Práctica 1: Estudio de la Fortaleza de Contraseñas



**INGENIERÍA DE LA CIBERSEGURIDAD**

Octubre 2023

Grado en Ingeniería Informática

Pablo Brasero Martínez. NIA: 100451247

Miguel Castuera García. NIA: 100451285

Índice

[**1. Generación de los datasets 3**](#_9bksiliobtoo)

[Generación de contraseñas sin diccionario 3](#_b2le01wiwg28)

[Proceso de Generación de Contraseñas 3](#_argfceq1722w)

[Generación de Listas de Contraseñas 3](#_1d0e2dmbq79j)

[Generación de contraseñas con diccionario 3](#_pyzgk9ijrg27)

[Carga del Diccionario 3](#_m7uykyp8c7k0)

[Proceso de Generación de Contraseñas 4](#_zg5azgmlyiyy)

[Hashing de Contraseñas 4](#_2c2ilppz4qta)

[Tipos de Contraseñas Generadas 4](#_an1vo9y9mnys)

[Datasets del punto extra 4](#_k3o7yi8tubkb)

[**2. Metodología 4**](#_f0u5xfcx6aia)

[**3. Resultados y análisis 6**](#_imbndpy8qea5)

[Fuerza bruta para contraseñas cortas (estrategia-1) 6](#_yck5i4ff9uh)

[Fuerza bruta para contraseñas de longitud media (estrategia-2) 8](#_f5hgnveo86fw)

[Fuerza bruta para contraseñas largas con caracteres en mayúsculas (estrategia-3) 11](#_fps1l72fh2wg)

[Fuerza bruta para contraseñas largas con caracteres en minúsculas y números (estrategia-4) 13](#_htnvqeg6vgpe)

[Ataque de diccionario con reglas especial (estrategia-5) 15](#_czefjohpjyop)

[**4. Conclusiones 17**](#_o8dzb8ub0zia)

[**5. Códigos 18**](#_ui53n2b7h033)

[generador\_sha256.py 18](#_jy89t3cjtpyr)

[generador\_diccionario\_sha256.py 19](#_egexpas1em6x)

[generador\_md5.py 21](#_yf03jfkpiaiw)

[generador\_diccionario\_md5.py 22](#_jpsuo0ryuplb)

[scriptJohn.sh 23](#_9ikf6inukxxx)

# Generación de los datasets

## Generación de contraseñas sin diccionario

Para la generación de los datasets de contraseñas sin elementos del diccionario, hemos utilizado un script en Python que emplea las bibliotecas "random", "string" y "hashlib". El proceso está dividido en dos funciones principales: "generate\_password" y "generate\_password\_list".

### Proceso de Generación de Contraseñas

La función "generate\_password", que toma dos argumentos: la longitud de la contraseña ("length") y el nivel de complejidad ("complexity"). Dependiendo del nivel de complejidad seleccionado, se elige un conjunto de caracteres específico:

- "lower": Solo letras minúsculas.

- "upper": Solo letras mayúsculas.

- "numeric": Solo números.

- "alphanumeric": Letras, números y símbolos de puntuación.

Una vez seleccionado el conjunto de caracteres, la función genera una contraseña aleatoria de la longitud especificada.

### Generación de Listas de Contraseñas

La función "generate\_password\_list" se encarga de generar un conjunto de contraseñas únicas. Toma tres argumentos: el nivel de complejidad ("complexity"), la longitud de la contraseña ("length") y el número total de contraseñas a generar ("num\_passwords"). Utiliza la función "generate\_password" para crear cada contraseña individual y su correspondiente hash SHA-256. Ambos se almacenan en un conjunto para garantizar la unicidad.

[Código *generador\_sha256.py*](#ogt6godevduc)

## Generación de contraseñas con diccionario

Para la generación de los datasets de contraseñas que incorporan elementos del diccionario, también hemos utilizado un script en Python, haciendo uso nuevamente de las bibliotecas "random", "string" y "hashlib". En este caso, el proceso se compone de tres funciones principales: "load\_dictionary", "generate\_password" y "hash\_password".

### Carga del Diccionario

La primera etapa implica cargar un diccionario de palabras desde un archivo de texto mediante la función "load\_dictionary". Este diccionario se emplea posteriormente para generar contraseñas basadas en palabras reales.

### Proceso de Generación de Contraseñas

La función "generate\_password" toma el diccionario cargado como argumento y genera una contraseña basada en una de ocho posibles combinaciones aleatorias. Estas combinaciones pueden incluir elementos como dígitos, letras y palabras del diccionario, en diferentes órdenes y formatos, incluyendo la inversión de las palabras del diccionario.

### Hashing de Contraseñas

De la misma forma que en el caso anterior, la función "hash\_password" se encarga de crear el hash SHA-256 de la contraseña generada.

