Universidad Carlos III de Madrid
Curso Ingeniería de la Ciberseguridad

Práctica Obligatoria 1

FECHA: 27/10/2024
ENTREGA: ÚNICA
GRUPO: 22

Nombre: Aitana Antonia Ortiz Guiño
NIA: 100472097

Nombre: Alba Vidales Casado
NIA: 100472236

ÍNDICE

Pistas para la elección de comandos....2
Metodología de la elección de comandos....2
Metodología Comando 1....2
Metodología Comando 2....3
Metodología Comando 3....3
Metodología Comando 4....4
Metodología Comando 5....4
Resultados obtenidos....5
Resultados obtenidos para foromotos....5
Resultados obtenidos para meneate....5
Resultados obtenidos finales....6
Análisis de resultados....6
Caso foromotos.txt....6
Caso meneate.txt....9
Explicación del script creado....9
Consideraciones....11
Anexo....12

Pistas para la elección de comandos

- [x] ~~1. En los primeros años de ForoMotos.com, la web generaba por defecto contraseñas alfanuméricas en minúsculas de 5 caracteres (después se subió a 6).~~
- [x] ~~2. Más adelante, se autogeneraban incluyendo mayúsculas y obligando a tener al menos un símbolo especial.~~
- [x] ~~3. La longitud mínima para las contraseñas de ForoMotos.com es 4 caracteres.~~
- [x] ~~4. Ambas webs se nutren de usuarios que hablan español, lo cual seguro que se ve reflejado en las contraseñas que usan.~~
- [x] ~~5. ForoMotos.com tiene un subforo dedicado a hacking donde los script kiddies usan la jerga propia de sus élite hacking skills.~~
- [x] ~~6. Algunos usuarios de la web suelen reutilizar la misma contraseña en varios sitios.~~

Metodología de la elección de comandos

Metodología Comando 1

Aplicación de la pista número 4, dado que se nos indica que los usuario hablan español consideramos que era buena idea usar el wordlist de ‘spanish.txt’ que se nos proporciona en el repositorio [1].

1. Primera Línea **(john --format=RAW-MD5 --wordlist=/wordlists/languages/spanish.txt --rules g22_foromotos.txt):** Usa el formato MD5 y un diccionario en español para generar y probar variaciones de cada palabra del diccionario contra los hashes de `g22_foromotos.txt`. Las reglas (--rules) añaden variaciones comunes (mayúsculas, números, símbolos).

2. Segunda Línea (**john --show ... | grep ':' >> "$temp_file"**): Extrae y guarda solo los hashes exitosamente descifrados en formato usuario:contraseña en un archivo temporal ($temp_file).

Esta secuencia optimiza el descifrado enfocándose en palabras comunes en español y variaciones típicas, maximizando las coincidencias probables.

```
Unset
john --format=RAW-MD5 --wordlist=/wordlists/languages/spanish.txt --rules
g22_foromotos.txt
john --show --format=RAW-MD5 g22_foromotos.txt | grep ':' >> "$temp_file"
```

Metodología Comando 2

Aplicación de la pista número 3, dado que se nos indica que la longitud mínima de las contraseñas es de 4 caracteres, implementamos un comando por fuerza bruta que vaya crackeando contraseñas de 4 a 5 caracteres.

1. Primera Línea (**john --format=RAW-MD5 --incremental --fork=3 g22_foromotos.txt --min-length=4 --max-length=5**): Usa el formato MD5 en modo incremental, generando combinaciones de caracteres de 4 a 5 caracteres. --fork=3 ejecuta 3 procesos en paralelo, distribuyendo la carga para una mayor eficiencia.

2. Segunda Línea **(john --show ... | grep ':' >> "$temp_file"**): Extrae los hashes descifrados en formato usuario:contraseña y los guarda en $temp_file.

