

Crypto 100: base64

Descripción del reto:

base64 (100pts) ✓

Antes de este CTF creía que conocía base64 encoding y decoding, pero creo que no, nos puedes ayudar a entender el algoritmo y entonces, decodificar la bandera?

Nos dan el código fuente del servicio con las funciones de codificación y decodificación!

Y aquí puedes interactuar con el servicio:

<http://3.19.72.219:3133>

 `b64.py` `0d89726cf65a2f79b6b4bce2ea94e7cc`

En el reto nos mencionan que es base64 y nos proporcionan un archivo con el cual muestran el algoritmo que utilizan para codificar el texto.

El contenido del archivo es el siguiente:

```
import socketserver
import base64

with open('flag.txt','r') as f:
    flag = f.readline()

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

def bASE64(cadena):
    return str.encode(magic(str(base64.b64encode(cadena))))

def decode_base64(cadena):
    return base64.b64decode(cadena)

class MyTCPHandler(socketserver.BaseRequestHandler):
    def handle(self):
        opcion = 0
        while opcion != 4:
            try:
                self.request.send(b"\r\n#####bASE64#####\r\n")
                self.request.send(b"Selecciona una opcion:\r\n")
                self.request.send(b"1. Codifica cadena\r\n2. Decodifica Base64\r\n3. Obtener bandera\r\n4. Salir\r\n >>>>")
```

```

opcion = int(self.request.recv(1024).strip())
if opcion == 1:
    self.request.send(b"\r\nDame algo para codificar: ")
    to_send = bASE64(self.request.recv(1024).strip())[2:-1]
    self.request.send(b"\r\nCadena codificada: ")
    self.request.send(to_send)
    self.request.send(b"\r\n\r\n")
elif opcion == 2:
    try:
        self.request.send(b"\r\nDame algo para decodificar: ")
        to_send = decode_base64(self.request.recv(1024).strip())
        self.request.send(b"\r\nCadena decodificada: ")
        self.request.send(to_send)
        self.request.send(b"\r\n")
    except:
        self.request.send(b'\r\nOperacion no permitida...')
        self.request.send(b"\r\n")
elif opcion == 3:
    self.request.send(b"\r\nAqui tiene su bandera: ")
    self.request.send(bASE64(str.encode(flag))[2:-1])
    self.request.send(b"\r\n")
elif opcion == 4:
    self.request.send(b"\r\nHasta pronto...")
else:
    self.request.send(b"\r\nOpcion incorrecta...")
    self.request.send(b"\r\n")
except:
    self.request.send(b'\r\nOperacion no permitida...')
    self.request.send(b"\r\n")

```

```

if __name__ == "__main__":
    HOST, PORT = "0.0.0.0", 3101

```

```

server = socketserver.ThreadingTCPServer((HOST, PORT), MyTCPHandler)
server.allow_reuse_address = True
server.serve_forever()

```

Al analizar el código podemos observar que las funciones para codificar y decodificar son muy diferentes

```

def bASE64(cadena):
    return str.encode(magic(str(base64.b64encode(cadena))))

```

```
def decode_base64(cadena):  
    return base64.b64decode(cadena)
```

El truco esta en entender el algoritmo que utilizan para codificar el texto a base64.

Nos conectamos al servicio para ver la flag, la cual es la siguiente:

“AgfJA2rLzNTlyxnLnJrFm3nFzdnTnhmXngqWx2y0yZfSx3a0CJrFyZbTm256nhj9cG==”

El mismo codigo nos dice como codificaron la flag:

```
self.request.send(b"\r\nAqui tiene su bandera: ")  
    self.request.send(bASE64(str.encode(flag))[2:-1])
```

Si para codificar se utiliza esta funcion,

```
def bASE64(cadena):  
    return str.encode(magic(str(base64.b64encode(cadena))))
```

Para decodificarlo seria lo mismo pero al reves

```
return base64.b64decode(str(magic(str.decode(cadena))))
```

Hacemos un script con en el cual decodificaremos el texto en base64.

```
import base64  
cadena =  
"AgfJA2rLzNTlyxnLnJrFm3nFzdnTnhmXngqWx2y0yZfSx3a0CJrFyZbTm256nhj9cG=="  
def magic(cryptic):  
    return "".join(c.lower() if c.isupper() else c.upper() if c.islower() else c for c in cryptic)  
  
def decode_base64(cadena):  
    return base64.b64decode(str(magic(str.decode(cadena))))  
  
print (decode_base64(cadena))
```

La flag es: `hackdef{base64_3s_d3m4s14d0_f4c1l_p4r4_c0m3nz4r}`

Crypto 200: RSA

Descripción del reto:

RSA (200pts) ✓🌟

Aquí vamos de nuevo con RSA, te damos la bandera encriptada: bandera_enc, y la llave publica: llave_publica.pem.
El reto es muy simple, desencriptar dicha bandera

Hint! El exponente es muy chico o es muy alto o es normal?

