INGENIERÍA DE LA CIBERSEGURIDAD

PRÁCTICA 1 - Estudio de la fortaleza de contraseñas

Doble Grado en Ingeniería Informática y ADE

Curso 2023/24

Grupo 80

Realizado por:

Arianna Potente Vázquez [100432091], 100432091@alumnos.uc3m.es

Ernesto Gracia Cancho [100432026], 100432026@alumnos.uc3m.es

ÍNDICE

1.- Introducción

En esta primera práctica de ingeniería de la ciberseguridad, se tratará el tópico de la fortalezas de las contraseñas. Como sabemos, hay innumerables formas de romper una contraseña, siendo la más básica de todas la fuerza bruta. Esta técnica es la que se va a emplear en la práctica y probar diferentes combinaciones de estrategias de fuerza bruta para poder las contraseñas de 25 datasets que crearemos según unas reglas establecidas en el enunciado.

Por último, el orden que seguirá la memoria del proyecto consiste en explicar las estrategias y lógica seguida a la hora de crear los datasets en “Generación de los datasets”, las estrategias implementadas para crackear las contraseña tanto las exitosas como las fallidas en “Metodología” y por último una descripción de los resultados obtenidos (tiempo, media, mediana y porcentaje de contraseñas rotas) de la mejor estrategia que ha funcionado para cada dataset y un análisis de estos mismo resultados en “Resultados y análisis”. Conviene subrayar que, para mayor información sobre las estrategias de crakeado y resultados de tiempo que no estén en forma tabular (es decir, una captura de pantalla) se encuentran en el anexo.

2.- Generación de los datasets

Para la generación de los datasets hemos empleado la herramienta de **Crunch**, que está específicamente diseñada para la generación de diccionarios que se pueden crackear utilizando fuerza bruta. Antes de nada, conviene subrayar que para que la tarea de crackear las contraseñas con JTR (John The Ripper) sea más sencilla más adelante hemos decidido generar datasets exclusivamente de longitud 3, longitud 4, longitud 5, longitud 6 y longitud 7 en vez de generar cinco datasets para cada parte con longitudes aleatorias de 3 a 7, a excepción de los últimos cinco datasets cuya longitud de contraseñas ya venía dada por el diccionario existente de internet.

A su vez, es importante remarcar que para la realización de contraseñas hemos hecho una búsqueda exhaustiva por el manual de Crunch para entender la herramienta y las diferentes funcionalidades que presenta con el objetivo de crear las contraseñas de la manera más eficiente posible.

Dicho esto procederemos a explicar la creación de los 25 datasets por orden:

1. **Parte 1 (letras minúsculas):** la herramienta de Crunch contiene una lista de datos (“.lst”) con conjuntos de caracteres ya establecidos. Con el fin de evitar poner las 26 letras del alfabeto para la creación de las contraseñas, nos hemos metido en la lista **charset.lst** que se encontraba en el directorio “/usr/share/crunch” y así encontrar el conjunto de caracteres que más se adecuaba con esta parte, “lalpha” (que corresponde únicamente a las 26

letras del alfabeto minúsculas, l=lower, alpha = alphabetical), quedando la regla como "**-f** /usr/share/crunch/charset.lst lalpha".

Luego se establece la longitud máxima y mínima de las contraseñas que contendrá el diccionario (como se ha mencionado anteriormente 3 y 3, 4 y 4 etc.). Para los tres primeros datasets hemos implementado lo que se acaba de explicar y además hemos definido el fichero de salida donde contendrán todas las combinaciones de contraseñas según las reglas establecidas con la regla "**-o** datasetX.txt" (siendo X el número de dataset correspondiente).

Por último, para los dos últimos datasets, además hemos especificado un patrón específico para que sigan las contraseñas creadas, de modo que, a la hora de crackearlas sea mucho más sencillo; puesto que podemos especificar a jtr el patrón que hemos seguido y **este descartará el resto de combinaciones posibles.** Esto lo hemos conseguido con la regla "**-t**", quedando el resultado como por ejemplo "-t @@uc@@" donde los dos primeros caracteres y los dos últimos son minúsculas y los dos caracteres del centro son fijos siendo las letras u y c.

Una ilustración de cómo quedaría el código para crear un diccionario teniendo en cuenta lo anteriormente mencionado sería: "crunch 6 6 -f /usr/share/crunch/charset.lst lalpha -t @@uc@@ -o dataset4.txt".

2. **Parte 2 (letras mayúsculas):** Nuevamente para la creación de los siguientes datasets, hemos añadido la longitud máxima y mínima de las contraseñas que se van a crear. Asimismo, hemos establecido el fichero de salida que contendrá todas las combinaciones de las contraseñas creadas (**-o** datasetX.txt). Para evitar escribir las 26 letras del alfabeto en mayúsculas hemos buscado en la lista **charset.lst** este conjunto de caracteres que se corresponde con "ualpha" (u = upper, alpha = alphabetical), quedando la regla como "**-f** /usr/share/crunch/charset.lst ualpha".

   Para terminar, para los dos últimos datasets, hemos establecido un patrón para que sigan las contraseñas creadas al igual que se ha explicado anteriormente. Sin embargo, en la regla en vez de utilizar las "@'s" ya que estas se corresponden con caracteres minúsculas, hemos utilizado las "**,**'s" que se corresponden con los caracteres mayúsculas. Un ejemplo de esto sería: ",Z,A,," donde el primer, tercer y dos últimos caracteres son mayúsculas del alfabeto, mientras que el segundo y tercer carácter son letras fijas, Z y A, que no se van a cambiar en la combinaciones.

