

WriteUp CTF

Final JOINTS 2021



Walkie O Talkie

Sabtu, 24 April 2021

```
[ vibonacci    ]  
[ camellia    ]  
[ aclostael   ]
```

table of contents

table of contents	1
— Binary Exploitation	2
— PassVault	2
— Cryptography	4
— Here We AES Again	4
— ReSAh	7
— P & S	9
— Forensic	13
— memory	13
— Reverse Engineering	16
— Hidden	16
— PNGWARE	18

Binary Exploitation

PassVault

Diberikan sebuah binary dengan proteksi sebagai berikut:

```
[*] '/ctf/work/jointsfinal/passVault/PassVault'
Arch:      amd64-64-little
RELRO:     Partial RELRO
Stack:     Canary found
NX:        NX enabled
PIE:       PIE enabled
```

Di dalam fungsi edit_creds() terdapat vuln, yaitu bisa melakukan overwrite di index negatif.

```
v2 = __readfsqword(0x28u);
printf("Index : ");
__isoc99_scanf("%d", &v1);
getchar();
if ( v1 <= 9 && credentials[64 * (__int64)v1] )
{
    printf("New username : ");
    read_str(&credentials[64 * (__int64)v1], 16);
    printf("New password : ");
    read_str(&credentials[64 * (__int64)v1 + 16], 16);
    printf("Note : ");
    read_str(&credentials[64 * (__int64)v1 + 32], 32);
}
```

Dengan begitu, bisa dimanfaatkan untuk mengoverwrite got. Disini saya meng overwrite memset dengan puts. Sehingga ketika melakukan delete, bisa digunakan untuk melakukan leak libc dengan mengarahkannya di got.

```
printf("Index : ");
__isoc99_scanf("%d", &v1);
getchar();
if ( v1 > 9 )
    puts("Invalid index!");
else
    memset(&credentials[64 * (__int64)v1], 0, 0x40uLL);
```

Setelah mendapatkan address libc, overwrite memset lagi dengan system untuk mendapatkan shell.

```
[*] '/ctf/work/jointsfinal/passVault/libc.so'
Arch:      amd64-64-little
RELRO:     Partial RELRO
Stack:     Canary found
NX:        NX enabled
PIE:       PIE enabled
[+] Starting local process './PassVault': pid 6788
[!] ASLR is disabled!
[+] Opening connection to dubwesub.joints.id on port 51707: Done
[*] 0x7f5f95318000
[*] Switching to interactive mode
$ ls
PassVault
flag.txt
$ cat flag.txt
JOINTS21{Ch3cK_F0r_n36At1V3_Valu3}
$
```

Flag : JOINTS21{Ch3cK_F0r_n36At1V3_Valu3}

—Cryptography

||— Here We AES Again

Diberikan sebuah service dan source codenya. Pada service kita dapat melakukan enkripsi menggunakan **AES GCM**. Kemudian kita juga dapat melakukan login sebagai guest atau admin dengan verifikasi cipher-login + extra_enc yang dienkripsi menggunakan **CTR** dengan padding **crc**.

```
print(f"Here is the admin code : {admin_code.hex()}")
print(f"Here is the extra code for regular user : {regular_extra_code.hex()}")
print("Here, we implement double protection for admin\n")
print("""Menu :
1. Generate Encrypted Code
2. Enter as agent""")
```

Jika dilihat dari source code, seharusnya challenge ini memiliki 2 tahap untuk dilewati, yaitu enkripsi GCM ketika membuat token admin, dan verifikasi enkripsi CTR ketika login. Namun pada source code, kita melihat :

```
code = bytes.fromhex(code)

if code == admin_code:
    print("You can't generate encrypted admin code !")

enc_code, code_tag = encrypt(code, key, nonce)
print(f"Encrypted Code : {enc_code.hex()}")
print(f"Code Tag : {code_tag.hex()}")
```

Yang sepertinya probset tidak sedang baik baik saja , sehingga terlewat memasukkan kondisi lolos kedalam filter enkripsi dengan inputan token admin langsung.

Jadi, langsung saja kita ke tahap 2. Disini tujuan kita adlah mengubah json admin bernilai true pada extra enc yang ada, kemudian melakukan enkripsi dengan CTR.

```
key = os.urandom(16)
another_key = os.urandom(16)
another_another_key = os.urandom(16)
nonce = os.urandom(16)
admin_code = os.urandom(16)
regular_extra_code = gen_extra_code(json.dumps({"adm00n": 0}), another_another_key, another_key)
print(regular_extra_code)
```

Karena enkripsi hanya menggunakan CTR, dimana enkripsinya dilakukan per byte. Kita dapat mengganti index bytes pada nilai **admin 0 menjadi 1**, dengan mengubahnya menjadi **character+1**.

```
def gen_extra_code(json_str, key, aes_key):
    code = json_str.encode() + key
    code = code + struct.pack('<L', crc(code))
    aes_obj = AES.new(aes_key, AES.MODE_CTR, counter = Counter.new(128))
    return aes_obj.encrypt(code)
```