📄 **bandera_enc** d95fc81f6c1258e1e65592e9868ff3ba

📄 **llave_publica.pem** 35371de4b80af1a1d1f3a3852ba61ec3

Nos proporcionan 2 archivos “bandera_enc” y “llave_publica.pem”.
Primero veremos que contiene el archivo de “llave_publica.pem” con el siguiente comando:

```
openssl rsa -noout -text -inform PEM -in llave_publica.pem -pubin
```

Y lo que no regresaría sería lo siguiente:

Public-Key: (6141 bit)

Modulus:

```
1a:ea:54:9a:0a:8d:62:f9:cc:f4:a7:da:c0:88:8a:
db:d5:a9:ee:cf:b5:6e:9c:b3:b7:26:5f:8e:99:b3:
a2:96:68:eb:41:e3:c9:dd:4d:59:8d:18:73:f8:8d:
02:8b:25:a1:1f:13:24:dd:f4:0d:d0:2b:18:8c:d0:
d2:be:f6:5e:7d:14:d3:a7:66:af:07:e5:86:df:39:
82:4c:b0:4e:5f:1a:4f:ae:95:1f:0f:3f:f0:40:e7:
0b:6c:d3:65:31:5c:11:2b:43:77:30:89:fd:8d:40:
40:72:43:33:24:14:2e:b3:46:c4:80:c5:88:0b:6f:
77:9e:df:07:a3:aa:4c:9f:34:24:e0:5e:01:21:2f:
2f:f3:cf:09:4d:60:02:d0:d8:ab:a7:49:a4:1f:d8:
2c:b3:ea:20:51:66:a6:f4:7c:4a:37:69:50:04:bb:
ae:0f:33:82:cd:0d:d4:61:8a:b0:a1:e4:88:8d:e3:
09:81:64:87:c9:22:d2:41:a0:df:42:cc:66:92:35:
db:a0:78:6e:0d:54:3d:4d:e7:91:c5:02:fa:97:f6:
b2:a4:a9:e8:84:16:4b:51:75:ed:d1:37:ad:87:67:
19:9c:bb:f6:23:e3:16:16:c3:b8:78:70:19:87:b7:
99:69:fc:cc:45:f4:46:0d:66:98:8f:99:85:e6:81:
26:77:dc:1d:39:ed:8c:fa:7b:0b:54:42:75:73:7f:
e8:3d:8c:65:1a:56:48:53:e8:26:3a:b1:4b:98:f6:
```

af:af:3a:b7:b4:12:93:8c:f5:2d:60:c2:16:f8:f6:
a9:87:5d:b7:19:95:c1:d7:00:b9:d1:7d:50:83:e8:
81:e0:63:83:0b:8b:26:19:33:ba:7b:07:4d:e6:dd:
41:d4:90:3c:55:f9:9e:39:8a:f4:e5:84:84:fd:b6:
86:8b:50:da:7a:0e:9d:1e:d4:c2:8c:63:39:7d:b1:
c8:3e:73:2e:e1:f2:e0:06:32:be:55:2e:92:3c:55:
fd:fc:86:06:46:c2:86:6a:a1:7f:00:89:ec:90:79:
48:59:4b:ca:22:57:66:43:e5:8f:28:99:9b:20:87:
2d:0b:8c:c7:f1:ef:37:d9:9a:5e:20:fb:df:0a:69:
4d:c3:27:fb:23:0f:54:77:4a:1d:5a:1a:38:25:25:
e6:05:e5:51:67:e0:7e:74:5c:d3:36:3b:b3:66:d5:
6b:bd:11:19:eb:b1:bd:1b:9f:bc:7c:7e:1f:1b:be:
78:18:e7:5c:97:1e:96:87:d4:91:97:0c:cc:0a:65:
78:a8:03:d5:7d:ce:bf:ef:14:07:76:6c:78:ef:a2:
4b:be:70:cf:0e:bf:2d:e2:e9:21:78:98:6b:4f:a8:
5b:84:91:de:1d:28:46:d2:a9:34:f5:bc:85:c8:c0:
c1:cf:cb:3b:21:a0:a8:37:48:fe:38:51:c3:24:67:
b1:66:98:81:18:86:20:84:02:0d:67:33:1c:21:2a:
bc:ef:fb:3d:41:21:e8:28:5c:0f:b1:3e:9c:46:96:
4e:47:79:32:e7:60:65:a5:d7:44:8e:22:1e:e6:d2:
86:4e:b5:ff:cf:83:49:51:49:17:0e:87:98:26:7f:
06:1b:c0:43:55:a8:73:0c:01:60:28:cd:4f:80:ee:
2b:77:79:ab:65:9d:de:41:0e:00:e1:f3:1a:06:ce:
01:03:a9:61:b8:fa:dc:ac:a8:fe:c1:00:db:ec:68:
98:c7:b0:7c:9f:ab:46:d0:70:39:e4:23:40:70:ec:
0c:8e:5d:d9:79:3e:85:21:a5:04:f3:f2:6c:f0:70:
4d:d1:f0:fb:46:13:9c:53:b1:a7:d5:69:c2:72:9e:
54:b6:f0:17:05:7b:c3:10:dc:8d:2b:d0:65:b1:9c:
9a:62:3b:28:65:33:41:83:8b:13:3b:9c:43:82:a8:
2f:4e:f1:71:34:b5:27:fc:4f:60:86:7e:2c:c5:02:
11:17:59:c5:73:77:7e:df:aa:72:31:2d:e0:d3:3f:
dc:24:bb:b4:ab:49:83:a7:c4:43:c9:c8:d2:8d:52:
96:14:65