Este método es útil para probar combinaciones de caracteres cortos que podrían no estar en diccionarios, aumentando las posibilidades de descifrar contraseñas simples y breves.

```
Unset
john --format=RAW-MD5 --incremental --fork=3 g22_foromotos.txt
--min-length=4 --max-length=5
john --show --format=RAW-MD5 g22_foromotos.txt | grep ':' >> "$temp_file"
```

Metodología Comando 3

Aplicación de la pista número 2, dado que se nos indica las contraseñas se empezaron a autogenerar con al menos un símbolo especial decidimos aplicar la regla ‘appendnumbers_and_special_simple’.

1. Primera Línea (**john --format=RAW-MD5 --wordlist=/wordlists/languages/spanish.txt --rules=appendnumbers_and_specials_simple --max-run-time=100 g22_foromotos.txt**): Usa el formato MD5 y un diccionario en español, aplicando reglas que añaden números y caracteres especiales simples al final de cada palabra. --max-run-time=100 limita la ejecución a 100 segundos, ideal para obtener resultados rápidos sin realizar un ataque prolongado.

2. Segunda Línea (**john --show ... | grep ':' >> "$temp_file"**): Extrae los hashes descifrados en formato usuario:contraseña y los guarda en $temp_file.

Este enfoque maximiza las probabilidades de descifrar contraseñas que contienen una palabra común seguida de números o símbolos, dentro de un tiempo controlado.

```
Unset
john --format=RAW-MD5 --wordlist=/wordlists/languages/spanish.txt
--rules=appendnumbers_and_specials_simple --max-run-time=100
g22_foromotos.txt
john --show --format=RAW-MD5 g22_foromotos.txt | grep ':' >> "$temp_file"
```

Metodología Comando 4

Aplicación de la pista número 5, dado que se nos indica que hay usuarios que usan jerga propia, vimos que la regla a aplicar sería l33t y además vimos adecuado añadir posibles variaciones como la de añadir un símbolo especial en cualquier posición.

1. Primera Línea (**john --format=RAW-MD5 --wordlist=/wordlists/languages/spanish.txt --rules=MyRule g22_foromotos.txt**): Usa el formato MD5, el diccionario en español, y aplica MyRule, que contiene estas subreglas:
    a. **l33t**: Realiza sustituciones de letras con caracteres comunes en "leet speak" (cómo cambiar "a" por "4", "e" por "3", etc.).
    b. **append1_addspecialeverywhere**: Agrega un número y un carácter especial en distintas posiciones de la palabra.
    c. **appendjustspecials**: Añade solo caracteres especiales al final de las palabras.

2. Segunda Línea (**john --show ... | grep ':' >> "$temp_file"**): Extrae los hashes descifrados en formato usuario:contraseña y los guarda en $temp_file.

Este método amplía las variaciones posibles al probar diferentes posiciones y combinaciones de caracteres especiales y números, especialmente útiles para contraseñas que mezclan palabras en español con modificaciones comunes.

```
Unset
john --format=RAW-MD5 --wordlist=/wordlists/languages/spanish.txt
--rules=MyRule g22_foromotos.txt
john --show --format=RAW-MD5 g22_foromotos.txt | grep ':' >> "$temp_file"
```

Metodología Comando 5

Aplicación de la pista número 5, dado que se nos indica que las contraseñas subieron a 6 caracteres, implementamos un comando por fuerza bruta que vaya crackease contraseñas de 6 caracteres.

1. Primera Línea (**john --format=RAW-MD5 --incremental --fork=4 --min-length=6 --max-length=6 --max-run-time=1320 g22_foromotos.txt**): Usa el formato MD5 y realiza un ataque incremental, generando todas las combinaciones posibles de contraseñas de exactamente 6 caracteres. Un --fork=4 que ejecuta 4 procesos en paralelo para mejorar la velocidad. Y un--max-run-time=1320 que limita el tiempo de ejecución a 22 minutos (1320 segundos) para evitar un ataque prolongado.

2. Segunda Línea (**john --show ... | grep ':' >> "$temp_file"**): Extrae los hashes descifrados en formato usuario:contraseña y los guarda en $temp_file.

