

**课 程 实 验 报 告**

**课程名称： 计算机系统基础实验**

**实验名称： 二进制程序分析**

**院 系 ：计算机科学与技术**

**专业班级 ： CS2205**

**学 号 ： U202215510**

**姓 名 ： 徐新飏**

**指导教师 ： 李专**

**2024年 4 月 7 日**

**一、实验目的与要求**

通过逆向分析一个二进制程序（称为“二进制炸弹”）的构成和运行逻辑，加深对理论课中关于程序的机器级表示各方面知识点的理解，增强反汇编、跟踪、分析、调试等能力。

实验环境：Ubuntu，GCC，GDB等。

**二、实验内容**

作为实验目标的二进制炸弹（binary bombs）可执行程序由多个“关”组成。每一个“关”（阶段）要求输入一个特定字符串，如果输入满足程序代码的要求，该阶段即通过，否则程序输出失败。实验的目标是设法得到得出解除尽可能多阶段的字符串。

为了完成二进制炸弹的拆除任务，需要通过反汇编和分析跟踪程序每一阶段的机器代码，从中定位和理解程序的主要执行逻辑，包括关键指令、控制结构和相关数据变量等等，进而推断拆除炸弹所需要的目标字符串。

实验源程序及相关文件：

bomb.c 主程序

phases.o 各个阶段的目标程序

support.c 完成辅助功能的目标程序

phases.h support.h 公共头文件

**阶段1： 串比较 phase\_1(char \*input);**

要求输出的字符串(input) 与程序中内置的某一特定字符串相同。提示：找到与input串相比较的特定串的地址，查看相应单元中的内容，从而确定input 应输入的串。

**阶段2：循环 phase\_2(char \*input);**

要求在一行上输入 6个整数数据，与程序自动产生的 6个数据进行比较，若一致，则过关。提示：将输入串input拆分成 6个数据由函数 read\_six\_numbers(input, numbers) 完成。之后是各个数据与自动产生的数据的比较，在比较中使用了循环语句。

**阶段3：条件分支 phase\_3(char \*input);**

要求输入一个整数数据，该数据与程序自动生成的 一个数据比较，相等则过关。提示：在自动生成数据时，使用了 switch … case 语句。

**阶段4：递归调用和栈 phase\_4(char \*input);**

要求在一行中输入两个数，第一个数表示在一个有序的数组（或者binary search tree）中需要搜索到的数，该数是在一定范围之内的；第二个数表示找到搜索数的路径（在树的左边搜索编码为二进制位0，在树的有边搜索编码为二进制位1）。

**阶段5：指针和数组访问 phase\_5(char \*input);**

要求在一行中输入一个串，该串与程序自动生成的串相同。在生成串和比较串时，使用了数组和指针。

**阶段6：链表、结构、指针的访问 phase\_6(char \*input);**

要求在一行中输入6个数，这6个数是一个链表中结点的顺序号（从 1 到6）。按照输入的顺序号，将对应链表结点中的值形成一个数组。若该数组是按照降序排列的，则过关。

**三、实验记录及问题回答**

**（1）实验任务1 的实验记录**

1.串比较

objdump -d bomb > bomb.s获得bomb程序的汇编代码并保存于文本文件bomb.s中供分析。

寻找phase\_1的反汇编代码，使用gdb bomb开始调试，b phase\_1设置断点，r开始运行程序，输入一个字符串。

在breakpoint1停下后，注意到反汇编代码中调用了string\_not\_equal函数,如图1-1。

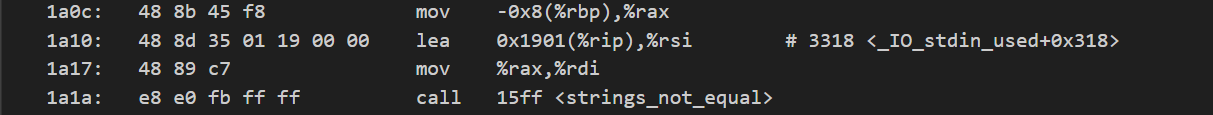


图1-1 调用string\_not\_equal

调用strings\_not\_equal函数前注意到将值移动到了rsi和rdi中，可以推理得知strings\_not\_equal函数是将两个寄存器中的值进行相互比较来决定是否通过第一关。通过gdb分别访问两个地址中的值，可以得知rdi中是自己输入的字符串，rsi是phase\_1任务中预设的字符串，如图1-2，将其存储下来作为第一关的答案。

