1. **Introducción**

Primero explicaremos los 4 comandos usados en John the Ripper para romper las contraseñas de g13\_foromotos.txt. Luego, explicaremos el script de Python que se usó para procesar los hashes de g13\_meneate.txt.

1. **Descripción de los comandos utilizados en John the Ripper**

**Primer comando:**

john --wordlist=rae\_words.txt --rules=dive --min-length=4 --fork=8 --format=Raw-MD5 g13\_foromotos.txt

- Explicación: En este primer paso, usamos un diccionario de palabras (‘rae\_words.txt’), que es el diccionario que más nos ha servido entre todos los que hemos probado.

- --rules=dive: Activa la regla ‘dive’, que aplica variaciones en las palabras del diccionario (por ejemplo, reemplazos de letras y números).

- --min-length=4: Establece una longitud mínima de 4 caracteres para las contraseñas, acorde a la política de contraseñas de foromotos.

- --fork=8: Ejecuta el proceso en 8 núcleos de CPU, mejorando la velocidad de procesamiento.

- --format=Raw-MD5: Especifica que el formato de los hashes es ‘MD5’.

- Resultados: Este comando rompió aproximadamente 40,000 contraseñas en unos 90 segundos.

**Segundo comando:**

john --wordlist=contraseñas.txt --rules=jumbo --min-length=4 --fork=8 --format=Raw-MD5 g13\_foromotos.txt

- Explicación: Utilizamos un segundo diccionario, ‘contraseñas.txt’, que contiene muchos términos en español, nombres y distintos tipos de contraseñas comunes.

- --rules=jumbo: Aplica la regla ‘jumbo’, que realiza muchas combinaciones posibles, como añadir números al final de palabras o cambiar letras por caracteres especiales.

- Otros parámetros (--min-length=4, --fork=8, --format=Raw-MD5) siguen igual que el primer comando, manteniendo los requisitos de longitud, el uso de múltiples núcleos y el formato ‘MD5’.

- Resultados: Este comando nos permitió romper alrededor de 17,000 contraseñas más en 60 segundos.

**Tercer comando:**

john --format=Raw-MD5 --min-length=4 --max-length=5 --fork=8 g13\_foromotos.txt

- Explicación: En este paso, utilizamos un ataque de fuerza bruta acotado, orientado a contraseñas cortas (entre 4 y 5 caracteres).

- --max-length=5: Limita la longitud máxima a 5 caracteres, lo cual es útil, ya que sabemos que algunos usuarios de ForoMotos pueden usar contraseñas muy cortas.

- El resto de las opciones siguen igual para aprovechar al máximo el hardware y trabajar en hashes ‘MD5’.

- Resultados: Este comando rompió aproximadamente 200 contraseñas adicionales en 45 segundos.

**Cuarto comando:**

john --max-run-time=1200 --format=Raw-MD5 --mask='?1?1?1?1?1?1' --fork=8 -1='[ -~1234567890aeiou]' g13\_foromotos.txt

- Explicación: Se trata de un ataque más largo que usa máscaras, atacando específicamente contraseñas de 6 caracteres con caracteres y símbolos usados frecuentemente en español.

- --max-run-time=1200: Limita el tiempo de ejecución a 1200 segundos (20 minutos) para que el proceso no sobrepase el tiempo total asignado.

- --mask='?1?1?1?1?1?1': Especifica una longitud fija de 6 caracteres para las contraseñas.

- -1='[ -~1234567890aeiou]': Define el conjunto de caracteres usados en el ataque, incluyendo números, letras en minúscula, y algunos caracteres especiales.

- Resultados: Este comando nos permitió romper unas 13,000 contraseñas más en 20 minutos.

1. **Descripción del script de Python para g13\_meneate.txt**

**Importación de módulos**

import bcrypt

import multiprocessing as mp

Primero, el código importa ‘bcrypt’, que ayuda a comparar una contraseña en texto plano con un hash bcrypt, y ‘multiprocessing’, que nos permite utilizar varios núcleos del CPU para acelerar el procesamiento. Al principio no usamos multiprocessing, y el archivo se nos ejecutaba pero sin romper ninguna contraseña.

