**课 程 实 验 报 告**

**课程名称： 计算机组成原理**

**专业班级： 计算机Ⅱ类2103班**

**学 号： 8008121077**

**姓 名： 陈映融**

**指导教师： 胡勇**

**报告日期： 2022/12/06**

# 实验2： 拆弹实验

## 2.1 实验概述

实验目的：增强对程序的机器级表示、汇编语言、调试器和逆向工程等方面原理与技能的掌握。

实验目标：需要拆除尽可能多的炸弹。

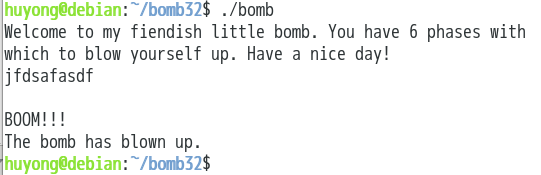
实验要求：使用gdb调试器和objdump来反汇编炸弹的可执行文件，并单步跟踪调试每一阶段的机器代码，从中理解每一汇编语言代码的行为或作用，进而设法“推断”出拆除炸弹所需的目标字符串。

实验语言：c。

实验环境：linux

## 2.2 实验内容

一个“binary bombs”（二进制炸弹，下文将简称为炸弹）是一个Linux可执行C程序，包含了6个阶段（phase1~phase6）。炸弹运行的每个阶段要求你输入一个特定的字符串，若你的输入符合程序预期的输入，该阶段的炸弹就被“拆除”，否则炸弹“爆炸”并打印输出 "BOOM!!!"字样。实验的目标是拆除尽可能多的炸弹层次。



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

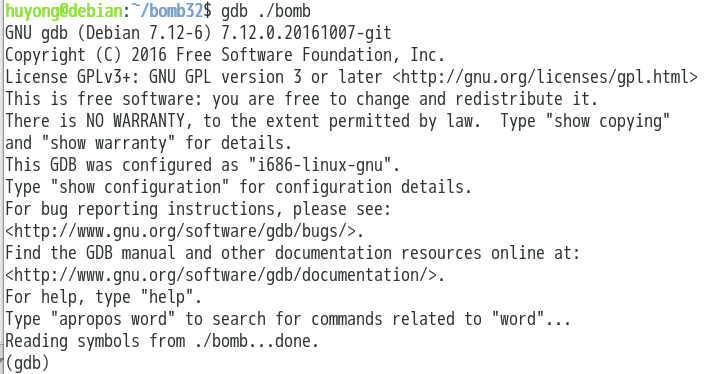
\* 阶段5：指针

\* 阶段6：链表/指针/结构

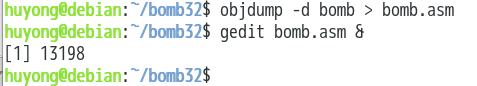
另外还有一个隐藏阶段，但只有当你在第4阶段的解之后附加一特定字符串后才会出现。

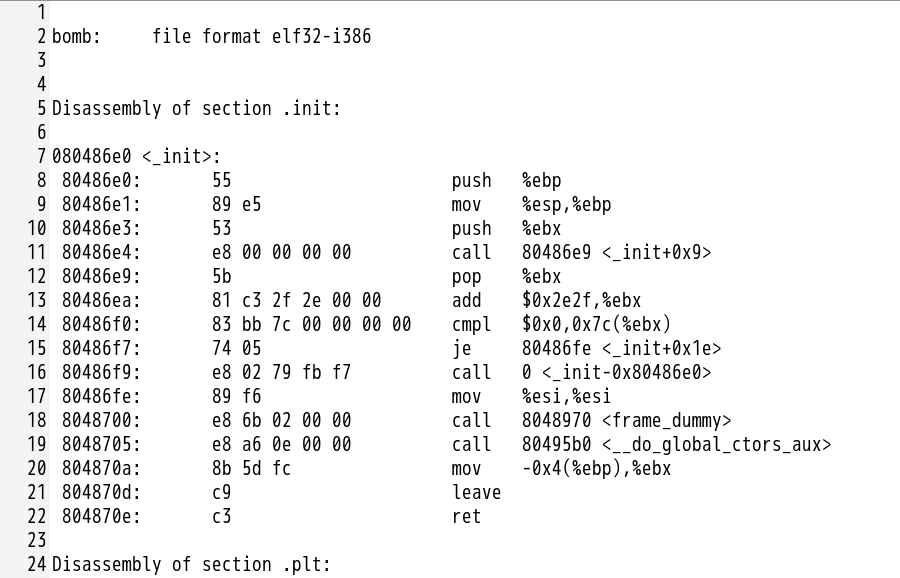
为了完成二进制炸弹拆除任务，你需要使用gdb调试器和objdump来反汇编炸弹的可执行文件，并单步跟踪调试每一阶段的机器代码，从中理解每一汇编语言代码的行为或作用，进而设法“推断”出拆除炸弹所需的目标字符串。这可能需要你在每一阶段的开始代码前和引爆炸弹的函数前设置断点，以便于调试。

