Rompiendo contraseñas con `Hashcat`

Carlos Ortega Marchamalo y Pablo Collado Soto
Puesto 4

1 Datos técnicos

Tal y como se explica en el guión de la práctica debemos escoger un fichero con *hashes* para analizar así como una herramienta de auditoría y un diccionario del que partir. En nuestro caso hemos optado por analizar los archivos `raw-md5.hashes4.txt` y `raw-md5.hashes5.txt`. Asimismo y tal y como se recomienda hemos trabajado con `hashcat` para intentar recuperar las contraseñas a partir de los *hashes* contenidos en dichos archivos. Finalmente señalamos que hemos empleado dos diccionarios base para nuestras pruebas. En primer lugar hemos empleado el archiconocido `rockyou.txt` así como el diccionario por defecto de la herramienta *John the Ripper* que, en nuestro caso, denominaremos `john.txt`. Este programa es muy similar a `hashcat` y tal y como aparece en [1] es venerado por su gran calidad. Destacamos asimismo que, tal y como comentaremos al analizar los distintos ataques en profundidad, las contraseñas contenidas en estos diccionarios no son las únicas que comprobaremos pues asistiremos a tipos de ataques que nos permiten recombinar el diccionario original para aumentar las candidatas a probar.

2 Problemas con la instalación

Antes de empezar a emplear `hashcat` para trabajar nos topamos con un ligero problema. Al invocar el programa para cerciorarnos de que todo funcionaba correctamente con `hashcat --benchmark` este nos informaba de no haber encontrado dispositivos compatibles para llevar a cabo su trabajo. Como nuestra intención era usar simplemente la `CPU` para calcular los *hashes* necesarios y la versión instalada se correspondía con este fin nos pareció realmente extraño estar en esta situación.

Tras navegar por Internet descubrimos que nuestro equipo no tenía el *OpenCL Runtime* instalado para procesadores *Intel Core*. Tal y como se aprecia en [2], este *framework* nos ofrece una *API* para ejecutar programas en máquinas heterogéneas. Nuestro sistema solo contaba con la implementación libre en vez de la oficial cosa que podría afectar a la velocidad de `hashcat`. Así, conseguimos encontrar y descargar el driver de [3]. Instalamos una serie de dependencias y tras ello ejecutamos el instalador.

Con todo listo simplemente ejecutamos `hashcat -I` para consultar los dispositivos detectados. El dispositivo que hace uso del driver que acabamos de instalar está asociado al identificador 2 en nuestro caso. Es por ello que una de las opciones que siempre aparecerá en nuestras órdenes es `-d 2` indicando que el dispositivo a emplear es aquel del que podremos obtener un mayor rendimiento.

3 Resumen ejecutivo

Si algo hemos sacado en claro tras realizar la práctica es la fragilidad de las contraseñas que como usuarios empleamos día a día. Como curiosidad probamos a atacar nuestras contraseñas generando el *hash* `MD5` de la misma con la siguiente orden:

```
echo 'nuestra_contraseña' | md5sum | awk '{print $1}' > my_pwd_hash.txt
```

Este es entonces el archivo a analizar y, sin sorpresa alguna, la contraseña se descubrió absurdamente rápido. Así, no solamente resulta inquietante lo rápido que se pueden romper las contraseñas sino también lo sencillo que resulta. Hacer lo que hemos hecho nosotros es cuestión de descargar una herramienta de Internet, instalarla y simplemente arramplar con lo que se desee. Esta facilidad pasmosa es una de las razones por la que tras acabar la práctica lo primero que hemos hecho ha sido cambiar las contraseñas de varios servicios. Saber que algo tan frágil es la primera línea de defensa ante los atacantes le quitaría el sueño a cualquiera...

Dejando de lado lo anterior también señalamos que hemos aprendido cómo auditar contraseñas de una manera sólida. Creemos que esta habilidad podría ser tremendamente útil de cara a la vida laboral y nos alegramos de haber podido comprender cómo funcionan estos procedimientos que, si nos guiamos por el cine y la televisión, parece estar al alcance de muy pocos.

