**中国海洋大学 计算机科学与技术系**

**实验报告**

**姓名：岳宇轩**  **学号：19020011038** **专业：**计算机科学与技术

**科目：**计算机系统原理 **题目：**缓冲区溢出攻击

**实验时间: 2020** 年 1月 15日

**实验成绩: 实验教师:** 孙鑫

**一、实验目的：**

1. 熟悉linux基本操作命令，其中常用工具和程序开发环境

2. 熟悉反汇编obnjump、调试指令gdb的操作

3. 加深对IA-32函数调用规则和栈结构的具体理解

**二、实验要求**

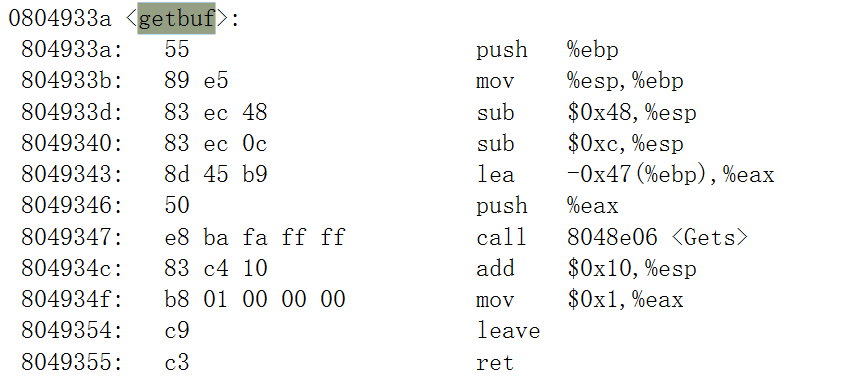
1. 熟悉linux基本操作命令，还有其中常用工具和程序开发环境。以及objdump、gdb指令

2. 灵活掌握各种汇编语句，以及查询内存中的信息的方法。

3.需要对目标可执行程序bufbomb分别完成5个难度递增的缓冲区溢出攻击。5个难度级分别命名为smoke (level 0)、fizz (level 1)、bang (level 2)、boom (level 3)和nitro (level 4)，其中smoke级最简单而nitro级最困难。

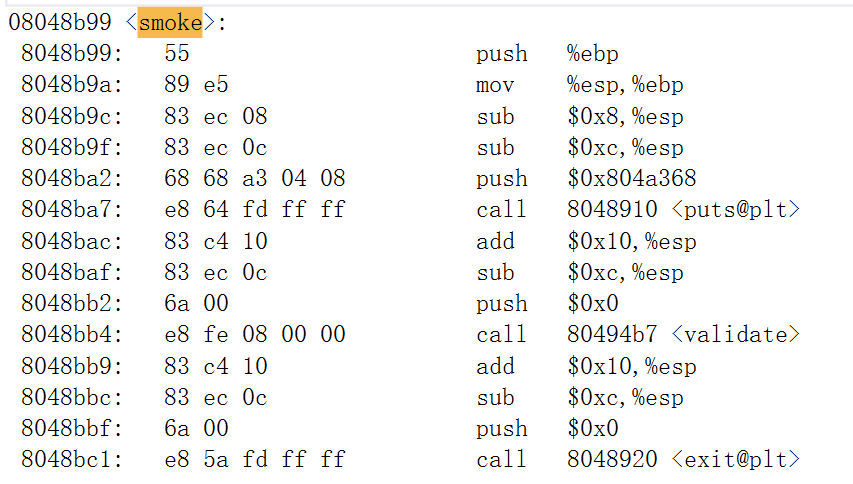
**三、实验内容（进行缓冲区溢出攻击方法的解释）：**

**smoke:**



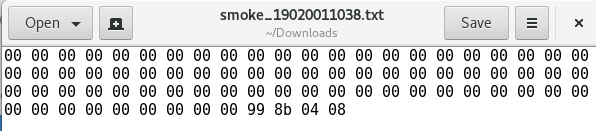
从8049343行指令可以看到

buf缓冲区的大小为0x47，即71个字节

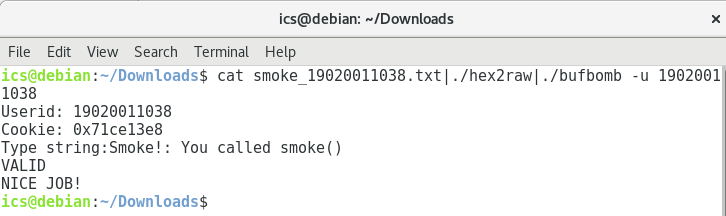


来到smoke函数，看到smoke函数的地址为0x08048b99

攻击字符串用来覆盖数组buf，进而溢出并覆盖ebp和ebp上面的返回地址，攻击字符串的大小应该是71+4+4=78个字节。攻击字符串的最后4字节应是smoke函数的地址0x08048b99。

