此lab针对的是原书3.10.3和3.10.4的内容，利用缓冲区溢出改变原系统的执行路径，实现系统入侵，有点黑客的意思了。通过这个lab可以对系统漏洞有更直观的认识，并且了解系统入侵的基本思想和思路，也能对程序的机器级表示有更加深入的理解。

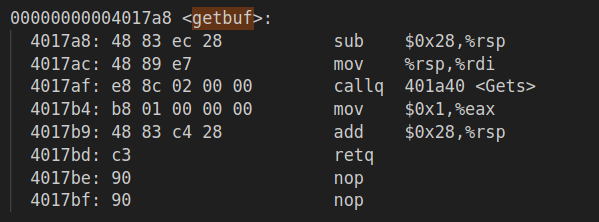
Phase1

通过插入字符串，让getbuf函数返回后执行touch1函数。这个比较简单，我们知道函数返回会从栈中弹出返回后下一条指令的地址，而且这个地址是调用该函数时压入栈的，在该函数调用栈的最顶端，所以我们只要改变这个地址为touch1函数的入口地址即可。

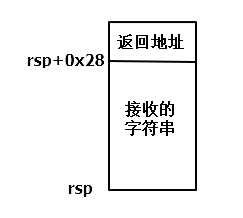
1. 反汇编ctarget

objdump -d ctarget > ctarget.d

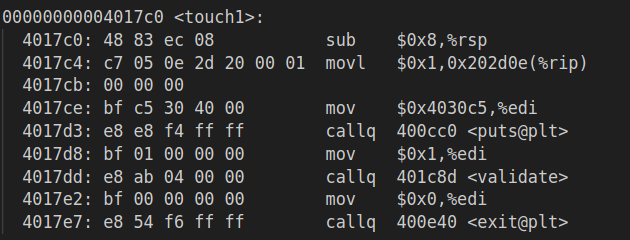
1. 找到getbuf函数



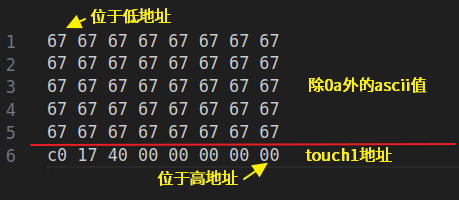
它将栈指针减了0x28（40），第2、3条表示将栈指针作为参数调用Gets函数，将得到的字符串放到栈上，由于只有0x28大小，所以超过这个数就会溢出，紧挨着的就是返回地址，所以只要在刚好溢出的地方放touch1的地址就可以。



touch1的地址是0x4017c0，

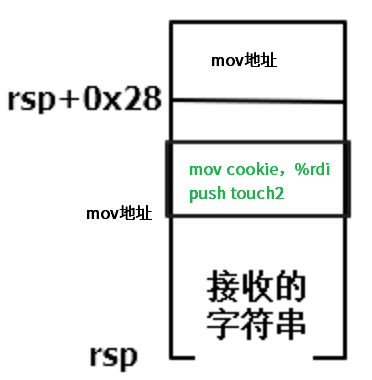


前0x28个字符可以是任意普通字符（不能是0a），最后8字节填入c0 17 40 00 00 00 00 00即可（实测高32位的0不填也行）。注意字节顺序，栈是从低到高存的，所以touch1地址也要从低到高排列。

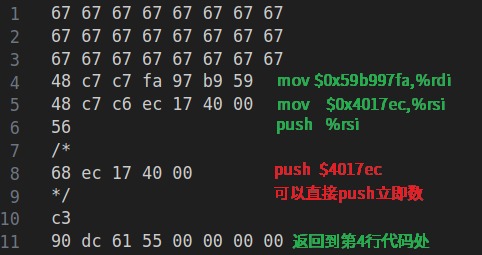


Phase2

通过插入字符串，让getbuf函数返回后执行touch2函数，并以cookie值作为参数。此题相对1多个参数传入，传入参数也就是将cookie值放到%rdi寄存器中，所以这里需要一条mov指令，所以整体思路是getbuf返回地址是栈上的某个地址，这个地址的内容是mov指令，将cookie值放到%rdi中，然后将touch2的地址push入栈，再ret返回到touch2函数。（push将值放入栈顶，ret将栈顶弹出作为返回地址）