También hemos interiorizado la diferencia clara que hay entre unos ataques y otros. Comprender este hecho es la clave para poder llevar a cabo estas auditorías en el menor tiempo posible manteniendo la mayor eficacia. Así, los ataques de diccionario son rápidos al ser sencillos pero no son capaces de "exprimir" los ficheros auditados al máximo. Otros métodos como la aplicación de reglas son más pesados al generar listas de palabras más largas pero por eso mismo obtienen una mayor cantidad de aciertos. Es por ello que resultan tremendamente atractivos porque en base a nuestros

resultados tienen una mayor cantidad de aciertos sin que ello incremente el tiempo de proceso en exceso. Ejemplos de reglas a aplicar son el cambio de mayúsculas por minúsculas y viceversa u otras más enrevesadas que incluyen rotaciones y sobreescrituras de caracteres como la `best64.rule`. Debido al espacio limitado del que disponemos no hemos aplicado esta última regla ya que hemos preferido ahondar más en cada tipo de ataque para intentar comprenderlo en su totalidad. Por último hemos intentado realizar un ataque de combinación en el que concatenamos cada palabra de un diccionario con todas las demás. Esta técnica genera diccionarios tremendamente extensos pero es por eso mismo que es capaz de encontrar muchísimas contraseñas si lo comparamos con otros tipos de ataques. Sin que ello suponga tampoco un aumento desmesurado del tiempo. En definitiva, debemos conocer estas sutilezas y tener claros nuestros objetivos para ser capaces de escoger el tipo de ataque que más posibilidades de éxito nos pueda aportar.

Finalmente señalamos que, motivados por la curiosidad, leímos acerca de métodos más avanzados de auditoría como pueden ser las `Rainbow Tables`. Nos pareció tremendamente interesante observar las estrategias para diseñar una estructura de datos que minimizara estos tiempos de búsqueda de contraseñas.

Con todo pasamos a comentar el análisis que hemos efectuado de cada uno de los 2 archivos.

4 Analizando `raw-md5.hashes5.txt`

4.1 Ataque de diccionario

Con este tipo de ataque pretendemos probar todas las contraseñas recogidas en un diccionario o *wordlist* para comprobar si se corresponde con alguno de los *hashes* contenidos en el archivo a analizar. En esta ocasión se probarán las palabras del diccionario "tal cual", no se modificarán de ninguna manera.

Llevar a cabo este tipo de ataque resulta sencillo empleando `hashcat`. Tras consultar el *manpage* de la herramienta con `man hashcat` vimos cómo teníamos que especificar una serie de parámetros. Los relacionamos a continuación:

- `-m 0`: El archivo analizado contiene *hashes* `MD5`.
- `-a 0`: Realizaremos un ataque de diccionario.
- `-o dict_hits.txt`: Almacenaremos las contraseñas obtenidas en el archivo `dict_hits.txt`.
- `--potfile-disable`: Impedimos que `hashcat` almacene las contraseñas ya descubiertas en un archivo interno (el `potfile`) para que cada vez que corramos un ataque partamos del mismo punto. Con ello podremos comparar los ataques entre sí evitando el sesgo que se introduce si alguno emplea *hashes* ya calculados.

Asimismo, ejecutamos el comando `time` para obtener de forma más cómoda los tiempos de ejecución del programa. En este caso es de $22,372\ s$. Resaltamos que hemos obviado parte de la salida que no aportaba información relevante. Además, esta salida no se aporta en posteriores ejemplos ya que discutimos los resultados en el informe.

```
$ time hashcat -m 0 -a 0 -d 2 -o dict_hits.txt --potfile-disable raw-md5.hashes5.txt rockyou.txt
hashcat (v5.1.0) starting...

* Device #1: Not a native Intel OpenCL runtime. Expect massive speed loss.
             You can use --force to override, but do not report related errors.
OpenCL Platform #1: The pocl project
====================================
* Device #1: pthread-Intel(R) Core(TM) i7-5500U CPU @ 2.40GHz, skipped.

OpenCL Platform #2: Intel(R) Corporation
========================================
* Device #2: Intel(R) Core(TM) i7-5500U CPU @ 2.40GHz, 3982/15928 MB allocatable, 4MCU

# Parte de la salida omitida por claridad

Session..........: hashcat
Status...........: Exhausted
Hash.Type........: MD5
Hash.Target......: raw-md5.hashes5.txt
Time.Started.....: Sat May  2 14:47:05 2020 (12 secs)
Time.Estimated...: Sat May  2 14:47:17 2020 (0 secs)
Guess.Base.......: File (rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#2.........:  1202.4 kH/s (0.54ms) @ Accel:1024 Loops:1 Thr:1 Vec:8
Recovered........: 156897/3500000 (4.48%) Digests, 0/1 (0.00%) Salts
Recovered/Time...: CUR:N/A,N/A,N/A AVG:787827,47269620,1134470896 (Min,Hour,Day)
Progress.........: 14344384/14344384 (100.00%)
```

```
Rejected.....: 0/14344384 (0.00%)
Restore.Point....: 14344384/14344384 (100.00%)
Restore.Sub.#2...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidates.#2....: $HEX[206b6d3831303838] -> $HEX[042a0337c2a156616d6f732103]

Started: Sat May  2 14:46:55 2020
Stopped: Sat May  2 14:47:18 2020

real    0m22.372s
user    0m32.978s
sys     0m8.800s
```

