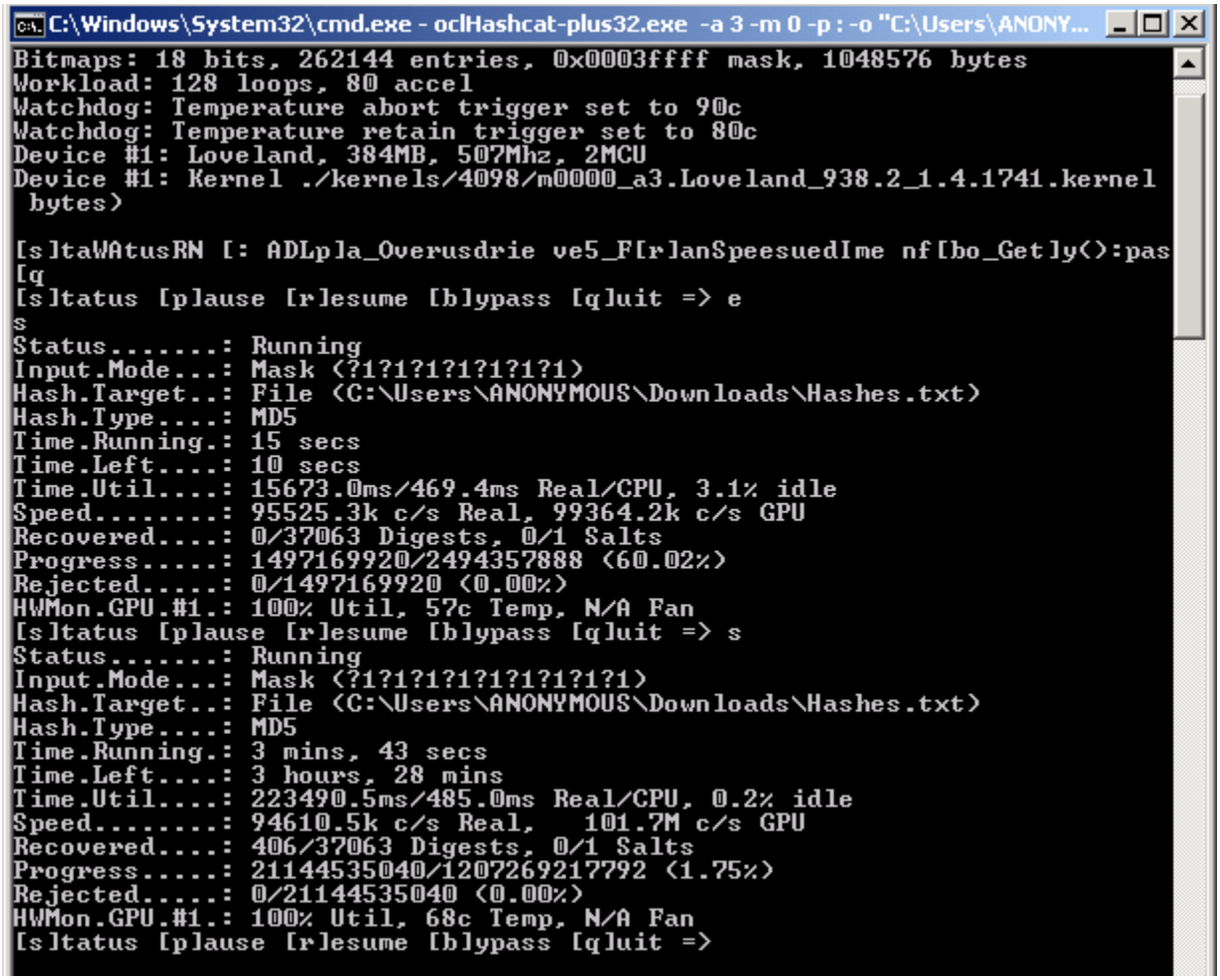
El code para los modos –a 6 seria:

./oclhashcat-plus32.bin -a 6 -m 0 -o "/user/Desktop/hashes_found.txt" --outfile-format=3 -n 80 -t 22 --remove --gpu-loops=128 -1 ?l?d?u "/user/Desktop/Hashes.txt" "/user/Desktop/darkc0de.lst" ?1?1?1?1