### Tipos de Contraseñas Generadas

Encontramos 8 tipos de contraseñas diferentes:

* Una palabra del diccionario seguida de un dígito y una letra.
* Un dígito y una letra seguidos de una palabra del diccionario.
* La inversa de una palabra del diccionario seguida de un dígito y una letra.
* Un dígito y una letra seguidos de la inversa de una palabra del diccionario.
* Una palabra del diccionario seguida de un dígito.
* Un dígito seguido de una palabra del diccionario.
* La inversa de una palabra del diccionario seguida de un dígito.
* Un dígito seguido de la inversa de una palabra del diccionario.

[Código *generador\_diccionario\_sha256.py*](#jpqiz4eylbia)

## Datasets del punto extra

Para los últimos 25 datasets incluidos en el apartado de punto extra del enunciado, hemos optado por emplear el algoritmo de hash md5. Hemos elegido este algoritmo puesto que representa un nivel diferente de complejidad y seguridad y consideramos suficientemente diferente como para suponer un reto mayor habiendo conseguido ya descifrar exitosamente los datasets anteriores.

Para la generación de las contraseñas hemos usado los mismos archivos y métodos que usamos para los primeros 25 datasets, y puesto que la librería "hashlib" incluía el algoritmo seleccionado únicamente hemos tenido que modificar la línea en la que se usaba dicho algoritmo.

[Código *generador\_md5.py*](#afr8y5rwpbsf)

[Código *generador\_diccionario\_md5.py*](#dqh0zmaqc1z9)

# Metodología

Cada una de estas estrategias tiene como objetivo maximizar el número de contraseñas rotas en los datasets utilizando diferentes técnicas y limitaciones.Se han empleado cinco estrategias diferentes con el programa John the Ripper, cada una con un propósito específico y dirigida a un tipo particular de contraseñas. En todas ellas se especifica el tipo de hashing para evitar problemas con john. A continuación, se describen las estrategias:

* Fuerza bruta para contraseñas cortas (estrategia-1):

*john --format=raw-sha256 --incremental --min-length=3 --max-length=5 dataset.txt*

Esta estrategia utiliza un ataque de fuerza bruta para romper contraseñas que tienen una longitud de entre 3 y 5 caracteres. El objetivo es romper todas las contraseñas en los datasets que cumplen con estos criterios de longitud, independientemente del tipo de contraseña. Está limitada por esas longitudes de manera que el tiempo de rotura no sea excesivo y estas no requieren de mucho tiempo.

* Fuerza bruta para contraseñas de longitud media (estrategia-2):

*john --format=raw-sha256 --incremental=ascii --min-length=6 --max-length=6 dataset.txt*

Esta utiliza un ataque de fuerza bruta pero está dirigida únicamente a contraseñas que tienen exactamente 6 caracteres de longitud. Es de longitud única ya que debe romper contraseñas formadas por números, símbolos y letras.

* Fuerza bruta para contraseñas largas con caracteres en mayúsculas (estrategia-3):

john --format=raw-sha256 --incremental=Upper --min-length=7 --max-length=7 dataset.txt

Esta estrategia está diseñada para romper contraseñas que tienen una longitud de 7 caracteres y todos ellos son letras mayúsculas. Las contraseñas de longitud 7 son muy difíciles de romper usando fuerza bruta sin especificar, por ello esta estrategia se centra tan solo en los dataset de letras mayúsculas.

* Fuerza bruta para contraseñas largas con caracteres en minúsculas (estrategia-4):

john --format=raw-sha256 --incremental=Lower --min-length=7 --max-length=7 dataset.txt

Esta estrategia tiene como objetivo romper contraseñas que tienen una longitud de 7 y contienen sólo letras minúsculas. Es útil para romper contraseñas muy complejas evitando exceder el límite de tiempo ya que sólo probará con 27 letras diferentes.

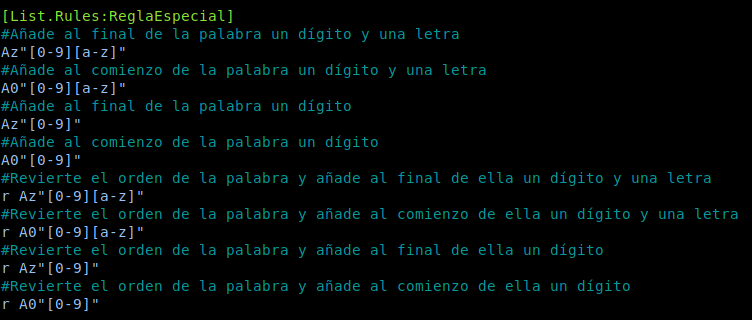
* Ataque de diccionario con reglas especiales (estrategia-5):

john --format=raw-sha256 --wordlist=/usr/share/wordlists/rockyou.txt --rules=ReglaEspecial dataset.txt

Esta estrategia utiliza un ataque de diccionario con la lista de palabras “rockyou.txt” y aplica una regla especial para probar posibles modificaciones de las contraseñas. Este método es eficaz para romper contraseñas que son palabras comunes y en este caso está destinado a romper los datasets creados con palabras de diccionario y que han sido ligeramente modificadas.