Lalu sebelum enkripsi, payload akan dipadding menggunakan crc. Namun karena padding crc hanya 4 bytes, kita dapat melakukan **bruteforce** untuk bypassnya. Yaitu pada **4 bytes terakhir** enc extra. Berikut full solver saya :

```
from Crypto.Util.number import *
from pwn import *

r = remote('dubwewsub.joints.id', 20001)

def kirim(payload):
    r.recvuntil(">" )
    r.sendline('2')
    r.recvuntil("ncrypted Code (in hex) : ")
    r.sendline(enc)
    r.recvuntil("Code Tag (in hex) : ")
    r.sendline(tag)
    r.recvuntil("Enter Extra Code : ")
    r.sendline(payload)
    result = r.recvline().strip()
    return result

def ganti(cipher):
    c = cipher.decode('hex')
    bitnya = ord(c[11:12])+1
    new = c[:11] + chr(bitnya) + c[12:]
    return new.encode('hex')

def ganti2(cipher, index, bitnya):
    c = cipher.decode('hex')
    new = c[:index] + chr(bitnya) + c[(index+1):]
    return new.encode('hex')

r.recvuntil("Here is the admin code : ")
admin = r.recvline().strip()
print(admin)
```

```

r.recvuntil("Here is the extra code for regular user : ")
extra = r.recvline().strip()
r.recvuntil("> ")
r.sendline('1')
r.recvuntil("Code (in hex) : ")
r.sendline(admin)
r.recvuntil("Encrypted Code : ")
enc = r.recvline().strip()
r.recvuntil("Code Tag : ")
tag = r.recvline().strip()

new_extra = ganti(extra)
index = [29,30,31,32]

while(1):
    for i in range(100):
        for j in range(100):
            for k in range(100):
                for l in range(100):
                    news = ganti2(new_extra,index[0],i)
                    news = ganti2(news,index[1],j)
                    news = ganti2(news,index[2],k)
                    news = ganti2(news,index[3],l)
                    print(kirim(news))

```

```

alvin@vibonacci /media/sf_Shared-Ubuntu
python solved.py
[+] Opening connection to dubwewsub.joints.id on port 20001: Done
5e007e37d2a6e6d607703f35c24f6535
Welcome admin, here is your flag : JOINTS21{Yea_its_Me_AeS_MaNIA}
[6] + 8994 suspended (signal) python solved.py

```

Flag : JOINTS21{Yea_its_Me_AeS_MaNIA}

Diberikan sebuah rsa dengan source enkripsi sebagai berikut :

```
1 #!/usr/bin/env python3
2
3 from Crypto.Util.number import *
4
5 flag = open("flag.txt").read().strip()
6 m1 = flag[:len(flag)//2].encode()
7 m2 = flag[len(flag)//2:].encode()
8
9
10 def encrypt_1(bits, m):
11     m = bytes_to_long(m)
12     p = getPrime(bits)
13     q = getPrime(bits)
14     n = p*q
15     c = pow(n+1, m, pow(n, getPrime(7) + 69))
16     return c, n
17
18 def encrypt_2(bits, m):
19     while 0x1337:
20         x = getRandomInteger(bits) << 2
21         p, q = pow(x, 3) + x + 1, pow(x, 2) + 3*x + 1
22         if isPrime(p) and isPrime(q):
23             break
24
25     n = pow(x, 8) + 3*pow(x, 7) + 3*pow(x, 6) + 8*pow(x, 5) + 9*pow(x, 4) + 7*pow(x, 3) + 8*pow(x, 2) + 5*x + 1
26     m = bytes_to_long(m)
27     c = pow(m, n, n)
28
29     return c, n
30
31
32 c1, n1 = encrypt_1(512, m1)
33 c2, n2 = encrypt_2(512, m2)
```

Disini flag dipecah menjadi 2 dan dienkripsi menggunakan 2 metode yang berbeda.

Pertama saya akan membahas mengenai **enkripsi mode 2**. Enkripsi ini sebenarnya cukup sederhana dengan hanya memanfaatkan *modular equations* . Untuk nilai n yang diketahui, kita dapat me recover nilai x pada persamaan di fungsi dengan menggunakan bantuan **solve()** sage tentu saja :).

Setelahnya kita hanya perlu mencari beberapa jurnal dan writeup untuk mereferensikan bagaimana **math** disini diimplementasikan untuk dekrip RSA nya.

Part1.sage

```
from Crypto.Util.number import *
c2 =
115442962284578820348962434955742551107120860834349539667118344161634002040
279672397823415915711888401228343614770732302376630907550602955509671173231
31914505218912940....
n2 =
146181803944171432081262768059014184074006446234359527023675400687177493623
530612781306199113661729481287342716457444263776820426875353943291430549253
7169845205305544385.....
```



```
#Sage Part :
x = var('a')
print(solve([ pow(x, 8) + 3*pow(x, 7) + 3*pow(x, 6) + 8*pow(x, 5) +
9*pow(x, 4) + 7*pow(x, 3) + 8*pow(x, 2) + 5*x + 1 == n2], x))

x =
442192797673300571985578863084456624314811044094134130206047154028185943398
014699155003939689047726519739456305738632474006017248704487471541271576550
46248
p, q = pow(x, 3) + x + 1, pow(x, 2) + 3*x + 1
phi = (p-1)*(q-1)
d = inverse(n2,phi)
flag1=(long_to_bytes(pow(c2,d,p*q)))
```

Pada bagian 2, saya mengambil [referensi](https://s0uthwood.github.io/2020/12/09/not-RSA-WriteUp/) penyelesaian dari writeup berikut :

<https://s0uthwood.github.io/2020/12/09/not-RSA-WriteUp/> .