Así, podemos intentar comprobar el número de contraseñas del diccionario `rockyou.txt` con `wc -l rockyou.txt`. Con ello vemos que, en teoría, hemos probado 14344391 contraseñas o lo que es lo mismo, el esfuerzo máximo del ataque habría sido de 14344391 *hashes*. Esta correspondencia directa entre el número de contraseñas y el esfuerzo máximo se debe a que, tal y como comentábamos, no estamos alterando el diccionario de ninguna manera. No obstante, la salida del programa nos indica que se han probado 14344384 contraseñas, 7 menos de las que esperábamos probar. Así, consultando el diccionario con `tail -n 20 rockyou.txt` veremos que este incluye algunas líneas en blanco que `hashcat` no estará comprobando como contraseña. En definitiva, el valor correcto es el que se indica en la salida del programa. Además, la correspondencia número de contraseñas-esfuerzo máximo se mantiene intacta.

Con este ataque probamos todas las contraseñas del diccionario. Por comodidad podemos ejecutar `head rockyou.txt` para consultar las primeras líneas de nuestra *wordlist*. Con ello vemos que una de las contraseñas probadas ha sido, por ejemplo `princess`. Podemos ver también por ejemplo como se ha probado la contraseña `harrypotter` en base a la salida del programa y esta está en el archivo diccionario, cosa que se puede comprobar con `cat rockyou.txt | grep 'harrypotter'`.

Solo nos queda comentar el número de contraseñas encontradas. Procediendo de la misma forma que para encontrar el esfuerzo máximo veremos que el número de líneas del archivo `dict_hits.txt` es 156897, esto es, hemos recuperado 156897 contraseñas en tan solo $27,372\ s$. Esto supone alrededor de un $4,5\%$ del total de contraseñas del archivo auditado.

4.2 Ataque con reglas

Ahora vamos a emplear el diccionario `john.txt` como base dado que al ser más pequeño que `rockyou.txt` resulta más manejable mientras que los conceptos siguen siendo idénticos.

En vez de usar el diccionario "tal cual" vamos a alterar los caracteres de sus palabras de uno en uno de manera que las mayúsculas pasen a ser minúsculas y viceversa. En otras palabras, si nuestro diccionario contuviera la palabra `abc` la nueva versión tendría: `Abc, aBc, abC`. No obstante, la regla que aplicaremos tiene en cuenta palabras de 15 caracteres con lo que cada palabra del diccionario generará 15 nuevas. El inconveniente es que si la palabra tiene menos de 15 caracteres, digamos 7, entonces tendremos 8 veces la misma contraseña repetida en la salida, esto es, comprobaremos la misma contraseña 8 veces. Teniendo esto en cuenta veremos que si muchas de las contraseñas son relativamente cortas haremos muchas comparaciones totalmente inútiles... Es por ello que nos planteamos trabajar con el diccionario para eliminar todos los duplicados, cosa que podemos lograr gracias al comando `awk`. Para poder obtener el diccionario modificado con la regla de interés haremos uso de la opción `--stdout` de `hashcat` que nos permite imprimir las palabras generadas por pantalla tal y como aparece en [4]. Aprovechando una redirección podemos generar el archivo. Señalamos que para aplicar esta regla en el ataque se debe emplear la opción `-r <ruta/a/la/regla>`. Hemos incluido la ruta absoluta para una mayor claridad.

```
# Contiene las originales y las modificadas con duplicados
hashcat -r /usr/share/hashcat/rules/toggles1.rule --stdout john.txt > john_tog.txt

# Contiene las originales y modificadas sin duplicados
hashcat -r /usr/share/hashcat/rules/toggles1.rule \
    --stdout john.txt | awk '!foo[$0]++' > john_tog_clean.txt
```

Si aplicamos el ataque con el diccionario original obtenemos un tiempo de ejecución de $11,164\ s$ y se recuperan 154 contraseñas. El esfuerzo máximo es de 46605 *hashes*, cosa que cuadra perfectamente con lo que cabría esperar. El diccionario `john.txt` contiene 3107 contraseñas y la regla genera 15 contraseñas por cada original con lo que, en efecto, contamos con $3107 \cdot 15 = 46605$ contraseñas. Teniendo en cuenta que la palabra `hello` se incluye en el diccionario original una de las contraseñas que esperamos probar es `Hello`, por ejemplo. Para ejecutar el ataque el comando ha sido:

```
time hashcat -m 0 -a 0 -d 2 -r /usr/share/hashcat/rules/toggles1.rule \
    --potfile-disable raw-md5.hashes5.txt john.txt
```