Y para el modo –a 7

./oclhashcat-plus32.bin -a 7 -m 0 -o "/user/Desktop/hashes_found.txt" --outfile-format=3 -n 80 -t 22 --remove --gpu-loops=128 -1 ?l?d?u "/user/Desktop/Hashes.txt"   ?1?1?1?1 "/user/Desktop/darkc0de.lst"

Una de las propiedades de las cadenas markov que también son aplicadas a las mascaras cuando usamos los modos –a 6 y –a 7



Como podemos observar en los primero minutos usando markov chains 22 recuperamos 406 hashes a una velocidad de aprox. 94610 millones de contraseñas por segundo, esta es la gran diferencia entre hashcat basado en CPU y Oclhashcat basado en GPU. lo que antes tomaria dias ahora nos tomara horas :D

Como puede ver estamos corriendo con el keyspace de 9 y markov 22 lo que seria 22^9 = 1207269217792 combinaciones.

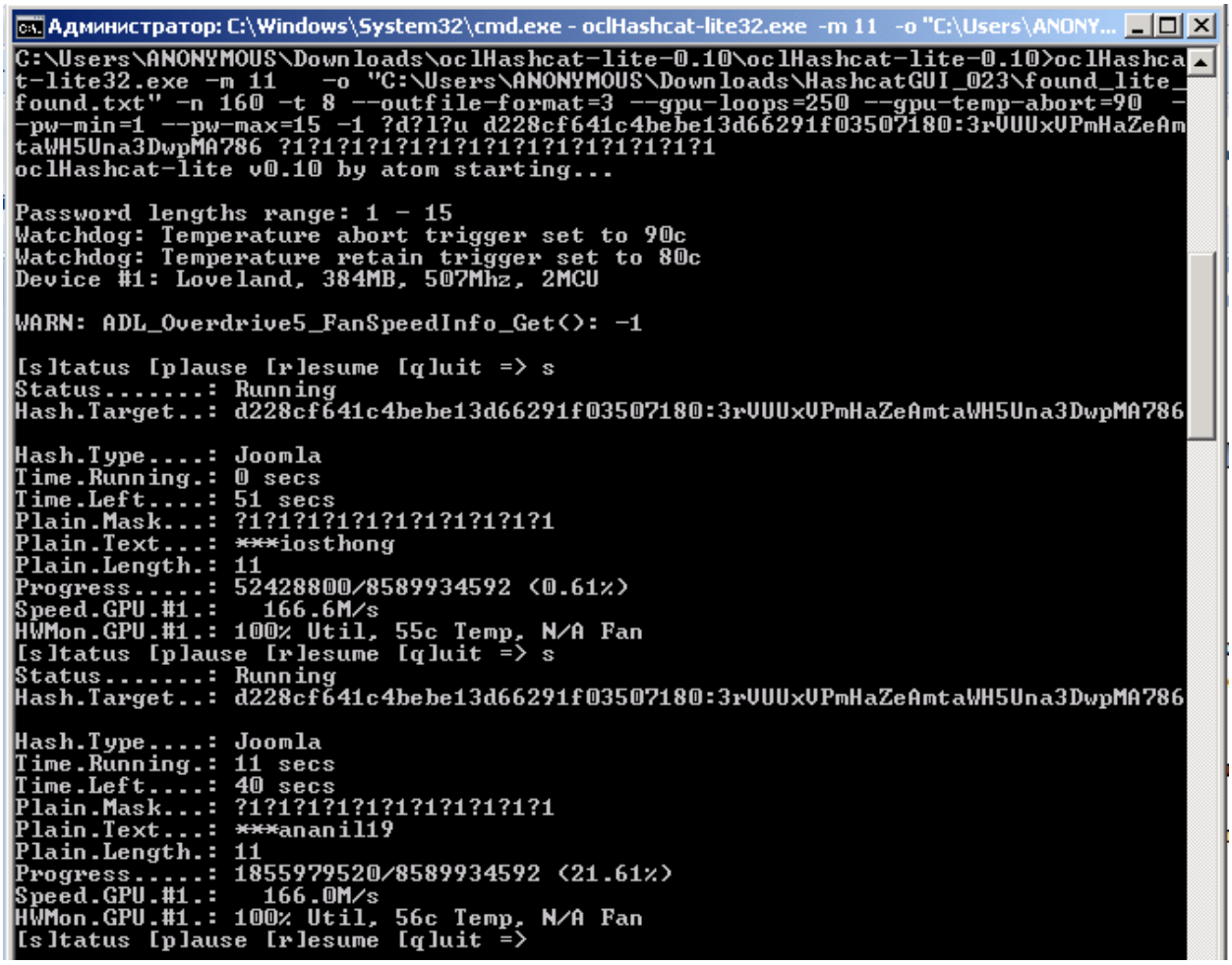
Con esto ya tendrá una idea básica de lo que es Oclhashcat, la idea principal de este manual es introducirle a esta herramienta y sus funciones básicas. Hay que remarcar que no siempre es fácil y hay que tener mucha paciencia y dedicación, puede que nosotros gastemos horas en tratar de recuperar un solo hash y no tengamos resultado, pero no hay que desanimarse, romper un hash puede tomar incluso semanas.