Jujur saya tidak mengerti mengapa penjabarannya demikian, namun karena flag berhasil di dekrip menggunakan arahan di writeup tersebut, yasudah kenapa tidak :)

Part1.py

```
c1 =
258085143787875507085833502240361486236271577653872590287892487687706747144
0174320794087919431....
n1 =
153853542672109158010590338332080477613396347390252124678950204761153212961
847364315678518727....

fn = c1 - 1
assert fn % n1 == 0
m = fn // n1
m = m % n1
flag2=(long_to_bytes(m))

print(flag2+flag1)
```

```
alvin@vibonacci: /media/sf_Shared-Ubuntu
python rsasolp.py
JOINTS21{Rsa_rs4_r5a_wh4t_is_that_thlng_actually?????}
alvin@vibonacci: /media/sf_Shared-Ubuntu
```

||— P & S

Diberikan kembali sebuah service dan source code didalamnya. Pada soal ini kita dibebaskan untuk melakukan **enkripsi flag** maupun pada **chosen plaintext** sebanyak yang kita mau. Alur enkripsi cukup sederhana,

```
var = flag
if choice == "2":
    var = input("Enter Plaintext : ")
    if len(var) != len(flag):
        print("The length of the plaintext must be same with the flag")
        break
for n in range(amount):
    cipher = ""
    for indeks in range(len(var)):
        cipher += chr(ord(var[indeks]) ^ random.choice(RANDOM_BOX[indeks]))
    arr_cipher.append(cipher.encode("latin-1").hex())
print(f"Cipher : {arr_cipher}")
```

Namun **tidak sederhana untuk dieksploitasi** :).

Setiap xor yang dilakukan benar benar random tanpa mengetahui urutan generate, isi random atau apapun yang mengarahkan kita untuk melakukan predict next random. Namun berikut **vuln** yang kami temukan :

1. Terdapat sebuah **9 identik karakter** yang disematkan pada 555 pilihan random untuk **indeks** yang kita **pilih**. Dan kita mengatur hal ini akan ada di indeks ke berapa.

```
try:
    ind_secret = int(input("Input Index for Secret : "))
    assert ind_secret in range(len(flag))
except:
    print("Invalid Index !")
    sys.exit(1)

for i in range(len(flag)):
    temp = [random.randint(0,255) for x in range(555)]
    if i == ind_secret:
        temp = temp[:-9] + secret
    RANDOM_BOX.append(temp)
```

2. Kita dapat dengan bebas melakukan enkripsi dengan percobaan **sebanyak** apapun baik untuk flag maupun chosen plaintext.

```
choice = input("How Many Times ? : ")
if choice == "1" or choice == "2":
    try:
        amount = int(input("How Many Times ? : "))
        if amount > 5000 or amount < 1:
            sys.exit(1)
    except:
        print("Something Error !")
        sys.exit(1)
```

Maka, disini saya mencoba menerapkan **frequent analysis** untuk solver nya.

Kita mengingat bahwa ada 9 identik karakter yang sudah pasti ada dari 555 list random, dengan metode generate random untuk lainnya, maka tentu jumlah ini kemungkinan besar akan bertambah. Dengan **probabilitas** yang lebih besar dari pada kejadian lainnya, apabila kita membesarkan **ruang sample** percobaan, maka tentu peluang kejadian semakin besar dan perbandingannya semakin jauh dibandingkan kejadian munculnya karakter lain. Sehingga sudah jelas dan pasti ada 1 karakter yang paling sering muncul, kita sebut sebagai **most_common_character**.

- Untuk enkripsi menggunakan **flag**, tentu **most_common_character** akan bernilai **most_common_key ^ flag[index_set_posisi_identik_karakter_yg_dipilih]**.
- Untuk enkripsi menggunakan **chosen plaintext**, jika kita meluncurkan payload **'\x00'*len(flag)**, **most_common_character** akan bernilai **most_common_key** saja.
- Jika kita melakukan xor kedua common_character tersebut, maka :

most_common_key ^ flag[index_set_posisi_identik_karakter_yg_dipilih] ^
bernilai most_common_key =
flag[index_set_posisi_identik_karakter_yg_dipilih] .