Además se va a utilizar para romper las contraseñas que falten de los datasets numéricos ya que entre las palabras comunes del rockyou hay una gran cantidad de números.

La regla utilizada para este ataque es la siguiente:



Las estrategias utilizadas para los datasets cifrados usando md5 son las mismas pero cambiando el formato por –format=RAW-MD5.

Para la obtención de las métricas de cada par estrategia-dataset se ha utilizado por cada estrategia un script que aplica el comando john a cada uno de los datasets y utilizando el john.log devuelve el número de contraseñas rotas y tanto la media como la mediana del tiempo de rotura de las contraseñas.

[Código *scriptJohn.sh*](#5sr691ztsfr)

# Resultados y análisis

A continuación se muestran la media y la mediana del tiempo requerido para romper una contraseña y el porcentaje de contraseñas rotas de cada par de dataset-estrategia. Para obtenerlos se ha usado un timeout de 20 minutos ya que algunos casos podrían tardar demasiado tiempo en romper todas las contraseñas o incluso podrían no romperlas nunca. Los datasets están ordenados por longitudes a excepción de los de diccionario cuya longitud no es una específica.

### Fuerza bruta para contraseñas cortas (estrategia-1)

| Datasets SHA-256 | Contraseñas rotas | Tiempo medio de rotura | Mediana de tiempo de rotura |
| --- | --- | --- | --- |
| Lower-3 | 100% | 0 | 0 |
| Upper-3 | 100% | 15 | 4 |
| Numeric-3 | 100% | 0 | 0 |
| Alphanumeric-3 | 100% | 77 | 132 |
| Dictionary-1 | 0% |  |  |
| Lower-4 | 100% | 2 | 1 |
| Upper-4 | 100% | 20 | 18 |
| Numeric-4 | 100% | 0 | 0 |
| Alphanumeric-4 | 100% | 116 | 60 |
| Dictionary-2 | 0% |  |  |
| Lower-5 | 100% | 2 | 1 |
| Upper-5 | 100% | 15 | 8 |
| Numeric-5 | 100% | 0 | 1 |
| Alphanumeric-5 | 100% | 135 | 145 |
| Dictionary-3 | 1% | 1 | 1 |
| Lower-6 | 0% |  |  |
| Upper-6 | 0% |  |  |
| Numeric-6 | 0% |  |  |
| Alphanumeric-6 | 0% |  |  |
| Dictionary-4 | 0% |  |  |
| Lower-7 | 0% |  |  |
| Upper-7 | 0% |  |  |
| Numeric-7 | 0% |  |  |
| Alphanumeric-7 | 0% |  |  |
| Dictionary-5 | 0% |  |  |

| Datasets MD5 | Contraseñas rotas | Tiempo medio de rotura | Mediana de tiempo de rotura |
| --- | --- | --- | --- |
| Lower-3 | 100% | 0 | 0 |
| Upper-3 | 100% | 6 | 2 |
| Numeric-3 | 100% | 0 | 0 |
| Alphanumeric-3 | 100% | 51 | 82 |
| Dictionary-1 | 0% |  |  |
| Lower-4 | 100% | 1 | 0 |
| Upper-4 | 100% | 10 | 9 |
| Numeric-4 | 100% | 0 | 0 |
| Alphanumeric-4 | 100% | 79 | 134 |
| Dictionary-2 | 0% |  |  |
| Lower-5 | 100% | 0 | 0 |
| Upper-5 | 100% | 10 | 3 |
| Numeric-5 | 100% | 0 | 0 |
| Alphanumeric-5 | 100% | 80 | 79 |
| Dictionary-3 | 0% |  |  |
| Lower-6 | 0% |  |  |
| Upper-6 | 0% |  |  |
| Numeric-6 | 0% |  |  |
| Alphanumeric-6 | 0% |  |  |
| Dictionary-4 | 1% | 0 | 0 |
| Lower-7 | 0% |  |  |
| Upper-7 | 0% |  |  |
| Numeric-7 | 0% |  |  |
| Alphanumeric-7 | 0% |  |  |
| Dictionary-5 | 0% |  |  |

La primera estrategia utilizada, en efecto, es suficiente para romper todas las contraseñas de longitudes entre 3 y 5 de manera muy rápida ya que con bajas longitudes no hace falta realizar tantas pruebas. El tiempo medio de rotura es menor a los dos minutos y en algunos casos como en los datasets numéricos no excede los dos segundos.

A pesar de ser muy útil para los casos mencionados cuando la longitud es mayor es una estrategia totalmente inservible, solo ha descubierto una contraseña de los datasets creados usando diccionario ya que el resto de las contraseñas de estos datasets son de longitud mayor.