Existe otra herramienta llamada Oclhashcat lite que esta optimizada, y por lo tanto solo funciona con un solo hash, para esos casos de urgencia que pueden surgir, aunque no soporta todos los algoritmos, una gran variedad pueden ser usados.

http://hashcat.net/oclhashcat-lite/

Les mostraremos en breve como es, ya que los comandos y las opciones son análogas a sus contrapartes basadas en CPU y GPU no entraremos en detalles, les mostraremos como es en GUI y les dejaremos en código para los usuarios Linux.

./oclhashcat-lite32.bin -m 11 -o "/user/Desktop/found_lite_found.txt" -n 160 -t 8 --outfile-format=3 --gpu-loops=250 --pw-min=1 --pw-max=15 -1 ?d?l?u d228cf641c4bebe13d66291f03507180:3rVUUxVPmHaZeAmtaWH5Una3DwpMA786 ?1?1?1?1?1?1?1?1?1?1?1?1?1?1?1



Vemos claramente una mejora en la velocidad, llegamos a 166 millones de contraseñas usando un joomla que es un algoritmo más lento que el MD5, ya que tiene un salt, para poder hacer más lenta la recuperación.

En ciertos casos como en el joomla o el md5 podremos llegar a ver que tienen algo raro al inicio o al final

Esto se llama “salt”

ceea918e0fb9da9ffbfcb2a13b91ac5b: fs978

No hay que asustarnos, solo hay que usar el modo –m 10 o 10 = md5($pass.$salt)

Existe una gran variedad de hashes y que al inicio podrían asustarle, pero la realidad es que no son tan terribles, en el siguiente link se presentan diversos tipos, de los más comunes:

http://forum.insidepro.com/viewtopic.php?t=8225

Si también se pregunta cual tarjeta grafica es mejor para usar GPU, los expertos en fuerza bruta de han hecho un estudio podra verlo en este link:

http://www.gat3way.eu/est.php

pero en pocas palabras, una tarjeta AMD es mejor que una NVIDIA.

2.5 HASHCAT UTILS

Las utilerías de hashcat es un conjunto de pequeñas utilidades que le serán útiles rompiendo contraseñas, están hechas en ejecutables separados y están diseñadas para realizar una sola función, están hechas para las plataformas Windows y Linux de 32 y 64 bits.