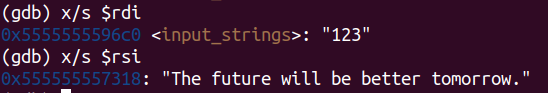


图1-2 string\_not\_equal的参数

答案：The future will be better tomorrow.

2.循环

输入第一关的答案后进入第二关，能够注意到phase\_2的反汇编代码中存在一个循环(1a72-1a9d)

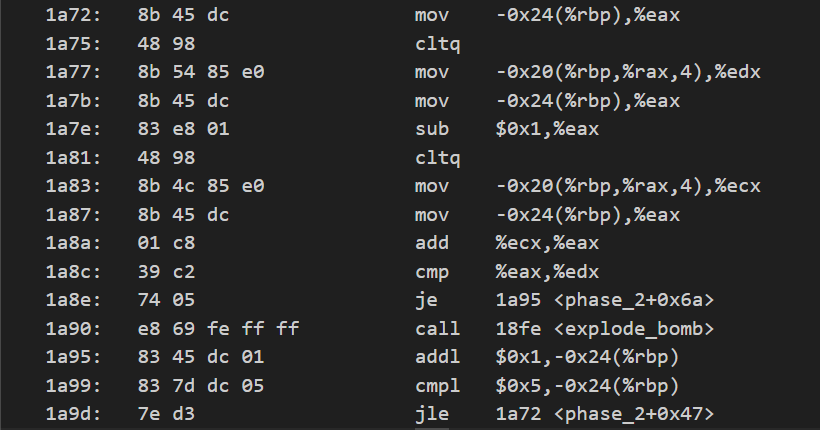


图2-1 循环部分

在循环语句中-0x24(%rbp)是一个循环计数器，在小于等于5时选择继续进行循环。由循环体中语句可以判断应该是一个比较累加和的操作，注意到mov -0x24(%rbp),%eax以及后面的cmp语句，即可以推理出eax的值是当前的循环轮数。

使用(gdb) display/i $pc显示当前执行到哪一条汇编语句，设置断点停在add %ecx,%eax语句时，可以查看此时寄存器ecx和eax中的值，继续步进，查看edx中的值。

通过追踪ecx和edx的反汇编代码，可以发现其都是从-0x20(%rbp,%rax,4)中来，那么查看其中的值便是该任务的关键。注意到eax是rax的低32位，那么在sub $0x1,%eax语句执行后即将rax的值减小了1，由于数组地址是连续的，由基址加变址寻址的方法，-0x20(%rbp,%rax,4)表示-0x20+rbp+rax\*4，rax减小1，地址减小4个字节，如图2-2。经过测试可以得出，edx即为ecx对应数组中的下一个元素，eax+ecx存储到eax中并与edx比较是否相同，如果相同即可通过测试。

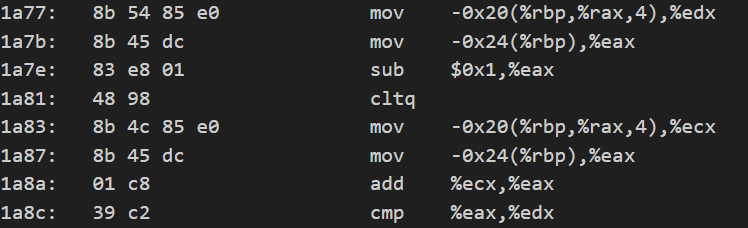


图2-2 取数和比较

根据以上推理，考虑该任务的数学形式，将输入假设为n1至n6,可以通过累加语句以及eax循环计数的信息得出以下公式：

答案为:(任意输入n1)n1 n1+1 n1+3 n1+6 n1+10 n1+15(例如:2 3 5 8 12 17)

