

**实验报告**



**题目： 拆解二进制炸弹**

**班 级： 2023211307**

**学 号： 2023211088**

**姓 名： 罗浩**

**学 院： 计算机学院**

**202 年 10 月 31 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. Windows PowerShell（10.120.11.12）
3. Linux
4. Objdump命令反汇编
5. GDB调试工具
6. 。。。。。

三、实验内容

登录bupt1服务器，在home目录下可以找到Evil博士专门为你量身定制的一个bomb，当运行时，它会要求你输入一个字符串，如果正确，则进入下一关，继续要求你输入下一个字符串；否则，炸弹就会爆炸，输出一行提示信息并向计分服务器提交扣分信息。因此，本实验要求你必须通过反汇编和逆向工程对bomb执行文件进行分析，找到正确的字符串来解除这个的炸弹。

本实验通过要求使用课程所学知识拆除一个“binary bombs”来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。 “binary bombs”是一个Linux可执行程序，包含了5个阶段（或关卡）。炸弹运行的每个阶段要求你输入一个特定字符串，你的输入符合程序预期的输入，该阶段的炸弹就被拆除引信；否则炸弹“爆炸”，打印输出 “BOOM!!!”。炸弹的每个阶段考察了机器级程序语言的一个不同方面，难度逐级递增。

为完成二进制炸弹拆除任务，需要使用gdb调试器和objdump来反汇编bomb文件，可以单步跟踪调试每一阶段的机器代码，也可以阅读反汇编代码，从中理解每一汇编语言代码的行为或作用，进而设法推断拆除炸弹所需的目标字符串。实验2的具体内容见实验2说明。

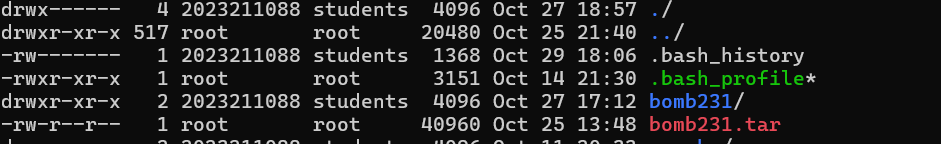
四、实验步骤及实验分析

建议按照：准备工作、阶段1、阶段2、…等来组织内容

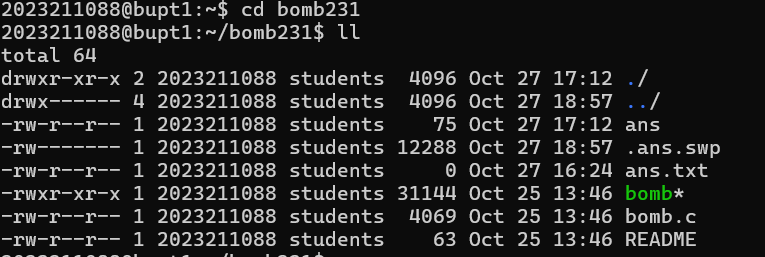
各阶段需要有操作步骤、运行截图、分析过程的内容

***准备工作：***

1. 远程登录服务器
2. 键入ll指令查询当前文件夹的bomb信息观察到要拆的是第几号的炸弹



1. 观察到是bomb231，则使用cd命令切入



1. 接下来启动gdb进行调试。



***Phase1：***

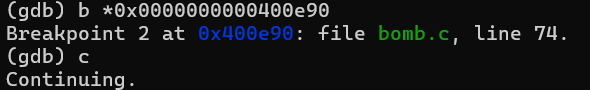
1. 设置断点：



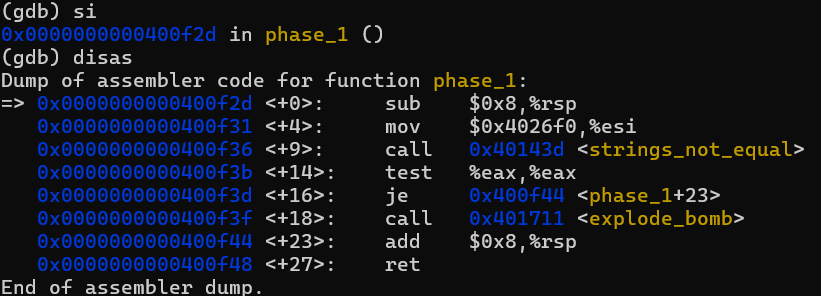
1. 输入r命令运行；
2. 输入disas即可查询到该源程序的汇编代码，找到phase1，即为拆炸弹的第一关：



