

**实验报告**



**题目： 拆解二进制炸弹**

**班 级： 2021211321**

**学 号： 2021212171**

**姓 名： 杨晨**

**学 院： 计算机学院**

**2022年 11月 7日**

一、实验目的

1.理解C语言程序的机器级表示。

2.初步掌握GDB调试器的用法。

3.阅读C编译器生成的x86-64机器代码，理解不同控制结构生成的基本指令模式，过程的实现。

1. 实验环境
2. 服务器
   1. 服务器：10.120.11.12（X86版本）：**2022年11月10日23:59分关机（二周）；提交作业截止时间2022年11月13日00：00，延迟扣分（每晚一天扣1.5分）**
   2. 服务器：10.99.0.230（华为arm版本）：**2022年11月25日23:59分关机（三周）；提交作业截止时间2022年11月27日00：00，延迟扣分（每晚一天扣1.5分）**
   3. 服务器密码找回指南：<https://khbgo05wz2.feishu.cn/docx/doxcnioT927j2gDSTwHoAQLGVpg>
3. MobaXterm
4. Linux
5. Objdump命令反汇编
6. GDB调试工具
7. 积分榜（http://10.120.11.13:19240/scoreboard）
8. 实验内容

四、实验步骤及实验分析

建议按照：准备工作、阶段1、阶段2、…等来组织内容

各阶段需要有操作步骤、运行截图、分析过程的内容

**准备工作**

首先新建一个文件夹，命名为lab2，将bomb.tar放入lab2中（**mv bomb.tar lab2**）

然后解压文件（**tar -xvf bomb556.tar**），得到文件夹bomb556

进入文件夹bomb556（**cd bomb556**），文件夹中有3个文件，可执行文件，c代码文件，readme文件

查看README文件（**cat README**）,确认炸弹属于bupt2021212171

文本

描述已自动生成

文本

描述已自动生成

用vi编辑器，创建一个answer，用以存储答案（**vi answer**）



总览bomb.c，发现有6个阶段，分别位于phase\_1，phase\_2……phase\_6几个函数中；

文本

描述已自动生成

观察最后一句，推测还存在隐藏关卡

图片包含 文本

描述已自动生成

1. **拆解phase\_1**

首先进入gdb调试阶段（**gdb bomb**）

文本

描述已自动生成

在第一阶段设置断点（**b phase\_1**）

在爆炸处设置断点（**b explode\_bomb**）

文本

描述已自动生成

反汇编phase\_1函数（**disas phase\_1**）

图形用户界面, 文本

描述已自动生成

观察函数strings\_not\_equal，猜测是比较字符串是否相等的函数

若函数返回值是1，则会引爆炸弹；若返回值是0，则成功拆除，因此接下来分析strings\_not\_equal函数

反汇编strings\_not\_equal函数（**disas strings\_not\_equal**）

文本

描述已自动生成

对应翻译为c代码

电脑萤幕画面

描述已自动生成

所以可以确定，这个函数就是判断两个字符串是否相等的。

第一个参数为标准答案，第二个参数为输入的字符串，只需要查看第一个字符串的内容，就可以知道第一阶段的答案了

观察phase\_1函数，可以发现第一个参数是存储在0x4026b0的字符串，将其输出（**x /s 0x4026b0**）



得到第一阶段的答案

打开answer文件（**vi answer**）

存入第一阶段的答案（**cata I am just a renegade hockey mom.**）

退出，保存（**Esc :wq**）

查看是否保存成功（**cat answer**）



重新进入gdb调试（**gdb bomb**）

输入答案（**r answer**）

第一阶段成功拆除

文本

描述已自动生成

* 1. **拆解phase\_2**

在爆炸处设置断点（**b explode\_bomb**）

在第二阶段设置断点（**b phase\_2**）

文本

描述已自动生成

反汇编函数phase\_2（**disas phase\_2**）

图形用户界面, 文本

中度可信度描述已自动生成

注意到函数read\_six\_numbers，反汇编这个函数（**disas read\_six\_numbers**）

图形用户界面, 文本

描述已自动生成

在read\_six\_numbers函数中调用了sscanf函数，int sscanf(char \*input, char \*format, arg1, …)用于将input字符串按照format模式串的形式，将字符输出到指定的变量地址，并返回成功输入的参数个数