http://www.hashcat.net/files/hashcat-utils-0.9-32.7z

http://www.hashcat.net/files/hashcat-utils-0.9-64.7z

en cuanto a lo que son no hay mucho que explicar, como podemos ver en la imagen

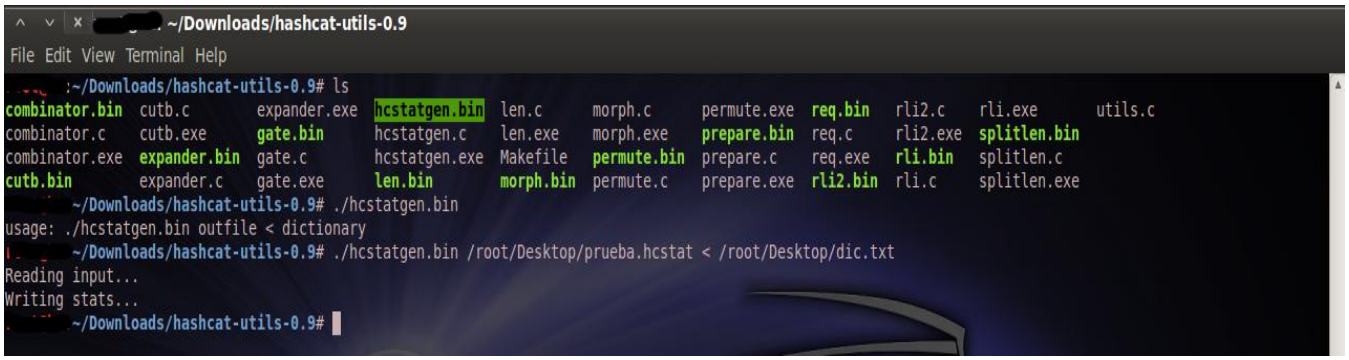
Los binarios en linux son .bin y los ejecutables en windows .exe

Para más detalles sobre que hace cada uno visite http://hashcat.net/wiki/doku.php?id=hashcat_utils

Aun sin saber mucho ingles los ejemplos explican bien que hace cada programa, les explicaremos únicamente hcstatgen.bin como les mencionamos antes que es con el procesaremos nuestro archivo .hcstat usando markov chains.

Antes que nada también tenemos que descargar el binario stastprocessor de http://hashcat.net/wiki/doku.php?id=statsprocessor

Primero seleccionemos un diccionario que tengamos, en nuestro caso usaremos uno que hicimos nosotros con passwords rotos.



Como vemos en la imagen fue algo sencillo, el resto de las utilidades lo son igual, solo tenemos que leer atentamente que función hace cada una y a nuestro juicio ver si esta acuerdo a nuestras necesidades. ya que tenemos nuestro .hcstat generado ahora lo procesaremos con statsprocessor.bin

Si tecleamos ./sp32.bin –help en la terminal, veremos que saldrán opciones parecidas a las de hashcat y oclhascat, estas las pueden ver en el homepage de la aplicación, por cuestiones de espacio no las mostraremos y nos enfocaremos en enseñarle como usarlo.

El código escrito fue ./sp32.bin --pw-min 5 --pw-max 5 --threshold=40 /root/Desktop/prueba.hcstat ?l?l?l?l?l | head -9

Solo para mostrarle los primero 9 resultados y podemos ver que las palabras que nos genera no son del todo no entendibles, esto es mas útil que generar todo el mapa de 5 caracteres nos ahorrara espacio y tiempo, esto lo podemos guardar en un archivo de salida usando –o /root/Desktop/dic.txt que podríamos usar como diccionario usando reglas o lo que se nos ocurra con él.

Para mas información visiten el foro del autor

http://hashcat.net/forum/thread-1265.html

2.6 SOLUCION DE PROBLEMAS

Primero asegúrate de que tu sistema está correctamente instalado, esto es que tengas todos los drivers, en el caso de usar la versión GPU se sugiere en AMD usar catalyst 12.8 y en NVIDIA usar el forceware correcto, si usas hashcat GUI debes tener el Framework version 2.4.

Dado que esta sección contiene demasiada información y los problemas que pueden surgir con cada usuario pueden ser únicos. Les sugerimos dirigirse al foro del autor si tiene algún problema

http://hashcat.net/forum

2.7 REFERENCIAS

md5decrypter http://forum.md5decrypter.co.uk/

Manual: http://hashcat.net/files/hashcat_user_manual.pdf

Hashcat suite. http://hashcat.net/forums

Rule-Fu: The art of word mangling. http://ob-security.info/?p=31

Rockyou dictionary http://www.skullsecurity.org/wiki/index.php/Passwords

Password exploitation class http://www.irongeek.com/i.php?page=videos/password-exploitation-class

Backtrack Linux http://backtrack-linux.org

2.8 MEJORAS

Consideramos que la sección de hashcat-utils y la sección de reglas de hashcat deberían ser mejoradas un poco en cuanto a explicación. Si hay algo más que no se entienda bien, nos enteraremos.