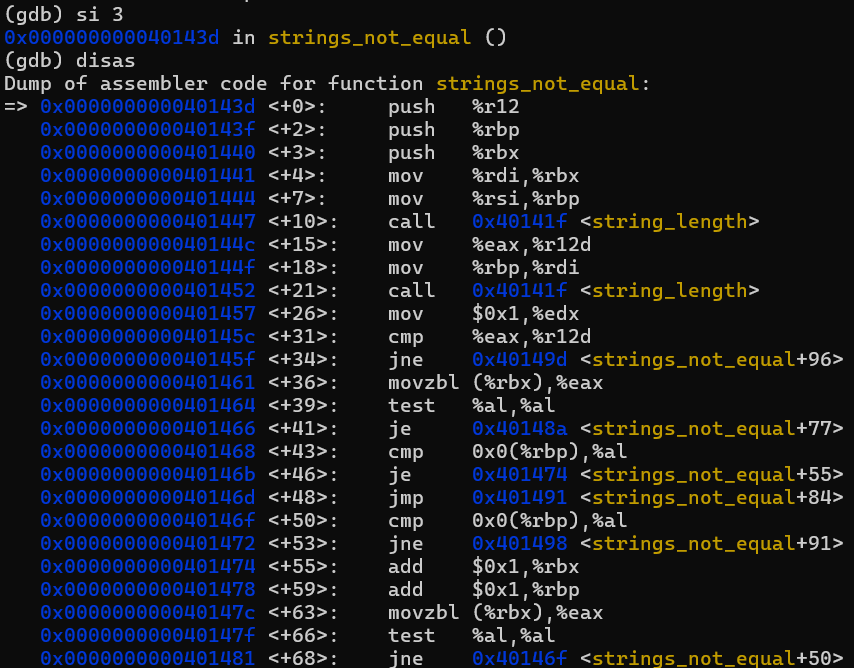
1. 设置该处断点，输入c跳转至该阶段：



1. 随意输入一些字符，单步调试si进入到phase1中
2. 接下来进行disas反汇编，得到该关卡相关信息：



1. 我们通过该函数的一些调用发现，它调用了一个叫做strings\_not\_equal的函数，然后接下来就进行了test指令根据返回值eax是否符合要求决定是否爆炸。因此strings\_not\_equal比较关键，单步进入该函数看看。

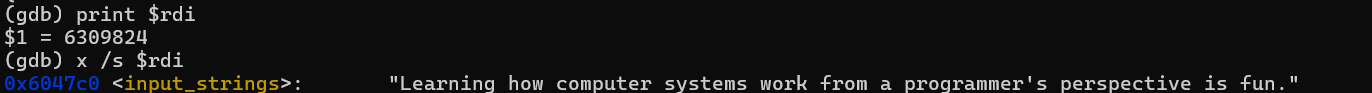


1. 里面有两个string\_length函数，显而易见，这是先判断两者长度是否相等，那么在其中一个寄存器中一定寄存着答案以用来得到长度。但究竟哪一个是答案呢，我们可以观察到，它有一个传参数的动作，