Este método es exhaustivo para contraseñas de 6 caracteres, probando todas las posibles combinaciones en un tiempo determinado, ideal para descifrar contraseñas cortas y complejas.

```
Unset
john --format=RAW-MD5 --incremental --fork=4 --min-length=6 --max-length=6
--max-run-time=1320 g22_foromotos.txt
john --show --format=RAW-MD5 g22_foromotos.txt | grep ':' >> "$temp_file"
```

Resultados obtenidos

Resultados obtenidos para foromotos

outputs > tiempos.txt

```
1  Tiempo wordlist --rules: 00:38
2  Tiempo incremental 4-5: 00:54
3  Tiempo wordlist con appendnumbers_and_specials_simple: 01:41
4  Tiempo wordlist con MyRule: 00:27
5  Tiempo incremental 6: 22:02
6  Tiempo total: 25:42
7  Total contraseñas crackeadas: 81668
```

Resultados obtenidos para meneate

```
Cargando credenciales...
Verificando contraseñas...
Total de contraseñas encontradas: 15101.
Resultados guardados en outputs/passwords_cracked_meneate.txt
Tiempo de ejecución: 132.34 segundos.
PS C:\Users\aitan\practica_ciber2\jtr_cracking_passwords>
```

Resultados obtenidos finales

outputs > results_total.txt

```
1  Contraseñas total crackeadas entre ambos Data Sets: 96769 contraseñas.
2  Tiempo total de la ejecución: 28:03 // 1682 segundos.
```

Análisis de resultados

[Repositorio al Jupyter Notebook]

En este cuaderno se realiza un análisis detallado de las contraseñas rotas del Dataset de *g22_foromotos.txt*, cubriendo los siguientes puntos:

- Identificación de contraseñas repetidas por pareja {Usuario, Contraseña}.
- Distribución de longitudes de contraseñas.
- Análisis de caracteres especiales en las contraseñas.
- Identificación de contraseñas repetidas {Contraseña}.
- Tiempo de ejecución para cada método de crackeo.

Y en la segunda parte realizamos una breve explicación del código implementado para el crackeo de las contraseñas y un análisis de las contraseñas rotas del Dataset de *g22_meneate.txt.*

Caso foromotos.txt

Identificación de contraseñas repetidas por parejas {Usuario:Contraseña}

En esta sección se realiza una comprobación de si existen contraseñas repetidas en un formato {Usuario:Contraseña}.

	Usuario	Contraseña
0	fastidioEiquematico7	acordonar
1	melodiaPerspicuo	cañón7&
2	fosaGoloso647	Sutura
3	vacaPlenipotenciario214	Epigrafista
4	fatidicoAstuto36393	4n4$4rc4
...	...	...
81664	asuntoFlacido70	rcnal
81665	tarjetaInfausto3047	trígono9#
81666	posteEmbriagador271	ambas!@
81667	masajePadecido143639	stvua
81668	velocidadFisionico3297	Magisterio

```python
# Agrupar por usuario y contraseña, contando las ocurrencias
user_password_counts = passwords_df.groupby(['user', 'password']).size().reset_index(name='count')

# Filtrar para encontrar usuarios con la misma contraseña más de una vez
repeated_user_passwords = user_password_counts[user_password_counts['count'] > 1]

# Imprimir el total de repeticiones de usuario-contraseña
total_repeated_user_passwords = repeated_user_passwords.shape[0]
print(f'Total de pares usuario-contraseña repetidos: {total_repeated_user_passwords}')
```

```
Total de pares usuario-contraseña repetidos: 0
```

Podemos observar que no es el caso, tras cargar el txt con las contraseñas crackeadas y buscar coincidencias {Usuario:Contraseña} observamos que no se dan.

Distribución de longitudes de contraseñas

Esta sección analiza la distribución de las longitudes de las contraseñas, centrándose en contraseñas con diferentes rangos (4-5 caracteres, 6 caracteres, etc.).