### Fuerza bruta para contraseñas de longitud media (estrategia-2)

| Datasets SHA-256 | Contraseñas rotas | Tiempo medio de rotura | Mediana de tiempo de rotura |
| --- | --- | --- | --- |
| Lower-3 | 0% |  |  |
| Upper-3 | 0% |  |  |
| Numeric-3 | 0% |  |  |
| Alphanumeric-3 | 0% |  |  |
| Dictionary-1 | 0% |  |  |
| Lower-4 | 0% |  |  |
| Upper-4 | 0% |  |  |
| Numeric-4 | 0% |  |  |
| Alphanumeric-4 | 0% |  |  |
| Dictionary-2 | 1% | 577 | 577 |
| Lower-5 | 0% |  |  |
| Upper-5 | 0% |  |  |
| Numeric-5 | 0% |  |  |
| Alphanumeric-5 | 0% |  |  |
| Dictionary-3 | 0% |  |  |
| Lower-6 | 100% | 72 | 59 |
| Upper-6 | 100% | 333 | 202 |
| Numeric-6 | 100% | 7 | 9 |
| Alphanumeric-6 | 31% | 599 | 639 |
| Dictionary-4 | 0% |  |  |
| Lower-7 | 0% |  |  |
| Upper-7 | 0% |  |  |
| Numeric-7 | 0% |  |  |
| Alphanumeric-7 | 0% |  |  |
| Dictionary-5 | 2% | 27 | 27 |

| Datasets MD5 | Contraseñas rotas | Tiempo medio de rotura | Mediana de tiempo de rotura |
| --- | --- | --- | --- |
| Lower-3 | 0% |  |  |
| Upper-3 | 0% |  |  |
| Numeric-3 | 0% |  |  |
| Alphanumeric-3 | 0% |  |  |
| Dictionary-1 | 1% | 13 | 13 |
| Lower-4 | 0% |  |  |
| Upper-4 | 0% |  |  |
| Numeric-4 | 0% |  |  |
| Alphanumeric-4 | 0% |  |  |
| Dictionary-2 | 0% |  |  |
| Lower-5 | 0% |  |  |
| Upper-5 | 0% |  |  |
| Numeric-5 | 0% |  |  |
| Alphanumeric-5 | 0% |  |  |
| Dictionary-3 | 1% | 6 | 6 |
| Lower-6 | 100% | 34 | 22 |
| Upper-6 | 100% | 203 | 182 |
| Numeric-6 | 100% | 6 | 7 |
| Alphanumeric-6 | 35% | 518 | 511 |
| Dictionary-4 | 0% |  |  |
| Lower-7 | 0% |  |  |
| Upper-7 | 0% |  |  |
| Numeric-7 | 0% |  |  |
| Alphanumeric-7 | 0% |  |  |
| Dictionary-5 | 2% | 17 | 17 |

Esta estrategia tenía como objetivo romper las contraseñas de longitud 6 y lo ha conseguido con las de los datasets numéricos, los de mayúsculas y los de minúsculas, sin embargo las contraseñas de esa longitud alfanuméricas requieren de más tiempo por ello solo se han roto menos de la mitad de las que posee el dataset alfanumérico y 3 de las que poseen los datasets creados a partir de el diccionario rockyou.txt.

A diferencia de la estrategia anterior que no necesitaba muchas pruebas esta está enfocada en una longitud más alta requiere de más tiempo como se observa comparando el tiempo de rotura en ambas tablas.

### Fuerza bruta para contraseñas largas con caracteres en mayúsculas (estrategia-3)

| Datasets SHA-256 | Contraseñas rotas | Tiempo medio de rotura | Mediana de tiempo de rotura |
| --- | --- | --- | --- |
| Lower-3 | 0% |  |  |
| Upper-3 | 0% |  |  |
| Numeric-3 | 0% |  |  |
| Alphanumeric-3 | 0% |  |  |
| Dictionary-1 | 0% |  |  |
| Lower-4 | 0% |  |  |
| Upper-4 | 0% |  |  |
| Numeric-4 | 0% |  |  |
| Alphanumeric-4 | 0% |  |  |
| Dictionary-2 | 0% |  |  |
| Lower-5 | 0% |  |  |
| Upper-5 | 0% |  |  |
| Numeric-5 | 0% |  |  |
| Alphanumeric-5 | 0% |  |  |
| Dictionary-3 | 0% |  |  |
| Lower-6 | 0% |  |  |
| Upper-6 | 0% |  |  |
| Numeric-6 | 0% |  |  |
| Alphanumeric-6 | 0% |  |  |
| Dictionary-4 | 0% |  |  |
| Lower-7 | 0% |  |  |
| Upper-7 | 100% | 132 | 133 |
| Numeric-7 | 0% |  |  |
| Alphanumeric-7 | 0% |  |  |
| Dictionary-5 | 0% |  |  |