很显然，一定是将字符串传进去了，那么我们依次查询一下寄存器即可得到答案：



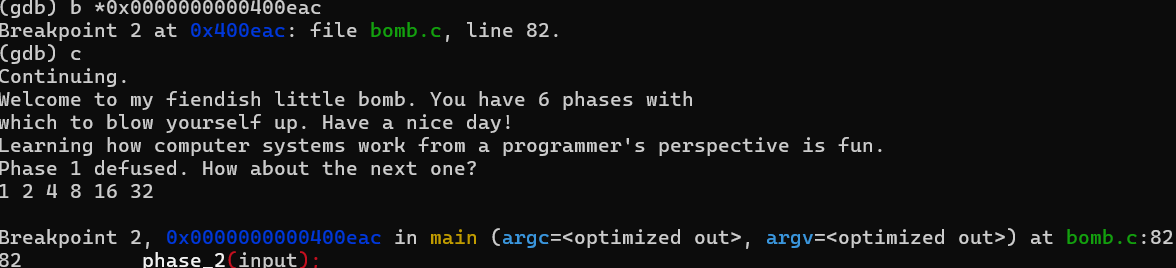


由此得到答案。

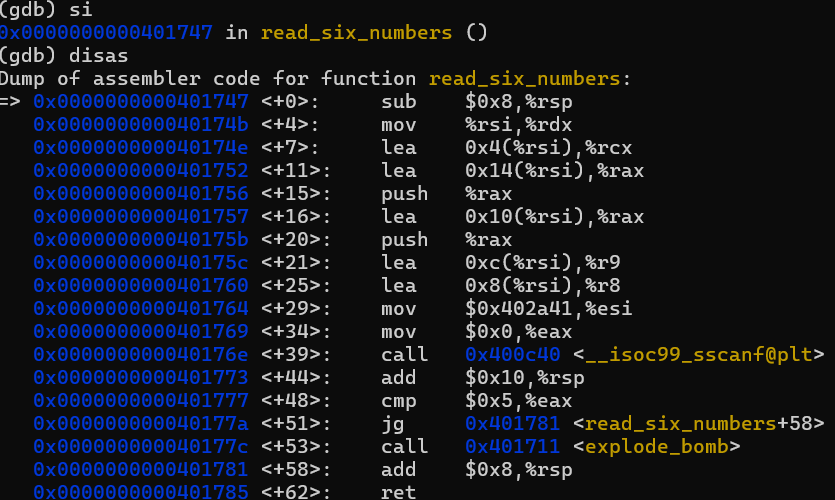
1. phase1破解成功

***Phase2：***

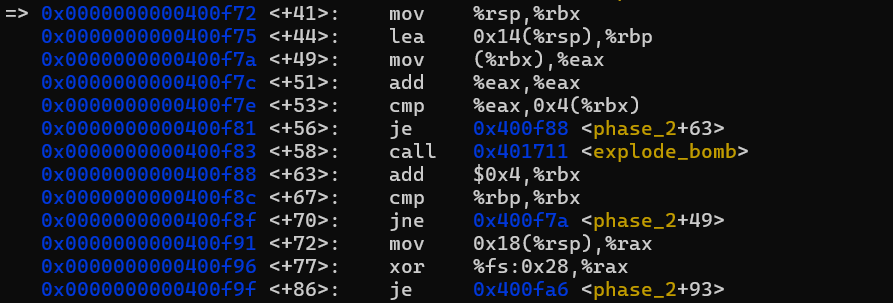
1. 按照phase1的流程设置断点，定位到phase2



1. 单步进入，观察函数发现，函数率先调用了一个<read\_six\_numbers>的函数，需要特别注意，因为后面紧跟一个炸弹。
2. 我们单步进入：



1. 很显然，该函数通过·读入用空格分隔的六个数，因此需要特别注意输入六个数以防炸弹爆炸。
2. 我们退出该函数，继续往下走：
3. 找到接下来的一块代码，可以发现在+70处出现了回跳，而每次回跳条件与rbx和rbp相关，可以从语境中读出两者为内存地址，读取出来看一下



1. 我们发现正是我们刚刚输入的数



1. 当rbx从a[0]处走到a[5]处时，该跳转失效，因此我们遭遇了循环代码，每次遍历a[]数组中的元素。
2. mov (%rbx),%eax、add %eax,%eax、cmp %eax,0x4(%rbx)这三段代码很关键，他们决定着炸弹是否爆炸，我们发现每次通过取a[i]的元素传入eax并乘2与a[i+1]的元素相比较，相等则符合条件。因此，该循环的主要目的搞清楚
3. 因此答案为1，2，4，8，16，32，注意第一位一定为1.

