**中南林业科技大学**



**课程实验报告**

课程名称： 微机原理与接口技术实验

姓 名： 刘军

学 号： 20212753

专业班级： 计算机科学与技术 3 班

指导老师： 周慧斌

学 院： 计算机与数学学院

|  |
| --- |
| 指导老师评语： 成绩：  签名：  年 月 日 |

目录

[**实验1：Data Lab** 3](#_Toc164078359)

[**一、 实验目的** 3](#_Toc164078360)

[**二、 实验环境** 3](#_Toc164078361)

[**三、 实验内容** 3](#_Toc164078362)

[1. bitXor 4](#_Toc164078363)

[1. tmin 5](#_Toc164078364)

[2. isTmax 5](#_Toc164078365)

[3. allOddBit 5](#_Toc164078366)

[4. negate 6](#_Toc164078367)

[5. isAsciiDigit 6](#_Toc164078368)

[6. conditional 6](#_Toc164078369)

[7. isLessOrEqual 7](#_Toc164078370)

[8. logicalNeg 7](#_Toc164078371)

[9. howManyBits 8](#_Toc164078372)

[10. floatScale2 8](#_Toc164078373)

[11. floatFloat2Int 9](#_Toc164078374)

[12. floatPower2 10](#_Toc164078375)

[**实验2：Bomb Lab** 12](#_Toc164078376)

[**一、 实验目的** 12](#_Toc164078377)

[**二、 实验环境** 12](#_Toc164078378)

[**三、 实验内容** 12](#_Toc164078379)

[1. phase\_1 14](#_Toc164078380)

[2. phase\_2 16](#_Toc164078381)

[3. phase\_3 17](#_Toc164078382)

[4. Phase\_4 18](#_Toc164078383)

[5. phase\_5 19](#_Toc164078384)

[6. phase\_6 19](#_Toc164078385)

[7. secret\_phase 20](#_Toc164078386)

[**实验3：Attack Lab** 22](#_Toc164078387)

[**一、 实验目的** 22](#_Toc164078388)

[**二、 实验环境** 22](#_Toc164078389)

[**三、 实验内容** 22](#_Toc164078390)

[1. phase1 25](#_Toc164078391)

[2. phase2 26](#_Toc164078392)

[3. phase3 27](#_Toc164078393)

**实验1：Data Lab**

课程名称：微机原理与接口技术课程实验 实验日期： 2024年3月22日

班级： 计算机科学与技术3班 姓 名： 刘军 学 号： 20212753

**一、 实验目的**

1. 熟悉整数和浮点数的位级表示；
2. 完成实验的bits.c文件内容。

**二、 实验环境**

* VMware Workstation虚拟机环境下的Ubuntu 64位

**三、 实验内容**

**实验准备阶段：**首先需要使用ubuntu联网环境跳转到链接下载实验所需的datalab：<http://csapp.cs.cmu.edu/3e/labs.html>

下载datalab压缩包并输入$ tar –xvf datalab-handout.tar进行解压缩，进入该目录所有文件如下所示：

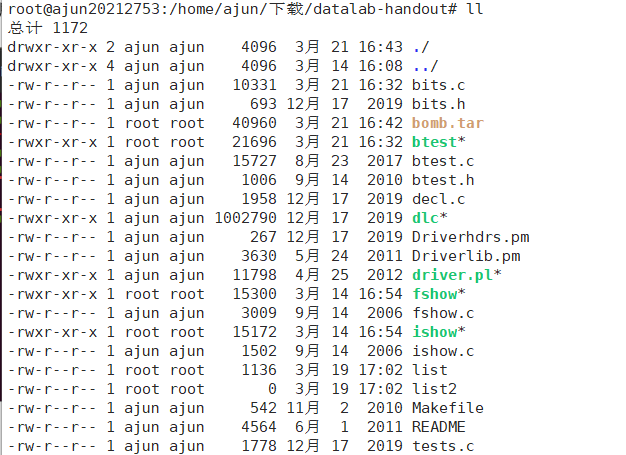


图1-1

在终端输入sudo apt install gcc-multilib安装make。

**实验过程阶段：**在终端输入vim bits.c进入bits.c文件，其中包含了13个编程谜题中每个谜题的骨架。任务是只使用整数谜题的直线代码（即没有循环或条件）和有限数量的C算术和逻辑运算符来完成每个函数骨架。明确禁止使用任何控制结构，如if，do，while，for，switch等；定义或使用任何宏；在此文件中定义任何其他功能；调用任何功能；使用任何其他操作，例如&&，||， - 或？：使用任何形式的转换使用除int之外的任何数据类型。

具体只能使用以下八个运算符：

! ˜ & ˆ | + << >>

需要解决12个函数，描述如下：

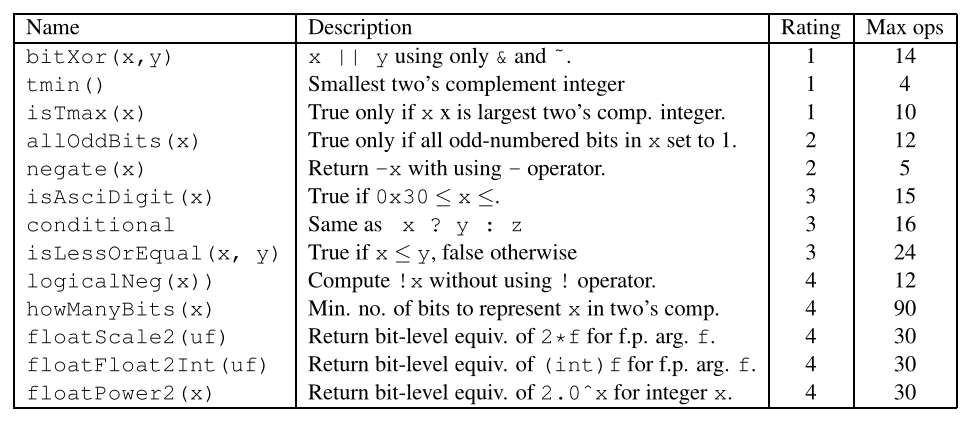
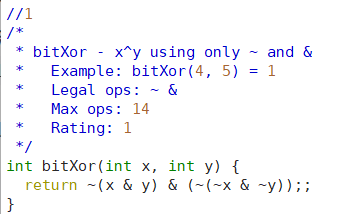