   Una ilustración de cómo quedaría el código para crear un diccionario teniendo en cuenta lo anteriormente mencionado sería: "crunch 6 6 -f /usr/share/crunch/charset.lst ualpha -t ,Z,A,, -o dataset9.txt".

3. **Parte 3 (números):** como los números son únicamente 10 caracteres en total pensamos que para esta parte no era necesario establecer ningún patrón en específico como sí que hicimos con las partes anteriores. Sin embargo, pudimos comprobar que para los dos últimos datasets (de longitud 6 y 7), sí que tardaba bastante tiempo a pesar de ser menos de la mitad de caracteres que el alfabeto. Dicho esto, la única diferencia por tanto que se ha implementado en esta parte es el conjunto de caracteres empleados de la lista **charset.lst** siendo este el "numeric", quedando la regla como "**-f** /usr/share/crunch/charset.lst numeric".

   Una ilustración de cómo quedaría el código para crear un diccionario teniendo en cuenta lo anteriormente mencionado sería: "crunch 3 3 -f /usr/share/crunch/charset.lst numeric -o dataset11.txt".

4. **Parte 4 (caracteres alfanuméricos y símbolos):** Nuevamente para la creación de los siguientes datasets, hemos añadido la longitud máxima y mínima de las contraseñas que se van a crear. Asimismo, hemos establecido el fichero de salida que contendrá todas las combinaciones de las contraseñas creadas (**-o** datasetX.txt). Para evitar escribir los 93 caracteres hemos buscado en la lista **charset.lst** este conjunto de caracteres que se corresponde con "mixalpha-numeric-all" (mixalpha = alfabeto mayusculas y minúsculas, numeric = números, all = símbolos), quedando la regla como "**-f** /usr/share/crunch/charset.lst mixalpha-numeric-all".

   Para terminar, excepto para el primer dataset, hemos establecido un patrón para que sigan las contraseñas creadas al igual que se ha explicado anteriormente. La razón de porque en el primer dataset no hemos especificado un patrón es simplemente que teníamos que elegir entre 4 conjuntos de caracteres distintos para poner en una contraseña de longitud 3 luego uno de los conjuntos quedaría inutilizado incumpliendo la exigencia del enunciado y por ende lo hemos dejado al azar. Para el resto de datasets hemos ido alternando entre caracteres fijos y aleatorios que estén dentro de los conjuntos de caracteres teniendo en cuenta que:
   - @ = minúscula.
   - , = mayúscula.
   - % = números.
   - ^ = símbolos.

   Un ejemplo de esto sería: ",@0%^" donde el primer carácter es una mayúscula, el segundo carácter es una minúscula, el tercer carácter es un número fijo,0, el cuarto carácter es un número y el quinto carácter un símbolo. Nuevamente, hemos decidido hacer esto así el número de combinaciones posibles de caracteres alfanuméricos + símbolos se ven reducidas

Una ilustración de cómo quedaría el código para crear un diccionario teniendo en cuenta lo anteriormente mencionado sería: “crunch 5 5 -f /usr/share/crunch/charset.lst mixalpha-numeric-all -t ,@0%^ -o dataset18.txt”.

5. **Parte 5 (palabras de diccionario):** Para los últimos cinco datasets se precisaba que las palabras fueran tomadas de un diccionario. Para facilitar esta tarea nos descargamos un diccionario de GitHub de los apellidos más comunes estadounidenses (top_family_names_usa.txt), de manera que solo lidiaramos con caracteres alfabéticos. A su vez, conviene subrayar que no establecimos ninguna regla de longitud, ni precisamos la herramienta de Crunch, simplemente seleccionamos 100 apellidos aleatorios del diccionario.

Hay que destacar, que una vez creados los 25 datasets, ya que estos estaban compuesto por muchas líneas y solo se requieren 100 contraseñas por dataset, escogimos 100 contraseñas aleatorias de cada dataset con la línea:

- “shuf -n 100 datasetX.txt > datasetfinalX.txt” (donde la X se corresponde con el número de dataset).

Por último, es importante mencionar que todas la líneas empleadas para la creación de los datasets de cada parte están recogidas en el fichero de texto “LÍNEAS DE CREACIÓN DE DATASETS”, por si se requiere su examinación.

3.- Metodología

En esta sección, vamos a describir las diferentes configuraciones de John the Ripper que hemos empleado para cada dataset generado en la práctica. Además, describiremos cuál es el propósito de cada estrategia y el tipo de contraseñas que queremos romper.

En primer lugar, para explicar correctamente este punto, vamos a detallar tanto las estrategias finales que hemos empleado en la práctica, como las estrategias que no hemos implementado pero que sí habíamos contemplado durante el desarrollo de esta prueba. Por tanto, dicho lo anterior, dividiremos esta sección en estrategias utilizadas (aplicadas en la práctica) y estrategias contempladas (pensadas pero no implementadas).

Estrategias utilizadas

1. **Mask + Min-length + Max-length + Format**

   Esta estrategia consiste en especificar a John the Ripper el aspecto que posee cada posición de las contraseñas que tiene que crackear. El propósito de esta estrategia consiste en especificar de forma detallada la estructura de las contraseñas con el objetivo de facilitar el recorrido de John the Ripper a la hora de buscarlas. Por ello, si le indicamos por ejemplo a John que las contraseñas únicamente tienen tres

posiciones y que solo están formadas por minúsculas, reducimos o limitamos el espectro de búsqueda considerablemente.