Exponent: 17 (0x11)

Podemos observar que nos muestra que el exponente es 17, esto nos indica que es un ataque de exponente bajo, tambien el modulo esta en hexadecimal, yo me siento más comodo pasando el modulo a decimal, así que haré eso.

Convirtiendo los datos tenemos que

e=17

n=3548570852663728357079604302372589381201155611964908273113078714717721
103798788580269734536344325766744392236763021310065826032298993843569791

939031029442597804169692367605142704300756013745047500127631840932695587
643322005997196969778632649565043163354828860491458218767080612770829980
026509870109098868762603198082319321573704115100663470529644919018030054
257283115767836135903201254808614607423979063600391158045013469954036198
610222325614801971211959942412011127617652666845636824654529215049120790
726965861181285380303607756827590198526457602891740831207810047206831728
151640049712745196099650879281018850331980999279630812011155773233282903
209145509125143111519198991785934980335568137130659554278237099925033495
337080283460071354073240709058978451709932296886843204277745880314243131
436233880996348134764511036323255810838545932441738425506879100966802258
405532199969238311795203131991093873872884551236213083417395040066858769
742832592233088642026249623966692442995248203405703660093573902432925915
195331110560502136042826389108413221687120295123639233382779380153168545
346223047960022358271541954693539516204845735727449836823608306886590756
067015582452580983535856617610328011724185438163747113739040419646096869
185071265971935844575219833870754955193082620044199646260052100098719775
440128567454795654348115130552024588363812257751225481699319728414593021
909298710538677752520858527174208559816545197407545293375909747205522249
663665817092459671818885568941516084763806466416538956854710191716367218
588892268183374520961876703874578298301450612423669992388236207370710508
987379344863477060226681021478165271921008190935948536416354078188395689
346291214758502373896310868662591310724883109915064581137755018210720885
752791447146466445898171299685277503596013583636123543518992461823844370
967911364266867277528929888295765522151733504382053

Y por ultimo tenemos c, que es nuestra bandera y que tambien es un archivo, haremos un pequeño script en el que obtendremos los bytes del archivo. Quedaría de la siguiente manera:

```
from Crypto.Util.number import bytes_to_long
c = bytes_to_long(open("c_enc","rb").read())
print (c)
```

Con esto obtenemos el valor de nuestro c el cual es,

C=7240686328395535790852402319233160169513748780184226940150046620732432
658334985466561837538982137639907964240123557950467159495311245514361303
714890035352731634056048494759553715847906284334490784586006501048308861
235710275283454086277237487371951509735602293285458740986923886938940342
267753100799362036150022677006776907754616667222171341565085876825449952
153336936105299966066033620533076592588570879148211496191371249659940337
319954620579141683494428611648301928879117697965248078414120854730552744
644410697684498360743160806104634656449915914434854753510998736328004586
964776423293941426563290590412638281448404201567172346488178246804822709