通过gdb调试bomb程序：



通过执行以下指令，可以反汇编bomb，并将结果保存到bomb.asm文件中，然后通过gedit或more可以方便地察看汇编代码：





**2.2.1 阶段1 字符串比较**

1. 任务描述：通过phase\_1的反汇编代码找出要输入的字符串。
2. 实验设计：利用gdb结合断点来动态地分析。
3. 实验过程：

观察phase\_1的反汇编代码，如图2.1.1所示：

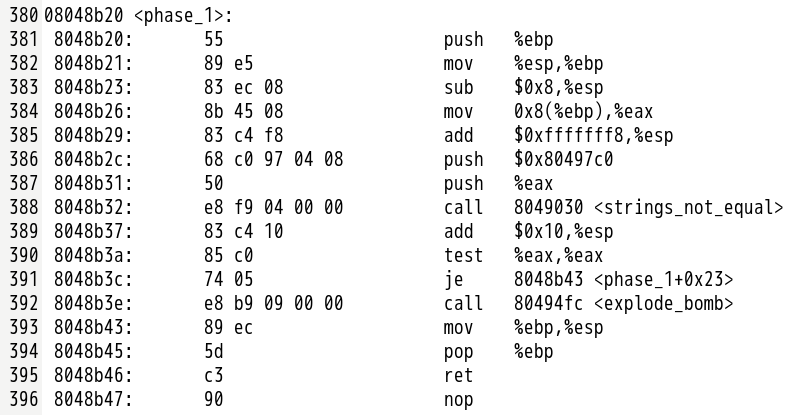


图2.1.1

通过分析，发现在调用strings\_not\_equal对比字符串之前，有两个压栈的指令，其中一个是将函数入参送入栈中，还有一个地址送入了栈，函数入参应该就是输入的字符串所在地址，猜测另一个地址就是正确字符串的首址，于是在phase\_1处下个断点，然后运行，随意输入一个字符，触发断点，再用x命令查看字符串。如图2.1.2所示：