Por consiguiente, hemos decidido emplear esta estrategia ya que la podemos emplear o aplicar a todos los datasets que nos hemos creado puesto que nosotros ya conocemos cual es la composición de cada dataset y por tanto, podemos detallar cual es la morfología de cada conjunto de contraseñas que tenía que descifrar. Por tanto, como hemos mencionado, el ámbito de aplicación de esta estrategia en este proyecto ha sido notable puesto que los hemos podido aplicar a 20 de los 25 datasets que nos debíamos crear. Los únicos que nos soportan esta estrategia son los datasets de la parte 5 (diccionarios). Esto se debe a que como hemos obtenido los diccionarios de internet, hemos decidido emplear otra estrategia más óptima que explicaremos posteriormente en el proyecto que consiste en la modificación del fichero de reglas de John the Ripper.

Debemos destacar también, que la especificación de la máscara varía en función de la parte de datasets en la que nos encontremos, pudiendo encontrar los siguientes casos:

- mask='?l?l?l' → minúsculas
- mask='?u?u?u' → mayúsculas
- mask='?d?d?d' → dígitos
- mask='?a?a?a' → alfanuméricos

Además, debemos mencionar que una subestrategia de la utilización de la máscara ha sido la creación de datasets que tenían algunas palabras fijas en la creación de las mismas como por ejemplo _ _ u c m _ _ (para contraseñas de longitud 7). En estos casos la utilización de la máscara ha sido clave puesto que el número de combinaciones que debe generar John se reduce considerablemente mejorando la eficiencia de esta estrategia.

En cuanto al resto de reglas, hemos decidido combinar mask, con min-length y max-length con el objetivo de acotar aún más el espectro de actuación de John. De esta forma reducimos considerablemente el tiempo de búsqueda y agilizamos el proceso.

Por último, hemos creído conveniente especificar el formato de todos los datasets a md5crpt puesto que este es el algoritmo de hasheado que hemos empleado en la creación de los datasets. Como vimos en clase, identificar y especificar el algoritmo de hasheo correctamente facilita aún más la tarea de John.

2. <u>Wordlist + Min-length + Max-length + Format + Rules</u>

Esta estrategia consiste en llevar a cabo el ataque por diccionario. Como vimos en clase, el ataque por diccionario consiste en que John the Ripper va a ir contrastando todas las palabras que se encuentran en el diccionario proporcionado, con las

palabras que debe descifrar. En términos de complejidad, esta estrategia es idónea ya que únicamente John tiene que contrastar las palabras generadas en el dataset, reduciendo exponencialmente el número de combinaciones que debería generar y posteriormente comparar en caso de no disponer de un wordlist. Esta regla, es más notable para aquellos datasets donde hemos especificado que algunas posiciones son fijas dentro de la formación de palabras, como por ejemplo: _ _ u c m _ _. En este caso el wordlist solo deberá comparar con aquellas palabras que contengan estos tres caracteres reduciendo considerablemente el número de comparaciones o generaciones (en caso de no hacer wordlist), que deberá realizar John the Ripper. Por tanto, como se puede observar, wordlist nos permite obtener una mayor eficiencia y una mayor probabilidad de éxito puesto que en nuestro caso, nos aseguramos que las contraseñas que debe descifrar se encuentren en el dataset proporcionado. Además, debemos resaltar que la utilización de recursos es menor ya que se reduce la carga del sistema.

Por consiguiente, hemos empleado esta estrategia en todos los datasets del proyecto puesto que como hemos mencionado, es una de las formas más eficientes y menos costosas computacionalmente hablando. Además, hemos podido proporcionar los datasets de una forma sencilla, puesto que hemos asignado los datasets creados desde la parte 1 hasta la 5, en el apartado de formación de datasets. Hemos decidido llevarlo a cabo de esta forma debido a que la selección de las 100 contraseñas ha sido de forma aleatoria. Por ello, la mejor manera de garantizar el éxito del crackeo era proporcionar el diccionario que ya habíamos creado previamente. La realización de la parte 5 ha sido de la misma manera que las partes anteriores, a excepción de que el diccionario lo hemos obtenido de internet; sin embargo, el tratamiento de las contraseñas aleatorias ha sido el mismo.

Con respecto al resto de reglas, hemos decidido combinar wordlist con min-length y max-length meramente por temas de facilidad a la hora de emplear los comandos en la terminal y llevar a cabo la práctica. Es decir, no lo hemos empleado por un tema de mejora de rendimiento y tiempo de John; ya que sabemos que estos dos parámetros no tiene sentido definirlos puesto que John va a comparar únicamente los los valores del diccionario y no le va a importar la longitud que le hemos especificado en nuestro caso concreto. Esto se debe, a que la longitud de las palabras del dataset ya se van a corresponder directamente con las palabras que tiene que crackear porque están sacadas de ese mismo diccionario.

Con respecto al format, hemos decidido volver a especificarlo con el objetivo de facilitar de nuevo la tarea de John the Ripper a la hora de descifrar las contraseñas como hemos explicado previamente.

Por último, hemos empleado una de las reglas predefinidas por John the Ripper denominada como “Single”.

