By {l4_p4nd1ll4_m4nt3q1ll4}
Base64

Al conectarse al servicio nos da 4 opciones una de ellas es obtener la bandera, esta opción regresa un texto que aparentemente es base64: AgfJA2rLzNTIyxnLnJrFm3nFzdnTnhmXngqWx2y0yZfSx3a0CJrFyZbTm256nhj9cG==

```
root@kali:~/ctf# nc localhost 3101

##############bASE64##############
Selecciona una opcion:
1. Codifica cadena
2. Decodifica Base64
3. Obtener bandera
4. Salir
 >>>>
```

pero al decodificarla no regresa la bandera en texto claro.

```
root@kali:~/ctf# echo AgfJA2rLzNTIyxnLnJrFm3nFzdnTnhmXngqWx2y0yZfSx3a0CJrFyZbTm256nhj9cG== | base64 -d
0j00000~0ŝy000300
00l0d00v00qənz00proot@kali:~/ctf#
```

Analizando un poco el Código fuente vemos la siguiente función que modifica una cadena después de ser codificada en base64

```
def magic(cryptic):
    return ''.join(c.lower() if c.isupper() else c.upper() if c.islower() else c for c in cryptic)
```

Esta función intercambia mayúsculas por minúsculas y viceversa. Entonces basta con hacer lo mismo con la flag que nos dan.

```
Python 2.7.16+ (default, Sep  4 2019, 08:19:57)
[GCC 9.2.1 20190827] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import base64
>>> flag='AgfJA2rLzNTIyxnLnJrFm3nFzdnTnhmXngqWx2y0yZfSx3a0CJrFyZbTm256nhj9cG=='
>>> base64.b64decode(''.join(c.upper() if c.islower() else c.lower() if c.isupper() else c for c in flag))
'hackdef{base64_3s_d3m4s14d0_f4c1l_p4r4_c0m3nz4r}\n'
>>>
```

FLAG: hackdef{base64_3s_d3m4s14d0_f4c1l_p4r4_c0m3nz4r}

RSA

En este reto nos dan 2 archivos, *llave_publica.pem* y *bandera_enc,* al ser un reto de RSA y al darnos esos archivos posiblemente se resuelva recuperando la llave privada a partir de la llave pública.

Desde Python se puede extraer los valores de la llave publica y en un primer vistazo el módulo es enorme pero el exponente es muy pequeño por lo que factorizar el módulo no sería posible y la primera idea queda descartada. Un ataque muy común es el ataque por exponente bajo, esto pasa cuando el exponente es tan bajo que hace posible calcular su raíz, un ejemplo claro es cuando **e**=3 o 17 como en el caso de este reto.

```
>>> from Crypto.PublicKey import RSA
>>> pub=open("llave_publica.pem","r")
>>> pub_key=pub.read()
>>> pubkey = RSA.importKey(pub_key)
>>> n=pubkey.n
>>> e=pubkey.e
>>>
>>> e
17L
>>> n
354857085266372835707960430237258938120115561196490827311307871471772110379878858
587643322005997196969778632649565043163354828860491458218767080612770829980026509
540361986102223256148019712119599424120111276176526668456368246545292150491207907
577323328290320914550912514311151919899178593498033556813713065955427823709992503
506879100966802258405532199969238311795203131991093873872884551236213083417395040
392333827793801531685453462230479600223582715419546935395162048457357274498368236
004419964626005210009871977544012856745479565434811513055202458836381225775122548
806466416538956854710191716367218588892268183374520961876703874578298301450612423
107248831099150645811377550182107208857527914471464664458981712996852775035960135
```

Recordando un poco de la forma de cifrado de RSA tenemos:

t=texto claro pasado a número entero
e=exponente publico
n=modulo
c=texto cifrado

$$c = t^e mod\ n$$

Entonces tendríamos que calcular la raíz 17 de nuestro texto cifrado el cual es el archivo: *bandera_enc* , para calcular la raíz podemos usar la función **iroot** para Python que está en la biblioteca **gmpy2**

Código completo:

