Universidad Carlos III

Ingeniería Informática Curso 2024-25

Ingeniería de la Ciberseguridad

Práctica 1: Estudio de la Fortaleza de Contraseñas

Grupo 13:

- **Jaime Vaquero Rabahieh** (NIA: 100472248, Email: 100472248@alumnos.uc3m.es)
- **Pablo Agudo Moreno** (NIA: 100429972, Email: 100429972@alumnos.uc3m.es)

ÍNDICE

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.

2. COMANDOS PARA FOROMOTOS.TXT

a. Primer comando:

john --wordlist=rae_words.txt --rules=dive --min-length=4 --fork=8 --format=Raw-MD5 g13_foromotos.txt

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. Este comando está formado por los siguientes elementos:

- **--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’.

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

b. Segundo comando:

john --wordlist=contraseñas.txt --rules=jumbo --min-length=4 --fork=8 --format=Raw-MD5 g13_foromotos.txt

Utilizamos un segundo diccionario, ***‘contraseñas.txt’***, que contiene muchos términos en español, nombres y distintos tipos de contraseñas comunes. Este comando está formado por los siguientes elementos:

- **--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’.

Con este comando rompimos alrededor de 17,000 contraseñas más en 60 segundos.

c. Tercer comando:

john --format=Raw-MD5 --min-length=4 --max-length=5 --fork=8 g13_foromotos.txt

En este paso, utilizamos un ataque de fuerza bruta acotado, orientado a contraseñas cortas (entre 4 y 5 caracteres). Este comando está formado por los siguientes elementos:

- **--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'.
Este comando rompió aproximadamente 200 contraseñas adicionales en 60 segundos.

d. Cuarto comando:

john --max-run-time=1200 --format=Raw-MD5 --mask='?1?1?1?1?1?1' --fork=8 -1='[-~1234567890aeiou]' g13_foromotos.txt

Se trata de un ataque más largo que usa máscaras, atacando específicamente contraseñas de 6 caracteres con caracteres y símbolos usados frecuentemente en español. Este comando está formado por los siguientes elementos:

- **--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, y algunos caracteres especiales.

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

3. DESCRIPCIÓN SCRIPT DE PYTHON PARA MENEATE.TXT

a. 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.

b. Definición de rutas de archivo:

```
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.

c. 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.

d. 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]'.

e. 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]'.

f. 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'.

g. 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.

h. 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]`.

i. 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'.

4. DECLARACIÓN DE USO DE IA GENERATIVA

Por último, queríamos declarar el uso de IA generativa, que nos ayudó a reducir el tiempo de ejecución del script de Python para las contraseñas de ‘g13_meneate.txt’.
La idea de comparar usuarios en el script de Python vino porque la IA generativa nos sugirió hacer una comparación de los hashes previa a la ejecución que no funcionó, pero que nos hizo llegar a la otra idea.