

**实验报告**



**题目： 拆解二进制炸弹**

**班 级： 2023211301**

**学 号： 2023210896**

**姓 名： 王书翰**

**学 院： 计算机学院（国家示范性软件学院）**

**2024年 10 月25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

5. Vim 8.2.2121

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。

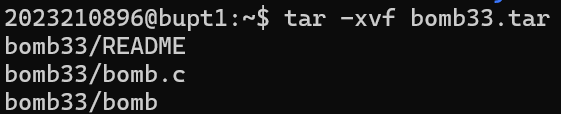
本实验通过要求使用课程所学知识拆除一个“binary bombs”来增强对程序的机器级表示、汇编语言、调试器和逆向工程等方面原理与技能的掌握。 “binary bombs”是一个Linux可执行程序，包含了5个阶段（或关卡）。炸弹运行的每个阶段要求你输入一个特定字符串，你的输入符合程序预期的输入，该阶段的炸弹就被拆除引信；否则炸弹“爆炸”，打印输出 “BOOM!!!”。炸弹的每个阶段考察了机器级程序语言的一个不同方面，难度逐级递增。

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

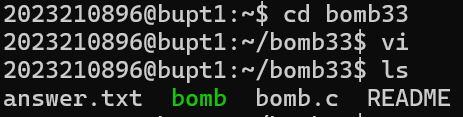
0、准备阶段

首先使用ls指令，查看压缩文件

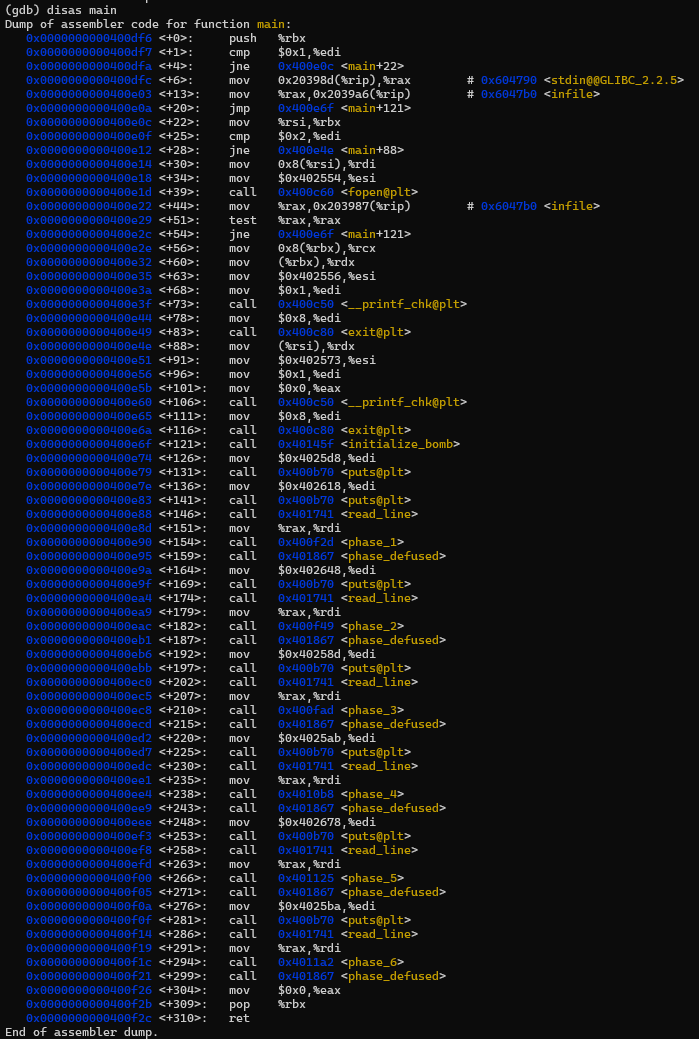