3.条件分支：

对于\_\_isoc99\_sscanf@plt，根据其内部调用了解到，其接收格式为“%d %d”，即需要输入两个整数。

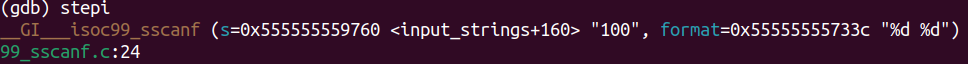


图3-1 输入格式

随后的cmpl语句即是比较第二个返回值（输入的数据的数量）是否大于1，如果不是则调用 explode\_bomb 函数。



图3-2 输入数据个数

如果第二个返回的值大于1，则根据第一个输入的值作为索引，如果其的值<=7，那么将不会跳转，继续执行接下来的语句，类似于c语言的switch-case语句。



图3-3 索引值

根据你输入的数0-7，根据。根据读取的值进行不同的跳转：

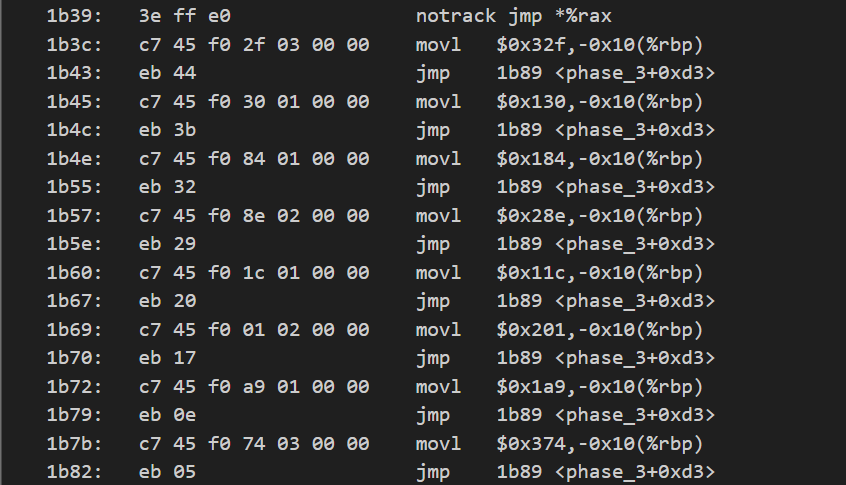


图3-4 switch-case跳转

如果读取的值不在上述范围内，则调用 explode\_bomb 函数。

最后，检查第二个输入的值与存储在栈上的值是否相等，如果相等则执行 nop 操作，否则调用 explode\_bomb 函数，随后清理栈帧并返回。



图3-5 比较值

答案为:0 815,1 304,2 388,3 654,4 284,5 513,6 425,7 884

4.递归调用和栈

同上的思路，首先查看输入的格式



图4-1 输入格式

得知要求输入两个整数，sscanf会返回输入整数的个数，如果为2，程序继续运行。



图4-2 输入数据个数

-0x1c+rbp的值为你输入的第一个数，根据后面比较的语句，要求其的值大于等于0，小于等于0xe，即十进制下的14，更改输入为13 5后继续测试。

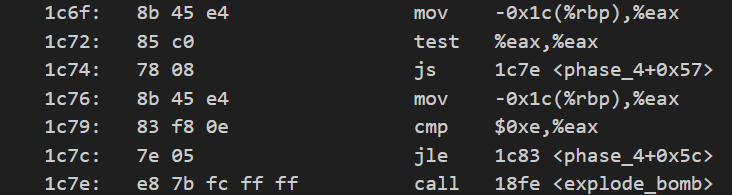


图4-3 输入数据范围

接下来程序准备给函数fun4传参，传入参数为14，0以及由用户输入的第一个数。

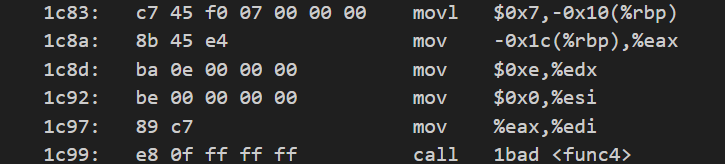


图4-4 func4传参

观察函数fun4的反汇编代码：

首先，函数从edi、esi和edx寄存器中读取参数，接着，从rbp-0x1c(valmax)处读取edx的值存入eax，将其减去ebp-0x18(valmin)处的值，即edx-esi，这个结果的在本次操作中为14-0。