所以，设计攻击字符串为

成功调用了smoke函数



**fizz.**



首先看到fizz函数的返回地址为0x08048bc6

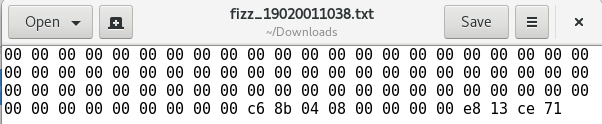


继续看，fizz函数使用了一个参数，这个参数需要是自己的cookie

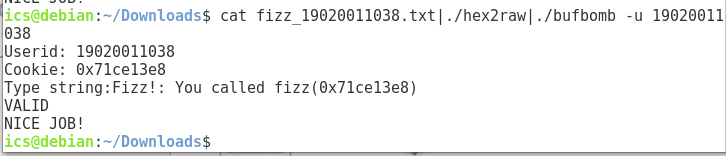
所以可以这样构造攻击串：71（冲掉buf）+4(冲掉ebp旧值）+4（用fizz函数地址替换，冲掉返回地址）+4（占位置）+4（这里到了ebp+8的位置，即需要的参数，把cookie值按照小端方式放到这里就行）

可以看到我的cookie是0x71ce13e8

所以可以得到第二题的答案为



成功！：

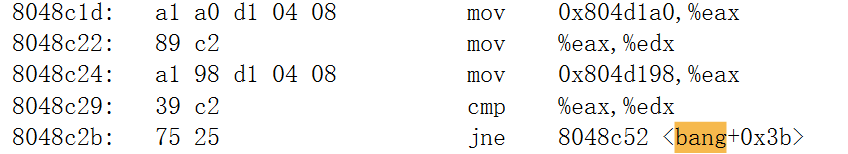


**Bang.**

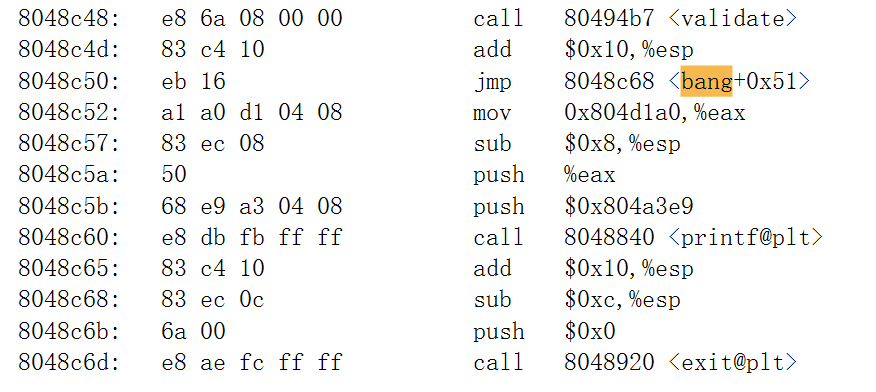
来到bang，首先记录一下返回地址是



接着看



比较了0x804d1a0和0x804d198处的内容，如果不相同则跳转至0x8048c52处



可以看到，如果跳转到了0x8048c52，则无法call calidate了，所以我们要让这两个内存处的内容相同

我先简单写了一个txt文件，前71个字节都是00，然后是00 00 00 00 17 8c 04 08，进入gdb调试，这样就可以进入bang函数了

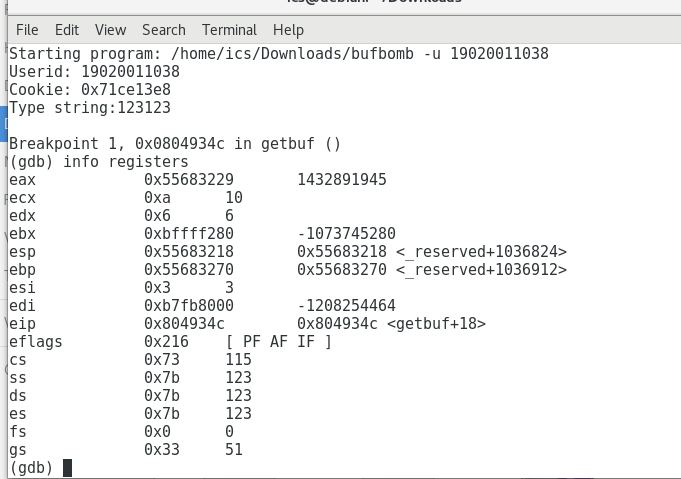
打印出0x804d198处的内容，发现是0x71ce13e8，就是我的cookie，所以只需要修改0x804d1a0处的内容为cookie值就可了