En esta gráfica se presenta la relación entre la frecuencia y la longitud de las contraseñas en el conjunto de datos. Se observa que las contraseñas con una longitud de 6 caracteres son las más comunes, alcanzando una frecuencia de hasta 40,000 unidades. A continuación, las contraseñas de 5 caracteres tienen una frecuencia cercana a 10,000 unidades. Por otro lado, las contraseñas de 6 caracteres en adelante muestran frecuencias más bajas, llegando hasta las contraseñas de longitud 17-18 como se puede apreciar en la gráfica.

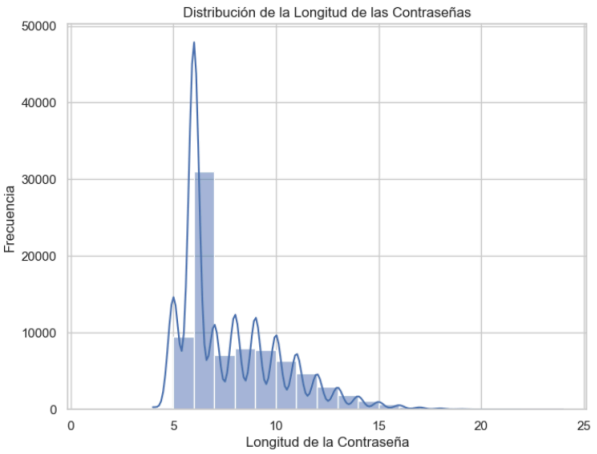

Imagen 1. Distribución de la Longitud de las Contraseñas

Análisis de caracteres especiales en las contraseñas

Determinar el número de contraseñas que contienen caracteres especiales y visualizar su distribución.

La gráfica presentada entonces muestra la distribución de contraseñas según la presencia o ausencia de caracteres especiales. Se ha realizado un análisis donde las contraseñas se dividen en dos categorías: aquellas que contienen caracteres especiales (por ejemplo, símbolos como "!", "@", "#", etc.) y aquellas que no los contienen, es decir, que solo están formadas por letras y números.

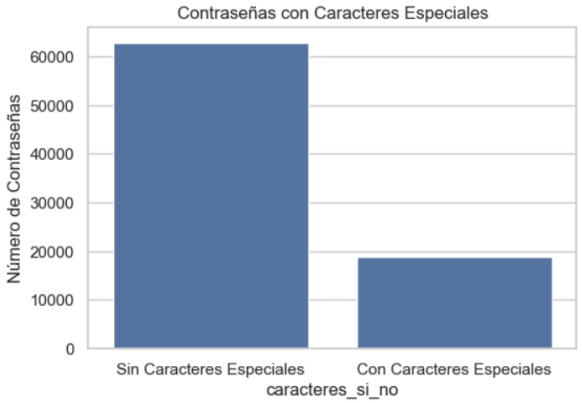

Imagen 2. Comparación de contraseñas: Caracteres Especiales vs. Sin Caracteres Especiales

El resultado es claro: una gran mayoría de las contraseñas (más de 60 mil) no contienen caracteres especiales, mientras que una menor cantidad (aproximadamente 20 mil) sí los incluye. Esto sugiere que, en el conjunto de datos analizado, las contraseñas que carecen de caracteres especiales son mucho más comunes, lo que puede implicar una menor complejidad y, potencialmente, menor seguridad en términos generales.

Identificación de contraseñas repetidas

Encuentra las contraseñas que aparecen más de una vez y visualiza las más repetidas.

En las imágenes se puede observar un análisis de contraseñas repetidas dentro de un conjunto de datos. El código identifica las contraseñas que aparecen más de una vez y presenta las más repetidas por longitud de contraseña.

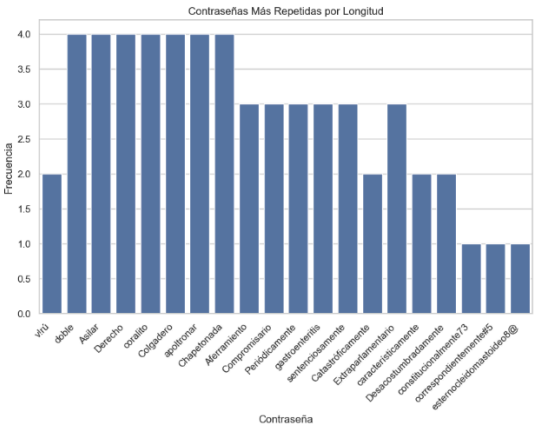

Imagen 3. Contraseñas más repetidas por longitud

Aquí por tanto podemos observar que las contraseñas más largas tienen muy pocas repeticiones, mientras a medida que reducimos el tamaño de ellas observamos una mayor repetición.