使用tar -xvf bomb33.tar 指令解压文件，得到3个文件

使用cd bomb33指令将目录更改为bomb33文件夹

用vi 指令启动vim编辑器，新建answer.txt文件用来储存答案



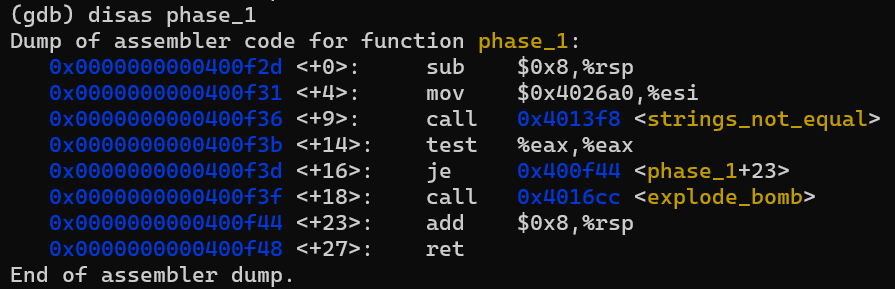
使用gdb 指令启动gdb调试器使用file bomb加载文件bomb，使用disas main指令查看main函数的反汇编语言，大概了解执行过程



由截图可知，拆炸弹的主要流程为先发现函数先通过调用readline函数读取输入的内容，将内容存在%rax寄存器中，随后将其复制到%rdi寄存器中，后将%rdi作为参数依次传入与调用phase\_1等6个函数，若成功拆除炸弹则继续执行调用phase\_defused函数并输出若干内容，重复执行6次。故接下来需要依次查看phase\_1等6个函数的反汇编代码与phase\_defused()的反汇编代码。

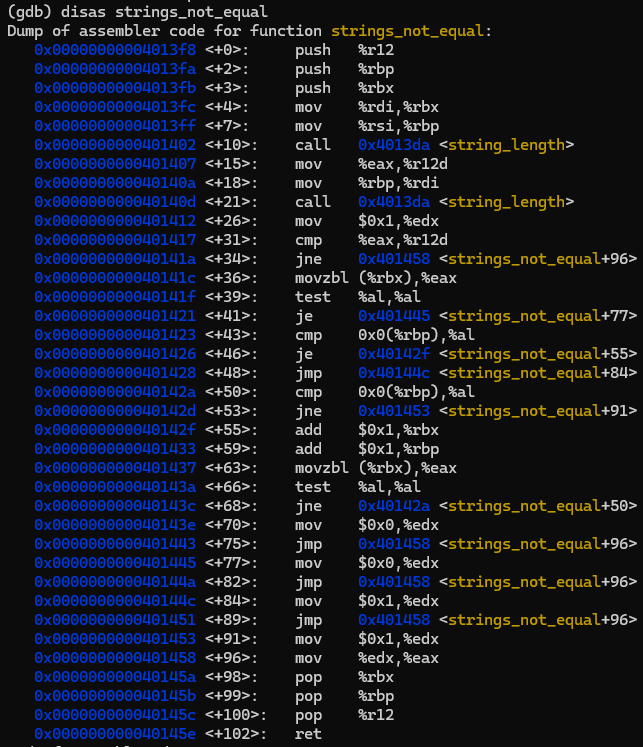
1、第一阶段

使用disas phase\_1指令查看其反汇编代码



分析反汇编语言得phase\_1函数先初始化了%esi寄存器的值为地址0x4026a0储存的值，紧接着将%esi作为参数调用了名为strings\_not\_equal函数，接着若该函数的返回值等于0则跳过炸弹，若不等于0则引爆炸弹。根据函数名称与调用的参数推测该函数的作用是比较%esi寄存器储存的字符串与输入的字符串是否相等，若相等则返回0.