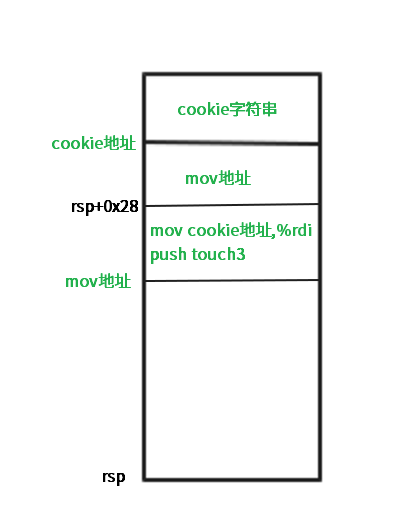
指令的机器码可以通过gcc -c exploit.s先编译为exploit.o，再用objdump -d exploit.o反汇编得到。



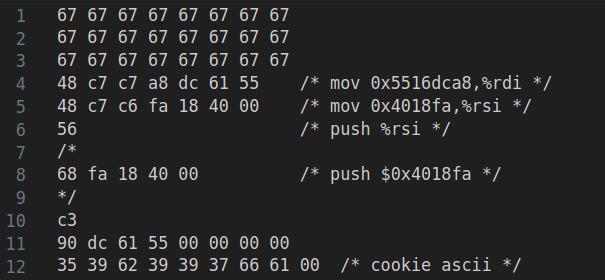
注释部分是另一种实现，可以取代5、6两行的指令。

Phase3

同Phase2类似，只是这次传入的参数是字符串指针，cookie作为字符串。此题的难点在于字符串放的位置，因为touch3在使用cookie前会调用其他函数，这个过程会对栈产生影响，有可能把字符串覆盖掉，所以字符串放的位置要注意。因为栈是向低地址生长的，所以把字符串放在高地址处，也就是getbuf返回地址的栈区之上。



其他跟2一样，只是mov的不是cookie了，而是cookie字符串的地址。



Phase4

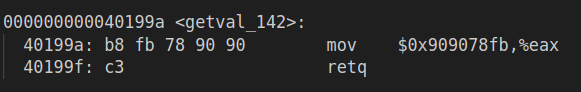
任务跟Phase2一样，只是这次的目标文件是rtarget，它不仅实现了栈随机化，而且代码不能在栈区运行，所以之前的方法就失效了。这次用的方法是Return-Oriented Programming（面向返回编程），利用程序已有的代码实现自己的目的。

我们的思路可以有多种，比如

1. 把cookie的值放到栈上，pop到rdi中；
2. 把cookie的值放到栈上，pop到rax中，再mov到rdi中；
3. 把cookie的值放到栈上，rsp+偏移得到cookie地址，mov到rdi中；
4. 把cookie的值放到栈上，rsp+偏移得到cookie地址，mov到rax，再mov到rdi中；

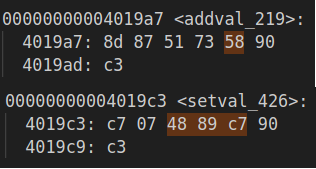
还有n多思路，之所以要搞这么多是因为目标代码不一定正好有我们需要的指令，比如此例中在规定的范围内就没有popq %rdi，所以1就排除了，然后官方说两个gadgets就能搞定，那很可能是2。（3、4涉及到加法，比较复杂，phase5会用到）