**BOOM!! Frequent analysis goes brrr*

Maka, langsung saja kita eksekusi menggunakan ful solver di bawah ini :

```
import collections
from Crypto.Util.number import *
from pwn import *

def enc(payload):
    r.recvuntil(">>> ")
    r.sendline('2')
    r.recvuntil("How Many Times ? : ")
    r.sendline('4999')
    r.recvuntil("Enter Plaintext : ")
    r.sendline(payload)
    r.recvuntil("Cipher : ")
    hasil= eval(r.recvline().strip())
    return hasil

def enc_flag():
    r.recvuntil(">>> ")
    r.sendline('1')
    r.recvuntil("How Many Times ? : ")
    r.sendline('4999')
    r.recvuntil("Cipher : ")
    hasil= eval(r.recvline().strip())
    return hasil

def get_key_flag():
    arr = []
    for i in range(30):
        l = enc_flag()
        arr += [i.decode('hex') for i in l]
    char_on_index = ''
    for j in range(len(arr)):
        char_on_index += arr[j][cobs]
    key_flag = collections.Counter(char_on_index).most_common(1)[0][0]
    return key_flag

def get_only_key():
    arr = []
    for i in range(30):
        l = enc('\x00'*26)
        arr += [i.decode('hex') for i in l]
    char_on_index = ''
```

```

    for j in range(len(arr)):
        char_on_index += arr[j][cobs]
    only_key = collections.Counter(char_on_index).most_common(1)[0][0]
    return only_key

flag = 'JOINTS21{'
for cobs in range(len(flag),26):
    r = remote('dubwewsub.joints.id', 20000)
    r.recvuntil("Input Index for Secret : ")
    r.sendline(str(cobs))
    lets = get_key_flag()
    go = get_only_key()
    flag += xor(lets,go)
    print(flag)
    r.close()

```

```

flag JOINTS21{jUsT_m05t_C0mM0n}
[*] Closed connection to dubwewsub.joints.id port 20000
alvin@vibonacci /media/sf_Shared-Ubuntu
python

```

Nice challenge anw :) .

Flag : JOINTS21{jUsT_m05t_C0mM0n}

Crypto

Here We AES Again ✓

462

P & S ✓

499

ReSAh ✓

500

Forensic

memory

Diberikan sebuah file zip yang berisi memory dump dengan profile Win7SP1x86_23418. Analisis dengan filescan, pslist, dan iehistory memberikan hasil menarik.

```
dhipz@vibonacci ~/Downloads/joints python2 ../volatility/vol.py -f memory.dmp --profile=Win7SP1x86_23418 iehistory
Volatility Foundation Volatility Framework 2.6.1
*****
Process: 1472 explorer.exe
Cache type "URL " at 0x615000
Record length: 0x100
Location: :2021040420210405: joints@file:///C:/Users/joints/Desktop/SECRET.txt
Last modified: 2021-04-04 12:40:03 UTC+0000
Last accessed: 2021-04-04 05:40:03 UTC+0000
File Offset: 0x100, Data Offset: 0x0, Data Length: 0x0
*****
Process: 1472 explorer.exe
Cache type "URL " at 0x615100
Record length: 0x100
Location: :2021040420210405: joints@Host: Computer
Last modified: 2021-04-04 12:10:43 UTC+0000
Last accessed: 2021-04-04 05:10:43 UTC+0000
File Offset: 0x100, Data Offset: 0x0, Data Length: 0x0
*****
Process: 1472 explorer.exe
Cache type "URL " at 0x615200
Record length: 0x100
Location: :2021040420210405: joints@file:///C:/Users/joints/Downloads/P4LrAZBlkGk0spBidT69cgR1Wno4910V8EM9aThd.zip
Last modified: 2021-04-04 20:25:38 UTC+0000
Last accessed: 2021-04-04 13:25:38 UTC+0000
File Offset: 0x100, Data Offset: 0x0, Data Length: 0x0
*****
Process: 1472 explorer.exe
```