3. **Incremental+Format+Max-length+Min-length**

Esta estrategia consiste en llevar a cabo ataques de fuerza bruta mediante la funcionalidad por defecto “incremental”. El objetivo de esta es la de crear y probar de forma sistemática todas las combinaciones entre un rango especificado. Por consiguiente, se caracteriza por ser un proceso bastante exhaustivo y eficaz cuando se acotan las características de las contraseñas y estas son de una longitud reducida. Esto quiere decir que por ejemplo, esta funcionalidad es idónea para obtener contraseñas que sean números, letras mayúsculas/minúsculas, símbolos o combinación de ellas pero con una longitud reducida. Además, como es lógico, no tiene sentido emplear esta técnica para contraseñas excesivamente largas pues el tiempo de ejecución incrementa exponencialmente. Por ello, en este proyecto, hemos decidido emplear esta estrategia en datasets cuya longitud es pequeña; siendo estos para nosotros los de longitud 3 y 4 como se puede observar en el fichero txt “LINEAS JTR CRACKEO” adjuntado en la carpeta de entrega de proyecto.

Por consiguiente, consideramos que esta estrategia era idónea para contraseñas cortas puesto que al ser un servicio más sistemático, podría llegar a sacar las contraseñas de una forma más rápida y así mejorar el resto de estrategias. De hecho, como explicaremos a continuación, se podrá ver que sí que existen algunos casos en los que mejora los tiempos para contraseñas cortas.

En cuanto al resto de comandos, “format” nos permite de nuevo facilitar el trabajo de John mostrando el algoritmo de hash del documento que le estamos pasando. Esta característica, como hemos mencionado antes, mejora la eficiencia del proceso.

Por último, como es lógico, hemos especificado la longitud máxima y mínima de las contraseñas para acotar el empleo del incremental y evitar que realice esfuerzos innecesarios como por ejemplo buscar contraseñas de longitud 1 o 2.

4. **Mask + Min-length + Max-length + Format + Fork / Wordlist + Min-length + Max-length+Format+Rules+Fork/ Incremental+Format+Max-length+Min-length + Fork**

Estas estrategias tienen las mismas características que la primera, segunda y tercera respectivamente. La única diferencia es que hemos intentado reducir el tiempo de ejecución de John mediante el comando fork. Esta estrategia surge debido a que nos dimos cuenta de que John corría siempre con dos hilos en OpenMP. Al ver esta situación, consideramos la posibilidad de ejecutar con más hilos ya que en teoría, si aumentamos el número de hilos de ejecución, la creación de contraseñas y posterior comprobación con las contraseñas sería más rápida.

Esta estrategia la llevamos a cabo en algunos datasets que se puede ver en el fichero de texto. Al observar los resultados nos quedamos atónitos puesto que el tiempo de ejecución aumentó en vez de reducirse.

5. **Ordenación de datasets con wordlist**

Esta estrategia tiene los mismos comandos que la anterior estrategia de wordlist. La única diferencia ha sido la ordenación del dataset de 100 contraseñas que obtenemos del dataset general creado con crunch. Esta estrategia únicamente la hemos probado para los datasets correspondientes a la Parte3 (números), donde las contraseñas que nos debíamos crear eran números aleatorios dentro de una longitud determinada.

Nuestra lógica detrás de esta estrategia era que ya que el diccionario que le pasamos a John se encuentra ordenado con todas las formaciones de números posibles dentro de un rango numérico, si las contraseñas que debía romper estaban ordenadas numéricamente de menor a mayor, john podía tardar menos en descifrarlas; ya que de esta forma no debía recorrer todo el diccionario al ejecutarse.

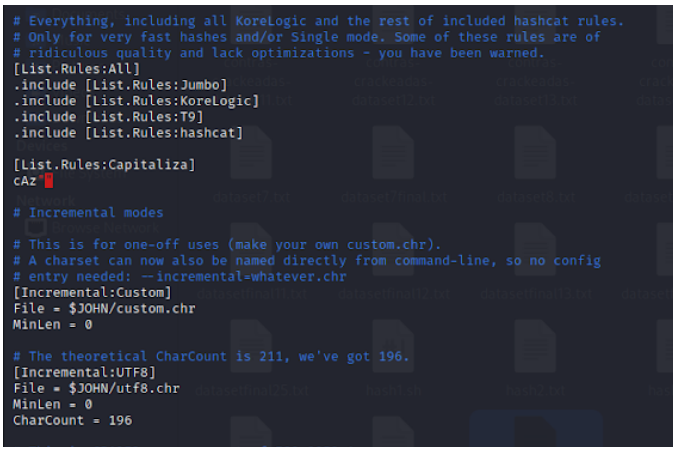
Sin embargo, tras realizar mediciones de tiempo, comprobamos que esta estrategia no fue efectiva; puesto que no redujo el tiempo de otras estrategias previas como la primera o la segunda.

6. **Modificacion de las reglas en archivo /etc/john/john.conf**

Esta última estrategia consiste en modificar el fichero de configuración de John the Ripper. En concreto, el fichero que se muestra en el título. En este fichero, hemos creado una regla llamada “Capitaliza”. Esta regla que hemos definido es como un máscara que reconoce la primera letra del diccionario descargado de internet como una mayúscula y el resto como minúsculas. Esta estrategia ha sido necesaria puesto que este diccionario está formado por los apellidos estadounidenses más famosos del país, y lógicamente, comienzan por mayúscula y el resto en minúsculas.

Es por ello, que queríamos probar esta nueva estrategia donde evitamos que se empleen las reglas del single y se lleve a cabo la nuestra regla especificando que solo debe observar mayúscula y minúsculas en ese orden. El comando en cuestión resultó de la siguiente manera:

```
--rules=Capitaliza --fomat=md5crypt --max-length=9 --min-length=3 -wordlist=top_family_names_usa.txt hash24.txt
```



Esta estrategia la realizamos en los últimos 5 datasets ya que son los únicos que requieren de esta regla para su correcto funcionamiento.