***Phase3：***

1. 定位到phase3。
2. 单步进入phase3函数，我们会发现这段代码有点长，因此，需要分块来看
3. 观察输入：





1. 需要输入两个整数和一个字符，因此我们选择随便填以满足该条件
2. 往下走，注意到有一个条件判断，将7和某个事物进行了比较，我们观察一下

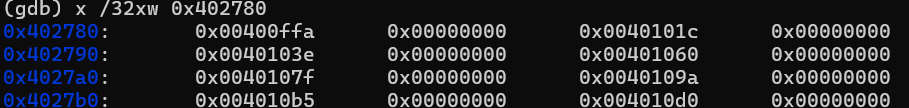






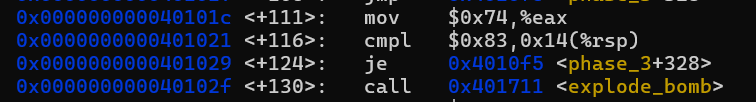
一个数和第二个数貌似是我们输入的两个整数，发现了一定的规律，意思是第一个整数不能大于7，否则跳转到炸弹函数

6.接下来有一个让人摸不清头脑的跳转：jmp \*0x402780(,%rax,8)，这个很象是switch语句中的跳转表，我们观察一下：



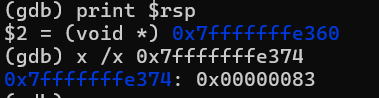
果然是跳转表，其中存储的地址都与phase3函数中的其中一行对应，分别是0-7条件，这也是为什么之前有一个与7的比较！！！，因此，这无疑是一个switch语句

1. 进一步观察得到，几乎每一个case都既能引爆炸弹又能过关，我们不妨选用第二个case：0x40101c

case2：

跳转到这个case只需要rax=1，也就是我们输入的第一个整数为1.

1. 然后有一个$0x83与内存中某个值的比较，我们查看一下：



似乎与输入的字符参数有关%c

1. 通过不断改变输入字符值，我们发现该值与输入字符的ASCII值相关，因此只需要找到合适的字符就能通关。
2. 这里我选用的是1 t 131，131是我随便输入的值。

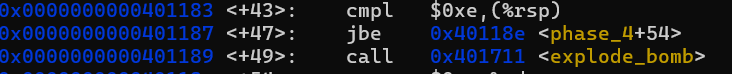
***Phase4：***

1. 我们来到phase4接口
2. 再来看一下系统要求我们输入的值有些什么

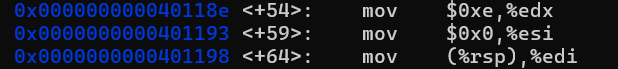


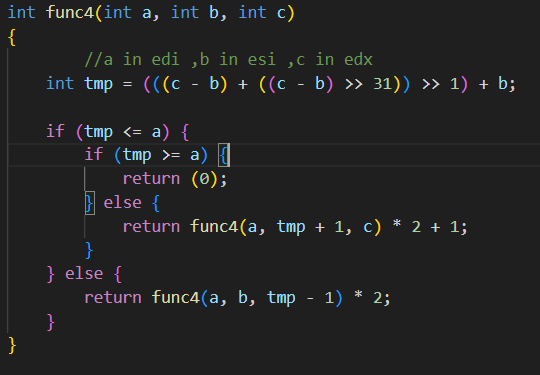
是两个整数，按照要求填写

1. 往下，这段代码稍微需要留意：



查看rsp指向的值发现这就是我们输入的第一个整数，要求小于等于14.

1. 再往下，来到了一个新函数fun4，先别急着进入，看看传参数：
2. 一共三个参数，rdx：14，esi：0，edi：7（我们输入的整数），再看返回的参数eax要和零作比较，也即是说要想通关，就得返回零。
3. 进入fun4观察，我对fun4函数进行还原得到如下C源码：



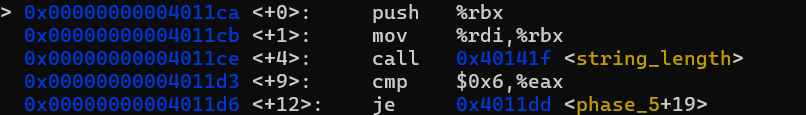
我们发现这是一个递归函数，但有办法不进行递归，也就是令tmp==a,也就是说让(((c - b) + ((c - b) >> 31)) >> 1) + b==a(我们输入的第一个整数)，计算得到a=7,也就是说，第一个整数为7即可。