编写攻击汇编代码，并进行反汇编

作用就是把cookie放到0x804d1a0处，然后调用bang函数

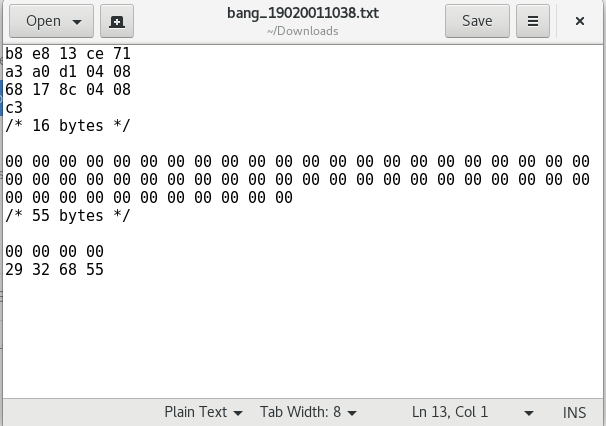
那么，怎么才能调用到这段函数呢？

可以把它写入buf数组中，buf数组会被加载进入缓冲区，找到缓冲区的地址，用它覆盖掉返回地址就行了。

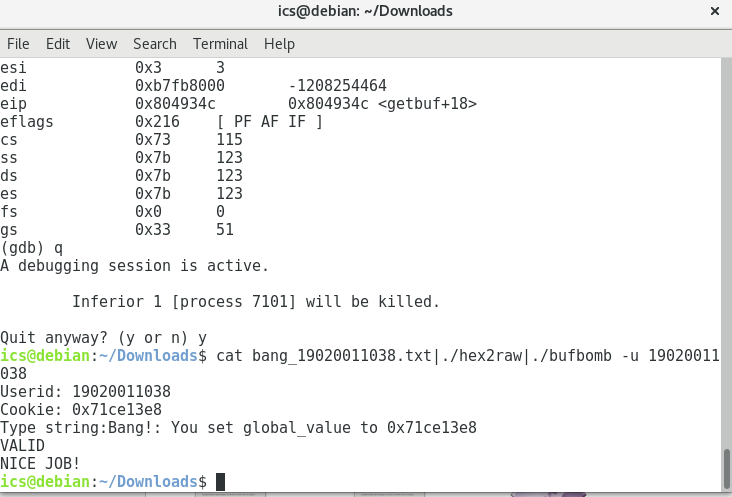


通过调试，可以找到ebp为0x55683270，那么根据

ebp-0x47 = 0x55683229就是缓冲区首地址了



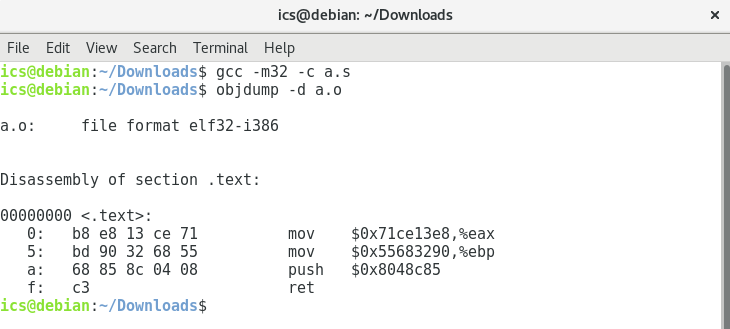
所以就可以构造出这样的攻击串



success！！！！

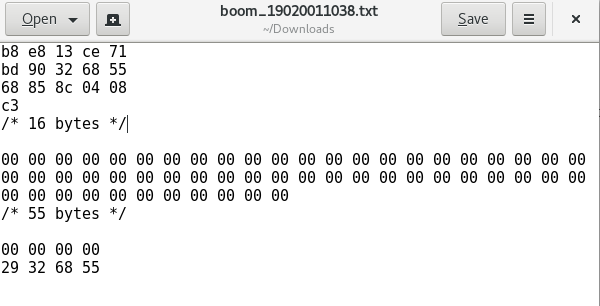
**Boom.**

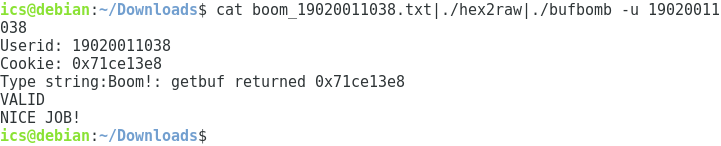