Por último, como se puede observar, el resto de comandos son los mismos que se han explicado previamente. en las estrategias anteriores.

Como hemos mencionado previamente, en esta sección, vamos a explicar aquellas estrategias que contemplamos pero que no pudimos implementar.

Estrategias contempladas

1. **GPU**

   Con el objetivo de mejorar el rendimiento de las ejecuciones que realizamos, intentamos asociar nuestra tarjeta gráfica a John the Ripper. Sin embargo, comprobamos que JTR ya la seleccionaba automáticamente y no dejaba cambiar el parámetro a mano.

2. **Encoding**

Esta estrategia la descartamos debido a que cuando se emplea Md5crypt (formato en el que se encuentran hasheadas las contraseñas), no es necesario llevará a cabo ningún tipo de conversión; puesto que el formato empleado produce un valor y tamaño fijo de hash que es completamente independiente a cualquier tipo de codificación de caracteres. En otras palabras, la codificación de caracteres no influye sobre el resultado de Md5crypt. Es por ello, que no continuamos con esta estrategia.

4.- Resultados y análisis

Para comenzar, hemos creado una tabla general donde hemos especificado la media y la mediana de los mejores resultados que hemos obtenido en cada uno de los datasets. Debemos resaltar que la media muestra el tiempo medio de creación de una contraseña y la mediana el valor central que separa ambos lados de la distribución. Por último, todos los resultados están expresados en minutos siendo por ejemplo 1:30 minutos → 1,5 minutos.

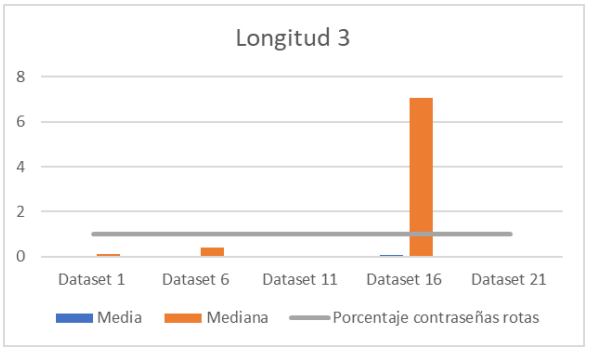
En ella, podemos destacar que existe una gran variedad de estrategias que priman en cada dataset; pero a simple vista podemos destacar que predomina el uso de wordlist y de mask principalmente.

Con respecto al número de contraseñas crackeadas, hemos obtenido un resultado asombroso. Todas las contraseñas se han crackeado.

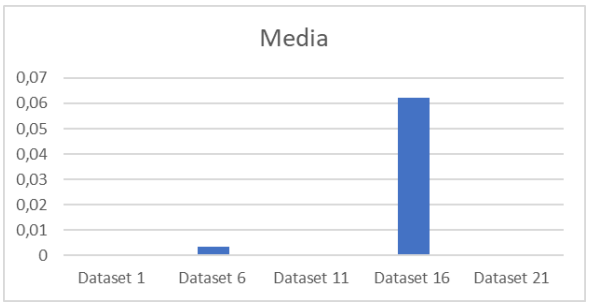
	Media	Mediana	Porcentaje contraseñas rotas	Estrategia
Dataset 1	9,7967E-05	0,1166667	100%	Mask/Wordlist
Dataset 2	0,022	2,475	100%	Mask
Dataset 3	0,7966745	88,525	100%	Mask
Dataset 4	0,0266	3,13	100%	Mask
Dataset 5	0,0231	2,48	100%	Mask
Dataset 6	0,00355833	0,4	100%	Mask/Incremental
Dataset 7	0,05689	6,45000	100%	Worldist
Dataset 8	0,72563	81,98335	100%	Worldist
Dataset 9	0,0283	3,18	100%	Wordlist
Dataset 10	0,023	2,5166	100%	Mask
Dataset 11	0,00007	0,00000	100%	Mask/Wordlist/Incremental + threads
Dataset 12	0,00075	0,08333	100%	Mask/Wordlist/Incremental + threads
Dataset 13	0,00555	0,61667	100%	Incremental
Dataset 14	0,0372	4,133	100%	Mask
Dataset 15	0,489	57,2167	100%	Mask
Dataset 16	0,06226	7,05835	100%	Incremental
Dataset 17	0,05858	7,00835	100%	Wordlist
Dataset 18	0,04572	5,30835	100%	Wordlist
Dataset 19	0,0634	6,8	100%	Wordlist
Dataset 20	0,073	8,325	100%	Wordlist
Dataset 21	0,00027167	0,0333333	100%	Regla personalizada
Dataset 22	0,000485	0,0666667	100%	Wordlist
Dataset 23	0,000435	0,05	100%	Wordlist/Regla personalizada
Dataset 24	0,000095	0,0166667	100%	Wordlist/Regla personalizada
Dataset 25	0,000095	0,0166667	100%	Wordlist/Regla personalizada

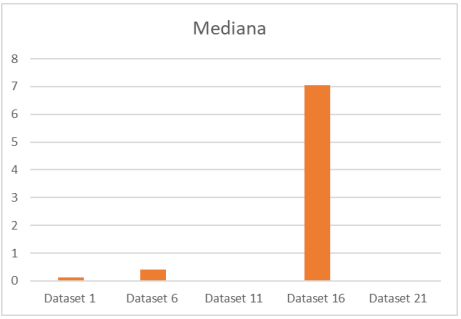
A continuación, vamos a mostrar una serie de gráficas que muestran el rendimiento de cada uno de los datasets en función de su longitud. Es decir todos los datasets de longitud 3 van a ser comparados en términos de dataset alfabeto minúsculas, mayúsculas, numérico, alfanumérico y diccionario externo. Con ello podremos comparar media, mediana y porcentajes en función de la estrategia que ha predominado en cada dataset.