0x8594c770	dllhost.exe	1968	492	14	193	0	0	2021-04-04 05:10:29 UTC+0000	
0x86caf030	msdtc.exe	2144	492	12	145	0	0	2021-04-04 05:10:30 UTC+0000	
0x86573d40	svchost.exe	3864	492	11	321	0	0	2021-04-04 05:21:34 UTC+0000	
0x85add590	svchost.exe	3888	492	12	156	0	0	2021-04-04 05:21:34 UTC+0000	
0x85375030	firefox.exe	2980	3776	0	-----	1	0	2021-04-04 13:28:00 UTC+0000	2021-04-04 13:28:23 UTC+0000
0x86461030	audiodg.exe	3636	792	5	127	0	0	2021-04-04 13:33:56 UTC+0000	
0x86b01c90	MpCmdRun.exe	2104	3864	7	124	0	0	2021-04-04 13:34:52 UTC+0000	
0x8536f030	conhost.exe	3744	336	2	31	0	0	2021-04-04 13:34:52 UTC+0000	
0x869d1a78	mobsync.exe	1996	600	7	154	1	0	2021-04-04 13:34:54 UTC+0000	
0x8561ad40	SearchProtocol	2404	1696	8	279	0	0	2021-04-04 13:34:58 UTC+0000	
0x86497030	SearchFilterHo	3836	1696	5	97	0	0	2021-04-04 13:34:58 UTC+0000	
0x86506d40	cmd.exe	1428	1936	0	-----	0	0	2021-04-04 13:35:07 UTC+0000	2021-04-04 13:35:07 UTC+0000

```
dhipz@vibonacci ~/Downloads/joints python2 ../volatility/vol.py -f memory.dmp --profile=Win7SP1x86_23418 filescan | grep "\\Desktop"
Volatility Foundation Volatility Framework 2.6.1
0x000000003e024428 2 0 RW-r-- \Device\HarddiskVolume1\Users\joints\Desktop\RANDOM.txt
0x000000003e028770 2 0 RW-r-- \Device\HarddiskVolume1\Users\joints\Desktop\RANDOM.txt
0x000000003e05fd30 8 0 R--rwd \Device\HarddiskVolume1\ProgramData\Microsoft\Windows\Start Menu\Programs\Accessories\Remote Desktop Connecti
on.lnk
0x000000003e09dc30 2 0 RW-r-- \Device\HarddiskVolume1\Users\joints\Desktop\RANDOM.txt
0x000000003e0b1598 2 0 RW-r-- \Device\HarddiskVolume1\Users\joints\Desktop\RANDOM.txt
0x000000003e0bc0e8 2 0 RW-r-- \Device\HarddiskVolume1\Users\joints\Desktop\RANDOM.txt
0x000000003e0cb868 2 0 RW-r-- \Device\HarddiskVolume1\Users\joints\Desktop\SECRET.txt
0x000000003e0cc290 2 0 RW-r-- \Device\HarddiskVolume1\Users\joints\Desktop\RANDOM.txt
0x000000003e2abb30 2 1 R--rwd \Device\HarddiskVolume1\Users\Public\Desktop
0x000000003e2c8038 2 0 RW-r-- \Device\HarddiskVolume1\Users\joints\Desktop\RANDOM.txt
0x000000003e306498 8 0 R--rwd \Device\HarddiskVolume1\Users\joints\Desktop\desktop.ini
0x000000003e31e888 8 0 R--rwd \Device\HarddiskVolume1\Users\Public\Desktop\TeamViewer.lnk
0x000000003e320920 8 0 R--rwd \Device\HarddiskVolume1\Users\joints\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Accessories\Access
ibility\Desktop.ini
0x000000003e327e48 8 0 R--rwd \Device\HarddiskVolume1\Users\joints\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Accessories\System
Tools\Desktop.ini
0x000000003e328c28 8 0 R--rwd \Device\HarddiskVolume1\ProgramData\Microsoft\Windows\Start Menu\Programs\Accessories\Accessibility\Desktop.i
ni
0x000000003e329c00 8 0 R--rwd \Device\HarddiskVolume1\ProgramData\Microsoft\Windows\Start Menu\Programs\Accessories\Desktop.ini
0x000000003e32a5a8 8 0 R--rwd \Device\HarddiskVolume1\ProgramData\Microsoft\Windows\Start Menu\Programs\Maintenance\Desktop.ini
0x000000003e32ac90 8 0 R--rwd \Device\HarddiskVolume1\Users\Public\Desktop\Firefox.lnk
0x000000003e333df0 8 0 R--rwd \Device\HarddiskVolume1\ProgramData\Microsoft\Windows\Start Menu\Programs\Games\Desktop.ini
0x000000003e33a908 8 0 R--rwd \Device\HarddiskVolume1\ProgramData\Microsoft\Windows\Start Menu\Programs\Accessories\System Tools\Desktop.in
i
```

Coba coba extract process firefox.exe, didapatkan secret.zip berisi secret.txt yang diberi password setelah dijalankan foremost.

```
dh1pz@vibonacci: ~/Downloads/joints$ python2 ./volatility/vol.py -f memory.dmp --profile=Win7SP1x86_23418 memdump --dump-dir=dump -p 2980
Volatility Foundation Volatility Framework 2.6.1
*****
Writing firefox.exe [ 2980] to 2980.dmp
dh1pz@vibonacci: ~/Downloads/joints$ foremost dump/2980.dmp
Processing: dump/2980.dmp
|foundat=secret/PK
|foundat=secret/secret.txt0
*foundat=manifest.json{
  "manifest version": 2,
  "name": "DoH Roll-Out",
  "description": "This used to be a Mozilla add-on that supported the roll-out of DoH, but now only exists as a stub to enable migrations.",
  "version": "2.0.0",

  "hidden": true,

  "applications": {
    "gecko": {
      "id": "doh-rollout@mozilla.org",
      "strict_min_version": "72.0a1"
    }
  }
}
PK
*|
```

Di direktori Desktop terdapat banyak file RANDOM.txt dan SECRET.txt yang berisi seperti potongan text base32. Selain itu ada beberapa file diantaranya yang merupakan duplikat. Karena kita dapat file secret.zip maka dicoba untuk memproses file file SECRET.txt saja. Karena terdapat banyak file yang berisi text singkat maka terpikir bahwa isi setiap file merupakan sebuah potongan dari satu text base32 sehingga cukup bruteforce saja sampai dapat.

```
# isi file SECRET.txt yang unik
text =
["FPEUJKRNBXW", "KKN52GI3KIH", "WORDCGFWEYT", "KU4TINJDKJB", "GOJGJQ4GGMRE"]

import base64
from itertools import permutations

perm = permutations(text, len(text))
for i in list(perm):
    print(str(base64.b32decode("".join(i))))
```