这里我刚开始理解错了，认为必须用反汇编后的完整指令，比如下面这个

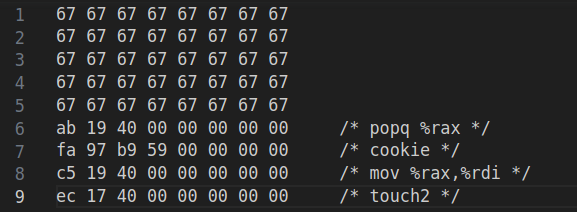


我以为必须以40199a为起始地址，用这个完整的指令，后来才知道可以任意选取起始地址，比如上面78 90 90就可以组成一个指令，地址是40199c。

OK，我们找找看有没有popq %rax和mov %rax,%rdi。下面的58和48 89 c7刚好就是，地址分别是0x4019ab和0x4019c5。



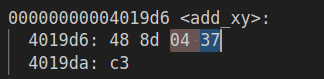
所以最终的路径是getbuf->popq->mov->touch2



Phase5

跟Phase3的任务一样，但是要用Return-Oriented Programming。基本思路很简单，得到rsp的值，加上cookie的位置偏移，放到rdi中，调用touch3。实现路径也是有很多，难点是能不能在目标代码中找到对应的机器码。有一点要注意，因为栈随机，64位栈地址高32位不一定是0，所以mov时只能用movq。

这里关键是处理加法，仔细的小伙伴会发现目标代码中有个add\_xy函数，这个可以利用；还可以找add和lea的机器码，最后找到addb的04指令，不过它直接加了0x37，而且巧合的是它刚好在add\_xy函数里。



所以有两个思路，一个是准备好rsi和rdi调用add\_xy，一个是将rsp放到rax，加0x37，所以要把cookie字符串放到rsp后0x37的地址处。

方法1：实现路径

rsp->rax->rdi，

popq %rax

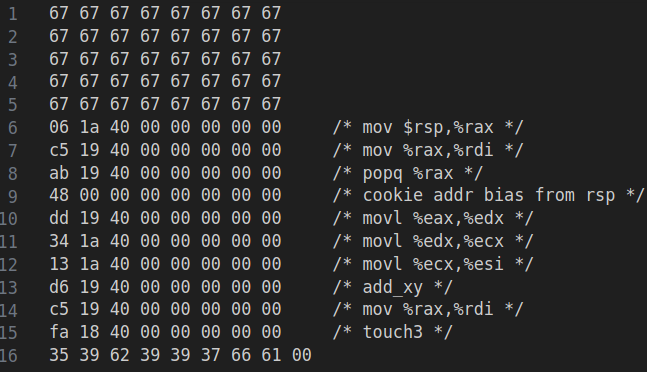
eax->edx->ecx->esi

add\_xy

rax->rdi

touch3

具体怎么找到的，就是一个个试，比如esi/rsi那一列的机器码一个个搜，就能找到哪个寄存器能给esi/rsi赋值，然后继续直到eax/rax，或者能用popq D得到值。这个刚好符合官方的8个gadget。



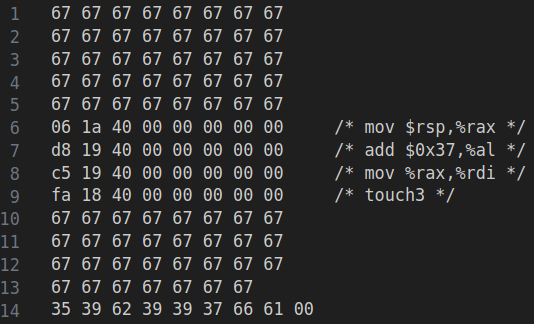
方法2：rsp->rax

rax+=0x37 这里虽然只加低8位，但不影响高24位

rax->rdi

touch3

这个是我网上找的，更简单，0x37的位置也OK，没有给程序带来崩溃、段错误等副作用。



OK，以上就是attacklab的5个Phase解答，这个lab说简单也简单，理论和思路不难，说复杂也复杂，需要找合适的gadget，尤其是Phase5，非常需要耐心。希望本文对大家有所帮助，欢迎大家讨论交流。