Datasets de Longitud 3:

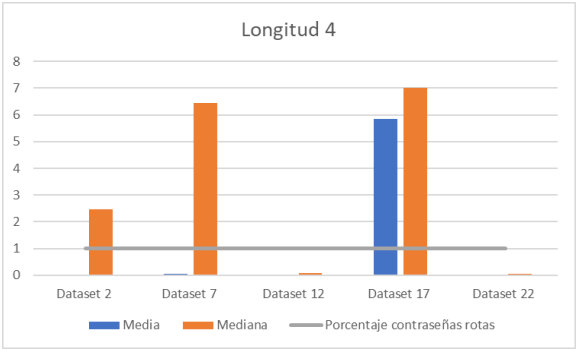


En esta gráfica podemos ver que en términos de media, es bastante mayor que el resto el dataset1 de los alfanuméricos siendo este el menos eficiente en términos de tiempo. Si nos fijamos en los valores de la media en la tabla inicial, tiene sentido este resultado puesto que lo lógico es que aumente la media (sea menos eficiente) conforme pasemos de busqueda de menos valores por parte de John a más combinaciones de valores como es el alfanumérico. Es importante resaltar que el mejor resultado de creación de media es el del dataset1 (minúsculas) empleando tanto mask como wordlist.

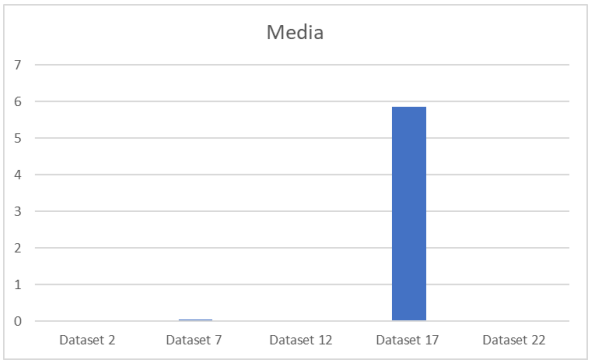


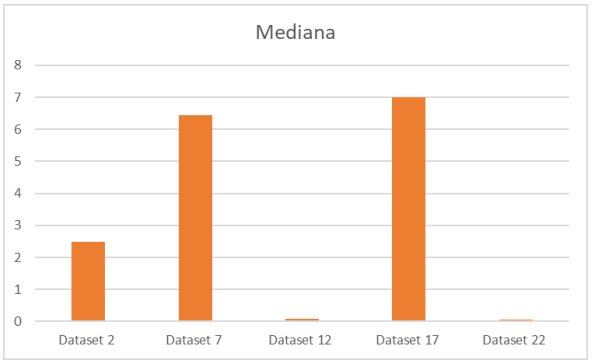


Datasets de longitud 4:

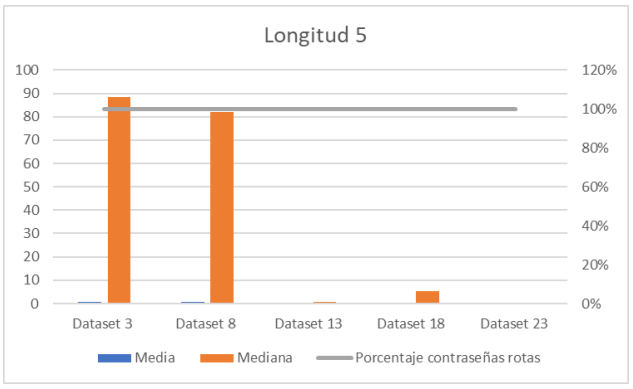


En cuanto a los datasets de longitud 4, podemos destacar que la menor media de crackeo de contraseñas corresponde de nuevo con el dataset 2, posteriormente la del dataset 22,12, 7 y 17 respectivamente. Vemos que de nuevo se cumple el mismo patrón donde se craquean las contraseñas más lentas en los datasets alfanuméricos. Del mismo modo, las contraseñas más rápidas se craquean en el dataset formado por minúsculas. Por tanto, en este caso el dataset más rápido es el referente a minúsculas (dataset2) con estrategia mask y el más lento es el alfanumérico (dataset 17) con estrategia Wordlist.

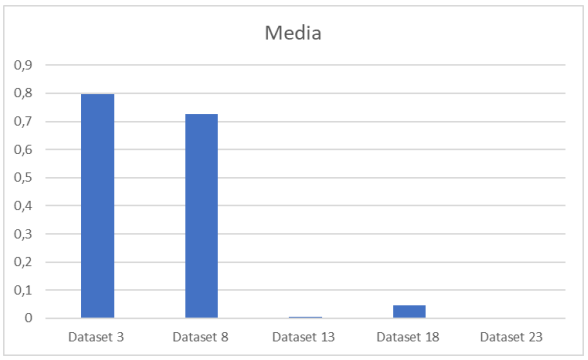


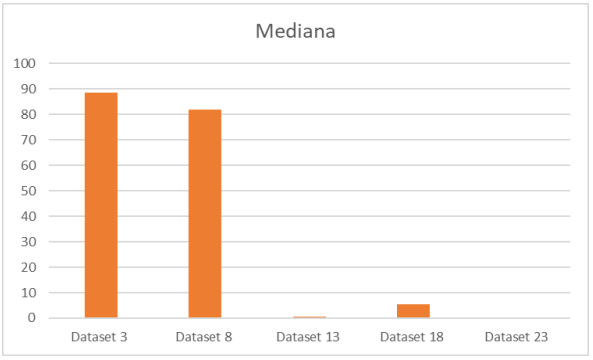


Datasets de longitud 5:

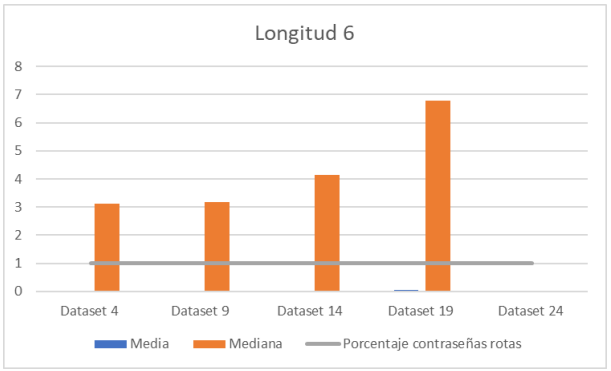


En referencia a los datasets de longitud 5, podemos encontrar una interpretación distinta en lo referente a las medias de crackeo. En este caso el orden de velocidad de crackeo de contraseñas sería el siguiente de menor a mayor: datasets → 23 - 13 - 18 - 8 - 3. La velocidad de crackeo ha sido más rápida en el caso de wordlist/regla personalizada y el más lento el dataset 3 con incremental.

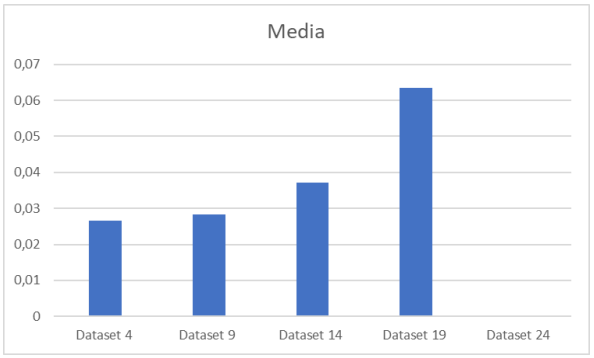


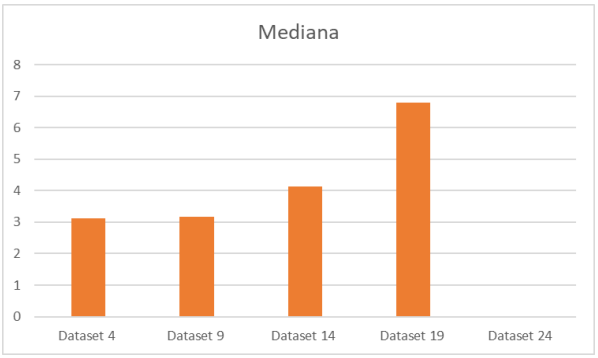


Datasets de longitud 6:

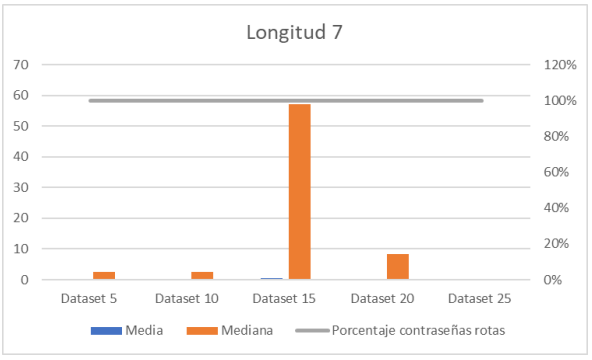


En los datasets de longitud 6 podemos encontrar que la velocidad media de crackeo de contraseñas es mayor para el dataset 19 (alfanuméricos) como es lógico y menor para el dataset24. Como se puede observar, se guardan los patrones que se están dando en las gráficas anteriores. Sin embargo en este cabe resaltar que el menor valor es el del dataset 24. Esta diferencia tiene sentido puesto que solo tiene que llevar a cabo el ataque por diccionario con 1000 valores del diccionario; mientras que el resto de datasets tiene que ejecutar john comparando un número más elevado de valores.

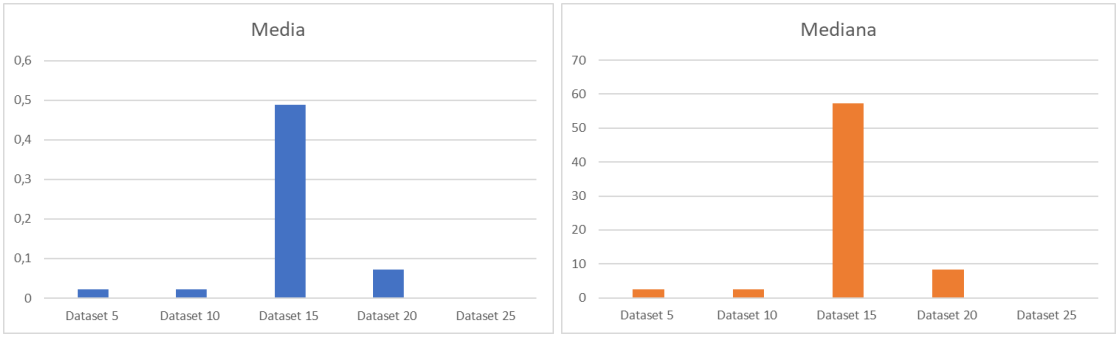




Datasets de longitud 7:



En esta longitud pasa exactamente lo mismo que en el caso anterior; es decir, la media de velocidad de craqueo para el dataset 25 es muy inferior con respecto el resto de datasets porque el diccionario sobre el que actúa es bastante menor en comparación. Por el otro lado, el dataset 15 es el más lento sacando contraseñas ya que tiene tiene que comprobar todas las combinaciones de alfanumericos con la máscara que le proporcionamos en el código.