使用disas strings\_not\_equal指令查看其反汇编代码



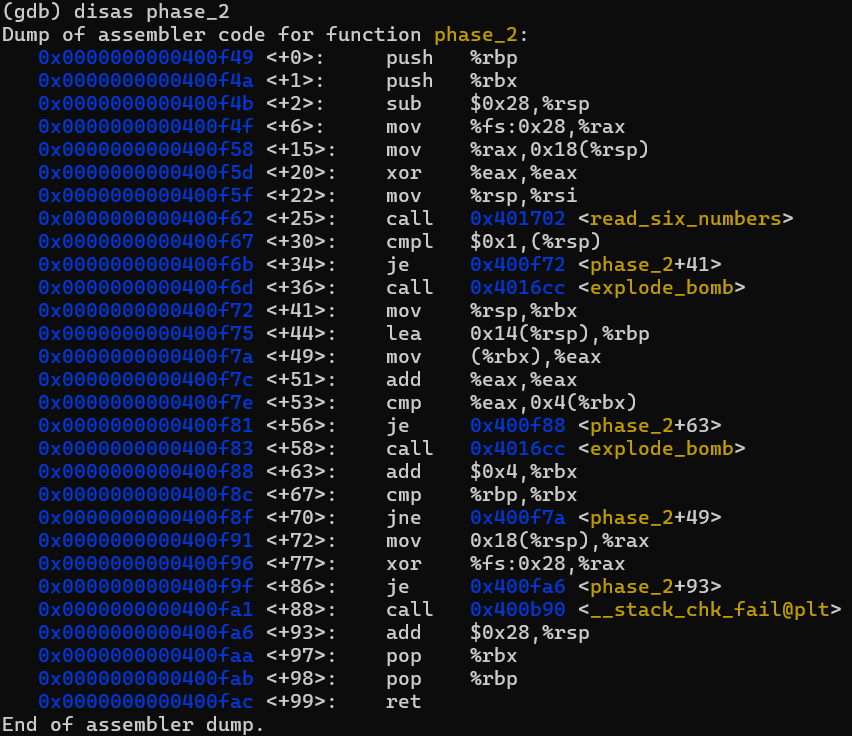
分析汇编语言得知该函数首先比较了两个字符串的长度，后继续比较其他内容。观察到该函数调用了两次string\_length函数，猜测该函数为输入一个字符串，返回该字符串的长度，在<+31>行对%r12d与%eax这两个函数的返回值的比较可验证猜想。两次调用的寄存器为%rbx与%rdi，分别储存的值为原先%rdi与%rsi寄存器储存的值，而储存的值分别为输入的字符串和地址0x4026a0储存的值，故输入的字符串一定为地址0x4026a0储存的值。

使用x/s 0x4026a0指令查看该地址储存的值，该值为阶段一的答案，将其输入answer.txt文件

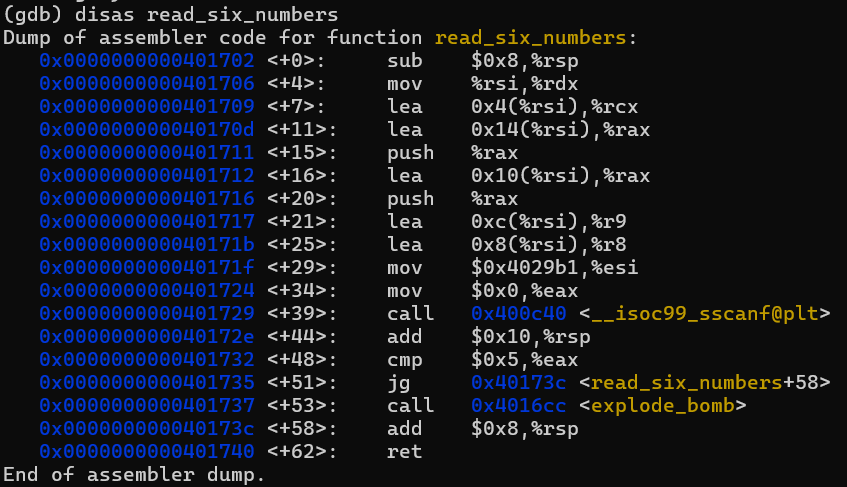
1729875979795(1)