图1-2

### bitXor

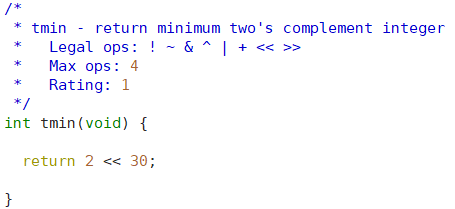
要求只用~和&两个运算符来实现异或运算

运用德摩根律



### tmin

要求返回二进制形式下补码表示的最小值，即0x80000000

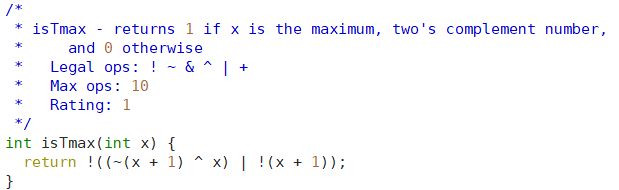


### isTmax

判断一个数是不是二进制形式下，补码表示的最大数,即0x7fffffff

该数取反后就是tmin，tmin和tmin的相反数一样。

注意：0和0的相反数也一样

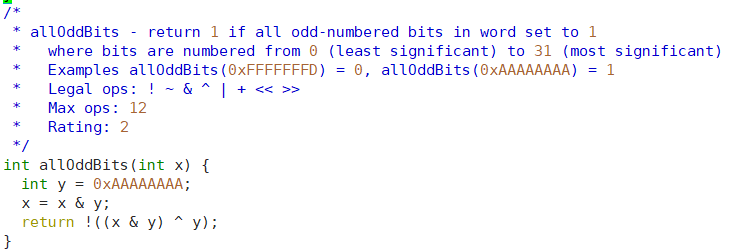


### allOddBit

判断一个给定的数在二进制表示下，是不是全部奇数位都是1

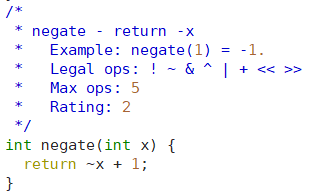
先生成一个奇数位都是1，偶数位都是0的掩码mask。

然后让x & mask，看看mask是否保持不变，如果不变，说明满足条件



### negate

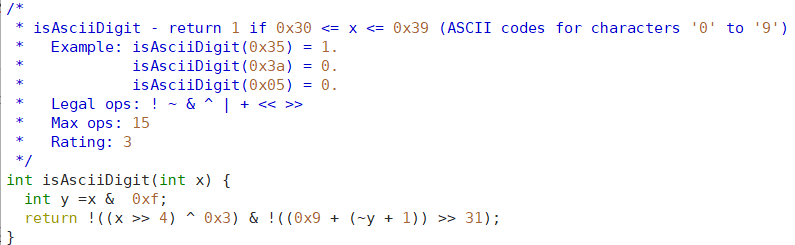
返回一个数的相反数



### isAsciiDigit

判断一个数是否位于0x30到0x39之间

可以分别进行判断，首先判断10位是否为3, 然后判断个位是否小于等于9

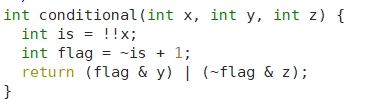


### conditional

实现一个条件表达式 x ? y : z

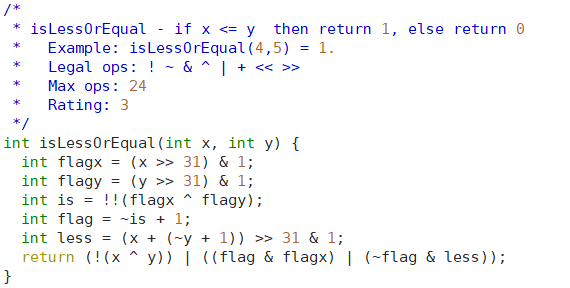
首先，设置一个变量notZero来判断x是否不为0，如果不为0，则该变量为1；否则为0。

然后，设置一个变量flag，当notZero为1时，该变量为0xffffffff；否则为0x00000000



### isLessOrEqual

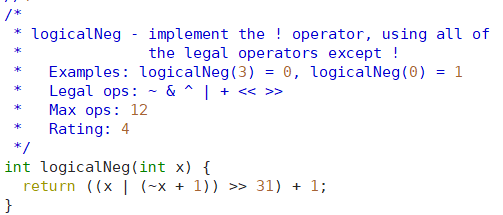
判断x<=y



### logicalNeg

实现!运算符——也就是x = 0时返回1，否则返回0

通过判断x是否为0来实现这个函数，不难发现，只有0和其相反数或之后，全部都位为0



### howManyBits

判断一个数在二进制补码表示下，最少需要多少位才能表示出来

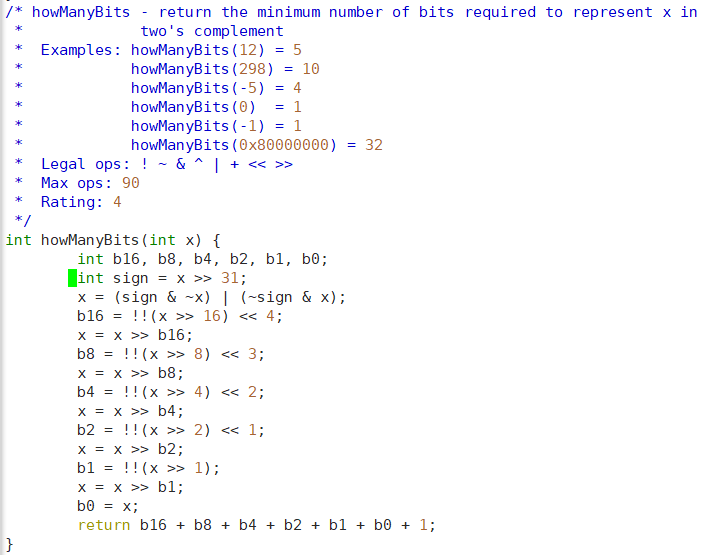
当x为正数时，假设idx表示为1的最高的位置，答案即idx + 1。

当x为负数时，我们将x取反，假设idx表示此时为1的最高的位置，答案即idx + 1。

因此，我们可以通过找出最高的1的位置来计算答案。

因为不能使用循环，所以我们利用二分查找。

首先，判断高16位是否存在1，如果存在1的话，我们就只需要在高16位上寻找最高位；如果高16位不存在1，那么我们去低16位寻找。就这样一直找下去，直到区间长度变为1停止。