要使返回值为cookie，需要把cookie值放到eax寄存器中，还需要设置ebp值为test的ebp值



push的地址是返回地址，是call getbuf行的pc，也就是0x8048c85





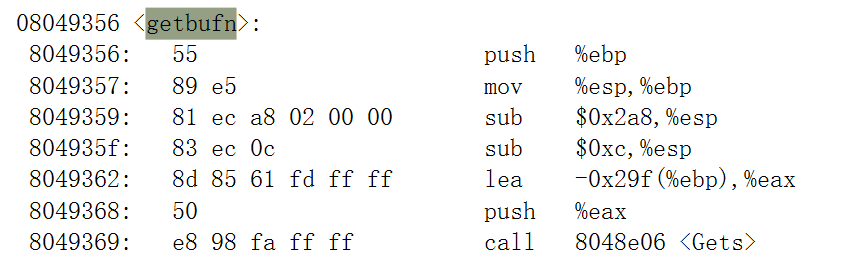
把它写到缓冲区中，并用缓冲区地址替换返回地址，构造出上图所示的攻击串

**nitro.**

这个题废了我老多功夫了

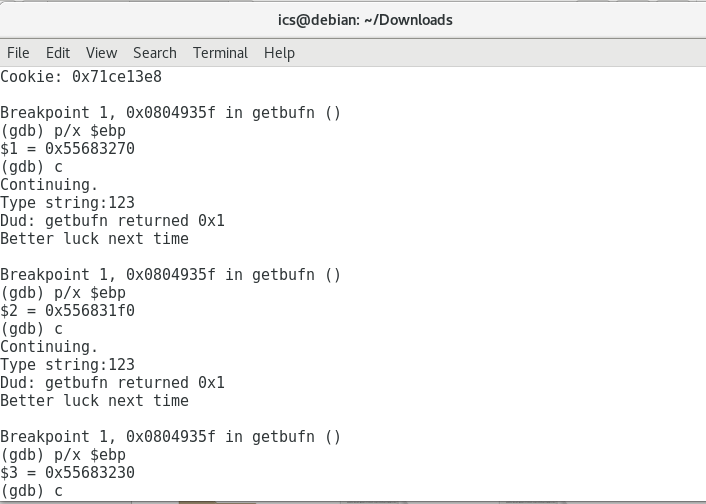
当堆栈地址不固定，但是只能指定覆盖ret时返回一个固定的地址，怎么办？

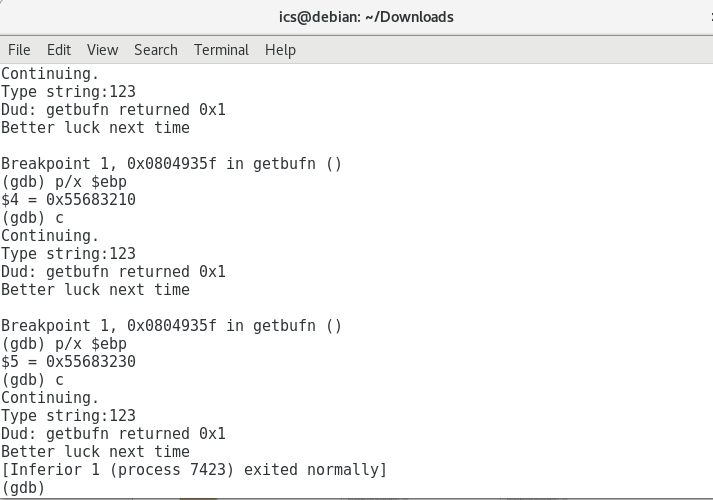
通过查阅资料，可以使用nop来尽可能的填充我们构造的执行攻击代码的堆栈，这样当我们分配的堆栈初始地址不同时仍可以通过若干个nop跳转到我们的攻击代码。



