Práctica Kali Linux Criptografía

Enrique Ulises Báez Gómez Tagle

21 de octubre de 2024

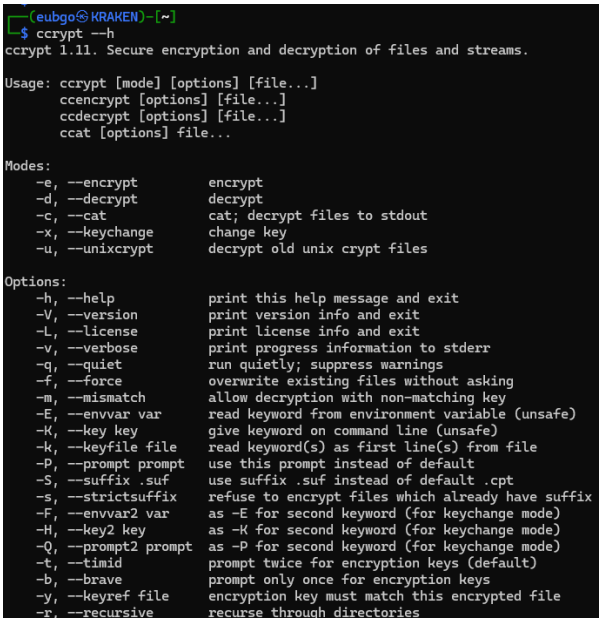
Índice

1. Crear Carpetas

```
mkdir openssl ccrypt
```

2. Investigación de los comandos

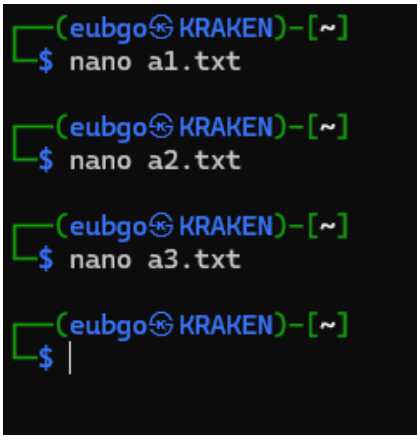
```
ccrypt --h
openssl -h
```



```
┌──(eubgo㉿KRAKEN)-[~]
└─$ openssl -h
help:

Standard commands
asn1parse         ca                ciphers           cmp
cms               crl               crl2pkcs7         dgst
dhparam           dsa               dsaparam          ec
ecparam           enc               engine            errstr
fipsinstall       gendsa            genpkey           genrsa
help              info              kdf               list
mac               nseq              ocsp              passwd
pkcs12            pkcs7             pkcs8             pkey
pkeyparam         pkeyutl           prime             rand
rehash            req               rsa               rsautl
s_client          s_server          s_time            sess_id
smime             speed             spkac             srp
storeutl          ts                verify            version
x509

Message Digest commands (see the `dgst' command for more details)
blake2b512        blake2s256        md4               md5
rmd160            sha1              sha224            sha256
sha3-224          sha3-256          sha3-384          sha3-512
sha384            sha512            sha512-224        sha512-256
shake128          shake256          sm3

Cipher commands (see the `enc' command for more details)
aes-128-cbc       aes-128-ecb       aes-192-cbc       aes-192-ecb
aes-256-cbc       aes-256-ecb       aria-128-cbc      aria-128-cfb
aria-128-cfb1     aria-128-cfb8     aria-128-ctr      aria-128-ecb
aria-128-ofb      aria-192-cbc      aria-192-cfb      aria-192-cfb1
aria-192-cfb8     aria-192-ctr      aria-192-ecb      aria-192-ofb
aria-256-cbc      aria-256-cfb      aria-256-cfb1     aria-256-cfb8
aria-256-ctr      aria-256-ecb      aria-256-ofb      base64
bf                bf-cbc            bf-cfb            bf-ecb
bf-ofb            camellia-128-cbc  camellia-128-ecb  camellia-192-cbc
camellia-192-ecb  camellia-256-cbc  camellia-256-ecb  cast
cast-cbc          cast5-cbc         cast5-cfb         cast5-ecb
```

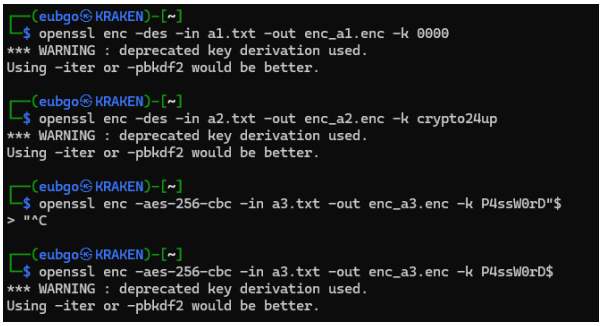
3. Creación de archivos

```
nano a1.txt
nano a2.txt
nano a3.txt
```