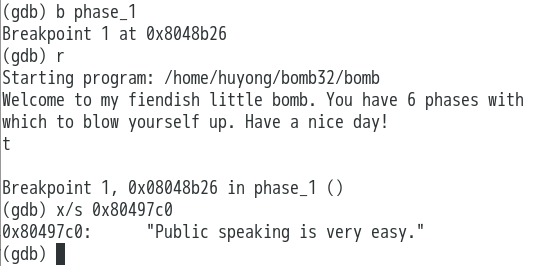


图2.1.2

故猜测“Public speaking is very easy.”就是所需字符串。重新执行该程序，直接输入该字符串，观察结果。如图2.1.3所示：

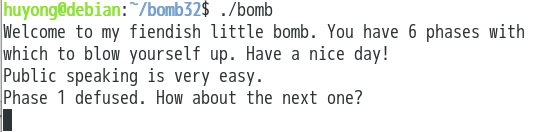


图2.1.3

1. 实验结果：如图2.1.3所示，阶段一拆弹成功！

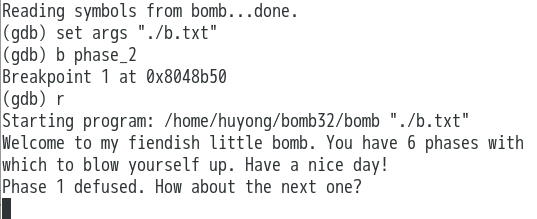
阶段一很简单，找出字符串首地址就可以了，没有什么大的难点。

为了提高调试效率，可以将解析的结果保存到文件中，在调试时以参数方式提供给程序运行，方法如下：

第一步：建立一个参数文件，比如“b.txt”文件，然后用vim或gedit编辑该文件，写入“Public speaking is very easy.”：



然后在gdb中调用该参数文件：



**2.2.2 阶段2 循环**

任务描述：通过phase\_2的反汇编代码推断第二阶段要输入的数据

**2.2.3 阶段3 条件/分支**

任务描述：通过phase\_3的反汇编代码推断第三阶段要输入的数据

**2.2.4 阶段4 递归调用和栈**

任务描述：通过phase\_4以及func4的反汇编代码推断第四阶段要输入的数据

**2.2.5 阶段5 指针**

任务描述：通过phase\_5的反汇编代码推断第五阶段要输入的数据

**2.2.6 阶段6 链表/指针/结构**

任务描述：通过phase\_6的反汇编代码推断第五阶段要输入的数据

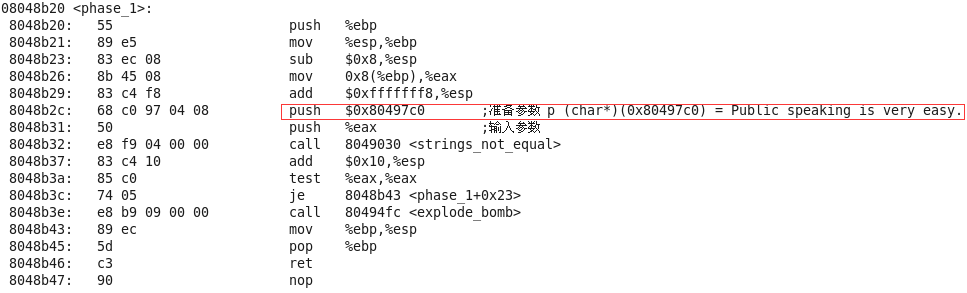
**2.2.7 阶段7 隐藏阶段**

任务描述：找出隐藏阶段开启方式并且拆除隐藏阶段的炸弹。

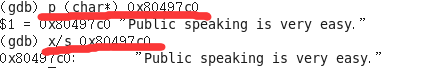
## 2.3实验过程

2.3.1 : 字符串比较

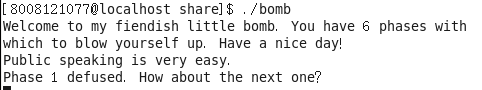
阅读汇编代码，call命令的参数准备阶段用命令查看立即数表示地址



两种查看方法



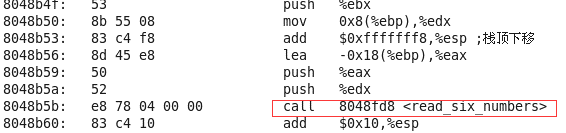
得出答案：Public speaking is very easy.



进入第二关

2.3.2 循环

分析汇编代码有提示读6个数 我们输入6个数字



我们以call 80494fc <explode\_bomb>为突破点开始分析。

ebp为栈底指针同时可以，ebp由下向上为参数1····n

着眼第一次比较的第一个参数。第一个数就是1

往后R[ebx] = 1 后面有R[ebx]++

同时看到寄存器esi参数地址给到了esi

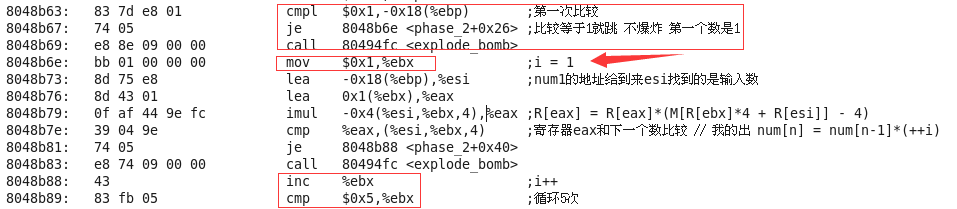
eax比ebx的值大一

最后最难分析的就是imul指令



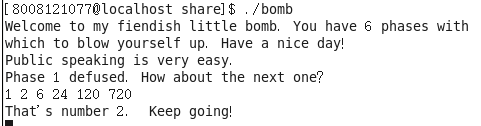
-0x4(%esi,%ebx,4),%eax 第一个源操作找到的是esi指向的输入参数的下一个数（int类型）