Si ejecutamos el ataque con el diccionario filtrado veremos que recuperaremos el mismo número de contraseñas en un tiempo ligeramente menor ya que en este caso solo probamos 20814 contraseñas al haber eliminado los duplicados. Para obtener el esfuerzo máximo en este caso dependemos de la orden `wc -l john_tog_clean.txt` ya que este depende de la

longitud de las propias palabras. El número de duplicados depende del número de caracteres de cada palabra con lo que solo podemos afirmar el esfuerzo máximo del diccionario generado con la regla sin modificar de manera determinista. Ejecutar este ataque supone lanzar:

```
time hashcat -m 0 -a 0 -d 2 --potfile-disable raw-md5.hashes5.txt john_tog_clean.txt
```

Finalmente comentamos que si llevamos a cabo un ataque de diccionario como en el apartado anterior obtendremos un total de 28 contraseñas. Al aplicar la regla obtenemos 154 con lo que $154 - 28 = 126$ son las contraseñas adicionales que hemos obtenido gracias a cambiar mayúsculas por minúsculas y viceversa, lo que es un 450% de las obtenidas sin modificar el diccionario. Para cerciorarnos de esto lanzamos:

```
time hashcat -m 0 -a 0 -d 2 --potfile-disable raw-md5.hashes5.txt john.txt
```

Señalamos que dada la discusión que hemos llevado a cabo no hemos incluido la salida explícita de las órdenes en aras de aprovechar el espacio limitado al máximo. Nos disculpamos en caso de haber errado con nuestra decisión. No obstante, estas salidas son idénticas a la que se adjuntaba en el primer caso variando solo los valores que hemos incluido en estos últimos párrafos.

5 Analizando `raw-md5.hashes4.txt`

5.1 Ataque de diccionario

Para establecer una comparación con el archivo anterior llevamos a cabo este ataque con el diccionario `rockyou.txt`. Los resultados muestran que se han recuperado 156806 contraseñas en un tiempo de $9,7\ s$, es decir, en torno a un $4,5\%$ del total también. Al haber discutido las vicisitudes de este tipo de ataque anteriormente pasamos a comentar otros tipos de ataque.

5.2 Ataque de combinación

Haciendo uso del ataque de diccionario se nos presenta una limitación muy importante pues simplemente probamos con las contraseñas contenidas en ese diccionario tal y como aparecen en él por lo que no se tienen en cuenta posibles claves que consisten en combinaciones o concatenaciones de esas contraseñas incluidas en el diccionario. Es por ello que hemos decidido a dar un paso más y tener esta posibilidad presente.

Así, llevamos a cabo el ataque de **combinación** el cual se apoya en dos diccionarios para construir las nuevas contraseñas, encadenando cada una de las palabras de uno con cada una del otro. Como consecuencia, se llegan a probar hasta $3107 \cdot 3107 = 9653443$ contraseñas teniendo en cuenta que empleamos el diccionario `john.txt`, o lo que es lo mismo, el esfuerzo máximo es de 9653443 *hashes* pues el diccionario, que es empleado dos veces, consta de 3107 palabras.

Para tener una mejor perspectiva de los beneficios de utilizar este método hemos recurrido antes al ataque de diccionario propiamente dicho, con el que hemos obtenido un resultado de **36** contraseñas rotas en un tiempo de $10,198\ s$. Para esto hemos seguido el mismo procedimiento que en el ataque de diccionario mencionado previamente con la variación de que en esta ocasión el archivo de *hashes* a auditar es *raw-md5.hashes4.txt*.

Posteriormente ponemos en marcha el ataque de combinación, estableciendo para ello la opción `-a 1`. Por consiguiente, el comando resultante a lanzar es:

```
time hashcat -m 0 -a 1 -d 2 --potfile-disable raw-md5.hashes4.txt john.txt john.txt
```

Observamos como, en tan solo $11,207\ s$, se han recuperado 39418 contraseñas. De este modo apreciamos los enormes beneficios y la inmensa diferencia entre usar un diccionario directamente y combinar entre ellas las palabras que en él se encuentran.

Estudiando la información que se muestra tras concluir el ataque vemos como, efectivamente, se han probado las 9653443 contraseñas esperadas.

Contemplando las posibles claves a probar advertimos como se intentará, por ejemplo, con `michaeljordan` ya que las palabras `michael` y `jordan` se encuentran en el diccionario. En este caso podemos comprobar como se ha llegado a romper la contraseña `adidaschelsea`, encontrándose cada una de estas dos palabras en el diccionario considerado.

6 Referencias

1. Bibliografía aportada por el guión de la práctica.
2. Enlace de descarga para el diccionario `john.txt`.
3. Información de `OpenCL`.
4. Enlace de descarga del driver `OpenCL`.
5. Ataques basados en reglas con `hashcat`.