Sin embargo, este punto de análisis no es del todo significativo dado que el que se repita una contraseña para diferentes usuarios puede tratarse de una mera casualidad.

Tiempo de ejecución para cada método de crackeo

Visualiza el tiempo de ejecución para cada método de cracking utilizado en el proceso.

La imagen superior muestra un análisis de los tiempos de ejecución de diferentes métodos de cracking que hemos utilizado. En la gráfica, se comparan cuatro enfoques:

1. Wordlists en español con las Rules generales : Este método tarda aproximadamente 38 segundos.
2. Incremental (4-5 caracteres): Toma alrededor de 54 segundos.
3. Anexar números/caracteres especiales: Este método es el más rápido, con un tiempo de 101 segundos.
4. Wordlist en español con MyRules: Este método tarda aproximadamente 27 segundos.
5. Incremental (6 caracteres): Es el más lento, con un tiempo considerable de 1322 segundos.

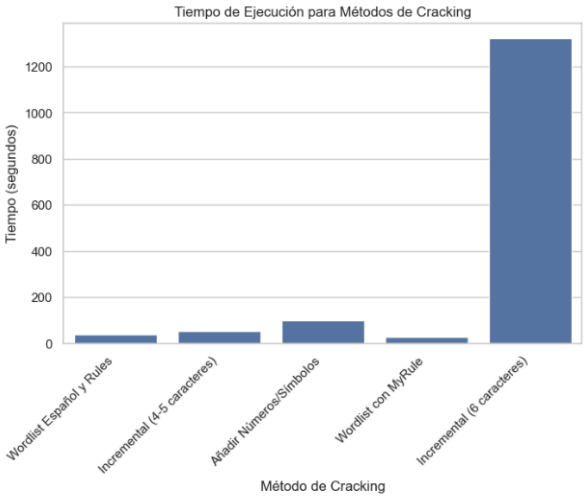

Imagen 4. Tiempo de ejecución de los Métodos de Cracking

A partir de estos datos, podemos concluir que los métodos que utilizan combinaciones más largas o complejas de caracteres (como el método incremental para contraseñas de 6

caracteres) requieren mucho más tiempo para ejecutarse. En cambio, los métodos que se basan en wordlist o en combinaciones más cortas resultan más rápidos.
Este análisis nos deja ver que, aunque los ataques más simples como el de diccionario son relativamente rápidos, las contraseñas más largas y complejas aumentan significativamente el tiempo de descifrado, lo cual resalta la importancia de usar contraseñas más largas y variadas para mejorar la seguridad.

Caso meneate.txt

Repositorio al Jupyter Notebook

En este cuaderno se realiza un análisis detallado de las contraseñas rotas del Dataset de *g22_meneate.txt*, cubriendo los siguientes puntos:

Explicación del script creado

La idea para trabajar con las contraseñas de meneate es la siguiente:

Siguiendo la pista número 6 donde se nos dice que hay usuario que repiten contraseña en diferentes sitios web, decidimos implementar un código que lo que hace es comparar contraseñas crackeadas del anterior DataSet con los hashes bcrypt del nuevo DataSet para verificar si coinciden. Utiliza archivos de entrada que contienen usuarios y contraseñas crackeadas (g22_foromotos.txt) y usuarios con sus hashes bcrypt (g22_meneate.txt).