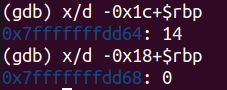


图4-5 参数运算

然后将结果逻辑右移31位以计算符号位，再将结果加到eax中，并对eax进行算术右移一位，结果相当于除以2，将结果存储到edx中。这一段的功能是计算(((valmax - valmin) >> 31) +(valmax - valmin))/2。

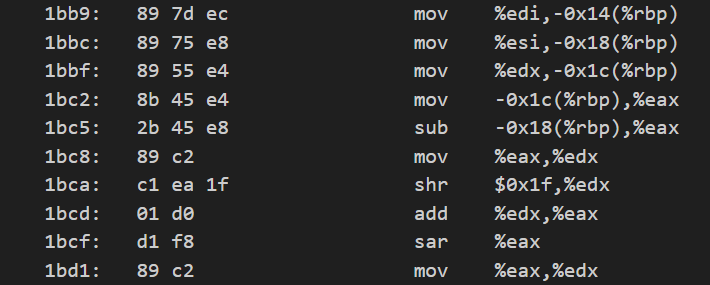


图4-6 参数运算

然后，它将ebp-0x18（即valmin）的值加到eax（此时为valmid）中，将结果存储到ebp-0x4处，比较ebp-0x14（为用户输入的第一个数）和ebp-0x4处的值。如果ebp-0x4(mid)处的值小于等于ebp-0x14(x)处的值，它会跳转到标签 1bfc，否则继续执行下面的代码。

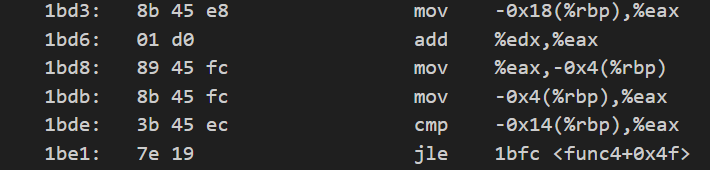


图4-7 mid与x比较

接下来先分析mid>x的情况，此时eax中存储着mid，将rax-1(即是eax-1)赋值给edx，edx中的值为mid-1。随后将参数修改为x,min,mid-1传入func4，将返回值乘2后存储到eax中。