得出num[n] = num[n – 1] \* [i + 1]

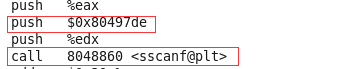


有第一个数是1 后一个数是前一个数\*（++i）

的出答案1 2 6 24 120 720

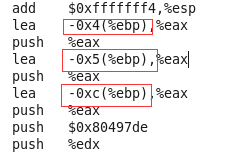


2.3.3 条件分支



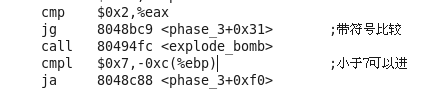
从scanf入手可知调用输入函数，要准备参数。猜测非寄存器值的功能  


查看得知要输入的数据为两整数和一个字符，第二个就是看数据特点了



esp栈顶指针下划了12Byte一个为4B一个为1B最后一个为4B。 字节对齐所以有从C开始进一步验证了数据输入的格式

继续



cmpl加ja可知第一个参数必须要小于7才可以使炸弹不爆炸

下面就是多个重复度很高的代码并且还有一个8位的寄存器bl来存储数据，推测是第二个参数字符参数。后面比较的数据为第三个参数。对数据用p (char) $bl来翻译

有7个数我从1或者0开始试第一个数据答案组合0 q 777 或者1 q 777

结果0 q 777正确 后续从其他重复度高代码可得出其他结果

1 b 214

2 b 755

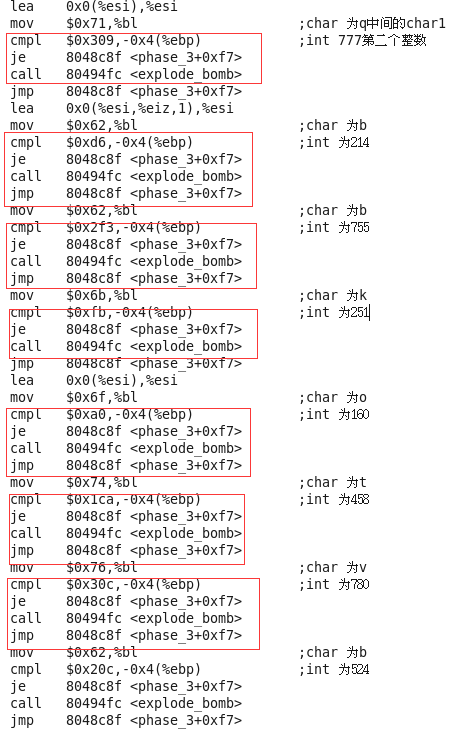
3 k 251

4 o 160

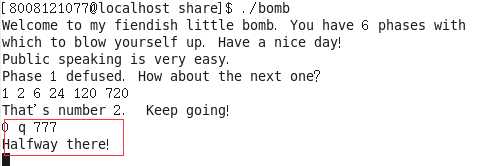
5 t 458

6 v 780

7 b 524



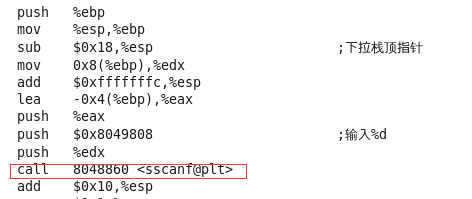
随便选择一个结果为



通过！！！测试其他结果同意也成立。推测C语言代码用的应该是switch case语句

2.3.4 递归

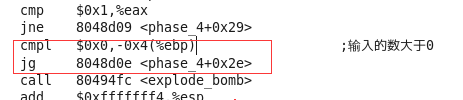
同3我们从输入函数入手从非寄存器参数入手



得出的输入的数据为%d

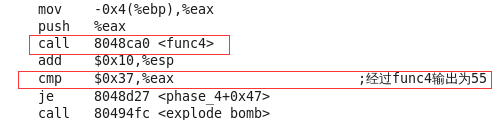


继续看到对于输入数据的限制输入的数据一定要大于0.