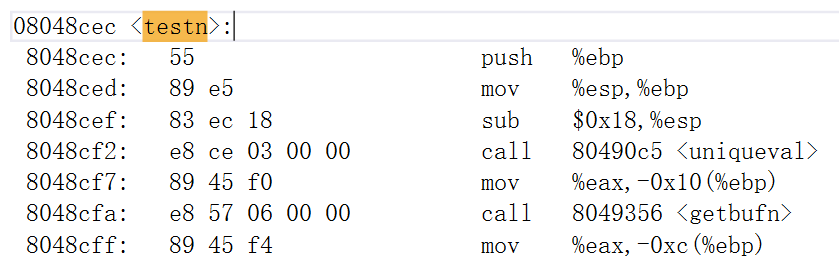
可以看到getbufn中开辟的数组大小为0x29f，即十进制671

每次运行testn的ebp都不同，所以每次getbufn里面保存的test的ebp也是随机的，但是栈顶的esp是不变的，我们就要找到每次随机的ebp与esp之间的关系来恢复ebp。我们先通过调试来看一下getbuf里面保存的ebp的值的随机范围为多少。





再来看一下testn中的代码



先是老套路，push ebp

mov esp,ebp

这样就是ebp==esp了

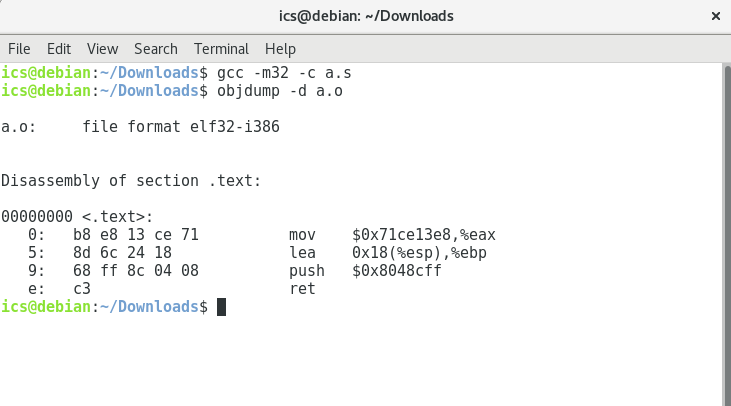
接着esp = esp - 0x18

这样就是 esp == ebp-18

即ebp == esp + 18 这就是ebp和esp的关系，这样我们就可以通过esp来恢复ebp了。

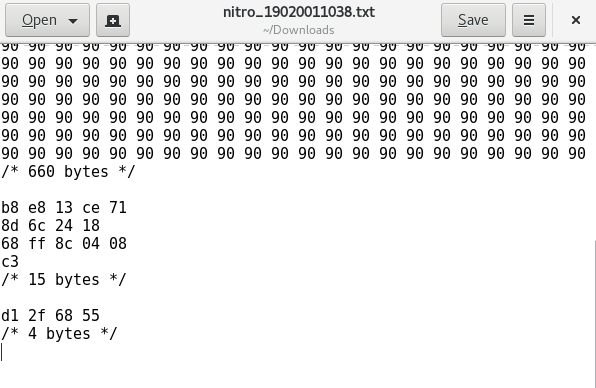
其余操作和boom中的类似

可以得到下面所示的攻击反汇编代码：



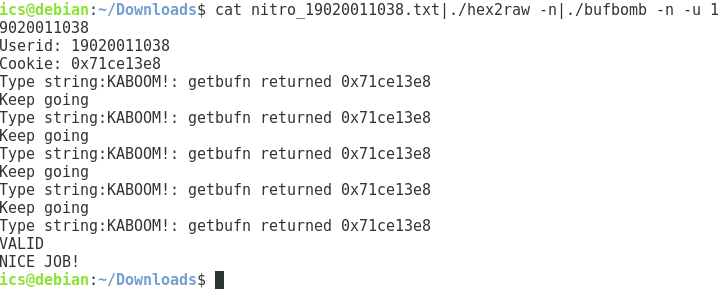
push的地址是testn中执行到getbufn时的pc

于是可以构造出这样的攻击串：



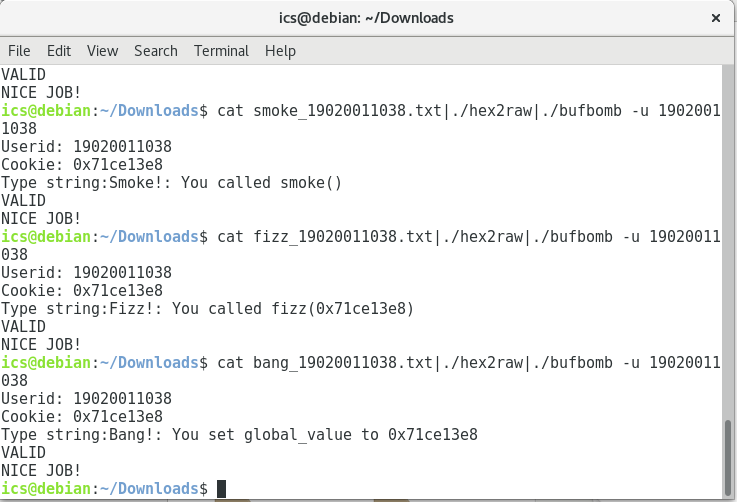