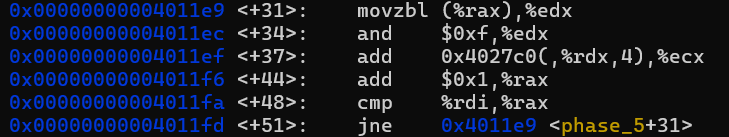
1. 再往下看，将零与我们输入的第二个值进行了比较，因此第二个数为0，通关
2. 答案为7 0

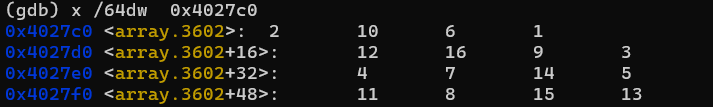
***Phase5：***

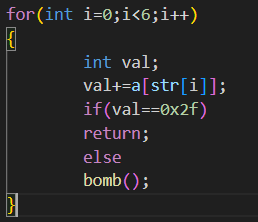
1.进入phse5接口

2.该函数调用了strlen函数，返回值为字串长度，观察到返回值与6相比较，得知输入为一个6字符的字串。



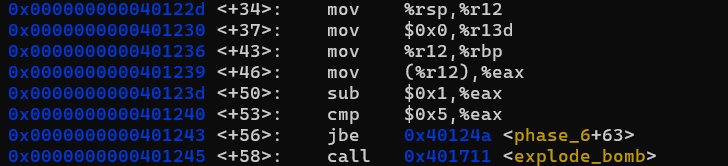
3.我们发现这是一个单循环，，如图为循环体主要部分，and操作相当于对edx进行模16操作，这个很关键。，rax可视为递增下标，对它进行观察：发现就是我们输入的字符串循环体依次遍历这六个字符。

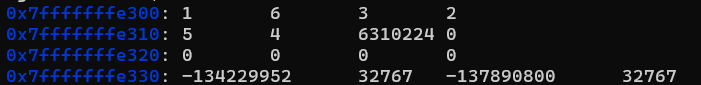
4.观察操作：add 0x4027c0(,%rdx,4),%ecx，我们发现ecx在不断的积累值，每次从一个地址中取数，我们进行观察：，发现每次从一个数组中取数，取数的地址又与每次遍历的字符有关，还原函数如下：



1. 因此我们只需要控制每次取数的地址就可以满足最总答案为0x2f的条件，我选用"0129:="
2. 通关

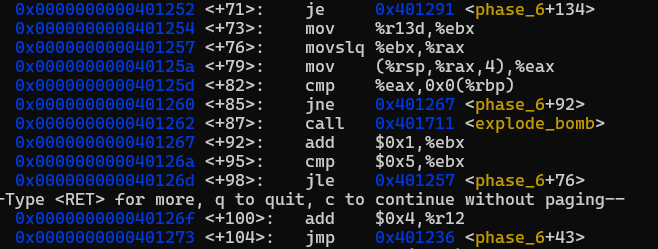
***Phase6：***

1. 来到boss关，进入phase6
2. 发现这段代码很长很长，我们要一段一段地看，首先根据题意，我们需要输入六个数
3. 来看第一段，在炸弹爆炸之前进行了一次检验，rax从r12中取数，读取r12：



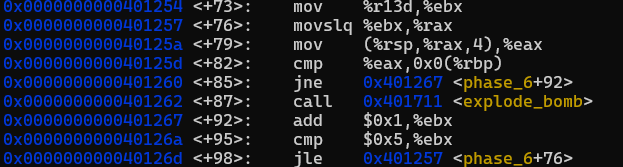
这很显然就是我们刚刚输入的数。而此时要求第一个数-1小于等于5，否则爆炸，也就是说不大于6即可。