| Datasets MD5 | Contraseñas rotas | Tiempo medio de rotura | Mediana de tiempo de rotura |
| --- | --- | --- | --- |
| Lower-3 | 0% |  |  |
| Upper-3 | 0% |  |  |
| Numeric-3 | 0% |  |  |
| Alphanumeric-3 | 0% |  |  |
| Dictionary-1 | 0% |  |  |
| Lower-4 | 0% |  |  |
| Upper-4 | 0% |  |  |
| Numeric-4 | 0% |  |  |
| Alphanumeric-4 | 0% |  |  |
| Dictionary-2 | 0% |  |  |
| Lower-5 | 0% |  |  |
| Upper-5 | 0% |  |  |
| Numeric-5 | 0% |  |  |
| Alphanumeric-5 | 0% |  |  |
| Dictionary-3 | 0% |  |  |
| Lower-6 | 0% |  |  |
| Upper-6 | 0% |  |  |
| Numeric-6 | 0% |  |  |
| Alphanumeric-6 | 0% |  |  |
| Dictionary-4 | 0% |  |  |
| Lower-7 | 0% |  |  |
| Upper-7 | 100% | 118 | 117 |
| Numeric-7 | 0% |  |  |
| Alphanumeric-7 | 0% |  |  |
| Dictionary-5 | 0% |  |  |

Los datasets restantes requieren de demasiada cantidad de pruebas por ello esta estrategia se enfoca únicamente en las contraseñas de longitud 7 mayúsculas. Se observa en las métricas que ha logrado romper todas las contraseñas de este dataset pero ninguna del resto como se podía prever.

### 

### Fuerza bruta para contraseñas largas con caracteres en minúsculas (estrategia-4)

| Datasets SHA-256 | Contraseñas rotas | Tiempo medio de rotura | Mediana de tiempo de rotura |
| --- | --- | --- | --- |
| Lower-3 | 0% |  |  |
| Upper-3 | 0% |  |  |
| Numeric-3 | 0% |  |  |
| Alphanumeric-3 | 0% |  |  |
| Dictionary-1 | 0% |  |  |
| Lower-4 | 0% |  |  |
| Upper-4 | 0% |  |  |
| Numeric-4 | 0% |  |  |
| Alphanumeric-4 | 0% |  |  |
| Dictionary-2 | 0% |  |  |
| Lower-5 | 0% |  |  |
| Upper-5 | 0% |  |  |
| Numeric-5 | 0% |  |  |
| Alphanumeric-5 | 0% |  |  |
| Dictionary-3 | 0% |  |  |
| Lower-6 | 0% |  |  |
| Upper-6 | 0% |  |  |
| Numeric-6 | 0% |  |  |
| Alphanumeric-6 | 0% |  |  |
| Dictionary-4 | 0% |  |  |
| Lower-7 | 100% | 109 | 112 |
| Upper-7 | 0% |  |  |
| Numeric-7 | 0% |  |  |
| Alphanumeric-7 | 0% |  |  |
| Dictionary-5 | 0% |  |  |

| Datasets MD5 | Contraseñas rotas | Tiempo medio de rotura | Mediana de tiempo de rotura |
| --- | --- | --- | --- |
| Lower-3 | 0% |  |  |
| Upper-3 | 0% |  |  |
| Numeric-3 | 0% |  |  |
| Alphanumeric-3 | 0% |  |  |
| Dictionary-1 | 0% |  |  |
| Lower-4 | 0% |  |  |
| Upper-4 | 0% |  |  |
| Numeric-4 | 0% |  |  |
| Alphanumeric-4 | 0% |  |  |
| Dictionary-2 | 0% |  |  |
| Lower-5 | 0% |  |  |
| Upper-5 | 0% |  |  |
| Numeric-5 | 0% |  |  |
| Alphanumeric-5 | 0% |  |  |
| Dictionary-3 | 0% |  |  |
| Lower-6 | 0% |  |  |
| Upper-6 | 0% |  |  |
| Numeric-6 | 0% |  |  |
| Alphanumeric-6 | 0% |  |  |
| Dictionary-4 | 0% |  |  |
| Lower-7 | 100% | 94 | 101 |
| Upper-7 | 0% |  |  |
| Numeric-7 | 0% |  |  |
| Alphanumeric-7 | 0% |  |  |
| Dictionary-5 | 0% |  |  |

Como se ha mencionado anteriormente las contraseñas restantes tienen una gran complejidad debido a su longitud por ello esta estrategia se encarga de romper las contraseñas minúsculas o formadas por números de la longitud más alta. Es una estrategia ligeramente más ambiciosa que la anterior y por ello los tiempos de media de rotura y mediana son también mayores pero ambas logran cumplir su tarea rompiendo el 100% de las contraseñas de los datasets numéricos y la mayoría de las contraseñas minúsculas. Posiblemente con un timeout mayor rompería el 100% de estas últimas.

Además se observa una pequeña cantidad de contraseñas rotas de los 5 datasets de diccionario, estas son contraseñas que estaban formadas por la combinación de minúsculas y números.