对于前一个cmp我是没看懂的经过我ni的单步调试每次比较时R[eax]的值都会变成1前期没有对eax的处理。这个跳转爆炸jne语句不会执行。

从结果入手经过func4的返回值要为55



进入func4函数

Eax经常作为返回值出现，从下图第一个比较跳转指令分析当i<=1时（下图有误）reuturn 1;

递归有两部分一部分是func(i - 1) 另一部分是func(i – 2)从寄存器add %esi % eax可知放回两数和

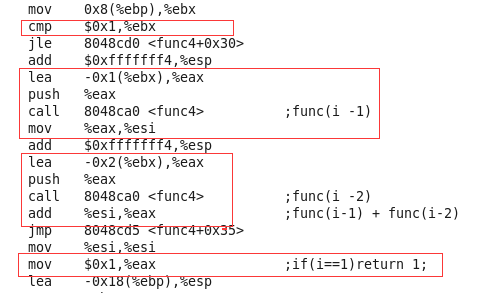
可以等同于下面代码

1. func4(num) {
2. if (i <= 1)
3. return 1;
4. return func4(num-1) + func4(num-2);
5. }

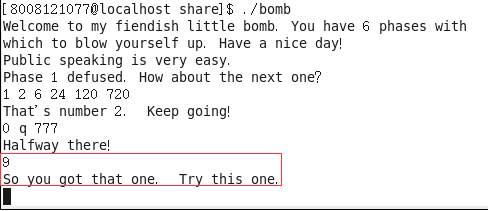
由func4（0）= fun4（1） = 1推出

Func4（9） = 55

其实就是斐波那契数列 。所以第4题的答案为9

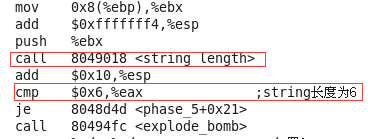


结果图：

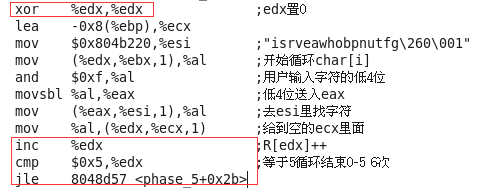


2.3.5 指针

从call string length入手结合返回结果和6比较推测需要输入的是长度为6的字符串



edx自身亦或置零，结合inc %edx推测edx为循环控制遍历i其中i = 0; i <= 5中间就是对数据的操作。我们通过ni还在si逐步分析



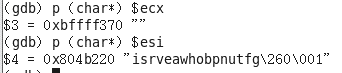


run的时候键入aaaaaa 6个字符保证第一个不爆炸。后面进行单步调试



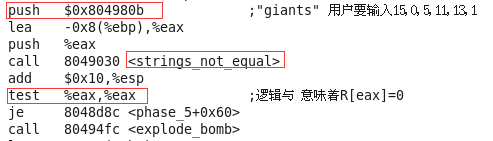


查看两个重要寄存器可知 ecx中是空串用 esi中为程序提供的字符串



看到中间的一系列mov操作可知用户输入的数据通过ebx + edx循环变量访问而且是底4位在去esi中找存在的字符送如ecx中循环结束。

看后续代码



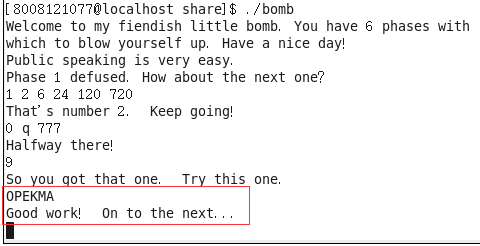
有了字符串比较，函数前面准备的参数中有未知地址推测为需要比较的目标参数。查看得知