1. 继续往下看，发现这是一个循环其中，r13寄存器充当循环标志数，一次检验我们输入的六个值，也就是说，我们这六个值都不能超过6
2. 下一段代码：

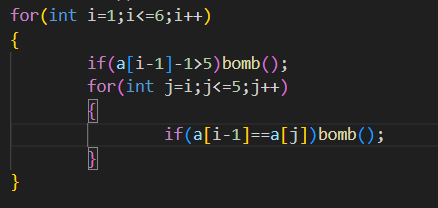


显而易见，依旧是一个循环，而且每次循环后会跳向+43处，也就是上一个循环的前面，循环的结束条件在+71的前面，也就是，也就是说我们的考虑欠妥，这是一个嵌套循环。

重新进行观察，对r13寄存器充当的角色重新进行定义，它时外层循环的标志数，而内存循环的标志数为寄存器ebx：

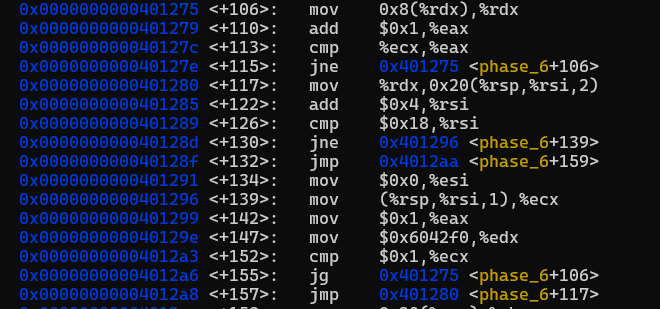


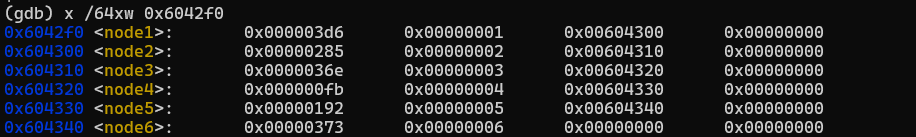
每次ebx在r13的基础上继续递增直至到5，也就是读入的第六个数，因此将这段代码写成一个嵌套循环的C语言为：



至此明了，这段代码检测输入是否两两不等，且都不大于6.

1. 下一段可以被划分为这一段，因为我们以循环跳出条件为分界点：



接下来我们被直接跳转到+134位置接下来ecx被赋予了我们输入的第esi个位置的值，eax=1，然后一个未知的地址出现了，我们查询一下：，

根据前缀提示，我们得知，这似乎是一个由结构体构成的链表因为，每个结构体的第三段都存储了一个指向下一行节点的指针，因此这是一个顺序链表，值分别为0x3d6,0x285,0x36e,0xfb,0x192,0x373。

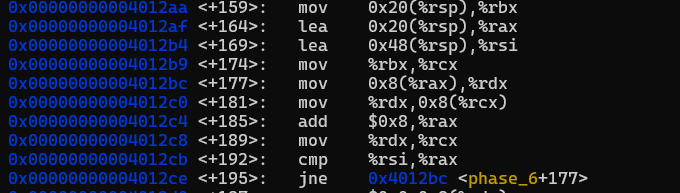
因此此时edx存储指向第一个结点的指针。

接下来有两个分支，ecx>1和其他情况跳转位置不同。

ecx>1时，edx跳转到指向下一个节点的指针，eax++，知道ecx和eax相等，相当于寻找节点值和ecx存储的值相等的节点，之后将rdx的值放入了一个内存里暂存了起来，这个内存位置与rsi呈正相关。Rsi迭代，直到遍历完六个节点。

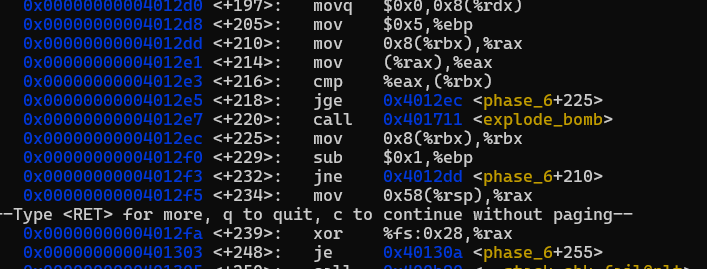
至此我们可以明白这段代码将node按照输入数据大小一一对应进行排序，存储至0x20(%rsp,%rsi,2)中（每个node的指针）。