bufn一共671字节，再+4，就是675，其中反汇编得到的字节码是15bytes，剩下的用nop填充也就是660bytes

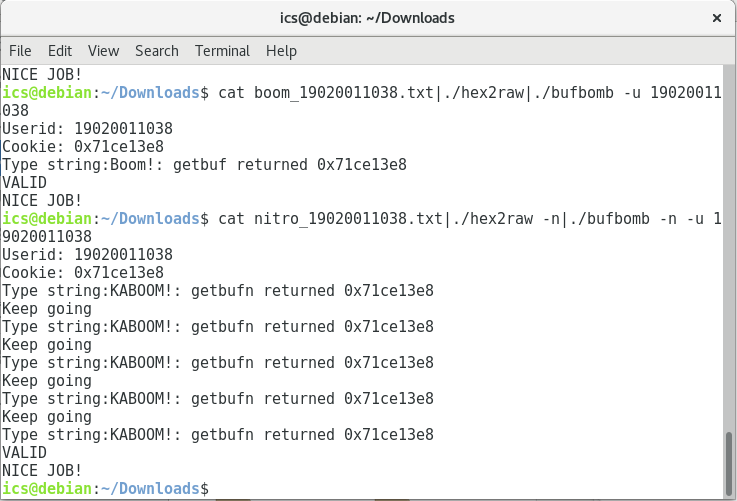
接下来的4个字节用来覆盖返回地址，这里我们取ebp为0x55683270，那么对应缓冲区的首地址就是0x55683270 - 0x29f = 0x556832d1



执行时./hex2raw后也要加 -n ，因为这被卡好久

最后记录一下实验成功的结果





**总结：**

1. 首先找到读入数组的长度，即缓冲区的大小
2. 攻击字符串首先要满足buf的长度，然后，还要再+4个字节，用来覆盖ebp旧值。再＋4个字节，即需要调用的函数的返回地址，替代函数返回地址。
3. 返回地址用小端格式
4. 当需要执行的函数要穿参数时，可以根据参数的数量冲掉ebp+8,ebp+0xc,ebp+0x10等位置，这样调用了函数后就可以直接根据ebp获得参数了
5. 可以先写一个简单的文件使得可以进入gdb调试
6. 攻击字符串需要进行一些其他的操作时，可以把汇编代码反汇编得到的字节序列放到buf字符串中，汇编代码中写要执行的操作和调用的函数。用缓冲区首地址替换函数返回地址
7. 注意返回地址的选择
8. 若要修改被攻击函数的返回值，就把要返回的内容写入eax中，还要记得修改ebp和pc
9. 堆栈地址不固定，找固定的值和ebp之间的关系