### floatScale2

计算出2 \* f的浮点数表示

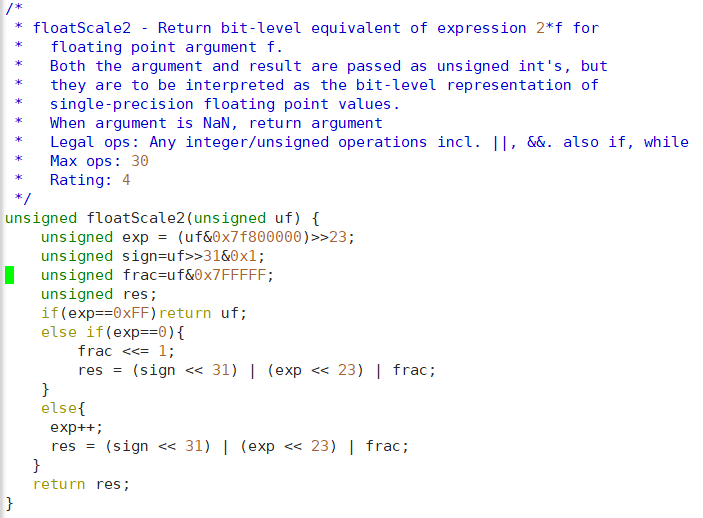
首先，把f中的符号位s，阶数exp，尾数frac全都扣出来。

然后，判断一下是否是非规格化的值，也就是判断阶数exp是否为0，如果为0，直接把尾数乘2然后配上符号输出即可。

如果exp = 255，也就是为NaN或者无穷，直接返回f。

否则，就把阶数exp加1，如果此时exp为255，也就是阶码位全为1，此时溢出了，返回无穷。

最后，代表没有溢出，返回正常的值即可。



### floatFloat2Int

把一个给定的浮点数f通过强制类型转换为int

首先，还是先把符号位s，阶码exp，尾数frac 都抠出来。

计算一下真实的阶数，E= exp — 127。

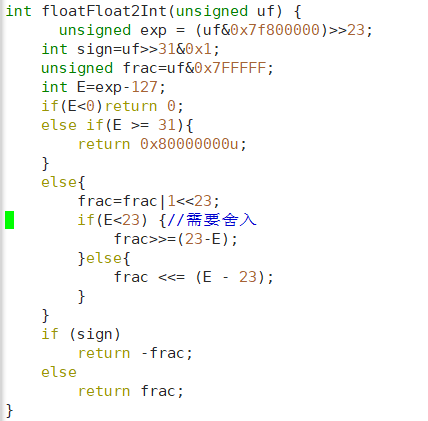
如果f是NaN，或者E>31，根据题意，应该直接返回0x8000000ou 。

如果E<0，代表f只有小数部分，直接返回0即可。

我们算出真实的尾数M = frac|(1<<23)。

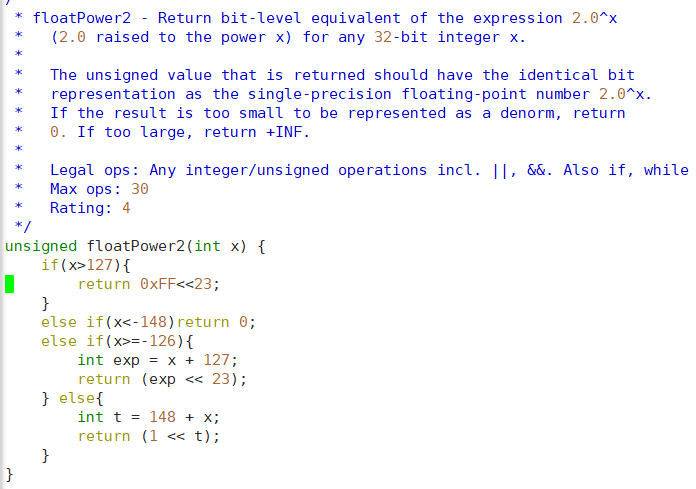
如果E>23就需要在后面补E-23个0，否则，需要舍去23-E位。

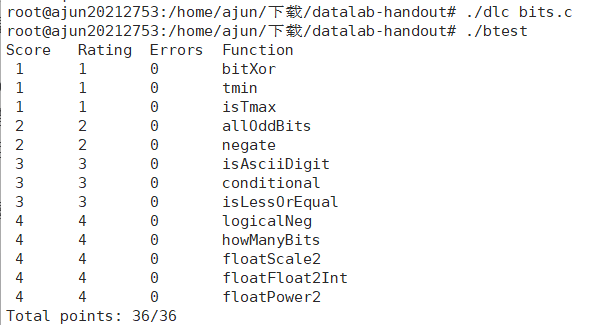
最后，搭配上符号位输出即可。



### floatPower2

算出2^x的单精度浮点数表示并返回





**实验2：Bomb Lab**

课程名称：微机原理与接口技术课程实验 实验日期： 2024年3月29日

班 级： 计算机科学与技术3班 姓 名： 刘军 学 号： 20212753

**一、 实验目的**

1. 提高解决问题的能力；

2. 更好地理解计算机系统的安全问题；

3. 提高编程技能和汇编语言的理解。

**二、 实验环境**

* VMware Workstation虚拟机环境下的Ubuntu 64位。

**三、 实验内容**

**实验准备阶段：**首先需要使用ubuntu联网环境跳转到链接下载实验所需的bomblab：<http://csapp.cs.cmu.edu/3e/labs.html>

下载bomblab压缩包并输入$ tar –xvf bomb.tar进行解压缩，进入该目录所有文件如下所示：

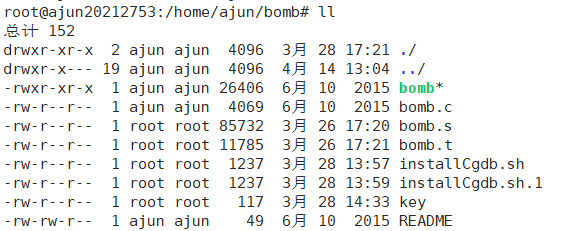


图2-1

在终端输入sudo apt-get install gdb安装调试器。基本用法参考下图：



图2-2

**实验过程阶段：**

“Binary bombs”是一个可在Linux系统上运行的C程序，它由6个不同的阶段（phase1~phase6）组成。在每个阶段，程序会要求输入一个特定的字符串。如果输入的字符串符合程序的预期输入，那么这个阶段的炸弹就会被“解除”，否则炸弹就会“爆炸”，并输出“BOOM!!!”的提示信息。实验的目的是尽可能多地解除这些炸弹的阶段。