### Ataque de diccionario con reglas especial (estrategia-5)

| Datasets SHA-256 | Contraseñas rotas | Tiempo medio de rotura | Mediana de tiempo de rotura |
| --- | --- | --- | --- |
| Lower-3 | 0% |  |  |
| Upper-3 | 0% |  |  |
| Numeric-3 | 91% | 548 | 479 |
| Alphanumeric-3 | 0% |  |  |
| Dictionary-1 | 82% | 587 | 480 |
| Lower-4 | 0% |  |  |
| Upper-4 | 0% |  |  |
| Numeric-4 | 67% | 553 | 478 |
| Alphanumeric-4 | 0% |  |  |
| Dictionary-2 | 80% | 593 | 476 |
| Lower-5 | 0% |  |  |
| Upper-5 | 0% |  |  |
| Numeric-5 | 99% | 546 | 475 |
| Alphanumeric-5 | 0% |  |  |
| Dictionary-3 | 79% | 613 | 476 |
| Lower-6 | 0% |  |  |
| Upper-6 | 0% |  |  |
| Numeric-6 | 86% | 587 | 480 |
| Alphanumeric-6 | 0% |  |  |
| Dictionary-4 | 79% | 646 | 470 |
| Lower-7 | 0% |  |  |
| Upper-7 | 0% |  |  |
| Numeric-7 | 86% | 617 | 484 |
| Alphanumeric-7 | 0% |  |  |
| Dictionary-5 | 81% | 620 | 476 |

| Datasets MD5 | Contraseñas rotas | Tiempo medio de rotura | Mediana de tiempo de rotura |
| --- | --- | --- | --- |
| Lower-3 | 0% |  |  |
| Upper-3 | 0% |  |  |
| Numeric-3 | 91% | 437 | 351 |
| Alphanumeric-3 | 0% |  |  |
| Dictionary-1 | 81% | 862 | 950 |
| Lower-4 | 0% |  |  |
| Upper-4 | 0% |  |  |
| Numeric-4 | 66% | 433 | 349 |
| Alphanumeric-4 | 0% |  |  |
| Dictionary-2 | 75% | 468 | 382 |
| Lower-5 | 0% |  |  |
| Upper-5 | 0% |  |  |
| Numeric-5 | 97% | 411 | 348 |
| Alphanumeric-5 | 0% |  |  |
| Dictionary-3 | 77% | 506 | 442 |
| Lower-6 | 0% |  |  |
| Upper-6 | 0% |  |  |
| Numeric-6 | 87% | 443 | 358 |
| Alphanumeric-6 | 0% |  |  |
| Dictionary-4 | 73% | 530 | 552 |
| Lower-7 | 0% |  |  |
| Upper-7 | 0% |  |  |
| Numeric-7 | 82% | 426 | 357 |
| Alphanumeric-7 | 0% |  |  |
| Dictionary-5 | 77% | 470 | 369 |

Se puede contemplar que con tan sólo esta estrategia se logra romper casi el 100% de las contraseñas creadas a partir de diccionarios. A pesar de ser de mayor longitud que las anteriores los tiempos son inesperadamente cortos, esto se debe a que los ataques de diccionario son más eficaces, pues no necesitan tantos intentos y combinaciones. Los ataques de fuerza bruta de longitud alta no lograban ni un 50% de las pruebas totales en los 20 minutos sin embargo los de diccionario han terminado en torno a los 16 minutos.

Es útil también para romper la mayoría de contraseñas de los datasets numéricos especialmente los de longitud 7 que todavía no se habían roto, esto se debe a que entre las palabras de rockyou.txt hay una gran cantidad de números.

# Conclusiones

Estas 5 estrategias maximizan el número de contraseñas rotas de esos 50 datasets teniendo en cuenta que el tiempo es limitado y que por ello algunos datasets no podrían ser rotos de ninguna manera como es el caso de los alfanuméricos de longitud 7.

Se observa en los resultados que al utilizar john en un ataque de fuerza bruta comienza probando con contraseñas más comunes ya que las primeras en romperse han sido las numéricas, seguidas de las que están totalmente en minúsculas y seguidas de las que están en mayúsculas; en cambio tarda más tiempo en romper las alfanuméricas. Por ello los datasets con menor cantidad de contraseñas rotas son los alfanuméricos de longitud 6 y 7.

Los resultados obtenidos en las tablas muestran que el tiempo necesario para romper las contraseñas de los datasets cifrados con SHA-256 es mayor que los cifrados con MD5. Esto se debe a que MD5 produce un valor hash de 128 bits, mientras que SHA-256 produce un hash de 256 bits que hace que se requiera de tiempos más largos para romperlo.