推测为目标串，结合test和string not equal可知我们输入的字符串经过操作要变为giants逆推可知我们需要输入的最简单数据为

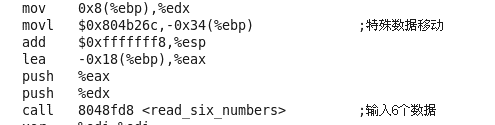
OPEKMA

键入测试

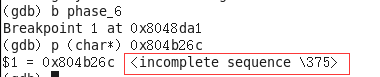


成功

2.3.6 链表

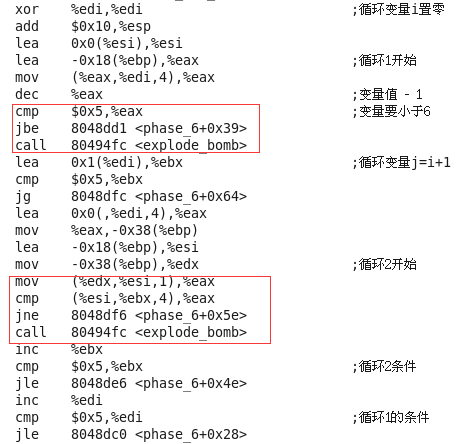


从特殊数据入手。查看可知



结合本题提示为链表

马上对于输入数据的要求数据数据要小于6，输入数据和相邻元素之间有比较也就是相邻元素不可以相等



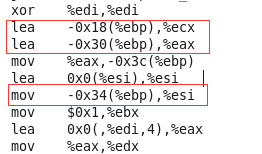
从上可以写出代码

1. for (int i = 0; i<=5; ++i) {
2. if (num[i] >= 6)
3. explode\_bomb();
4. for (int j = i+1; j <= 5; ++j) {
5. if (num[i] == num[j])
6. explode\_bomb();
7. }
8. }

上述是对输入数据的操作。其实到这里就可以代码测试了1-6数字的排类组合不过有A66种结果就是

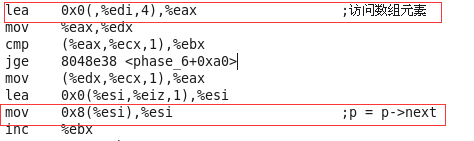
继续

观察汇编代码下面的处理无非就是围绕esi和eax两个寄存器的处理和对于他们指向数据和靠数据偏移量访问的数据。对esi和eax分析



多的不说直接开始调试如下：

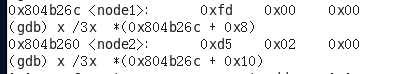
ecx寄存器明显就是输入元素的地址。esi的偏移量大猜测他存储的就是链表数据





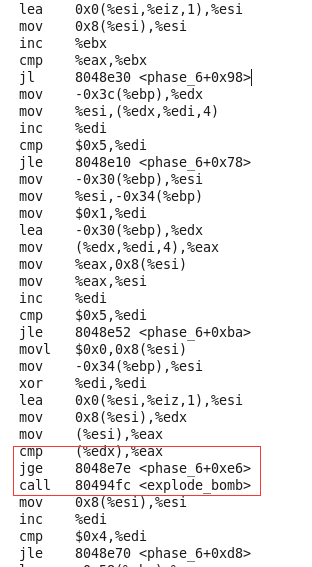
地址可知

当然通过偏移量和对自身的处理也可以猜测esi就是存储链表的节点我们根据+0x8来验证链表的节点



依次访问的到节点的大小fd、2d5、12d、3e5、d4、1b0;

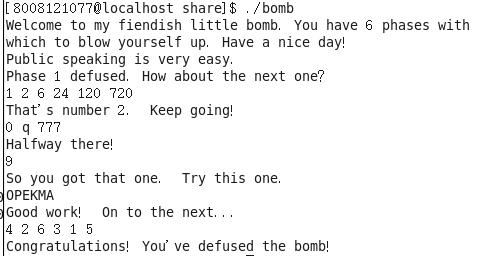
继续分析后续汇编



用call bomb入手比较的是edx 和eax的大小关系换而言之就是链表中数值的关系大小，由上对链表数值排序4 2 6 3 1 5 符合从数值1 – 6 相邻两个没有相同。