2、第二阶段

使用disas phase\_2指令查看其反汇编代码



分析汇编语言得知该函数先调用了read\_six\_numbers函数。使用disas read\_six\_numbers指令查看其反汇编代码



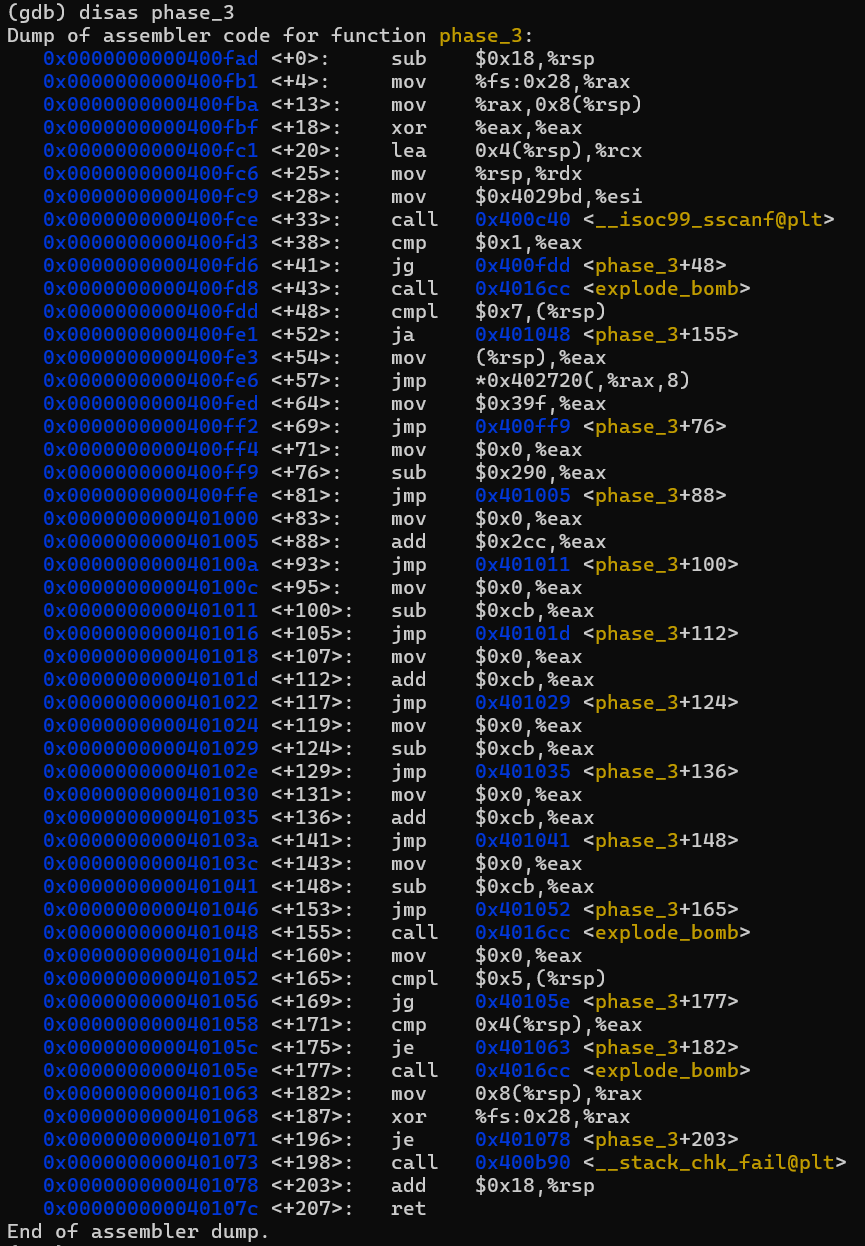
使用x/s 0x4029b1 指令查看该地址储存的值

1730621519714

联系函数名称与该信息得知该函数输入6个整数并返回栈底指针%rsp储存第一个数的地址，即%rsp储存这个长度为6的一维数组的首地址。随后检验栈底指针%rsp的值是否为1，若为1接着设置%rbx的值为%rsp，栈顶指针%rbp的值为栈底指针%rsp的地址加上16+4=20位（即第六个数字的地址），接着将%eax设置为%rbx，将%eax的值翻倍后与%rbx+4储存的值（即下一个数字）进行比较，若相同则将%rbx的值加上4（即让其指向数组的下一个元素），接着比较%rbx和%rbp的值（即检验%rbx是否指向最后一个数），若相同则跳出循环，若不同回到<+49>接着执行以上动作。通过分析可知输入的第一个数一定为1，之后的数是前一个数的两倍，即阶段二答案为1 2 4 8 16 32，将其输入answer.txt文件

3、第三阶段

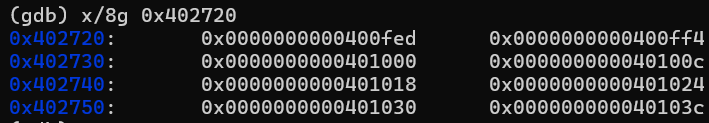
使用disas phase\_3 指令查看其反汇编代码



1730622211621使用x/s 0x4029bd 指令查看该地址储存的值

根据上文的分析得知该阶段首先输入两个数，第一个数储存在%rdx（%rsp）中，第二个数储存在%rcx（%rsp+4）中。

接着从<+48>行开始分析，该行将%rsp即第一个数与7进行比较，若其大于7则炸弹爆炸。故第一个数必须小于等于7且大于0。之后将%eax的值设置为第一个数，进行了switch的操作，即将第一个数乘以8加上0x402720的地址储存的值作为下一步跳转的目标地址。因为第一个数小于等于7且大于0，输入x/8g 0x402720 指令查看0x402720开始的8个地址的值：



由此可知不同的第一个数字对应不同的地址，又发现所有可能执行完后到<+165>行，该行将%rsp即第一个数与5进行比较，若其大于5则炸弹爆炸.故第一个数还要满足小于等于5的条件。之后将%eax（即不同情况中赋不同值的数）与%rsp+4（即第二个数）进行比较，必须相同才能不爆炸。