查看模式串中第二个参数%esi=0x4029c1（**x /s 0x4029c1**）

文本

描述已自动生成

所以第二阶段应该输入6个int类型的整数，分别存在rsp+0,rsp+4,rsp+8,rsp+12,rsp+16,rsp+20

手机屏幕的截图

中度可信度描述已自动生成

由此处的汇编代码，容易看出，前两个数字是0,1

黑板上的文字

描述已自动生成

汇编代码化简，可以得出（%rbx）+(%rbx+4) == (%rbx)+8，即前两个数字相加要等于第三数字

同时这是一个循环结构，因此这是一个斐波那契数列





可得第二题的答案 0 1 1 2 3 5

存入answer,运行

第二阶段成功拆除

文本

描述已自动生成

* 1. **拆解phase\_3**

在爆炸处设置断点（**b explode\_bomb**）

在第三阶段设置断点（**b phase\_3**）

图形用户界面, 文本

描述已自动生成

反汇编phase\_3函数（**disas phase\_3**）

图片包含 图形用户界面

描述已自动生成

图形用户界面, 文本, 应用程序, 聊天或短信

描述已自动生成

观察到phase\_3中同样调用了sscanf，第二个参数rsi在调用前被赋值0x4029cd

gdb中查看0x4029cd的值（**x /s 0x4029cd**）

文本

描述已自动生成

因此阶段三应该输入两个int类型的整数

注意到*cmpl $0x7,(%rsp)*，因此第一个整数不能大于7

后面有一个关键语句*jmpq \*0x402700(,%rax,8)*，由于rax为第一个整数，因此这个指令代表跳转到地址（0x402700 + 8 \* rax）中存放的地址（代码）

这里的功能类似于switch语句

查看此处的的跳转代码（**x /8xg 0x402700**）显示8个单元，每个单元占8字节，十六进制显示

图形用户界面, 文本, 应用程序, 聊天或短信

描述已自动生成

不妨就让第一个输入的数字为0，接下来则跳转到0x400ff5地址（代码）处

eax = 0;