键入4 2 6 3 1 5如下

结果：

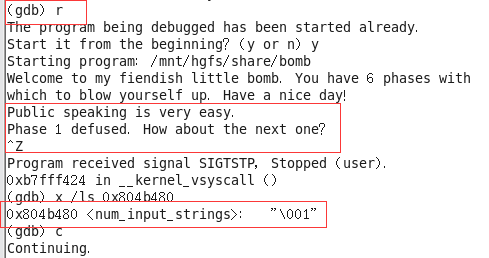


2.3.7 隐藏关卡

到这里./bomb运行的炸弹程序已经结束了，在gdb调试给secret\_phase打上断点也没有停止。猜测代码没有执行。

上网借鉴了一下：现在目光聚焦在phase\_defused上其中第一个cmp就是对6比较推测拆除6个炸弹以后分数会增加。

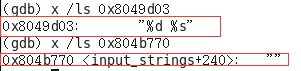
验证：



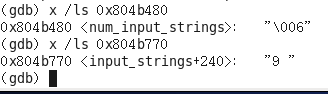
完成一个任务以后由””变成了”\001” \0结束符号推测到6时\0去除



前面5个阶段后0x804b480的值变为了



输入的字符串为空 输入的数据也未知



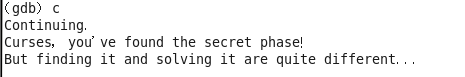
等num\_input\_string到了6查看得到9，这时第4题的答案推测第4题答案加个字符串可以进入隐藏关卡，在这同样也解释了为什么第四关的cmp 1 eax一直成立是在phase\_defused设置的。

看到字符串的首地址为0x8049d09



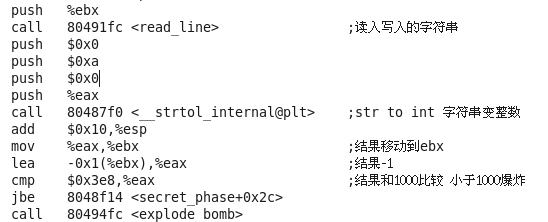
这是我们要的字符串

去阶段4输入9 austinpowers

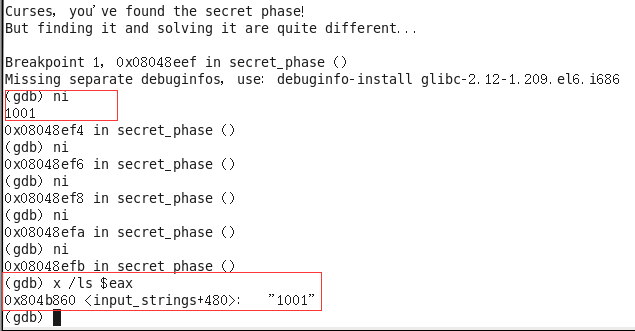


进入隐藏关卡成功

开始分析：

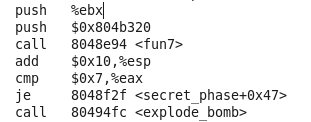


上面的代码都是在限制输入输入一定要大于等于1001

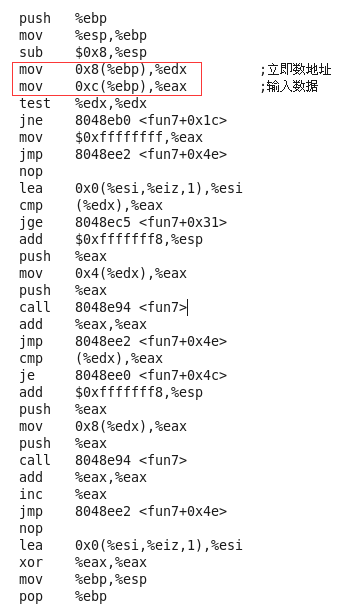


调试输入程序

进入fun7以后要返回值为7



看fun7的代码



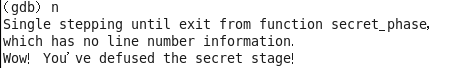
输入的数据和’$’比较并且由递归调用就是为了求

if (R[eax] == R[edx])结束 ：如下代码

1. if(addr <= input){
2. if ( addr == input ) return 0;
3. else 2\*fun7(addr+8,input)+1;
4. }

得到1001可以

结果：键入1001



## 2.4实验小结

本次实验可以说是有难度的，但是一旦理解了其中的原理可以全面的理解计算机系统基础的前4章。从数据的存储到位运输，再到不同位数的寄存器结合到数据的存储。特别是进程调用的理解。原先不理解的也很好多完事了。

同样重要的就是gdb调试。如果你不懂哪个寄存器或者哪个地址什么用。调试他ni或是si单步调试。但是ni和si也有不同点si会进入更加细致的文件（如.o文件）。如果你有什么不会调试你就什么都知道了。

总体而言本次实验我学习到了很多，是对前半个学期很好的总结和复习。相比实验1实验2更加全面，细致。当然本实验的难点—隐藏关卡的寻找我借鉴了网络资源。通过理解实现了实验。

实验同样也包含了多个寄存器辨识，和条件转移指令的记忆。