分析各种情况可发现地址较小的情况在执行完后会继续跳转到下一种情况，但不会重新给%eax赋值。

当%rsp=5时；执行跳转到0x0000000000401024，在该情况中%eax的值为0-203=-203。

当%rsp=4时；执行跳转到0x0000000000401018，在该情况中%eax的值为-203+203=0。

当%rsp=3时；执行跳转到0x000000000040100c，在该情况中%eax的值为0-203=-203。

当%rsp=2时；执行跳转到0x0000000000401000，在该情况中%eax的值为-203+716=513。

当%rsp=1时；执行跳转到0x00000000004010ff4，在该情况中%eax的值为513-256=257。

当%rsp=0时；执行跳转到0x00000000004010fed，在该情况中%eax的值为257+927=1184。

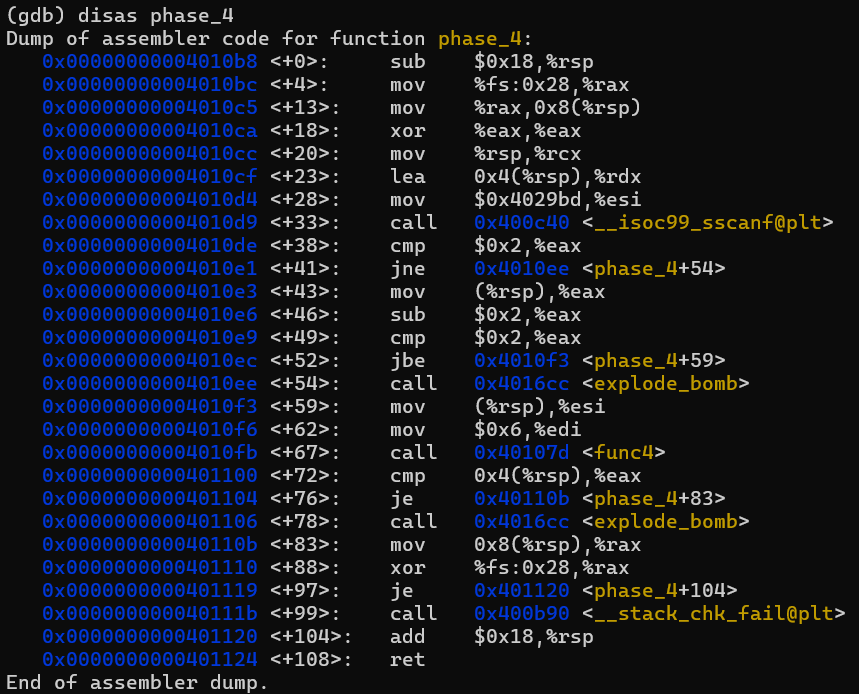
将6中可能与其对应地址的值还有对应的%eax的值列举得：

|  |  |  |
| --- | --- | --- |
| 第一个数的值 | 对应的跳转的地址 | %eax的值（第二个数的值） |
| 0 | 0x0000000000400fed | 1184 |
| 1 | 0x0000000000400ff4 | 257 |
| 2 | 0x0000000000401000 | 513 |
| 3 | 0x000000000040100c | -203 |
| 4 | 0x0000000000401018 | 0 |
| 5 | 0x0000000000401024 | -203 |