eax = 626

eax += 555

eax -= 657

eax += 657

eax -= 657

eax += 657

eax -= 657

最终eax = -465

观察可知，第二个输入的整数必须和eax相等；

同时，后面还有限制条件，需要保证第一个整数不能大于5，否则会爆炸

电脑屏幕的照片上有文字

中度可信度描述已自动生成

因此，第三阶段的答案是0 -465

将答案保存到answer中，运行

第三阶段成功拆除

文本

描述已自动生成

1. **拆除phase\_4**

在爆炸处设置断点（**b explode\_bomb**）

在第四阶段设置断点（**b phase\_4**）

图形用户界面, 文本

描述已自动生成

反汇编phase\_4函数（**disas phase\_4**）

图形用户界面, 文本

描述已自动生成

发现也调用了sscanf函数

查看0x4029cd，发现第四个阶段需要读入两个整数

此处和其他地方有些不同，第一个数存储在rsp+4里，第二个数存储在rsp里

文本

描述已自动生成

第二个数字在减2后，要小于等于2

图片包含 图形用户界面

描述已自动生成

之后进入函数func4，func4有两个参数，第一个参数是rdi初始值为5，第二个参数是我们输入的第二个数字

观察后续代码可知，要让函数的返回值等于输入的第一数字

接下来进入func4函数分析（**disas func4**）

图形用户界面, 文本

描述已自动生成

改写为c代码，即

屏幕上有字

描述已自动生成

在c语言编译器中，将t1,t2参数分别置为5,2，得到返回值是24

因func里的参数只和第二个数字有关，不妨令输入的数字是1和2

将func执行后的返回值输出，是24，验证了还原的func4无误

图形用户界面, 文本

描述已自动生成

综上，若第二个数字是2，那么第一个数字必须是24

第四阶段的一组答案即为24 2

将答案保存到answer中，运行

第四阶段成功拆除

文本

描述已自动生成

1. **拆解phase\_5**

在爆炸处设置断点（**b explode\_bomb**）

在第五阶段设置断点（**b phase\_5**）

文本

描述已自动生成

反汇编phase\_5函数（**disas phase\_5**）

图形用户界面

中度可信度描述已自动生成

发现调用了函数sscanf

查看0x429cd的值（**x /s 0x4029cd**）

文本

描述已自动生成

要求输入2个整数

往下分析，发现第一个数字只取低四位

文本

描述已自动生成

注意到*cmp $0xf,%eax*，因此，第一个数字的低四位不能全为1

接下来是一个循环，是每次以eax的值为下标，用数组a（地址为0x402740）的值替换eax，同时用ecx记录这些数字的和

文本

描述已自动生成

所以我们需要查看这个数组内的值（**x /16dw 0x402740**）输出16个单元，每个单元占4字节，十进制显示

图形用户界面

低可信度描述已自动生成

从*cmp $0xf,%edx*可以看出，循环次数必须15次，而循环结束的条件是eax==15

分析知，eax最开始必须为5,即第一个数字必须是5

图片包含 图示

描述已自动生成

第二个数字必须和ecx相等，而ecx=12+3+7+11+13+9+4+8+0+10+1+2+14+6+15=115

得第二个数字115

将答案存入answer,运行

第五阶段成功拆除

文本

描述已自动生成

1. **拆解phase\_6**

在爆炸处设置断点（**b explode\_bomb**）

在第六阶段设置断点（**b phase\_6**）

反汇编phase\_6函数（**disas phase\_6**）

文本

描述已自动生成

观察到调用了函数read\_six\_numbers，这个函数在phase\_2中出现过，这里同理

输入6个int类型的整数，分别存在rsp+0,rsp+4,rsp+8,rsp+12,rsp+16,rsp+20

接下来是一个双层循环，要保证第一个数字减1后要小于等于5（无符号小于等于，因此隐含条件是这个数字减一后不能为负数）

第二个数字不和第一个数字相等，第三个数字不和第二个数字相等……即六个数字要各不相同

下面%r13保存栈顶指针，+4后又跳回去，即要求第二个数字减1后要小于等于5，之后要判断有没有相等的数字；%r13+4后又跳回去，要求第三个数字减1后要小于等于5……

故，6个数字要在区间[1, 6]，且各不相同

图片包含 文本

描述已自动生成

对应的c代码为

文本

描述已自动生成

接下来是一个小循环，将a[i]变为7-a[i]

电脑屏幕的照片上有文字

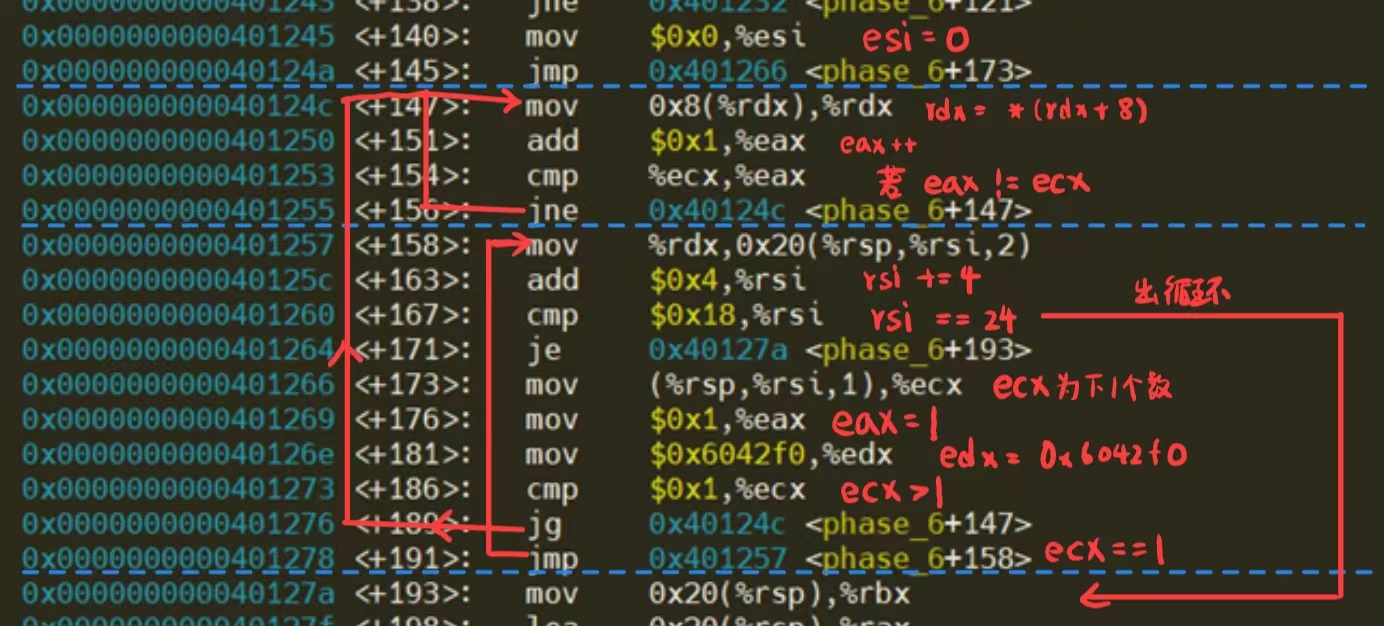
低可信度描述已自动生成

写成c代码就是

黑色的钟表

描述已自动生成

第三阶段是一个二重循环，较为复杂



注意到mov 0x8(%rdx),%rdx，每次移动8个字节，因此猜测这是个结构体，且0x8为该结构体指针偏移量

用gdb查看0x6042f0内的值

手机屏幕的截图

描述已自动生成

从图中看出，0~3字节是一个数字，4~7字节是节点编号，8~15字节一个是指针，打印可以发现，6308608指向node2，同样node2中的指针指向node3……可以推测出，这个结构体是一个链表