Output :

```
...
b'U945#RCgDb11LL\xaf%\x12\xa8\xb47\xb2\x94\xde\xe8\xc8\xda\x90s9&L8c2$'

b'U945#RCgDb11LL\xaf%\x12\xa8\xb47\xb1\x9c\x93&\x1c1\x99\x12)M\xee\x8c\x8d\
xa9\x07'
```

```
b'U945#RCgDb11LMJotdmH9^J%Qhoc9&L8c2$'
```

```
b'U945#RCgDb11LMJotdmH9\x9c\x93&\x1c1\x99\x12\x15\xe4\xa2U\x16\x86\xf6'
```

```
b'U945#RCgDb11LL\xceI\x93\x0e\x18\xcc\x89\n\xef2Q*\x8bC{)M\xee\x8c\x8d\xa9\x07'
```

```
b'U945#RCgDb11LL\xceI\x93\x0e\x18\xcc\x89\x14\xa6\xf7FF\xd4\x83\x95\xe4\xa2U\x16\x86\xf6'
```

```
...
```

String yang didapat merupakan password dari secret.zip yang berisi flagnya.

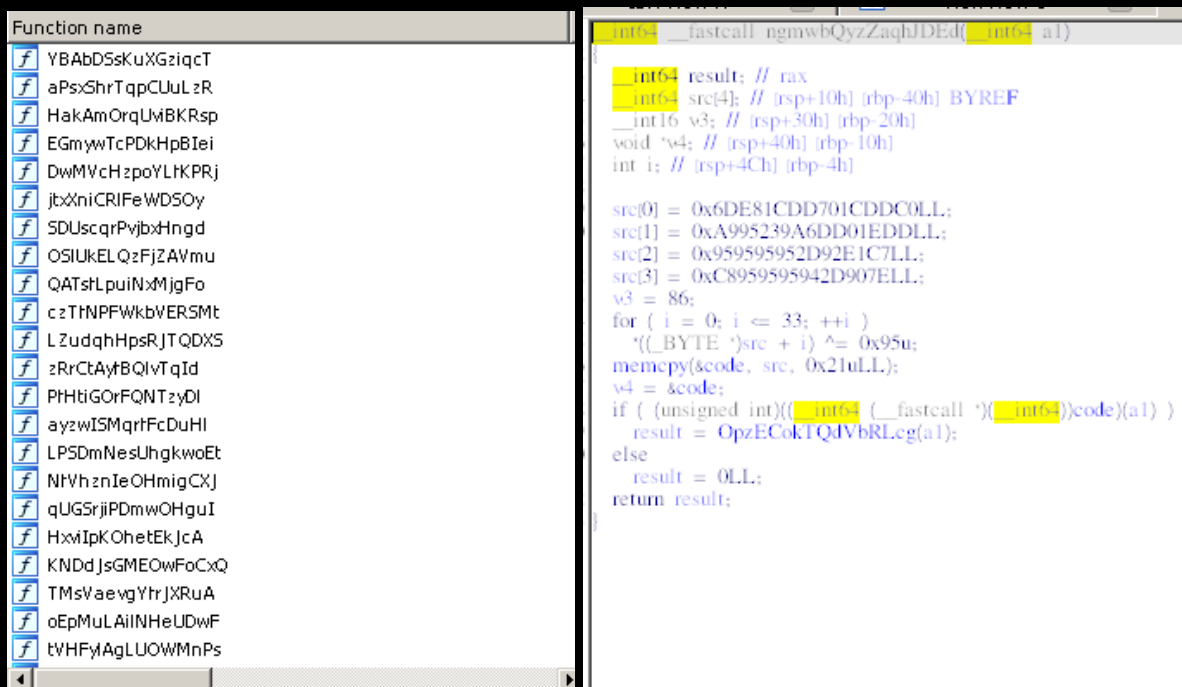
Flag : JOINTS21{cr3at3_a_m3mdump_th3y_5a1d_it_w1ll_b3_fun_th3y_5a1d}

Reverse Engineering

Hidden

Diberikan sebuah file binary ELF 64-bit. Ketika file dijalankan, file meminta inputan untuk dilakukan pengecekan. Ketika didecompile menggunakan IDA, kita melihat bahwa input di cek melalui kurang lebih 60an fungsi yang berbeda, satu per satu.

Pada setiap fungsi, terdapat variabel variabel yang berbeda yang akan dieksekusi dengan operasi yang berbeda beda juga tiap fungsinya. Hingga diakhir akan dilakukan pengecekan per byte untuk hasil operasi dan paramter fungsi yang dikirim dari fungsi sebelumnya.



The screenshot shows the IDA Pro interface. On the left, a list of functions is displayed under the 'Function name' header. The functions listed include: YBAbDSsKuXGziqcT, aPsxShrTqpCUuLzR, HakAmOrqUwIBKRsp, EGmywTcPDkHpBiei, DwMVcHzpoYLHKPRj, jtxXniCRIFeWDSOy, SDUscqrPvjbxHngd, OSIUKELQzFjZAVmu, QATstLpuINxMjgFo, czTINPFwkbVERSmt, LZudqhHpsRJTDQXS, zRrCtAytBQlvTqId, PHtiGOrFQNTzyDI, ayzwISMqrFfcDuHI, LP5DmNesUhgkwoEt, NfVhznIeOHmigCXJ, qUGSrjiPDmwOHguI, HxviIpKOhetEkJcA, KNDdJsGMEOWFoCxQ, TMsVaevgYtrjXRuA, oEpMuLAiINHeUDwF, and tVHFyAgLUOWMnPs. On the right, the decompiled code for the function 'fastcall ngmwbQyzZaqhJDEd' is shown. The code includes variable declarations for 'result', 'src', 'v3', 'v4', and 'i', followed by a loop that iterates from 0 to 33, performing a bitwise XOR operation on 'src' and a memory copy operation. The function returns 'result'.