Funciones principales:

1. **Cargar credenciales**: Lee los archivos de texto y guarda los datos en diccionarios para un acceso rápido.
2. **Verificar contraseña**: Compara una contraseña crackeada con un hash bcrypt, utilizando el algoritmo bcrypt.checkpw().
3. **Procesar en paralelo**: Usa ThreadPoolExecutor para ejecutar hasta 10 hilos en paralelo y acelerar la verificación.
4. **Salida de resultados**: Almacena las coincidencias de usuario y contraseña correctas en un archivo de salida g22_meneate.txt.

El **flujo del programa** es:

1. Carga las contraseñas y los hashes bcrypt.
2. Verifica si la contraseña crackeada coincide con el hash.
3. Ejecuta las comparaciones en paralelo.
4. Guarda los resultados encontrados en un archivo.

Identificación de contraseñas repetidas por parejas {Usuario:Contraseña}

En esta sección se realiza una comprobación de si existen contraseñas repetidas en un formato {Usuario:Contraseña}.

	user	password
0	fundacionAtorrante921127	chuchuy
1	popAfanoso67	Aquijes
2	vacaPlenipotenciario214	Epigrafista
3	concretoCapitulante43	6u56r
4	ascensorIndefinible612050	gulam
...	...	...
15096	genioDespuntado	uxdbpk
15097	humaredaPigre9	mazatepec30
15098	tarjetaInfausto3047	trígono9#
15099	posteEmbriagador271	ambas!@
15100	velocidadFisionico3297	Magisterio

```
# Agrupar por usuario y contraseña, contando las ocurrencias
user_password_counts = passwords_df.groupby(['user', 'password']).size().reset_index(name='count')

# Filtrar para encontrar usuarios con la misma contraseña más de una vez
repeated_user_passwords = user_password_counts[user_password_counts['count'] > 1]

# Imprimir el total de repeticiones de usuario-contraseña
total_repeated_user_passwords = repeated_user_passwords.shape[0]
print(f'Total de pares usuario-contraseña repetidos: {total_repeated_user_passwords}')
```

```
Total de pares usuario-contraseña repetidos: 0
```

Podemos observar que no es el caso, tras cargar el txt con las contraseñas crackeadas y buscar coincidencias {Usuario:Contraseña} observamos que no se dan.

Distribución de longitudes de contraseñas

Esta sección analiza la distribución de las longitudes de las contraseñas, centrándose en contraseñas con diferentes rangos (5 caracteres, 6 caracteres, etc.).

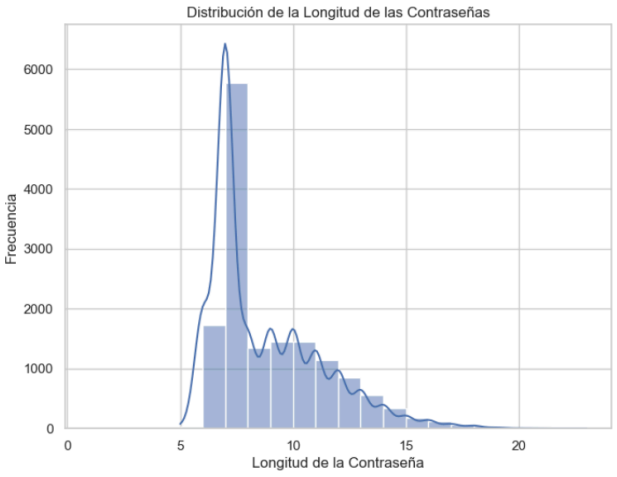

Imagen 5. Distribución de la Longitud de las Contraseñas

En esta gráfica se presenta la relación entre la frecuencia y la longitud de las contraseñas en el conjunto de datos. Se observa que las contraseñas con una longitud de 6 caracteres son las más comunes, alcanzando una frecuencia de hasta 6,000 unidades. A continuación, las contraseñas de 5 caracteres tienen una frecuencia cercana a 2,000 unidades. Por otro lado, las contraseñas de 7 caracteres en adelante muestran frecuencias más bajas, llegando hasta las contraseñas de longitud 17-18 como se puede apreciar en la gráfica.