每个炸弹阶段考察了机器级语言程序的一个不同方面，难度逐级递增：

\* 阶段1：字符串比较

\* 阶段2：循环

\* 阶段3：条件/分支

\* 阶段4：递归调用和栈

\* 阶段5：指针

\* 阶段6：链表/指针/结构

在炸弹拆除任务中，还存在一个隐藏阶段。然而，只有在第四个阶段解决后添加特定的字符串后，该隐藏阶段才会出现。为了完成任务，需要使用gdb调试器和objdump反汇编炸弹的可执行文件，然后单步跟踪每个阶段的机器代码，理解每个汇编语言的行为或作用。这将帮助“推断”出拆除炸弹所需的目标字符串。为了调试，可以在每个阶段的开始代码前和引爆炸弹的函数前设置断点。

在终端输入objdump -d bomb > bomb.asm得到bomb的反汇编文件bomb.asm如下所示。

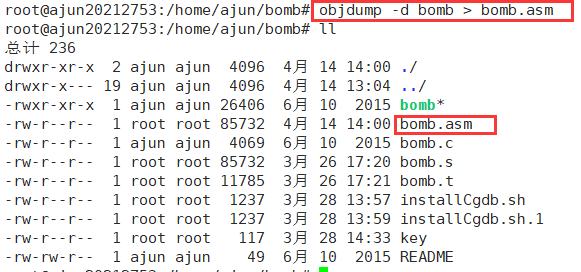
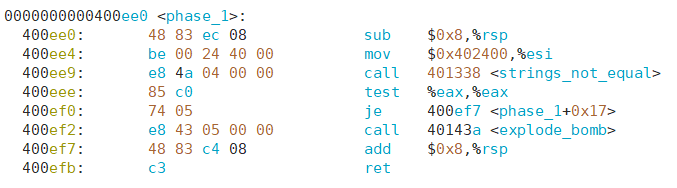


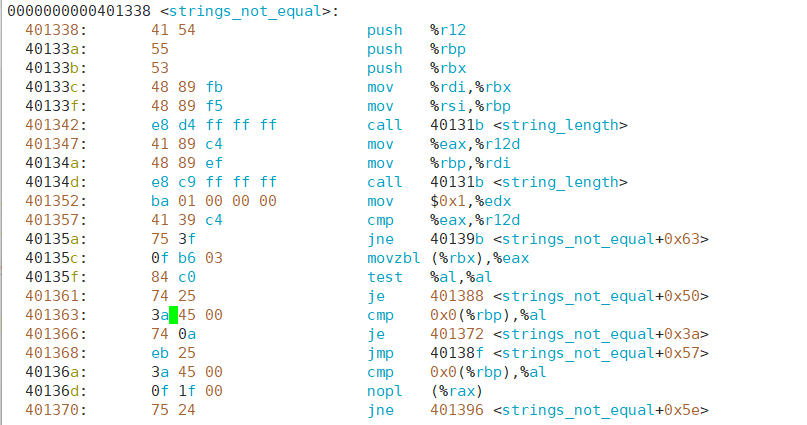
图2-3

### phase\_1



首先看到explode\_bomb函数，猜测调用这个函数会引爆炸弹，所以需要执行之前的je指令跳过它。

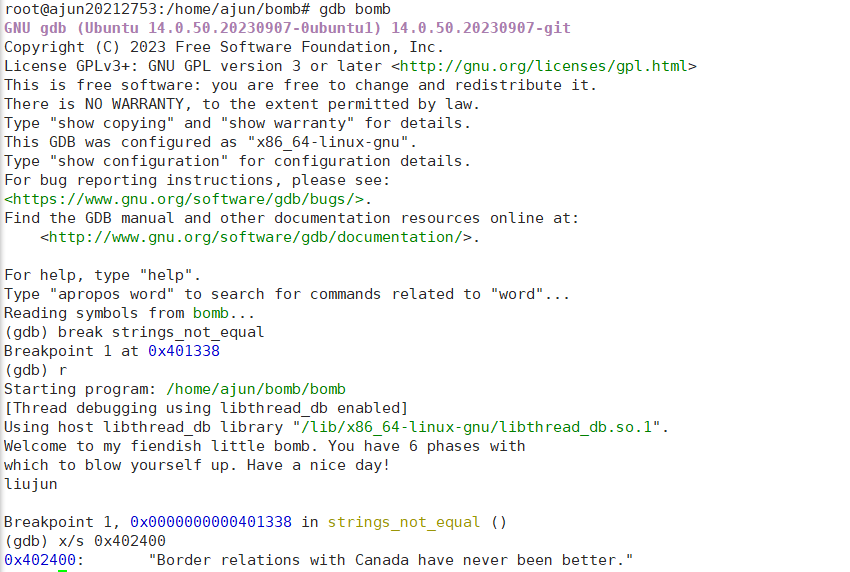
je指令能跳转所满足的条件为%eax = 0，猜测strings\_not\_equal函数的功能用于比较字符串是否相同。