Dikarenakan fungsi tersebut sangat banyak , kami mencoba mencari cara cepatnya namun sedang tidak dapat memikirkannya, sehingga memutuskan untuk melakukannya dengan cara manual dengan mengecek satu-satu pada setiap fungsi :).

Kami melakukannya dengan menggunakan gdb untuk mengetahui instruksi apa yang sedang dilakukan.

```

pwndbg> disassemble &code
Dump of assembler code for function code:
0x00000000407080 <+0>:  push    rbp
0x00000000407081 <+1>:  mov     rbp,rsp
0x00000000407084 <+4>:  mov     QWORD PTR [rbp-0x8],rdi
0x00000000407088 <+8>:  mov     rax,QWORD PTR [rbp-0x8]
0x0000000040708c <+12>: add     rax,0x28
0x00000000407090 <+16>: movzx   eax,BYTE PTR [rax]
0x00000000407093 <+19>: cmp     al,0x4e
0x00000000407095 <+21>: je      0x40709e <code+30>
0x00000000407097 <+23>: mov     eax,0x0
0x0000000040709c <+28>: jmp     0x4070a3 <code+35>
0x0000000040709e <+30>: mov     eax,0x1
0x000000004070a3 <+35>: pop     rbp
0x000000004070a4 <+36>: ret
0x000000004070a5 <+37>: add     BYTE PTR [rax],al
0x000000004070a7 <+39>: add     BYTE PTR [rax],al
0x000000004070a9 <+41>: add     BYTE PTR [rax],al
0x000000004070ab <+43>: add     BYTE PTR [rax],al

```

```

R   e   s   i   d   e   n   t   S
0x52, 0x65, 0x73, 0x69, 0x64, 0x65, 0x6e, 116, 83,
1, 2, 3, 4, 5, 6, 7, 8, 9,

l, e, e, p, e, r, -, E, Z, -, C
108, 0x65, 0x65, 112, 101, 114, 95, 0x45, 90, 95, 63
10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20

l, a, p, -, T, r, i, H, a, r
0x6c, 0x61, 112, 0x5f, -, 0x72, 0x69, 0x48, 0x61, 0x72
21, 22, 23, 24, 25, 26, 27, 28, 29, 30

d, -, H, A, C, K, E, R, M, A, N
0x64, 95, 0x48, 0x41, 67, 0x4b, 0x45, 0x52, -, 0x41,
31, 32, 33, 34, 35, 36, 37, 38, 39, 40,

S, -, N, o, t, L, i, k, e, T, h, i, s, -,
41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54,

B, a, b, y, R, a, g, e
55, 56, 57, 58, 59, 60, 61, 62

```

*kami sempat panik karena sisa waktu sedikit, dan pengerjaan manual sangat memakan waktu yang lama. Flag yang sudah didapatkan pada menit menit akhir masih wrong answer, sehingga ketiga anggota kami membrute-force-guessing flag yang sudah didapatkan bersama sama pada menit menit akhir :) alhamdulillah solved pada akhirnya setelah sekian banyak percobaan.

```

root@pwning:/ctf/work/jointsfinal/hidden# ./hidden
JOINTS21{ResidentSleeper_EZ_Clap_TriHard_HACKERMANS_NotLikeThis_BabyRage}
Nice! JOINTS21{ResidentSleeper_EZ_Clap_TriHard_HACKERMANS_NotLikeThis_BabyRage}
root@pwning:/ctf/work/jointsfinal/hidden#

```

Flag :
JOINTS21{ResidentSleeper_EZ_Clap_TriHard_HACKERMANS_NotLikeThis_BabyRage}

||—PNGWARE

Diberikan binary executable windows yang mengubah string yang terdapat di flag.txt dan menyembunyikannya di dalam gambar flag.txt.png. Kami mencoba Strings binarynya dan menemukan tulisan py &python38.dll, yang artinya kemungkinan dipack dengan py2exe.

Untuk meng-unpack bytecode python kami menggunakan <https://github.com/extremecoders-re/pyinstxtractor>, setelah diunpack terdapat file pngware.pyc.

```
ms-win-core-debug-l1-1-0.dll      _bz2.pyd
-ms-win-core-errorhandling-l1-1-0.dll  _ctypes.pyd
-ms-win-core-file-l1-1-0.dll        _decimal.pyd
-ms-win-core-file-l1-2-0.dll        _elementtree.pyd
-ms-win-core-file-l2-1-0.dll        _hashlib.pyd
-ms-win-core-handle-l1-1-0.dll      Include/
-ms-win-core-heap-l1-1-0.dll        libcrypto-1_1.dll
-ms-win-core-interlocked-l1-1-0.dll  libffi-7.dll
-ms-win-core-libraryloader-l1-1-0.dll  libssl-1_1.dll
-ms-win-core-localization-l1-2-0.dll  _lzma.pyd
-ms-win-core-memory-l1-1-0.dll      _multiprocessing.pyd
-ms-win-core-namedpipe-l1-1-0.dll    _overlapped.pyd
-ms-win-core-processenvironment-l1-1-0.dll  PIL/
-ms-win-core-processthreads-l1-1-0.dll  pngware.exe.manife
-ms-win-core-processthreads-l1-1-1.dll  pngware.pyc
-ms-win-core-profile-l1-1-0.dll      pyexpat.pyd
-ms-win-core-rtlsupport-l1-1-0.dll    pyiboot01_bootstrap
-ms-win-core-string-l1-1-0.dll      pyimod01_os_path.p
-ms-win-core-synch-l1-1-0.dll      pyimod02_archive.p
-ms-win-core-synch-l1-2-0.dll
lore--^C
```

