Writeup P3 Tarea 3

Writeup P3 Tarea 3

Nombre: Benjamín Aguilar Osorio
Equipo: L05 1NCR31BL35

Persona a cargo de la pregunta **Benjamín Aguilar Osorio**

Descripción general del problema

Descripción general: Problema de reversing enfocado en la decompilacion de un codigo, y en entender el funcionamiento de este

Metodología de resolución

A continuación se darán los **pasos** que se llevaron a cabo para resolver el problema:

1. Inicialmente, al leer el enunciado se puede suponer que se tratará de un problema de cifrado y descifrado de la flag. Al descargar el archivo adjuntado, se reconocen 2 tipos en particular: por un lado, un archivo de código compilado en Python (pyc) y un archivo que presuntamente posee la flag, el cual está como mensaje cifrado en lo que parece ser base64. El mensaje es: **=wUYkIDZwIWMxADM2IjN0IjNyMmZwMDMjJWZzEWYzMjN1gTNjFzM1ImZ1UWOzYjYjFzN0kzYzIjYygHM**.
2. A continuación, para proseguir con el problema, se utilizó un decompilador sobre el archivo pyc. En particular, se utilizó el programa "uncompyle6". Esto entrega el siguiente programa (notar que el programa es de Python 2.7).

```
import zlib, base64, requests
FLAG_FILE = 'flag.txt'
with open(FLAG_FILE, 'r') as (f):
    FLAG = f.read()
SEED = requests.get('https://random.uchile.cl/beacon/2.0-beta1/pulse?
time=1685437365000').json()['pulse']['localRandomValue']
x = 0
for c in FLAG:
    if ord(c) & 1:
        x += 3
    else:
        x /= 2

COMPRESSED_FLAG = zlib.compress(FLAG[::-1]).encode('hex')
```

```
XORED = hex(int(COMPRESSED_FLAG, 16) ^ int(SEED[:len(COMPRESSED_FLAG)], 16))
ENCODED = base64.b64encode(XORED)
print ENCODED[::-1]
```

3. El código entregado describe un proceso de cifrado basado en el uso de la operación XOR, en particular, apoyándose en un generador aleatorio (https://random.uchile.cl/beacon/2.0-beta1/pulse?time=1685437365000). En un principio, este método de cifrado debiese ser altamente demandante en tiempo computacional, pero en este caso, el generador aleatorio entrega un valor correspondiente a una unidad de tiempo, y al ser este una variable no cambiante ni dependiente del tiempo, el valor aleatorio es siempre constante.
4. Teniendo lo anterior en consideración, se obtiene el valor aleatorio correspondiente al SEED simplemente ingresando a la URL entregada, siendo este: **53bf2aed297ff3d60f602af72299a6968a363df42f0820002f04dfcb68e57541257c5c84fad94290619aef20f173705567a091de222c8d8b8cd35b17c6390144**.
   5.Finalmente, se realiza el proceso inverso de encriptación con el fin de desencriptar la flag. Para esto, se aplica la propiedad inversa de la operación XOR, obteniéndose el siguiente programa (notar que el programa es de Python 3.9):

```
ENCODED =
"=wUYklDZwIWMxADM2IjN0IjNyMmZwMDMjJWZzEWYzMjN1gTNjFzM1ImZ1UWOzYjYjFzN0kzYzIj
YygHM"
SEED =
"53bf2aed297ff3d60f602af72299a6968a363df42f0820002f04dfcb68e57541257c5c84fad
94290619aef20f173705567a091de222c8d8b8cd35b17c6390144"
REVERSE_ENCODED = ENCODED[::-1]



DECODED = base64.b64decode(REVERSE_ENCODED)
DECODED2= "0x2b23c9471cb639e5fb531c585633aa3ebc030fc26246260011b0d9daL"
DECODED = DECODED.decode("utf-8")
XORED = DECODED.strip("0xL")
XORED
COMPRESSED_FLAG = hex(int(XORED,16) ^ int(SEED[:len(XORED)],16))
COMPRESSED_FLAG= COMPRESSED_FLAG.strip("0x")
ZIP_FLAG = bytes.fromhex(COMPRESSED_FLAG)
flag = zlib.decompress(ZIP_FLAG)
print(flag[::-1])
```

6. Finalmente, se encuentra la flag, siendo esta:**cc5325{PyBy73N1nj4}**
7. Como último paso, se vuelve a cifrar la flag, buscando verificar que el valor obtenido sea igual al valor de la flag cifrada original.