A modo de resumen, podemos observar que existe una tendencia en cuanto a la velocidad media de craqueo. En datasets con baja longitud, todos los datasets menos el 4 en cada gráfica, presentan velocidades de craqueo bastante rápidas (predominando el primer dataset de cada parte) en comparación al del dataset 4 (en cada gráfica) cuya diferencia es notablemente mayor. Es más lenta esta última. Esta tendencia pasa a las contraseñas de longitud 3, 4 y 5.

Una vez nos hemos metido en contraseñas de longitud 6 y 7, la tendencia cambia debido a que los primeros datasets en cada longitud deben realizar John que son más costosos computacionalmente pues han crecido de forma exponencial. Esto hace que el más lento siga siendo el dataset 4 de ambas longitudes. Sin embargo, en este caso el más rápido es en ambos casos el dataset 24 y 25, ya que , el número de comparaciones que debe hacer John con el ataque con diccionario es significativamente menor.

En cuanto a las medianas, sólo debemos resaltar que hacen justificación a los valores encontrados y representados en la primera tabla.

La tabla de a continuación muestra un resumen de todos los tiempos de cada dataset con todas las estrategias utilizadas y el porcentaje de contraseñas rotas en cada caso. En los casos en los que no hay tiempo es porque no se ha implementado esa estrategia para el dataset.

	Mask + Min-length + Max-length + Format	Wordlist + Min-length + Max-length + Format + Rules	Mask + Min-length + Max-length + Format + Fork	Wordlist + Min-length + Max-length + Format + Rules+Fork
Dataset 1	0:00:09 -100%	0:00:09 -100%	0:00:13 -100%	0:01:45 -100%
Dataset 2	00:03:16 -100%	0:03:55 -100%	0:04:07 -100%	0:05:53 -100%
Dataset 3	01:57:00 -100%	2:46:00 -100%	-	-
Dataset 4	0:03:49 -100%	0:03:52 -100%	0:04:12 -100%	0:06:36 -100%
Dataset 5	0:03:39 -100%	0:03:49 -100%	0:04:17 -100%	0:05:59 -100%
Dataset 6	0:00:33 -100%	0:00:34 -100%	0:00:51 -100%	0:01:05 -100%
Dataset 7	0:11:55 -100%	0:07:24 -100%	0:12:57 -100%	0:14:50 -100%
Dataset 8	1:49:11 -100%	1:49:09 -100%	-	-
Dataset 9	0:04:06 -100%	0:04:03 -100%	-	-
Dataset 10	0:03:25 -100%	0:03:28 -100%	-	-
Dataset 11	0:00:01 -100%	0:00:01 -100%	0:00:01 -100%	0:00:05 -100%
Dataset 12	0:00:07 -100%	0:00:07 -100%	0:00:10 -100%	0:00:42 -100%
Dataset 13	0:01:01 -100%	0:00:55 -100%	0:01:25 -100%	0:03:04 -100%
Dataset 14	0:05:47 -100%	0:07:24 -100%	-	-
Dataset 15	1:11:18 -100%	1:15:50 -100%	-	-
Dataset 16	0:24:23 -100%	0:23:37 -100%	-	-
Dataset 17	0:10:20 -100%	0:08:19 -100%	-	-
Dataset 18	0:09:15 -100%	0:07:13 -100%	-	-
Dataset 19	0:10:05 -100%	0:09:23 -100%	-	-
Dataset 20	0:11:09 -100%	0:09:38 -100%	-	-
Dataset 21	-	0:00:04 -100%	-	-
Dataset 22	-	0:00:04 -100%	-	-
Dataset 23	-	0:00:04 -100%	-	-
Dataset 24	-	0:00:01 -100%	-	-
Dataset 25	-	0:00:01 -100%	-	-

Ordenación de datasets con wordlist	Incremental+Format+Max-lentgth+Min-length	Modificacion de las reglas en archivo /etc/john/john.conf
-	0:00:30 -100%	-
-	0:17:55 -100%	-
-	-	-
-	-	-
-	-	-
-	0:00:33 -100%	-
-	0:12:21 -100%	-
-	-	-
-	-	-
-	-	-
-	0:00:01 -100%	-
-	0:00:07 -100%	-
-	0:00:53 -100%	-
0:07:24 -100%	-	-
1:24:31 -100%	-	-
-	0:09:08 -100%	-
-	-	-
-	-	-
-	-	-
-	-	-
-	-	0:00:03 -100%
-	-	0:00:05 -100%
-	-	0:00:04 -100%
-	-	0:00:01 -100%
-	-	0:00:01 -100%

Para mejor visualización ver la hoja de cálculo:
https://docs.google.com/spreadsheets/d/1QutTG7XG9ShZ3Zm822wPp0nexj10qFSokzlod_I8gIk/edit?usp=sharing

5.- Anexo

Las capturas de pantallas de todas las estrategias implementadas a los 25 datasets se encuentran en el siguiente word:

https://docs.google.com/document/d/1gE2h29yItOo0Wl0w5Aq5dBUu2-R4ESd1QQppNQK18-I/edit?usp=sharing