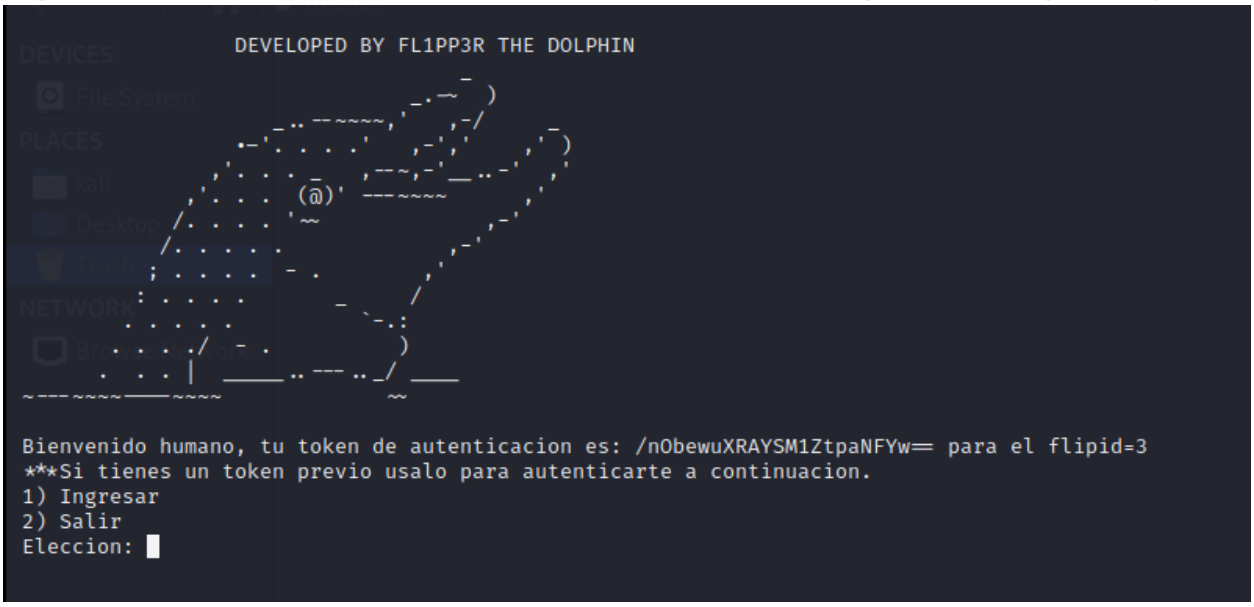
```
from Crypto.PublicKey import RSA
from Crypto.Util.number import long_to_bytes,bytes_to_long
import gmpy2

pub=open("llave_publica.pem","r")
pub_key=pub.read()
pubkey = RSA.importKey(pub_key)
n=pubkey.n
e=pubkey.e
enc=open("bandera_enc","r")
c=enc.read()
c=bytes_to_long(c)
t=gmpy2.iroot(c,17)[0]
flag=long_to_bytes(t)
print "Flag: "+ flag
```

```
root@kali:~/ctf# python sol_RSA.py
Flag: hackdef{S13mpr3_m4n3j4r_3xp0n3nt3s_4lt0s}
```

Fl1pp3r

Ingresando al servicio nos muestra un menú con 2 opciones, Ingresar y Salir



Pero aparte nos da un supuesto token de invitado para ingresar junto con el ID al que pertenece ese token, el ID del token de invitado se genera aleatoriamente y van desde el 0 al 5, pero para obtener la bandera debemos ingresar con un token con un

ID=6.
El token que se genera está cifrado con AES en modo CBC

```
def delfin_decrypt(iv,key,msg):
    try:
        ciphertext = base64.b64decode(msg)
        cipher = AES.new(bytes(key,"utf-8"),AES.MODE_CBC,iv)
        el_string = unpad(cipher.decrypt(ciphertext),AES.block_size,style="pkcs7")
        return str(el_string,"ascii")
    except:
        return "Padding Incorrecto"
```

Y en texto claro el token tiene lo siguiente: **flipid=#**

```
mensaje = "flipid=" + str(random.choice(range(100)) % 5)
```