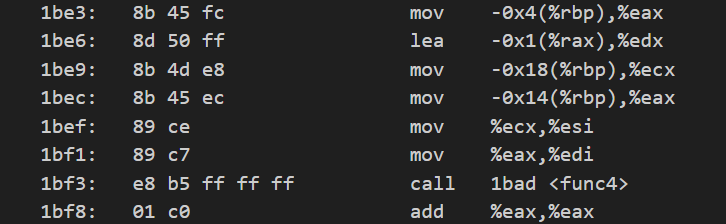


图4-8 mid>x的情况

接下来先分析mid=x的情况，此时直接返回0。

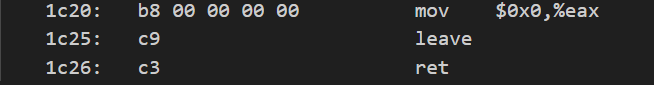


图4-9 mid=x的情况

接下来先分析mid<x的情况，与大于的情况相似。将参数修改为x,mid+1,max传入func4，将返回值乘2后再加1存储到eax中。

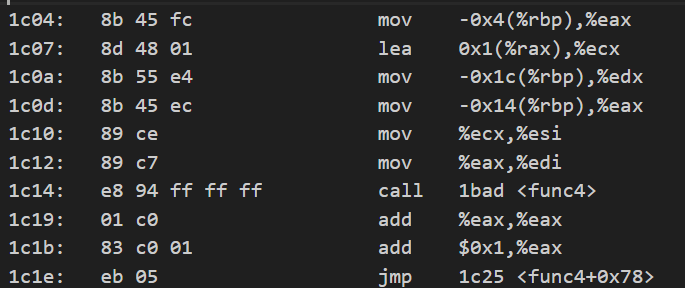


图4-10 mid小于x的情况

根据前面一开始传入7，14，0的推理，加上刚才的论证，可以推理得出fun4为一个类似于二分查找的函数。

可以推测fun4函数的c语言伪代码如下：

int func4(int x, int min, int max) {

int mid = (((max - min) >> 31) + (max - min))/ 2 + min;

//如果max>=min, mid=(max+min)/2

//如果max<min, mid=(max+min+1)/2

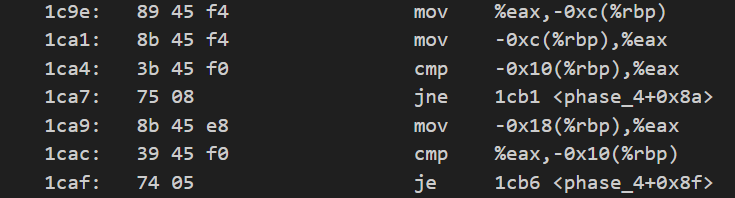
if(mid == x) return 0;

else if(mid > x) return 2 \* func4(x, min,mid - 1);

else return 2 \* func4(x, mid + 1, max) + 1;

}

返回phase\_4函数中，eax为整个递归函数的返回值，-0x10+$rbp中的值为7, -0x18+$rbp为你输入的第二个值，要求最后返回值与7相等，第二个值要输入7。



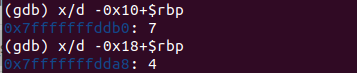


图4-11 递归函数返回值比较

从ide中编写相应函数，打印输出所有的解。

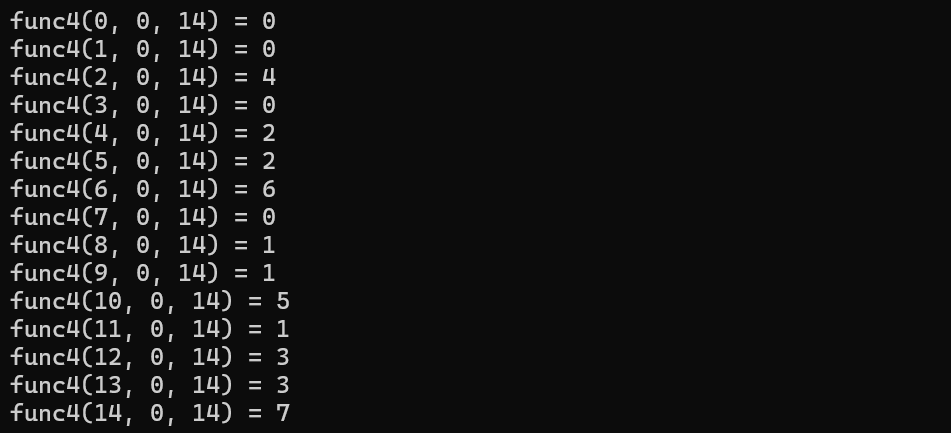


图4-12 c语言模拟运行结果

从中选择结果为7的解，输入14，7，通过测试。

答案为:14 7

5.指针和数组访问

首先注意到其调用了string\_length函数，并将返回的结果和6进行比较，推测其应当要求输入一个长度为6的字符串



图5-1 输入长度比较

随后跳转到1d3f处，比较-0x18+$rbp中的值是否小于等于5，在小于等于5的情况时，跳转到1d0f,这是一个循环。总体来看，这段代码是在一个数组中进行循环处理，每次循环从数组中取出一个字节，经过一系列操作后存储到另一个数组中，现在来详细分析一下操作内容。