1. 再下一段，被划分为这一段：



可以理解为遍历这存储在0x20(%rsp,%rsi,2)的六个值，按照次序将node进行重新链接

1. 最后一段：（做了一个注释）



//此时rbx指向按顺序存储的第一个位置，也就是0x6042f0

0x00000000004012d0 <+197>: movq $0x0,0x8(%rdx)//rdx指向最后一个被排序的节点

0x00000000004012d8 <+205>: mov $0x5,%ebp

0x00000000004012dd <+210>: mov 0x8(%rbx),%rax//将下一个结点的指针传给rax

0x00000000004012e1 <+214>: mov (%rax),%eax

0x00000000004012e3 <+216>: cmp %eax,(%rbx)//也就是将第一个节点与第二个节点的值作比较，第一个节点值大于等于第二个即视为通关

0x00000000004012e5 <+218>: jge 0x4012ec <phase\_6+225>

0x00000000004012e7 <+220>: call 0x401711 <explode\_bomb>

0x00000000004012ec <+225>: mov 0x8(%rbx),%rbx//指向下一节点

0x00000000004012f0 <+229>: sub $0x1,%ebp//，ebp为循环标志数，相当于按照连接关系依次遍历每个节点，应该是每个节点都与后一个节点做比较

0x00000000004012f3 <+232>: jne 0x4012dd <phase\_6+210>//循环

0x00000000004012f5 <+234>: mov 0x58(%rsp),%rax

0x00000000004012fa <+239>: xor %fs:0x28,%rax

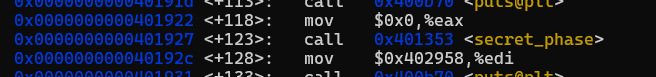
0x0000000000401303 <+248>: je 0x40130a <phase\_6+255>

这段码在检验根据输入的数的相对大小进行排序的节点是否满足递增，若是，视为通关。

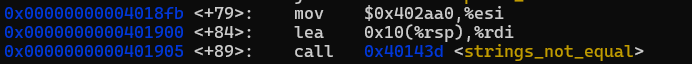
1. 综上，解得1 6 3 2 5 4。

***Phase7（隐藏关）：***

1. 通过观察phase\_defused中的汇编发现secret\_phase:



1. 逐步调试，发现有一个判断条件在阻止我们进入该函数：

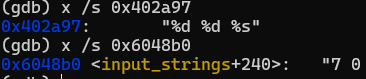


在和什么字符呢，查看参数地址：



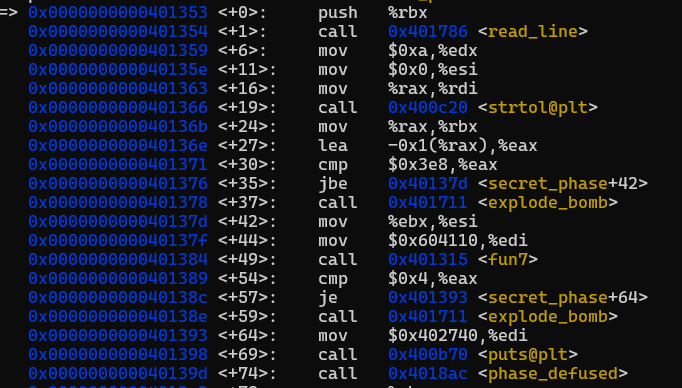
发现神秘字符DrEvil，又在和哪个字符比呢？？？

首先在这之前有一个判断，，eax,也就是sscanf的参数需要有三个才能进入下面的代码行，我们查看一下它的参数，

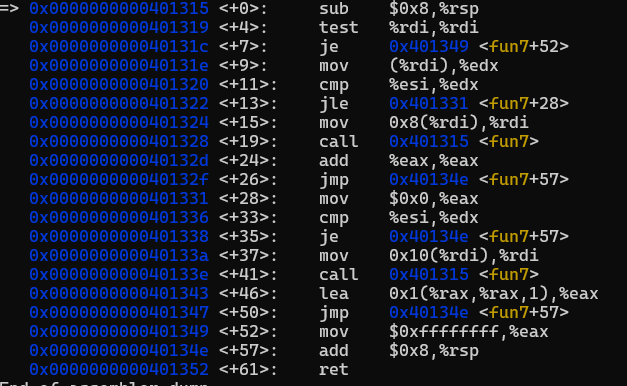


答案显而易见，这就是我们phase4隐藏的第三个参数，我们在phase4输入DrEvil,果然通过了条件判断，来到了phase7

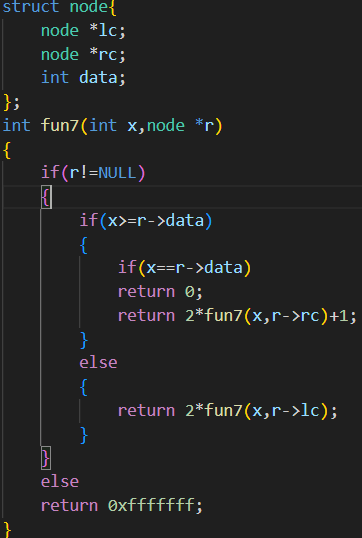
3.进入secret\_phase函数，strtol函数的返回值要求不能大于0x3e9,我们注意一下。

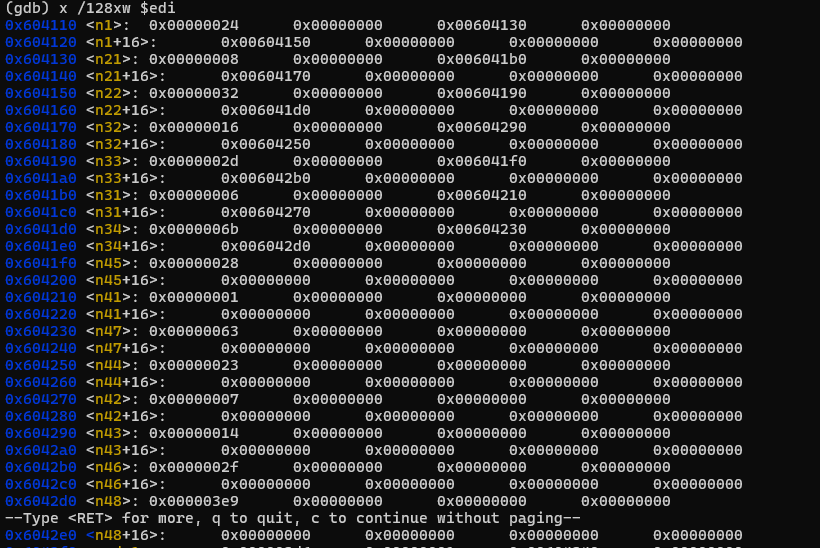


1. 接下来就是fun7函数，进入：



Fun7的两个参数一个是一个指针，另一个就是我们输入的转化为值的数字字符串。

写出对应C源程序：，这是建立在一颗树上的递归，树的汇编代码，每个节点第一个字节存储值，第二个字节和第三个字节分别存储指向左孩子与右孩子的节点：



观察发现，这是一颗满二叉树，深度为4，并且按照从上发到下，从左到右的顺序编好了号，可以不经C语言的翻译直接读取，完成题目的破解。

4.根据C语言代码经过一系列比较，最终应遍历到节点n42, 得到答案为7，结束。

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）

遭遇的挫败：我在phase3卡了很久，几乎比phase6还久，主要原因是我未能从gdb调试结果中发现规律，经过了多次试错无果。Switch语句中的数字比较还是比较令人迷惑的，究竟是以一个怎样的方式得到的结果我至今还未能搞清它的奥秘，但当我把10进制换到16进制观察时，我能找到一些蛛丝马迹，并以动态调试的方式通过了这关。因此，在拆炸弹的过程真的很象做出一道奥数题，有些时候更多的是需要灵光一现的时刻。

感受：每个phase的难度逐级递增，能让人有一个很好的过渡，很适合初学者。但phase7确实是需要一定的积累和深刻的理解的。当给人的满足感更是无与伦比的。