从strings\_not\_equal函数可以初略看出，若字符串不相等会返回1，反之。

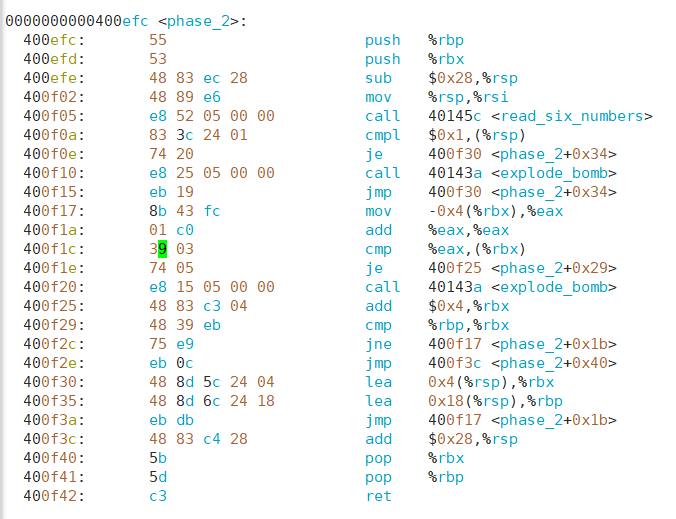
该函数所需要的参数为两个字符串首地址，用%rsi和%rdi寄存器存放。

使用gdb调试，查看这两个字符串存放的数据是什么。

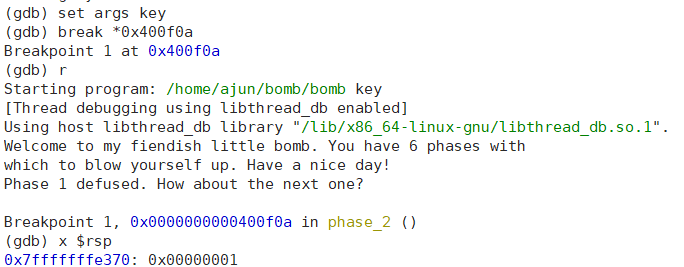


****

### phase\_2



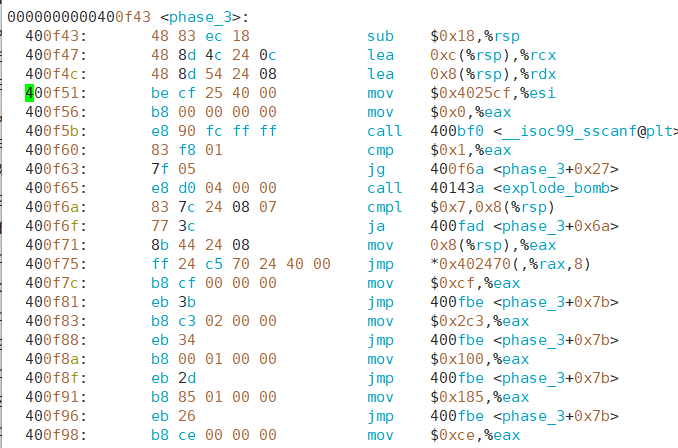
猜测read\_six\_numbers会读入六个数字，我们将断点设置在0x400f0a处，在文件key.txt中写入测试数据。

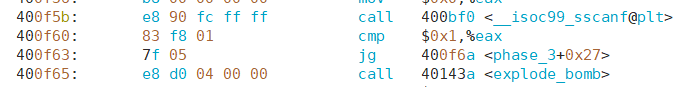


推测输入的六个数存放在首地址为rsp的数组中

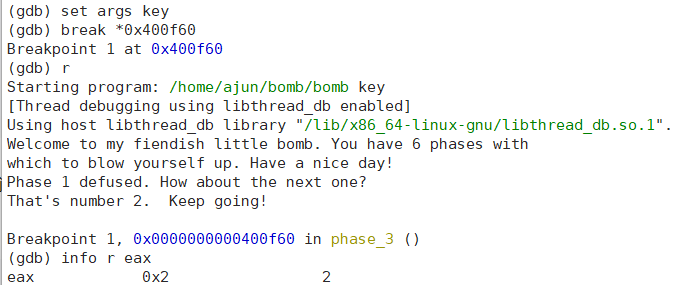
推导出序列为1 2 4 8 16 32

### phase\_3





sscanf函数是库函数，返回值eax表示正确格式化的数据个数，在测试文件中随便写入几个数0 1 2 3 4 5，断点在0x400f60处查看下eax中的值



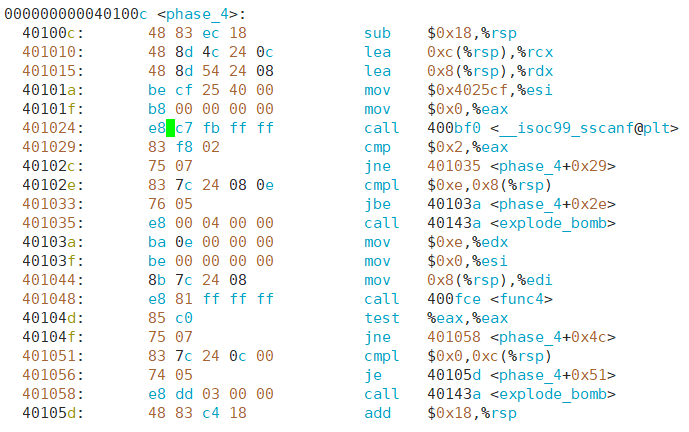
eax=0x2，而当前输入参数是多于两个的，所以第三关要求输入两个参数。并且通过调试(第二关)我们可以知道输入的两个参数存放的位置