很容易发现-0x18+$rbp充当整个循环计数器的作用，从0开始计数，循环首先将其存储到rdx中，将栈上的偏移地址-0x28(rbp)处的值（通常是数组的起始地址）加载到rax寄存器中，随后将起始地址和计数器相加，来实现数组指针的移动。

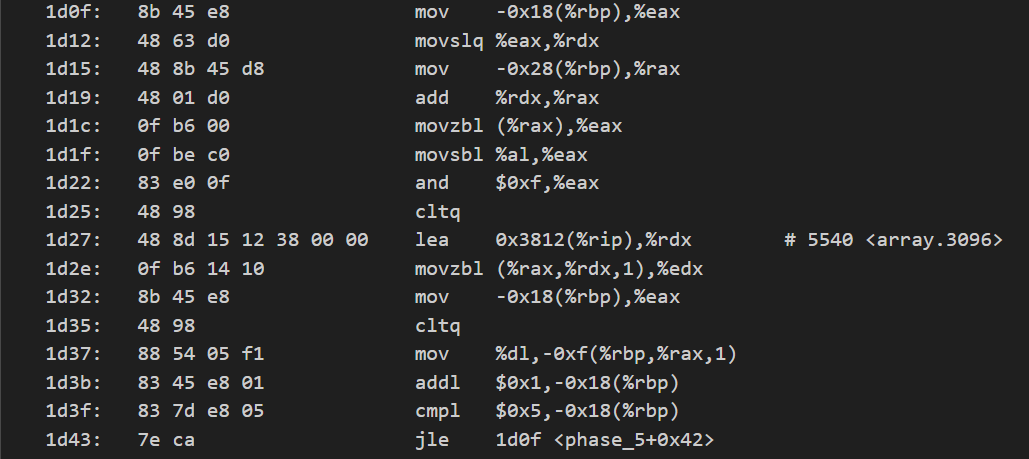


图5-2 数组指针的移动

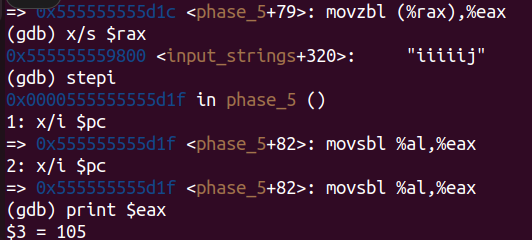
movzbl (%rax),%eax，从rax寄存器中指向的地址中读取一个字节的值，然后将其零扩展到eax寄存器中，即将其存储为无符号值。movsbl %al,%eax，将 al 寄存器中的值（一个字节）符号拓展到 eax 寄存器中，将无符号字节转换为有符号字节。随后与0xf进行按位与操作（相当于对16取余）。在这次测试中，i的ASCII码为105，可见其作用为逐步取出输入的字符串，取最低4位后作为新的数组中的映射。

图5-3 输入字符的ASCII码

跳出循环后，发现程序通过string\_not\_equal函数来进行输入正确性的判断，通过调试得到最后处理后要求字符串应该为bruins。

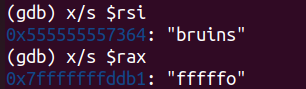


图5-4 目标字符串

查看数组中的映射规则，其中0-15为你输入字符ASCII码对应你所输入的字符对16的取余，bruins对应的ASCII码为98 114 117 105 110 115，对应数组下标为13 6 3 4 8 7。