# Códigos

## generador\_sha256.py

import random

import string

import hashlib

def generate\_password(length, complexity):

if complexity == 'lower':

characters = string.ascii\_lowercase

elif complexity == 'upper':

characters = string.ascii\_uppercase

elif complexity == 'numeric':

characters = string.digits

elif complexity == 'alphanumeric':

characters = string.ascii\_letters + string.digits + string.punctuation

else:

pass

return ''.join(random.choice(characters) for \_ in range(length))

def generate\_password\_list(complexity, length, num\_passwords):

password\_list = set()

while len(password\_list) < num\_passwords:

password = generate\_password(length, complexity)

sha256\_password = hashlib.sha256(password.encode()).hexdigest()

password\_list.add((password, sha256\_password)) # Almacenar la contraseña en claro y su hash

return list(password\_list)

# Generacion de las contraseñas

num\_passwords\_per\_length = 100

complexities = ['lower', 'upper', 'numeric', 'alphanumeric']

password\_lengths = [3, 4, 5, 6, 7]

for complexity in complexities:

for length in password\_lengths:

clear\_file\_name = f'{complexity}\_len{length}\_clear\_SHA256.txt'

hash\_file\_name = f'{complexity}\_len{length}\_hashes\_SHA256.txt'

# Generamos archivo con los hashes y con las contraseñas en claro para poder comparar

with open(clear\_file\_name, 'w') as clear\_file, open(hash\_file\_name, 'w') as hash\_file:

password\_list = generate\_password\_list(complexity, length, num\_passwords\_per\_length)

for password, password\_hash in password\_list:

clear\_file.write(password + '\n')

hash\_file.write(password\_hash + '\n')

[Volver arriba](#icu4g9ug043w)

## generador\_diccionario\_sha256.py

import random

import string

import hashlib

def load\_dictionary(filename):

with open(filename, 'r', encoding='iso-8859-1') as file:

dictionary = [line.strip() for line in file]

return dictionary

def generate\_password(dictionary):

choice = random.choice([1, 2, 3, 4, 5, 6, 7, 8])

if choice == 1:

password = random.choice(dictionary) + random.choice(string.digits) + random.choice(string.ascii\_letters)

elif choice == 2:

password = random.choice(string.digits) + random.choice(string.ascii\_letters) + random.choice(dictionary)

elif choice == 3:

password = ''.join(reversed(random.choice(dictionary))) + random.choice(string.digits) + random.choice(string.ascii\_letters)

elif choice == 4:

password = random.choice(string.digits) + random.choice(string.ascii\_letters) + ''.join(reversed(random.choice(dictionary)))

elif choice == 5:

password = random.choice(dictionary) + random.choice(string.digits)

elif choice == 6:

password = random.choice(string.digits) + random.choice(dictionary)

elif choice == 7:

password = ''.join(reversed(random.choice(dictionary))) + random.choice(string.digits)

elif choice == 8:

password = random.choice(string.digits) + ''.join(reversed(random.choice(dictionary)))

password = password.strip()

return password

def hash\_password(password):

sha256 = hashlib.sha256(password.encode()).hexdigest()

return sha256

# Generación de las contraseñas

dictionary\_file = 'rockyou.txt' # Reemplazar con la ruta al archivo de diccionario

dictionary = load\_dictionary(dictionary\_file)

num\_datasets = 5

num\_passwords\_per\_dataset = 100

for dataset\_num in range(1, num\_datasets + 1):

clear\_passwords = []

hashed\_passwords = []

for \_ in range(num\_passwords\_per\_dataset):

password = generate\_password(dictionary)

clear\_passwords.append(password)

hashed\_password = hash\_password(password)

hashed\_passwords.append(hashed\_password)

# Crear un archivo para almacenar contraseñas en claro

with open(f'clear\_passwords\_dataset\_{dataset\_num}\_SHA256.txt', 'w') as clear\_file:

clear\_file.write('\n'.join(clear\_passwords))

# Crear un archivo para almacenar contraseñas hasheadas

with open(f'hashed\_passwords\_dataset\_{dataset\_num}\_SHA256.txt', 'w') as hashed\_file:

hashed\_file.write('\n'.join(hashed\_passwords))

[Volver arriba](#axbem2n0q9m0)

## generador\_md5.py

import random

import string

import hashlib

def generate\_password(length, complexity):

if complexity == 'lower':

characters = string.ascii\_lowercase

elif complexity == 'upper':

characters = string.ascii\_uppercase

elif complexity == 'numeric':

characters = string.digits

elif complexity == 'alphanumeric':

characters = string.ascii\_letters + string.digits + string.punctuation

else:

pass

return ''.join(random.choice(characters) for \_ in range(length))

def generate\_password\_list(complexity, length, num\_passwords):

password\_list = set()

while len(password\_list) < num\_passwords:

password = generate\_password(length, complexity)

md5\_password = hashlib.md5(password.encode()).hexdigest()

password\_list.add((password, md5\_password))

return list(password\_list)

# Generacion de las contraseñas

num\_passwords\_per\_length = 100

complexities = ['lower', 'upper', 'numeric', 'alphanumeric']

password\_lengths = [3, 4, 5, 6, 7]

for complexity in complexities:

for length in password\_lengths:

clear\_file\_name = f'{complexity}\_len{length}\_clear\_MD5.txt'

hash\_file\_name = f'{complexity}\_len{length}\_hashes\_MD5.txt'