可以获取到第一个条件a<7

他们都会跳转到0x400fbe，最后会比较0xc(%rsp)和eax的大小。

通过前面的分析知道0xc(%rsp)便是输入的参数b，我们再讨论输入不同的a，最终得到的eax值是多少。

### Phase\_4



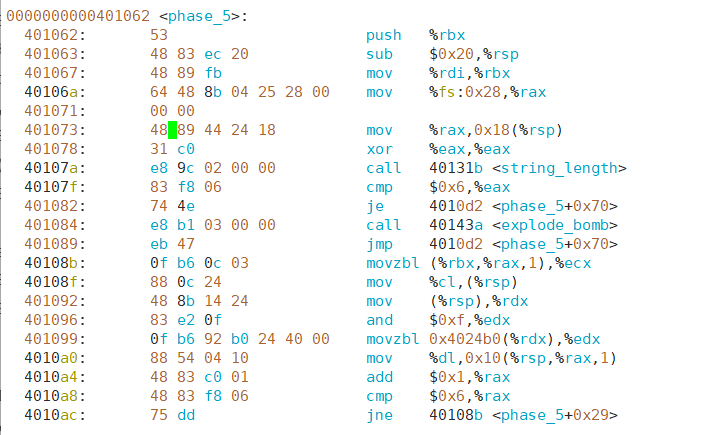
a<=0xe，b=0，这两个参数范围确定。

func4函数需要三个参数，%edx=0xe，%esi=0x0，%edi=a。

func4函数反汇编的倒数第二行，可能是一个递归程序，最终发现其实没有调用。

func4函数并没有改变%edi寄存器的值。

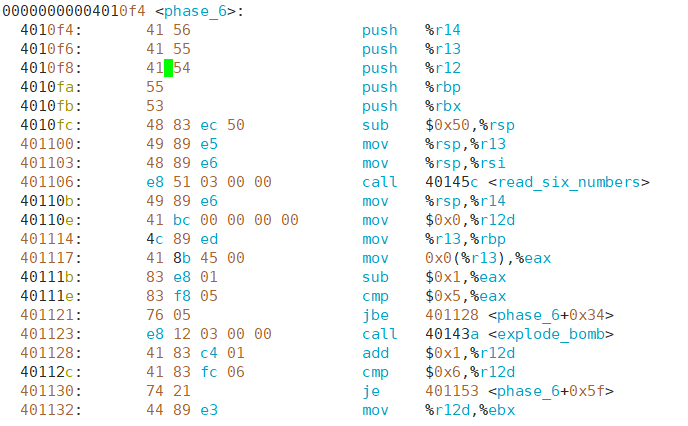
### phase\_5



又相同的string\_length函数，通关第一关的经验，肯定要输入一个字符串并且长度为6，在测试文件中写入6个字符，调试程序

输入6个字符，把每个字符 & 0xf，所得结果作为下标，从一个字符数组中取字符，最终组成"flyers"

### phase\_6



401106 读出6个int到rsp中

40110e 检测六个int是否为[1,2,3,4,5,6]的任意排序

401153 将这个六个int 变为 7 - int

40116f 将 第 7 - int个链表元素 放到 rsp0x20 + int 的相对位置 \* 2的位置

4011bd 链表， 被换成了 咱们数组中的顺序 ， 和题目无关

4011d2 检测没一个 都curArrayValue >= nextArrayValue 即要求现在的数组降序

需要读入六个数字，随意写1 2 3 4 5 6测试，断点在0x401117处调试

所以需要保证其数据为排列为3 4 5 6 1 2，又因为在第2步用7将数据去补数，所以转换后为4 3 2 1 6 5

### secret\_phase

在bomb.c注释说明了存在彩蛋关，查看汇编代码，发现phase\_defused函数会调用函数secret\_phase

只有输入的参数前两个为整数，第三个为字符串DrEvil时才不会跳过隐藏关。再看看运行结束时0x603870里面的字符串是什么

是第四关选择的答案。那么在第四关后面加上字符串DrEvil开启隐藏关卡，查看secret\_phase函数代码。

简单的分析可以发现这个一个二叉搜索树，再看看他的规律

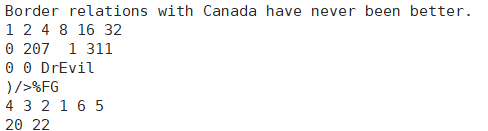
左会乘以2 ， 右会变成0， 不相等则rax = rax + rax + 1

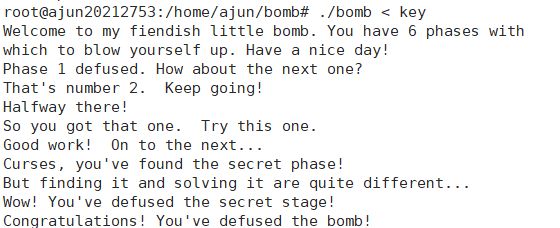
所以我们发现当eax为0的时候 左边可以走任意多次，并且在后面走的左边都无所谓。

可以发现右边是初始化的起点，因为答案需要eax == 2，参考下面这个树 我们可以 "左右"两步搞定，当然在走一个左也无所谓。

所以最后的序列是： 左右（左）

因此即答案为：20 22





**实验3：Attack Lab**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

课程名称：微机原理与接口技术课程实验 实验日期： 2024年4月11日

班级： 计算机科学与技术3班 姓名： 刘军 学 号： 20212753

**一、 实验目的**

1. 防止攻击者定位攻击；
2. 掌握使用金丝雀防护。

**二、 实验环境**

* VMware Workstation虚拟机环境下的Ubuntu 64位。

**三、 实验内容**

**实验准备阶段：**首先需要使用ubuntu联网环境跳转到链接下载实验所需的attacklab：<http://csapp.cs.cmu.edu/3e/labs.html>

下载target1压缩包并输入$ tar –xvf target1.tar进行解压缩，进入该目录所有文件如下所示：

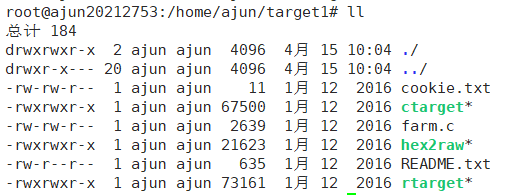


图3-1

当前提供材料包含一个攻击实验室实例的材料:

1.ctarget

带有代码注入漏洞的Linux二进制文件。用于作业的第1-3阶段。

2.rtarget

带有面向返回编程漏洞的Linux二进制文件。用于作业的第4-5阶段。

3.cookie.txt

包含此实验室实例所需的4字节签名的文本文件。（通过一些Phase需要用到的字符串）

4.farm.c

rtarget实例中出现的gadget场的源代码。您可以编译(使用标志-Og)并反汇编它来查找gadget。

5.hex2raw

生成字节序列的实用程序。参见实验讲义中的文档。（Lab提供给我们的把16进制数转二进制字符串的程序

**实验过程阶段：**

使用objdump -d ctarget > ctarget.asm以及objdump -d rtarget > rtarget.asm对ctarget以及rtarget进行反汇编，得到ctarget.asm和rtarget.asm。