186288745189838589499060337255724147340359355043088569614273303308731831
280346505130369527649488152646009870934833595918868931249663492683278521
120593791153979040070416376997568059994603062336979846841561238190481750
268609334349355269429808522553455873207367732840349311564230689263426716
344571097469788467126003813246711646316976781071176544169952449171847126
678610354774219685862412319566406571038750218780054608999912221842240588
956188118930386981422422334968361971021540487482631924324243250595025740
771388895836078274842058753663329944503580744505274121263964790969523356
240127845258484104397229708665907320802536107914846378401036598100402733
519284325608042912607036103317620101155537230120056109170671423961275965
222178322151562458091921007635227222839871493940054183633852064682382512
761283653296647171036845408983075980589938200155780508848544915730886858
707734123837082454959920938725447782268032766504892779847576081367881748
600487871716740257387500479155132510231551177257937569642525121186880656
97062939687782394984531853335512278823873665278645742731264

Teniendo los valores de e, n, y c se puede realizar un ataque de exponente bajo, el cual quedaría de la siguiente manera

```
import sys
try:
    import gmpy2
except ImportError:
    print("Install gmpy2 first to run this program")
    sys.exit()
```

n=3548570852663728357079604302372589381201155611964908273113078714717721
103798788580269734536344325766744392236763021310065826032298993843569791
939031029442597804169692367605142704300756013745047500127631840932695587
643322005997196969778632649565043163354828860491458218767080612770829980
026509870109098868762603198082319321573704115100663470529644919018030054
257283115767836135903201254808614607423979063600391158045013469954036198
610222325614801971211959942412011127617652666845636824654529215049120790
726965861181285380303607756827590198526457602891740831207810047206831728
151640049712745196099650879281018850331980999279630812011155773233282903
209145509125143111519198991785934980335568137130659554278237099925033495
337080283460071354073240709058978451709932296886843204277745880314243131
436233880996348134764511036323255810838545932441738425506879100966802258
405532199969238311795203131991093873872884551236213083417395040066858769
742832592233088642026249623966692442995248203405703660093573902432925915
195331110560502136042826389108413221687120295123639233382779380153168545
346223047960022358271541954693539516204845735727449836823608306886590756
067015582452580983535856617610328011724185438163747113739040419646096869
185071265971935844575219833870754955193082620044199646260052100098719775

```
440128567454795654348115130552024588363812257751225481699319728414593021
909298710538677752520858527174208559816545197407545293375909747205522249
663665817092459671818885568941516084763806466416538956854710191716367218
588892268183374520961876703874578298301450612423669992388236207370710508
987379344863477060226681021478165271921008190935948536416354078188395689
346291214758502373896310868662591310724883109915064581137755018210720885
752791447146466445898171299685277503596013583636123543518992461823844370
967911364266867277528929888295765522151733504382053
```

```
n=hex(n)
```

```
e=17
```

```
cipher_str=724068632839553579085240231923316016951374878018422694015004662
073243265833498546656183753898213763990796424012355795046715949531124551
436130371489003535273163405604849475955371584790628433449078458600650104
830886123571027528345408627723748737195150973560229328545874098692388693
894034226775310079936203615002267700677690775461666722217134156508587682
544995215333693610529996606603362053307659258857087914821149619137124965
994033731995462057914168349442861164830192887911769796524807841412085473
055274464441069768449836074316080610463465644991591443485475351099873632
800458696477642329394142656329059041263828144840420156717234648817824680
482270918628874518983858949906033725572414734035935504308856961427330330
873183128034650513036952764948815264600987093483359591886893124966349268
327852112059379115397904007041637699756805999460306233697984684156123819
048175026860933434935526942980852255345587320736773284034931156423068926
342671634457109746978846712600381324671164631697678107117654416995244917
184712667861035477421968586241231956640657103875021878005460899991222184
224058895618811893038698142242233496836197102154048748263192432424325059
502574077138889583607827484205875366332994450358074450527412126396479096
952335624012784525848410439722970866590732080253610791484637840103659810
040273351928432560804291260703610331762010115553723012005610917067142396
127596522217832215156245809192100763522722283987149394005418363385206468
238251276128365329664717103684540898307598058993820015578050884854491573
088685870773412383708245495992093872544778226803276650489277984757608136
788174860048787171674025738750047915513251023155117725793756964252512118
688065697062939687782394984531853335512278823873665278645742731264
```

```
import gmpy2
```

```
gs = gmpy2.mpz(cipher_str)
```

```
gm = gmpy2.mpz(n)
```

```
ge = gmpy2.mpz(e)
```

```
root, exact = gmpy2.iroot(gs, ge)
```

```
print (format(root, 'x'))
```


Esto nos regresara un número en hexadecimal el cual será la flag.