手机屏幕的截图

描述已自动生成

大致可以确定，这个结构体为

图形用户界面, 文本

描述已自动生成

再将上述汇编代码转为c代码

电脑萤幕画面

描述已自动生成

简化c代码为

文本

描述已自动生成

接下来进入下一个循环

电脑屏幕的照片上有文字

中度可信度描述已自动生成

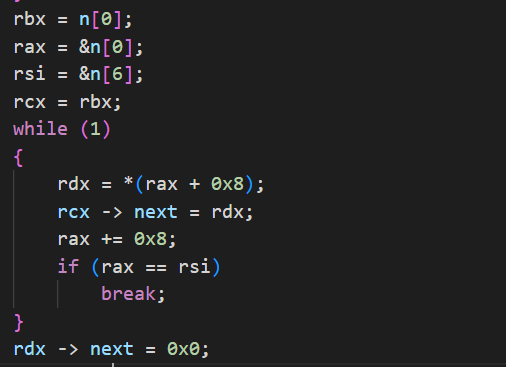
第一次循环开始，rcx=rbx=n[0]，rax=&n[0]，rdx=n[1]，让rcx->next=rdx，即n[0]->next=n[1];

第二次循环，rcx=n[1]，rax=&n[2]，rdx=n[2]，让n[1]->next=n[2]

……

最后一次循环时，rcx=n[4]，rax=&n[5]，rdx=n[5]，让n[4]->next=n[5]，再将n[5]的next指向地址0，结束

对应的c代码为



显然，这里是按照数组n的顺序去修改链表的next，使得链表按照数组的元素顺序，一一对应重新构成一个链表

最后一个循环的逻辑很清晰，即排序需要按照val递减，否则爆炸

电脑屏幕的照片上有文字

中度可信度描述已自动生成

从之前的分析，我们知道val的值依次为55,880,499,878,780,477

从大到小，排序为6，1，4，2，3，5

由于之前做了一次a[i]=7-a[i]的操作

所以phase\_6的答案应该为5 3 2 4 1 6

将答案保存到answer中，运行

phase\_6成功拆除

文本

描述已自动生成

1. **拆解secret\_phase**

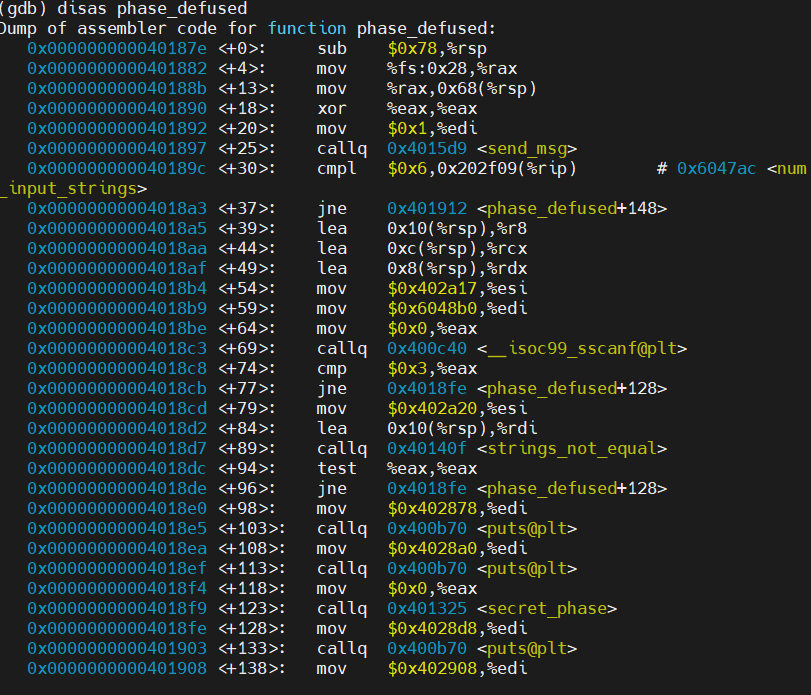
在爆炸处设置断点（**b explode\_bomb**）

在第隐藏阶段设置断点（**b phase\_defused**）

文本

描述已自动生成

反汇编phase\_dufused函数（**disas phase\_defused**）



注意到语句cmpl $0x6,0x202f09(%rip)，所以需要将前六个阶段都拆完才可以进入隐藏阶段

注意到调用了函数strings\_not\_equal，所以进入隐藏阶段需要输入一个字符串

查看0x402a20存储的字符串内容（**x /s 0x402a0**）

文本

描述已自动生成

因此进入隐藏阶段的钥匙是“DrEvil”