Setelah itu saya menggunakan uncompyle6 untuk decompile pngware.pyc nya. Berikut sourcodenya

```
# uncompyle6 version 3.7.4
# Python bytecode 3.8 (3413)
# Decompiled from: Python 3.8.5 (default, Jan 27 2021, 15:41:15)
# [GCC 9.3.0]
# Embedded file name: pngware.py
from PIL import ImageDraw
from PIL import Image
import random
```

```

def llhohlo(111111111):
    lul = 256
    bruh = dict(((chr(i), chr(i)) for i in range(lul)))
    brah = ''
    yes = []
    for yooyo in 111111111:
        moo = brah + yooyo
        if moo not in bruh:
            yes.append(bruh[brah])
            bruh[moo] = lul
            lul += 1
            brah = yooyo
        else:
            brah = moo
        else:
            if brah:
                yes.append(bruh[brah])
    return yes

W, H = (256, 256)
new = Image.new('RGB', (W, H), (128, 0, 0))
d = ImageDraw.Draw(new)
flag = 'flag.txt'
w, h = d.textsize(flag)
d.text(((W - w) / 2, (H - h) / 2), flag, fill='black')
y = new.load()
x = open(flag).read()
tes = llhohlo(x)
b_message = ''.join([format(ord(i) if type(i) == str else i, '012b') for i
in tes])
index = 0

for p in range(new.size[0]):
    for q in range(new.size[1]):
        val = []
        for i in range(3):
            if index >= len(b_message):
                val.append(y[(p, q)][i])
            else:
                val.append(y[(p, q)][i] | int(b_message[index]))
                index += 1

```

```

        else:
            y[(p, q)] = tuple(val)

else:
    new.save(flag + '.png', 'PNG')
# okay decompiling pngware.pyc

```

Intinya, isi dari flag.txt akan dicompress dengan menggunakan lzw encoding lalu di sisipkan pada gambar menggunakan teknik LSB.

Untuk decompress dan mendapatkan kembali isi flag.txt, berikut full solver yang kami gunakan :

```

from PIL import Image

def decompress(compressed):
    dict_size = 256
    dictionary = dict((chr(i), chr(i)) for i in xrange(dict_size))
    w = result = compressed.pop(0)
    for k in compressed:
        if k in dictionary:
            entry = dictionary[k]
        elif k == dict_size:
            entry = w + w[0]
        else:
            raise ValueError('Bad compressed k: %s' % k)
        result += entry
        dictionary[dict_size] = w + entry[0]
        dict_size += 1
        w = entry
    return result

im = Image.open('flag.txt.png')
flag = ""
for i in range(im.size[0]):
    for j in range(im.size[1]):
        r,g,b = im.getpixel((i,j))
        flag += str(r & 1) + str(g) + str(b)

data = map(''.join, zip(*[iter(flag)]*12))
data2 = []

```

```

for i in data:
    if int(i, 2) < 256:
        data2.append(chr(int(i, 2)))
    else:
        data2.append(int(i, 2))

print decompress(data2)

```

The least significant bits (plural) are the bits of the number closest to zero. They have the useful property of changing rapidly if the number changes. For example, if the number 3 (binary 00000011), the result will be 4 (binary 00000100) and 8 (binary 00001000). By contrast, the three most significant bits (MSBs) stay unchanged. The least significant bits are frequently employed in pseudorandom number generators.

LempelZivWelch (LZW) is a universal lossless data compression algorithm. It was published by Welch in 1984 as an improved implementation of the LZ77 algorithm. The algorithm is simple to implement and has the potential for very high compression ratios. It is the algorithm of the widely used Unix file compression utility compress and its successor gzip.

Tomorrow X Together, commonly known as TXT, is a five-member South Korean boy band. The group consists of five members Soobin, Yeonjun, Beomgyu, Taehyun and HueningKai.

Portable Network Graphic is a raster-graphics file format that supports lossy and lossless compression. It is a proposed, non-patented replacement for Graphics Interchange Format (GIF).

```
JOINTS21{TXT_LZW_LSB_PNG}
```

```
root@pwning:/ctf/work/jointsfinal/PNGWARE#
```

Flag : JOINTS21{TXT_LZW_LSB_PNG}

Bonus :

Final Feedback For Us

Yey. JOINTS21{Bababoey_semangat_finalnya_canda_final_xixixi}

[Kirim jawaban lain](#)

Terima Kasih