4. Cifrar archivos

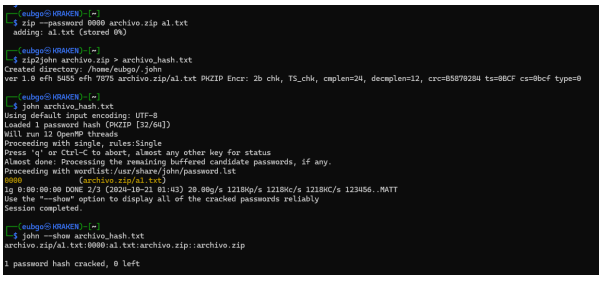
```
openssl enc -des -in a1.txt -out enc_a1.enc -k 0000
openssl enc -des -in a2.txt -out enc_a2.enc -k crypto24up
openssl enc -aes-256-cbc -in a3.txt -out enc_a3.enc -k
    P4ssW0rD$
```



5. Obtener Contraseñas

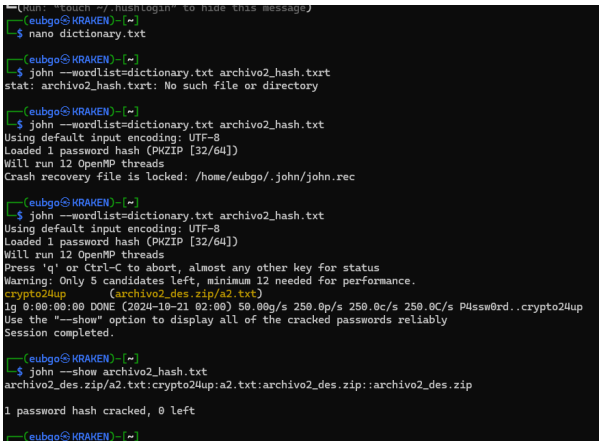
```
zip --password 0000 archivo.zip a1.txt
zip2john archivo.zip > archivo_hash.txt
john archivo_hash.txt
john --show archivo_hash.txt
```

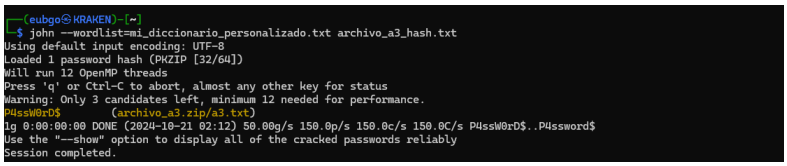
1. Se generó un ZIP con el archivo txt
2. Se obtuvo el hash del zip
3. Ser atacó con John
4. Se visualzan las contraseñas



Para el segundo y tercero, íbamos a usar crunch, pero estaba tardando mucho, por lo que optamos por demostrar el funcionamiento de una wordlist, en la que escondimos la contraseña,y le pedimos a john que la encontrara

```
nano dictionary.txt
john --wordlist=dictionary.txt archivo2_hash.txt
john --wordlist=dictionary.txt archivo2_hash.txt
john --show archivo2_hash.txt
```





6. Resultados y Desafíos Encontrados

Durante esta práctica, se realizaron diversas actividades de cifrado y descifrado utilizando herramientas como **OpenSSL**, **John the Ripper** y el uso de diccionarios de contraseñas. A continuación, se detallan los resultados obtenidos y los principales desafíos encontrados.

Para el **archivo a1.txt**, cifrado con el algoritmo **DES** y la contraseña de baja complejidad **"0000"**, se utilizó **John the Ripper** para descubrir la contraseña mediante un ataque directo. Debido a la sencillez de la contraseña, el proceso fue exitoso en pocos segundos. Este experimento destaca la importancia de evitar el uso de contraseñas triviales, ya que pueden ser descifradas fácilmente mediante ataques de fuerza bruta.

En el caso del **archivo a2.txt**, cifrado también con **DES**, pero con la contraseña de complejidad media **çrypto24up"**, se intentó realizar un ataque utilizando **Crunch** para generar combinaciones. Sin embargo, debido al alto número de combinaciones posibles, el proceso resultó ineficaz en términos

de tiempo y recursos. Como alternativa, se optó por utilizar una lista de contraseñas (**wordlist**) personalizada que contenía la contraseña escondida. Al utilizar esta wordlist con **John the Ripper**, la contraseña fue descubierta exitosamente.

Finalmente, para el **archivo a3.txt**, cifrado con **AES-256-CBC** y la contraseña **"P4ssW0rD$"**, se realizó un ataque similar, utilizando también una wordlist. Aunque la contraseña era más compleja y requería más tiempo de procesamiento, **John the Ripper** pudo encontrar la contraseña con éxito, demostrando la eficiencia de los ataques basados en diccionarios en estos casos.

Un desafío importante fue el manejo de contraseñas más complejas y el ajuste del tamaño de las wordlists. En particular, para el cifrado con **AES-256-CBC**, fue crucial optimizar las listas de contraseñas y evitar el uso de ataques de fuerza bruta completos, que eran imprácticos debido al alto número de combinaciones.

7. Conclusión

A lo largo de esta práctica, se realizaron ejercicios de cifrado y descifrado con herramientas populares como **OpenSSL** y **John the Ripper**. Aunque se enfrentaron dificultades al realizar ataques de fuerza bruta en contraseñas más complejas, el uso de diccionarios personalizados permitió superar estos obstáculos y completar el proceso de descifrado exitosamente. Esta práctica subraya la importancia de utilizar contraseñas fuertes y complejas para proteger los datos y la necesidad de optimizar los ataques de descifrado para contraseñas de diferentes niveles de complejidad.