通过在phase\_defused处设置断点，每次执行的时候看0x6048b0的值，发现这里是第4个输入的存储区，因此在第四个输入后面，需要输入DrEvil



成功进入隐藏阶段

文本

描述已自动生成

反汇编secret\_phase函数（**disas secret\_phase**）

图形用户界面, 文本, 应用程序

描述已自动生成

发现调用了函数fun7，且函数的返回值必须是4

函数的第一个参数存储在0x604110中，查看0x604110（**x /d 0x604110**）

文本

描述已自动生成

确定了第一参数是36，容易看出，第二个参数就是我们输入的值

接下来反汇编函数fun7（**disas fun7**）

图形用户界面, 文本, 应用程序, 聊天或短信

描述已自动生成

fun7函数自身调用自身，是一个递归函数

对应的c代码为

电脑的屏幕截图

描述已自动生成

容易看出，这是一个二叉树结构，根节点下标是0，访问左节点乘二，访问右节点乘二加一，当在二叉树树中找到数据时，返回二叉树下标

因为我们想要函数的返回值为4，根据分析

4 = 2 \* 2

2 = 1 \* 2

1 = 0 \* 2 + 1

所以，需要向左找两次，再向右找一次

通过设置不同参数，进行试验，可以画出二叉树的结构

图示

描述已自动生成

所以答案应该是7

将答案保存到answer中，运行

隐藏阶段成功拆除

文本

描述已自动生成

五、总结体会

总结心得（包括实验过程中遇到的问题、如何解决的、过关或挫败的感受、实验投入的时间和精力、意见和建议等）

在最开始实验的时候，会逐行逐句地看每一条指令，分析每一条的含义，这导致了我会在某一条指令上卡很久，后来发现这其实是没有必要的，很多语句是对栈指针进行操作，而非是真正的数据，这样的语句不用太过纠结。

在最初的时候，对于gdb语句不够熟悉，导致炸弹爆炸了一次，后来重新复习了gdb调试的相关语句，对整个过程的把控也更加稳定了

分析过程中可以根据跳转指令将程序分段，判断出循环部分、分支判断部分，这样分割可以快速判断出这段汇编代码的作用

分析函数时遇到看到不懂的地方就先向下看，根据后面的逻辑反推前面的逻辑，大胆的猜测，可以帮助我们更好地理解函数。

递归函数和循环较为复杂，这时，将对应的汇编代码转换为C语言伪代码或者代码，借助C语言编译器帮助我们寻找答案

拆解二进制炸弹的过程是对汇编指令的一次熟悉，在这次实验里，我对跳转指令，赋值指令，比较指令等等都有了更深入的理解，并且初步具备了举一反三的能力。

六、诚信声明（不签扣10分）

需要填写如下声明，并在底部给出手写签名的电子版。

在完成本次实验过程中，我曾分别与以下各位同学就以下方面做过交流：

1、简单描述交流内容，例如：来自\*\*\*的建议，采用\*\*\*方式\*\*\*

2、

此外，我还参考了以下资料：

1. 网址等
2. [Linux 命令大全 | 菜鸟教程 (runoob.com)](https://www.runoob.com/linux/linux-command-manual.html)
3. [C 库函数 – sscanf() | 菜鸟教程 (runoob.com)](https://www.runoob.com/cprogramming/c-function-sscanf.html)

在我提交的程序中，还在对应的位置以注释形式记录了具体的参考内容。

我独立完成了本次实验除以上方面之外的所有工作，包括分析、设计、编码、调试与测试。

我清楚地知道，从以上方面获得的信息在一定程度上降低了实验的难度，可能影响起评分。

我从未使用他人代码，不管是原封不动地复制，还是经过某些等价转换。

我未曾也不会向同一课程（包括此后各届）的同学复制或公开我这份程序的代码，我有义务妥善保管好它们。

我编写这个程序无意于破坏或妨碍任何计算机系统的正常运行。

我清楚地知道，以上情况均为本课程纪律所禁止，若违反，对应的实验成绩将按照0分计。

（签名）卡通人物

低可信度描述已自动生成