那么可以推出一组可行解为mfcdhg。



图5-5 数组中存储的字符ASCII码

根据循环中得到的规则进行反推，bruins反推得到字符串

答案为：ASCII码取余16恰巧为13 6 3 4 8 7的六个字符（例如mfcdhg）

6. 链表、结构、指针的访问

观察汇编代码，可以发现读取六个数之后程序进入了一个二重循环，-0x78(rbp)作为外循环的计数器（0到5），-0x74(rbp)作为内循环的计数器（0到5）。

在循环中，输入的六个值分别存储在-0x60+rbp中，每四个字节存储一个数据。

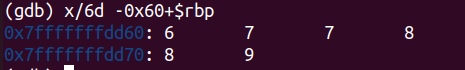


图6-1 输入数据存储

要求读取的数大于0，小于等于6。



图6-2 读取数据的要求

随后读取当前循环计数，加一后计入内循环计数（类似j=i+1），然后观察j是否小于等于5，如果符合该条件则进入内层循环，edx中存储输入的第“外层循环”个数，然后将第“外层循环”个数和第“内层循环”个数相比较，要求其不相等。综上所述，这个二重循环的作用是要求输入1-6之间六个不相等的数。

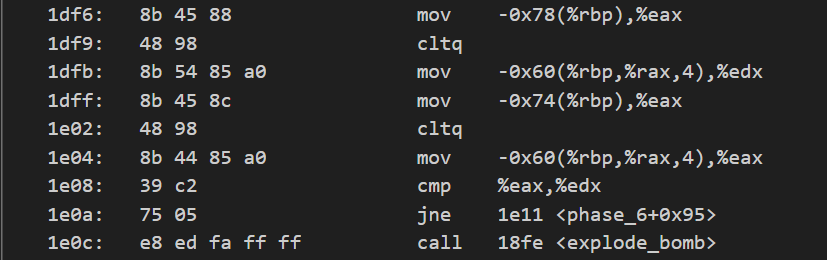


图6-3 判断数据输入是否符合要求

跳出循环后，计数器清零，给rax赋值-0x68(rbp)，即是node1的地址，给-0x74(rbp)赋值为1，接下来比较程序开始输入的值和-0x74(rbp)中的值，在eax<0x74(rbp)的情况下，程序不跳转，在此情况下，额外扩充出了16个字节，构建了一个链表。可以推测出链表的结构如下。

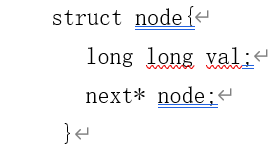


图6-4 构建链表

接下来的代码是一个简单的循环，用于复制一个数组的内容。将内存地址-0x40(rbp)中的值（假设是数组的起始地址）加载到寄存器rax中，并将其保存到内存地址-0x68(rbp)中。然后再将该值保存到内存地址-0x70(rbp)中，由此实现链表顺序的重排。

当输入数据为3 4 5 1 6 2时，经过该段程序可以发现都链接到了一起，构建节点的序号顺序即为一开始输入的数字。

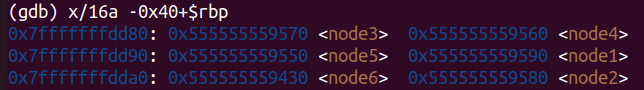




图6-5 重排链表

随后按顺序去取链表节点，每个链表中的地址对应了一项相应的值，通过调试，可以知道node1-6对应的值为：281 907 322 121 824 528。

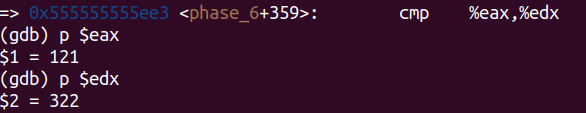


图6-6 node1-6的映射值

最后通过一个比较,要求链表中地址表示的值由大到小排列，才不会引爆炸弹。结合刚才各个节点中地址表示值的大小的表，输出顺序应当为node2(907), node5(824), node6(528), node3(322), node1(281), node4(121)。

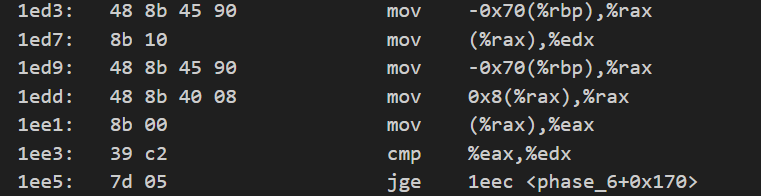


图6-7 数据比较要求

答案为：2 5 6 3 1 4

**（2）拆除炸弹的过程中关键操作**

1.查看call中string\_not\_equal函数的部分并分析其调用的参数位置，需要通过反汇编工具来查看程序的汇编代码，理解程序的内部运行逻辑。通过分析汇编代码，定位到call指令，指令后面紧跟着被调用函数的地址。向前查找即可以可以找到函数参数是如何传递的。

