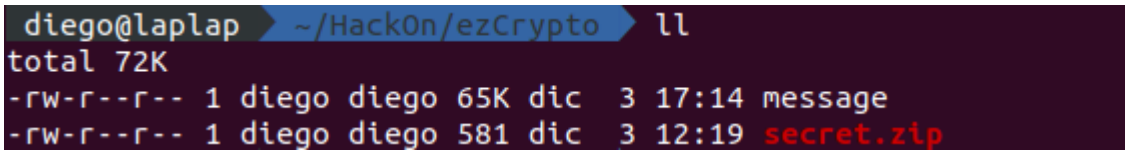
Se nos proporciona un zip "ezCrypto.zip". Si lo descomprimimos podemos ver lo siguiente:



Un fichero "message" que contiene texto y un fichero comprimido que pide contraseña.

Si miramos el contenido del fichero "message" podemos ver muchos ceros y unos. Al principio, podemos pensar que se trata de alguna codificación extraña, pero no. Es un código QR, donde cada valor representa un pixel. En particular, el cero representa un pixel en blanco y el uno un pixel en negro. Podemos crear un pequeño script en python que haga esta conversión o usar una herramienta online como: https://bahamas10.github.io/binary-to-qrcode/

Si miramos el contenido del QR podemos ver el siguiente texto:

`MeGust4n_l0s_Tr4baL3nguAS`

Como teníamos un fichero comprimido con contraseña, rápidamente podemos pensar que esta frase es la contraseña del zip. Efectivamente, era la contraseña del zip y ahora tenemos dos archivos más:

```
diego@laplap ~/HackOn/ezCrypto ll
total 8,0K
-rw-r--r-- 1 diego diego 172 dic  3 10:25 flag.enc
-rw-r--r-- 1 diego diego 280 dic  3 12:18 scriptEnc.py
```

El archivo `flag.enc` contiene un texto que parece estar cifrado con el algoritmo definido en el fichero `scriptEnc.py`

Vamos a mirar el script:

```
import string


def enc(flag):
    cipher = []
    for character in flag:
        cipher.append((11*ord(character)%256))
    return bytes(cipher)

if __name__ == "__main__":
    flag = ""
    cipher = enc(flag)
    f = open('flag.enc','w')
    f.write(cipher.hex())
    f.close()
```

Si nos fijamos abre el fichero flag.enc en modo escritura y escribe en él el resultado de llamar a la funcion `enc` pasando como argumento la flag.
Por tanto, tenemos que ser capaces de darle la vuelta al proceso de cifrado para poder descifrar.

El verdadero proceso de cifrado se lleva a cabo en esta operación.

```
cipher.append((11*ord(character)%256))
```

A cada carácter de la flag (pasado a ASCII) se le multiplica por 11 y a este resultado se le aplica la operación de módulo, concretamente, módulo 256. Dado que esta operación es "determinista" podemos programar un script que mediante fuerza bruta saque la flag. Va probando cada uno de los caracteres de una wordlist e iterando por cada carácter del archivo `flag.enc` . Si encontramos un carácter de nuestra wordlist que cifrado dé el mismo carácter que hay en el archivo `flagn.enc` habremos encontrado el carácter original. De esta manera podemos sacar la flag. Pero ¿y si aplicamos las mates?

Bueno, en este punto entran en juego algunos conceptos de aritmética modular.
Existe un inverso modular para el 11 mod 256. Esto podemos verlo de la siguiente forma:

```
pow(11,-1,256)
```

```
>>> pow(11,-1,256)
163
```

Como existe un inverso, tan solo tenemos que reescribir la ecuación congruente, pero debemos multiplicar por este valor. De esta mera obtendremos la flag.

El solver final es:

```python
def dec(flagEnc):
    plain = []
    for character in flagEnc:
      character = 163 * character % 256
      plain.append(character)
    return bytes(plain)

if __name__ == "__main__":
  with open('flag.enc') as f:
    cipher = bytes.fromhex(f.read())
  plain = dec(cipher)
  print(plain)
```

```
diego@laplap ~/HackOn python3 solver.py
b'HackOn{el_volc4n_d3_parangaricutirimicuar0_est4_4_punt0_d3_desparangaricutimiricuarse}'
```

La flag es:

`HackOn{el_volc4n_d3_parangaricutirimicuar0_est4_4_punt0_d3_desparangaricutimiricuarse}`

Con esto terminamos el reto. Espero que os haya gustado y, lo más importante, que hayáis aprendido.

Que la Fuerza os acompañe

DiegoAltF4