El número hexadecimal sería este

6861636b6465667b5331336d7072335f6d346e336a34725f337870306e336e7433735f346
c7430737d0a

Convirtiendo el texto del hexadecimal, la flag sería

hackdef{\$13mpr3_m4n3j4r_3xp0n3nt3s_4lt0s}

Crypto 300: fl1pp3r

Descripción del reto:

fl1pp3r (300pts) ✓🚩

Los delfines son animales muy inteligentes, o eso se dice, así que le pedimos a uno ayudarnos a implementar un sistema de tickets para facilitar la autenticación. Estamos empezando a dudar de su conocimiento en implementaciones criptográficas, pues varios usuarios se han quejado de alertas de inicio de sesión no reconocidas en sus cuentas.

Por alguna razón a todo nuevo usuario se le entrega, una llave "AAAAAAAAAAAAAAAA" a la que se le dice quizás incorrectamente "llave de inicialización", se cuenta con capacidad para 6 IDs de usuarios invitados (0-5), siendo el usuario con id 6 el administrador del sistema y con esto el que te dará la flag si te haces pasar por el ;-)


Nos preocupa que nuestro administrador del sistema pueda ser comprometido.

Puedes ayudarnos a descubrir el error en la implementación criptográfica? Te damos el código!

Y obtén la flag aquí:

IP: 3.19.72.219

Puerto: 3166

 **fl1pper.py** c4ffa22c147b5447e58cc66a654c40a6

Al analizar el programa encontramos que es un cifrado AES CBC, entonces buscamos el cómo funciona aquí

[https://es.wikipedia.org/wiki/Modos de operaci3n de una unidad de cifrado por bloques](https://es.wikipedia.org/wiki/Modos_de_operaci3n_de_una_unidad_de_cifrado_por_bloques)

La misma descripción nos comenta que el usuario con id 6 es el que nos dará la flag, siendo nosotros usuarios invitados solo podremos conseguir los id del 0 al 5.

Cuando nos conectamos al servidor este nos da un "token" y en la descripción tenemos la llave de inicialización, al meter estos datos nos regresará un id completamente random.

```
El token pertenece al usuario: fl1pid=4
Quieres seguir intentando? (S/N): s
```

Queremos que el 4 sea un 6, entonces el ataque que haremos será "Bit-Flipping Attack" con el cual modificaremos un bit de la llave de inicialización (vector de inicialización), para que el sistema nos regrese el id 6 y con esto obtener la flag.

f	l	i	p	i	d	=	4
1	2	3	4	5	6	7	8

“Bit-Flipping Attack” trata de modificar un byte de la cadena de inicialización, entonces modificaremos el byte número 8 de la cadena de inicialización para que nos regrese un 6.

A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
A	A	A	A	A	A	A		A	A	A	A	A	A	A	A

Al momento de enviar una I en la cadena de inicialización nos regresa un “;”

```
***Si tienes un token previo usalo para autenticarte a continuacion.
1) Ingresar
2) Salir
Eleccion: 1
Ingresa tu token: NLjfdtMwTFmhOjeHScn7nA==
Ingresa la llave de inicializacion: AAAAAAIAAAAAAA
El token pertenece al usuario: flipid=;
```

Buscando en la tabla ascii vemos que hay 5 posiciones del “;” y del “6”.

Si una “I” nos dio un “;” y queremos un 6 entonces debemos de enviar una letra que esta 5 posiciones arriba de la “I”, la cual es la “D”.

Enviamos la cadena de inicialización de la siguiente forma “AAAAAADAAAAAA” y obtenemos la flag.

```
dylan — nc 3.137.144.10 3166 — 80x24
~ — nc 3.137.144.10 3166
1) Ingresar
2) Salir
Eleccion: 1
Ingresa tu token: NLjfdtMwTFmhOjeHScn7nA==
Ingresa la llave de inicializacion: AAAAAADAAAAAA
El token pertenece al usuario: flipid=;
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: NLjfdtMwTFmhOjeHScn7nA==
Ingresa la llave de inicializacion: AAAAAADAAAAAA

Felicidades: hackdef{_nunC4_dej3s_3l_c0ntR0l_d3l_IV_a_uN_usuAr1o_qU3_3nt1end3_3L
_m0d0_d3_op3r4ci0n_CBC_}
```