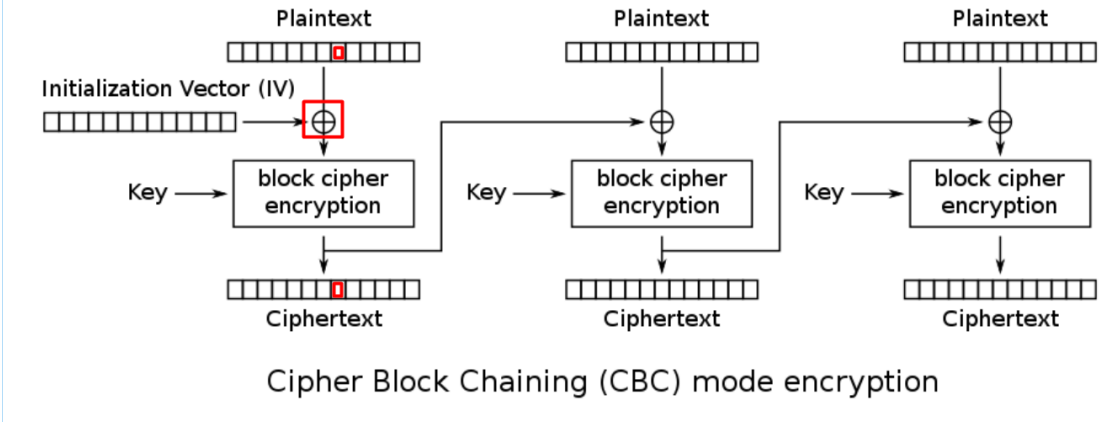
```
(str(delfin_encrypt(iv,key,mensaje),"utf-8"),mensaje),"utf-8")
```

Otra cosa que nos mencionan es que al IV del cifrado AES lo llaman **Llave de inicialización** y esta son 16 letras 'A'

```
iv = "\x41"*16
```

Ya que el texto del token es muy pequeño cuando se cifra solo nos regresa un bloque cifrado entonces, uniendo las pistas que nos dan, podemos asegurar que debemos modificar un solo byte del mensaje cifrado así cuando el sistema lo descifre nos dará el ID que necesitamos, gracias al modo de operación CBC donde el bloque que se descifra se le hace una operación XOR con el IV da como resultado el texto claro.

*'flipid=6**PADDING**'* **XOR '***AAAAAAAAAAAAAAA'* = TOKEN
Si recordamos una propiedad del XOR donde:
A ***XOR*** *B = C*
A ***XOR*** *C = B*
Tenemos
? ***XOR*** *'6' = Byte 8 del token*
ya que podemos manipular el IV que se usa, podemos obtener el valor que necesitamos y conseguir esta igualdad:
? ***XOR*** *? = 6*



Cipher Block Chaining (CBC) mode encryption

1er paso, obtener un Token "/nObewuXRAYSM1ZtpaNFYw==" y elegir la opción de ingresar e introducir el token.

```
Bienvenido humano, tu token de autenticacion es: /nObewuXRAYSM1ZtpaNFYw== para el flipid=3
**Si tienes un token previo usalo para autenticarte a continuacion.
1) Ingresar
2) Salir
Eleccion: 1
Ingresa tu token: /nObewuXRAYSM1ZtpaNFYw==
Ingresa la llave de inicializacion:
```

2do paso, modificar la llave de inicialización, reemplazando la octava A por el numero 6 el cual es el número que necesitamos en el ID.

```
Bienvenido humano, tu token de autenticacion es: /nObewuXRAYSM1ZtpaNFYw== para el flipid=3
**Si tienes un token previo usalo para autenticarte a continuacion.
1) Ingresar
2) Salir
Eleccion: 1
Ingresa tu token: /nObewuXRAYSM1ZtpaNFYw==
Ingresa la llave de inicializacion: AAAAAAA6AAAAAAAA

El token pertenece al usuario: flipid=D
Quieres seguir intentando? (S/N):
```

3er paso, ahora que tenemos el texto claro modificando el byte correspondiente al ID ya tenemos el valor que necesitamos para el XOR y para obtener el ID 6

? **XOR** 6 = D
? **XOR** D = 6 -> Listo!!!
ahora intercambiamos ese byte en la llave de inicialización.

AAAAAAADAAAAAAAA y obtendremos la bandera.

Flujo completo:

```
Bienvenido humano, tu token de autenticacion es: /nObewuXRAYSM1ZtpaNFYw== para el flipid=3
**Si tienes un token previo usalo para autenticarte a continuacion.
1) Ingresar
2) Salir
Eleccion: 1
Ingresa tu token: /nObewuXRAYSM1ZtpaNFYw==
Ingresa la llave de inicializacion: AAAAAAA6AAAAAAAA

El token pertenece al usuario: flipid=D
Quieres seguir intentando? (S/N): s

**Si tienes un token previo usalo para autenticarte a continuacion.
1) Ingresar
2) Salir
Eleccion: 1
Ingresa tu token: /nObewuXRAYSM1ZtpaNFYw==
Ingresa la llave de inicializacion: AAAAAAADAAAAAAAA

Felicidades: FLAG{Hola_35y4_EsUn4_Fl4g_d3_Prueb@}
```