# Generamos archivo con los hashes y con las contraseñas en claro para poder comparar

with open(clear\_file\_name, 'w') as clear\_file, open(hash\_file\_name, 'w') as hash\_file:

password\_list = generate\_password\_list(complexity, length, num\_passwords\_per\_length)

for password, password\_hash in password\_list:

clear\_file.write(password + '\n')

hash\_file.write(password\_hash + '\n')

[Volver arriba](#qtthz189z8vu)

## generador\_diccionario\_md5.py

import random

import string

import hashlib

def load\_dictionary(filename):

with open(filename, 'r', encoding='iso-8859-1') as file:

dictionary = [line.strip() for line in file]

return dictionary

def generate\_password(dictionary):

choice = random.choice([1, 2, 3, 4, 5, 6, 7, 8])

if choice == 1:

password = random.choice(dictionary) + random.choice(string.digits) + random.choice(string.ascii\_letters)

elif choice == 2:

password = random.choice(string.digits) + random.choice(string.ascii\_letters) + random.choice(dictionary)

elif choice == 3:

password = ''.join(reversed(random.choice(dictionary))) + random.choice(string.digits) + random.choice(string.ascii\_letters)

elif choice == 4:

password = random.choice(string.digits) + random.choice(string.ascii\_letters) + ''.join(reversed(random.choice(dictionary)))

elif choice == 5:

password = random.choice(dictionary) + random.choice(string.digits)

elif choice == 6:

password = random.choice(string.digits) + random.choice(dictionary)

elif choice == 7:

password = ''.join(reversed(random.choice(dictionary))) + random.choice(string.digits)

elif choice == 8:

password = random.choice(string.digits) + ''.join(reversed(random.choice(dictionary)))

password = password.strip()

return password

def hash\_password(password):

md5\_password = hashlib.md5(password.encode()).hexdigest()

return md5\_password

# Generación de las contraseñas

dictionary\_file = 'rockyou.txt' # Reemplazar con la ruta al archivo de diccionario

dictionary = load\_dictionary(dictionary\_file)

num\_datasets = 5

num\_passwords\_per\_dataset = 100

for dataset\_num in range(1, num\_datasets + 1):

clear\_passwords = []

hashed\_passwords = []

for \_ in range(num\_passwords\_per\_dataset):

password = generate\_password(dictionary)

clear\_passwords.append(password)

hashed\_password = hash\_password(password)

hashed\_passwords.append(hashed\_password)

# Crear un archivo para almacenar contraseñas en claro

with open(f'clear\_passwords\_dataset\_{dataset\_num}\_MD5.txt', 'w') as clear\_file:

clear\_file.write('\n'.join(clear\_passwords))

# Crear un archivo para almacenar contraseñas hasheadas

with open(f'hashed\_passwords\_dataset\_{dataset\_num}\_MD5.txt', 'w') as hashed\_file:

hashed\_file.write('\n'.join(hashed\_passwords))

[Volver arriba](#7gw6lom54095)

## scriptJohn.sh

#!/bin/bash

echo "ESTRATEGIA 1-----------------------------------------------------------------------------------"

# Directorio donde están los archivos

directorio="/home/miguel/nousar"

rm /home/miguel/registro\_john.txt

# Archivo de registro

registro="/home/miguel/registro\_john.txt"

# Recorre cada archivo en el directorio

for archivo in "$directorio"/\*

do

rm ~/.john/john.pot

rm ~/.john/john.log

rm ~/.john/john.rec

# Ejecuta el comando john

timeout 20m john --format=raw-sha256 --incremental=ascii --min-length=3 --max-length=5 "$archivo"

touch times.txt

cat ~/.john/john.log | grep Cracked | awk '{print $1}' > times.txt

tiempos\_en\_segundos=()

suma=0

while IFS= read -r linea

do

IFS=':' read -ra tiempo <<< "$linea"

segundos=$((10#${tiempo[1]}\*3600 + 10#${tiempo[2]}\*60 + 10#${tiempo[3]}))

tiempos\_en\_segundos+=("$segundos")

suma=$(($suma + $segundos))

done < times.txt

# Calcular la media

if (( $suma == 0 )); then

media=0

else

media=$(($suma / ${#tiempos\_en\_segundos[@]}))

fi

# Calcular la mediana

sorted\_tiempos=( $( printf "%d\n" "${tiempos\_en\_segundos[@]}" | sort -n ) )

longitud=${#sorted\_tiempos[@]}

if (( $longitud % 2 == 0 )); then

mediana=$(( (sorted\_tiempos[$((longitud/2))] + sorted\_tiempos[$((longitud/2-1))]) / 2 ))

else

mediana=${sorted\_tiempos[$((longitud/2))]}

fi

# Imprimir el nombre del archivo y la media

echo "$archivo"

echo "Media: $media"

# Convertir la mediana a formato de tiempo y mostrarla

printf 'Mediana: %d\n' $(($mediana))

done

[Volver arriba](#9ivra3aby32d)