正确使用x指令来查看相应内存或寄存器中，确定参数的内容和表现形式。根据参数的类型和值，查找正确的输入。

2.在拆除二进制炸弹的过程中需要注意寄存器的结构。特别是要注意不同寄存器之间的关系，比如eax是rax的低32位部分，如果使用x指令或p指令只查看eax，只能看到rax的低32位，而剩下的高32位则可能包含重要的信息，甚至于无法解读该信息的含义。

另外，假设程序中某个操作影响了eax寄存器，若忽略了rax的变化，可能会错过一些重要的信息，导致分析出错。

因此，在使用调试工具时，要特别注意指定寄存器的大小，以确保获取到完整的寄存器内容，不会遗漏任何信息。

3.查找输入数据的格式，使用gdb b设置断点然后使用si逐步调试，进入函数，查看输入数据的格式，也可以多次进行调试，尝试不同的输入数据，并观察程序的反应，通过不断迭代调试，逐步理解程序的行为和输入数据的格式。

4.由于递归函数本身结构的特殊性，使用伪代码表示递归函数的工作原理并模拟和跟踪各个变量的值对于问题来说更容易理解和分析。

使用伪代码可以简化问题的描述，突出递归调用的核心思想。通过简洁清晰的描述，可以更好地理解递归函数的执行过程，其着重于描述算法的逻辑结构，可以更专注于理解算法的工作原理，而不会被语言特性所干扰。

5.正确使用x指令查看某些连续地址的值，其基本格式为x/<n/f/u> <addr>，其对于程序整体和局部的分析都能达到事半功倍的效果。

6.画示意图和变化图，分析出rbp各个位置值的变化。尤其是在phase\_6中，分析rbp寄存器各个位置值的变化才可以正确理解链表的构建和存取顺序。

可以画出程序执行过程中rbp寄存器的示意图，包括rbp寄存器的各个位置以及它们随着程序执行的变化情况。按照程序的执行顺序，记录rbp寄存器每个位置的值随着时间的变化情况，清晰地调试rbp寄存器在不同阶段的值变化情况，更好地分析程序的执行流程。

通过观察流程图和变化图，可以发现rbp寄存器在程序执行过程中的关键变化点，以及这些变化点对程序执行的影响，这有助于我们更好地理解程序的逻辑和结构，并为进一步的调试和分析提供线索。

**四、体会**

完成这个实验后，我获得了许多宝贵的经验和深切的体会。

通过逆向分析二进制炸弹程序，我加深了对程序机器级表示各方面知识点的理解。通过分析程序的构成和运行逻辑，我学会了在Linux系统中如何使用gdb进行反汇编、跟踪、分析和调试等技能。

此外，我意识到动态调试在解决问题中的重要性。使用x p si b r c等指令，我学会了动态地观察程序的行为，跟踪变量的值变化，定位和解决程序中的错误，使我更加熟悉了调试技术和汇编语言，并提高了我解决实际问题的能力。

完成这个实验还加强了我的团队合作能力和学习能力。在遇到困惑的困难时，通过交流思路，共同探讨、查找资料来解决问题，提高了实验的完成效率。

最重要的是，通过这个实验，我学会了如何理解和分析程序的汇编语言执行逻辑，如何找出隐藏在程序中的关键信息，并通过细致调试，推断出解除炸弹所需的目标字符串。

完成这个实验是一次非常有价值的经历。我通过实践提升了自己的技能水平，获得了宝贵的经验，并对未来的学习和工作有了更清晰的认识。我期待着将这些经验和技能应用到更广泛的领域中。