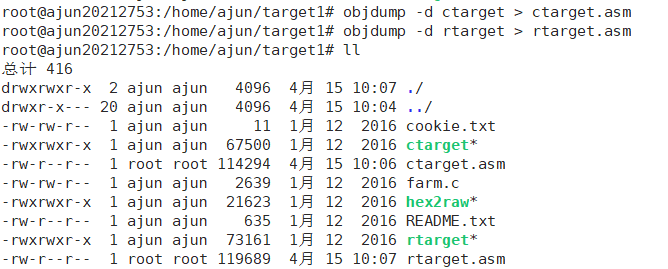


图3-3

在官方文档的目标程序给出，CTARGET和RTARGET都从标准输入读取字符串。它们使用下面定义的函数getbuf来执行此操作：

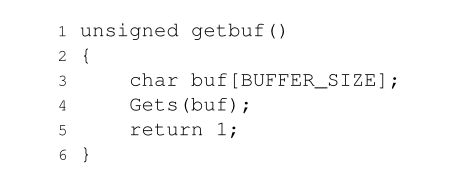


图3-4

函数Gets类似于标准库函数gets—它从标准输入中（从缓冲区）读取字符串 (以’ \n '或文件结束符结束) 并将其(连同空结束符)存储在指定的目的地。即空格/Tab/回车可以写入数组文本文件，不算作字符元素， 不占字节，直到文件结束， 如果是命令行输入的话，直到回车结束（区别getchar ()：是在输入缓冲区顺序读入一个字符 (包括空格、回车和 Tab)结束，scanf：空格/Tab/回车都当作结束。函数Gets()无法确定它们的目标缓冲区是否足够大，以存储它们读取的字符串。它们只是复制字节序列，可能会超出在目的地分配的存储边界（缓冲区溢出）对应汇编代码：

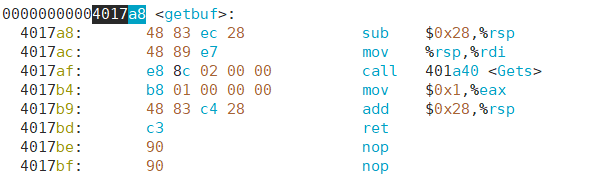


图3-5

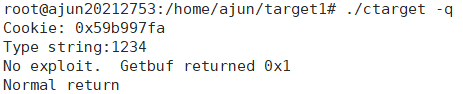
因为Ctarget就是让我们通过缓冲区溢出来达到实验目的，所以可以推断sub $0x28,%rsp的40个字节数就等于输入字符串的最大空间，如果大于40个字节，则发生缓冲区溢出（超过40个字节的部分作为函数返回地址，如果不是确切对应指令的地址，则会误入未知区域（报错：Type string:Ouch!: You caused a segmentation fault!段错误，可能访问了未知额内存））。

### phase1

首先执行ctarget。

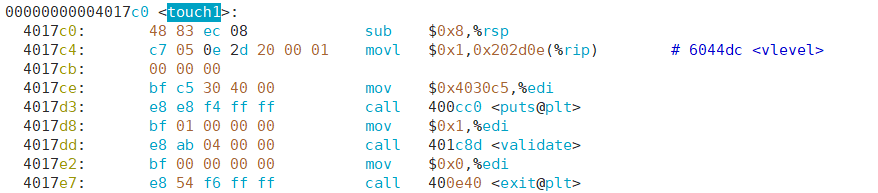
注意我们在执行ctarget程序的时候默认是连接到cmu的服务器，但是我们不是cmu的学生所以连不上服务器也就无法执行代码，所以执行的时候要加命令行参数 -q 以阻止连接到服务器的行为。这里随便填了几个数字，报错。

ctarget的流程大概就是执行test函数，然后输入字符串。这里level 1的要求是调用touch1函数即可。找到ctarget里getbuf函数的反汇编。

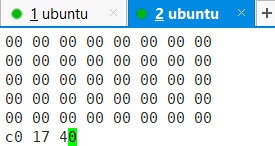


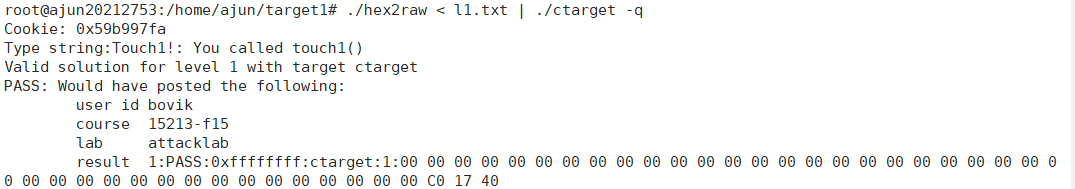
ctarget的流程大概就是执行test函数，然后输入字符串。这里level 1的要求是调用touch1函数即可。

查看touch1的反汇编



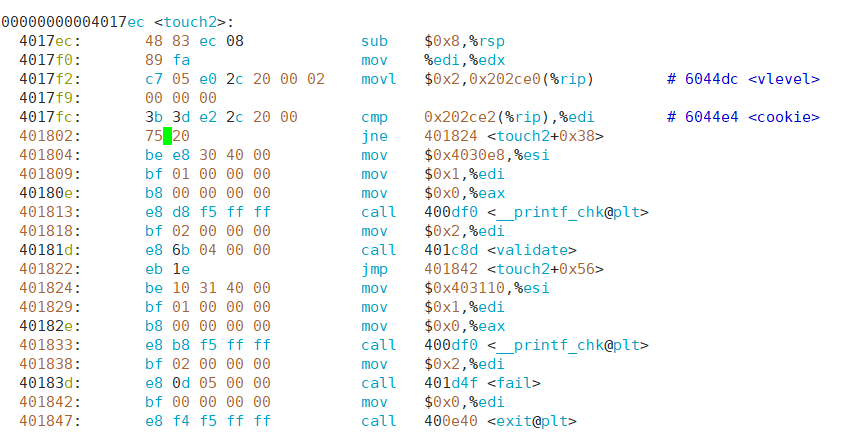
发现touch1的首地址是 0x4017c0，注意我们这里要用小端法，即输入的字符串要写成c0 17 40这样，开辟的40字节空间里面填什么无所谓。最后我们可以填成这样：





### phase2

touch2的反汇编



用touch2即成功。我们看见touch2传了一个参数，这个参数值如果等于cookie的话就成功。cookie的值在给定的文件cookie.txt里有0x59b997fa