**Definición de rutas de archivos**

hash\_file\_meneate = "./g13\_meneate.txt"

passwords\_file = "./hashes.txt"

output\_file = "./contraseñas\_rotas.txt"

Aquí se especifican las rutas de tres archivos:

- ‘g13\_meneate.txt’: contiene los hashes de las contraseñas que queremos descifrar.

- ‘hashes.txt’: incluye contraseñas en texto claro que ya han sido descifradas previamente.

- ‘contraseñas\_rotas.txt’: es el archivo donde se guardarán las contraseñas descifradas.

**Función ‘check\_password’**

def check\_password(hashes, actual\_password):

password = actual\_password[1]

for hash\_bcrypt in hashes:

if hash\_bcrypt[0] == actual\_password[0]:

return bcrypt.checkpw(password.encode('utf-8'), hash\_bcrypt[1].encode('utf-8'))

return False

Esta función recibe una lista de hashes y una contraseña en texto plano, y compara la contraseña con los hashes bcrypt en ‘g13\_meneate.txt’. Usa el método ‘bcrypt.checkpw’, que devuelve ‘True’ si hay coincidencia.

**Función ‘load\_hashes’**

def load\_hashes(file\_path):

hashes = []

with open(file\_path, 'r') as file:

for line in file:

user1, hash\_bcrypt = line.strip().split(':')

hashes.append([user1, hash\_bcrypt])

return hashes

Esta función carga los hashes bcrypt desde ‘g13\_meneate.txt’ y los formatea en una lista de pares ‘[usuario, hash]’.

**Función ‘load\_passwords’**

def load\_passwords(file\_path):

passwords = []

with open(file\_path, 'r') as file:

for line in file:

user2, password = line.strip().split(':')

passwords.append([user2, password])

return passwords

De la misma forma que ‘load\_hashes’, esta función carga contraseñas en texto claro desde el archivo ‘hashes.txt’, guardándolas en pares ‘[usuario, contraseña]’.

**Función ‘crack\_single\_hash’**

def crack\_single\_hash(hashes, passwords):

if check\_password(hashes, passwords):

print(f"Contraseña correcta para {passwords[0]}: {passwords[1]}")

return passwords

return None

Esta función recibe los hashes y una contraseña específica de un usuario. Si la contraseña es correcta para un hash, la imprime y la devuelve; de lo contrario, retorna ‘None’.

**Función ‘crack\_passwords\_parallel’**

def crack\_passwords\_parallel(hashes, passwords):

cracked\_passwords = []

num\_cores = mp.cpu\_count() # Calcula el número de núcleos

with mp.Pool(num\_cores) as pool:

# Genera una lista de tuplas (hashes y passwords)

results = pool.starmap(crack\_single\_hash, [(hashes, password) for password in passwords])

# Filtra los resultados válidos

cracked\_passwords = [result for result in results if result]

return cracked\_passwords

Esta función distribuye el trabajo de verificación de múltiples contraseñas al mismo tiempo, aprovechando todos los núcleos de la CPU disponibles. Con ‘pool.starmap’, el programa compara contraseñas en paralelo para cada hash.

**Función save\_cracked\_passwords**

def save\_cracked\_passwords(cracked\_passwords, output\_file):

with open(output\_file, 'w') as file:

for user, password in cracked\_passwords:

file.write(f"{user}:{password}\n")

Aquí se escriben las contraseñas descifradas en el archivo `contraseñas\_rotas.txt`, en el formato `[usuario:contraseña]`.

**Bloque main**

if \_\_name\_\_ == "\_\_main\_\_":

hashes = load\_hashes(hash\_file\_meneate)

passwords = load\_passwords(passwords\_file)

cracked\_passwords = crack\_passwords\_parallel(hashes, passwords)

save\_cracked\_passwords(cracked\_passwords, output\_file)

print(f"Proceso completado. Las contraseñas rotas se han guardado en {output\_file}.")

Este bloque principal carga los hashes y contraseñas en listas, las compara usando ‘crack\_passwords\_parallel’, y finalmente guarda los resultados en ‘output\_file’.

Con este script de Python hemos conseguido romper alrededor de 13,000 contraseñas de ‘g13\_meneate’.