Análisis de caracteres especiales en las contraseñas

La gráfica presentada entonces muestra la distribución de contraseñas según la presencia o ausencia de caracteres especiales. Se ha realizado un análisis donde las contraseñas se dividen en dos categorías: aquellas que contienen caracteres especiales (por ejemplo, símbolos como "!", "@", "#", etc.) y aquellas que no los

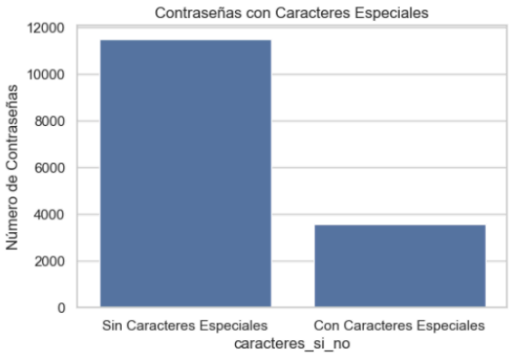

Imagen 6. Comparación de contraseñas: Con Caracteres Especiales vs. Sin Caracteres Especiales

contienen, es decir, que solo están formadas por letras y números.

El resultado es claro: una gran mayoría de las contraseñas (más de 11 mil) no contienen caracteres especiales, mientras que una menor cantidad (aproximadamente 4 mil) sí los incluye. Esto sugiere que, en el conjunto de datos analizado, las contraseñas que carecen de caracteres especiales son mucho más comunes, lo que puede implicar una menor complejidad y, potencialmente, menor seguridad en términos generales.

Identificación de contraseñas repetidas

En las imágenes se puede observar un análisis de contraseñas repetidas dentro de un conjunto de datos. Identificamos las contraseñas que aparecen más de una vez y presenta las más repetidas por longitud de contraseña.

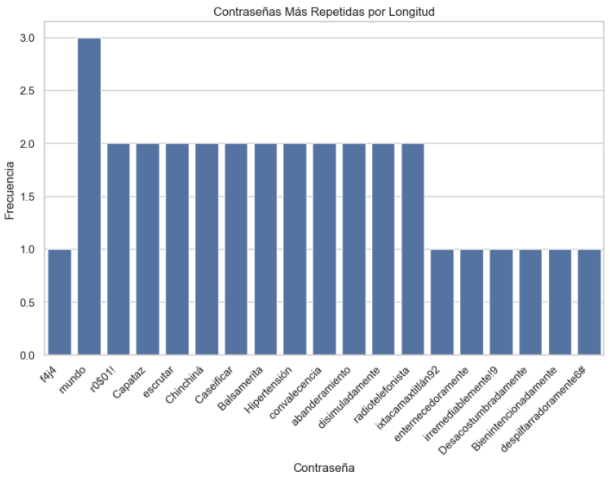

Imagen 6. Contraseñas más repetidas por longitud

Aquí por tanto podemos observar que las contraseñas más largas tienen muy pocas repeticiones, mientras a medida que reducimos el tamaño de ellas observamos una mayor repetición.

Sin embargo, este punto de análisis no es del todo significativo dado que el que se repita una contraseña para diferentes usuarios puede tratarse de una mera casualidad.

Consideraciones

Para la ejecución del script debemos tener en cuenta lo siguiente:

- Los archivos necesarios para ejecutarlo son:

```
# Archivos de entrada y salida
archivo_passwords_cracked = 'g22_foromotos.txt' # DataSet crackeado de foromotos
archivo_meneate = 'g22_meneate_entrada.txt' # DataSet sin crackear
archivo_resultados = 'g22_meneate.txt' # DataSet crackeado de meneate
```

  El contenido de *g22_foromotos.txt*, que son las contraseñas de foromotos crackeadas.
  El contenido de *g22_meneate.txt*, que son las contraseñas de meneate crackeadas.
  Y *g22_meneate_entrada.txt* es el archivo de entrada necesario para ejecutar el .py

Anexo

---

[1] kkrypt0nn. (n.d.). kkrypt0nn/wordlists [Repositorio GitHub]. GitHub. Recuperado de https://github.com/kkrypt0nn/wordlists