函数的第一个参数放在 %rdi 寄存器里面，这里显然不能直接输入字符串，需要借助汇编语言来实现。具体解题思路如下：

将正常的返回地址设置成为注入代码的地址，这次注入直接在栈顶注入，即设置为%rsp

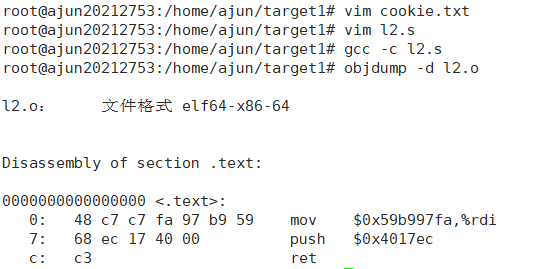
cookie的值写在%rdi里

获取touch2的首地址，这个已经有了

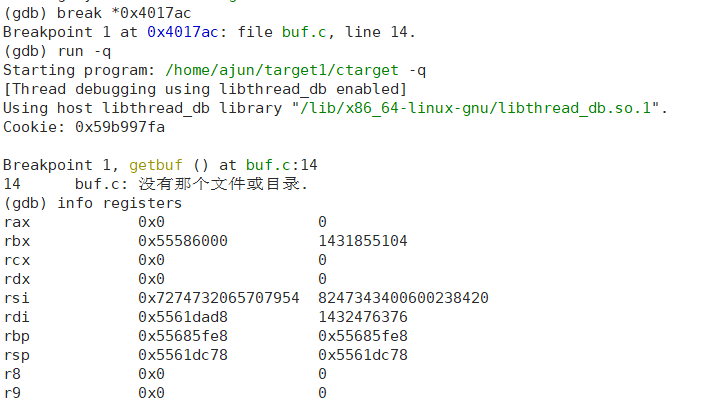
要调用touch2，却不能用call jmp等命令，所以只能用ret弹出，在弹出之前要先把touch2地址压栈。



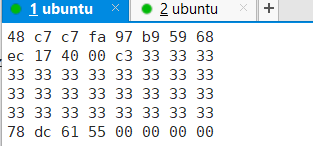
命令行里编译再查看其反汇编

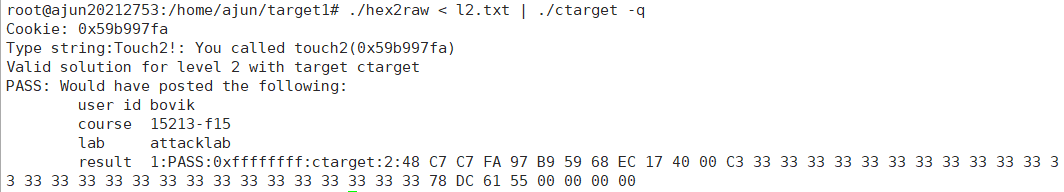


这三条指令地址有了，就是每行反汇编最前面的一长串十六进制数字，接着去找%rsp在哪，借助gdb。



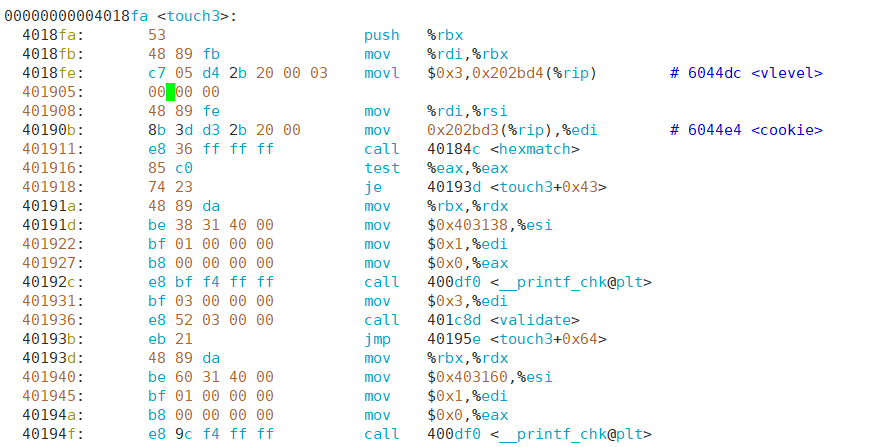
得到%rsp地址，然后写注入字符串就好了，字符串为 注入代码地址——无用字符——返回地址。





### phase3

touch3的代码



这次还是要调用touch3，与前面不同的是，这次传进的参数是一个字符串，同时函数内部用了另外一个函数来比较。本次要 比较的是"59b997fa"这个字符串。

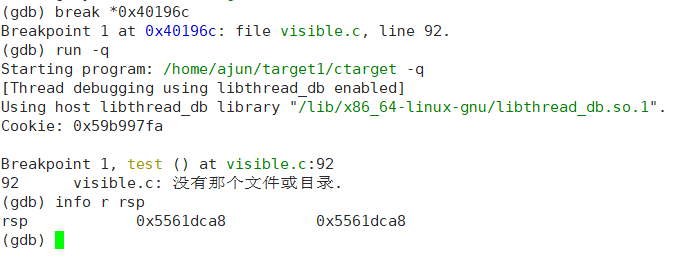
writeup里给了几个提示：

在C语言中字符串是以\0结尾，所以在字符串序列的结尾是一个字节0

man ascii 可以用来查看每个字符的16进制表示

当调用hexmatch和strncmp时，他们会把数据压入到栈中，有可能会覆盖getbuf栈帧的数据，所以传进去字符串的位置必须小心谨慎。

这次与上一次的最大区别就是多了一个函数，hexmatch也开辟了110字节的栈帧，strncmp也会开辟空间，但是就代码来看，\*s存放的地址是随机的，如果我们将数据放在getbuf的栈空间里面，很有可能就被这两个函数cover了。所以我们要把数据放到一个相对安全的栈空间里，这里我们选择放在父帧即test的栈空间里。gdb看一下test栈空间地址。



解题思路：

cookie转化为16进制

将字符串写到不会被覆盖的test栈空间，再将该地址送到%rdi中

将touch3首地址压栈再ret

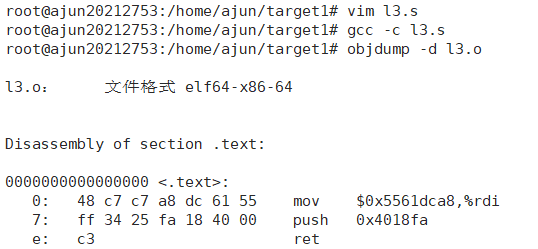
汇编代码

movq $0x5561dca8, %rdi

pushq 0x4018fa

ret

编译查看



使用man ascii命令得到cookie的十六进制

最后的输入文件