故阶段三的答案从以上6种情况任选一即可。因为4 0比较简单故取4 0，将其输入answer.txt文件

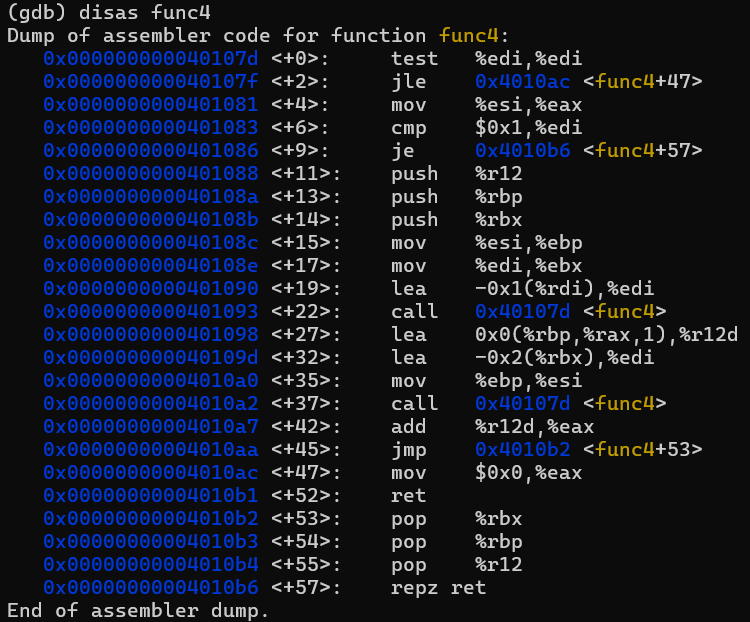
4、第四阶段

使用disas phase\_4 指令查看其反汇编代码



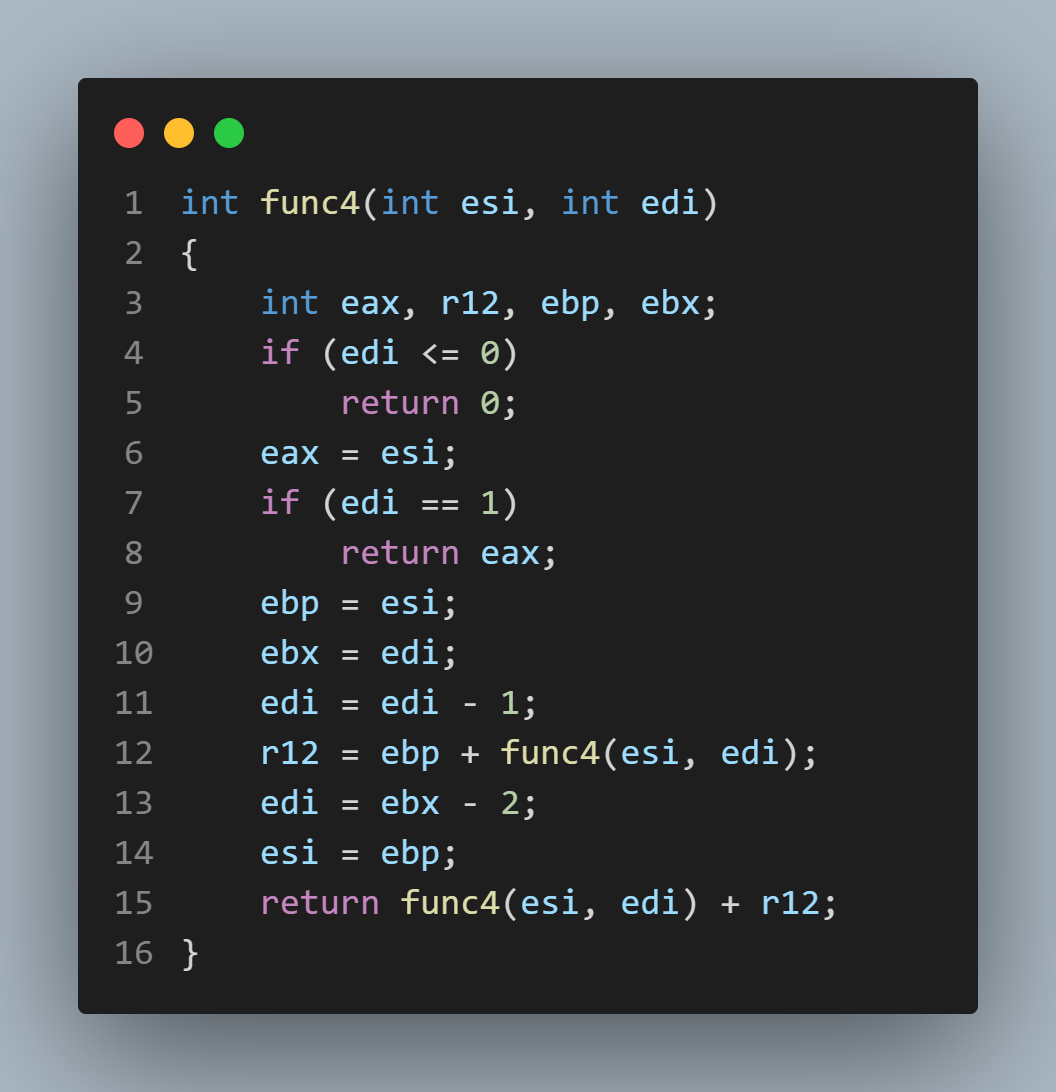
根据阶段3的分析得知该阶段首先输入两个数，第一个数储存在%rdx（%rsp+4）中，第二个数储存在%rcx（%rsp）中。接着判断%eax的值是否为2，若不同则炸弹爆炸；若相同则继续执行程序。接着将%eax的值设为%rsp（即第二个数），将其值减去2后再与2比较，若其大于2则炸弹爆炸，若其小于等于2则继续执行。随后将%esi的值也设为%rsp（即第二个数），将%edi的值设为6，将这两个数作为参数传入func4函数中。在其func4函数执行接收后将其返回值与%rsp+4（即第一个数）进行比较，若不同则炸弹爆炸，若相同则继续执行。

经过如上的分析，我们可以确定第二个数一定小于等于4且大于0，且func4的返回值为第一个数。

输入disas func4 指令查看func4函数的反汇编代码

不难看出该函数出现了递归操作，故逐步分析汇编语言比较困难，不妨将其转化为C语言分析。

将汇编语言翻译成C语言为



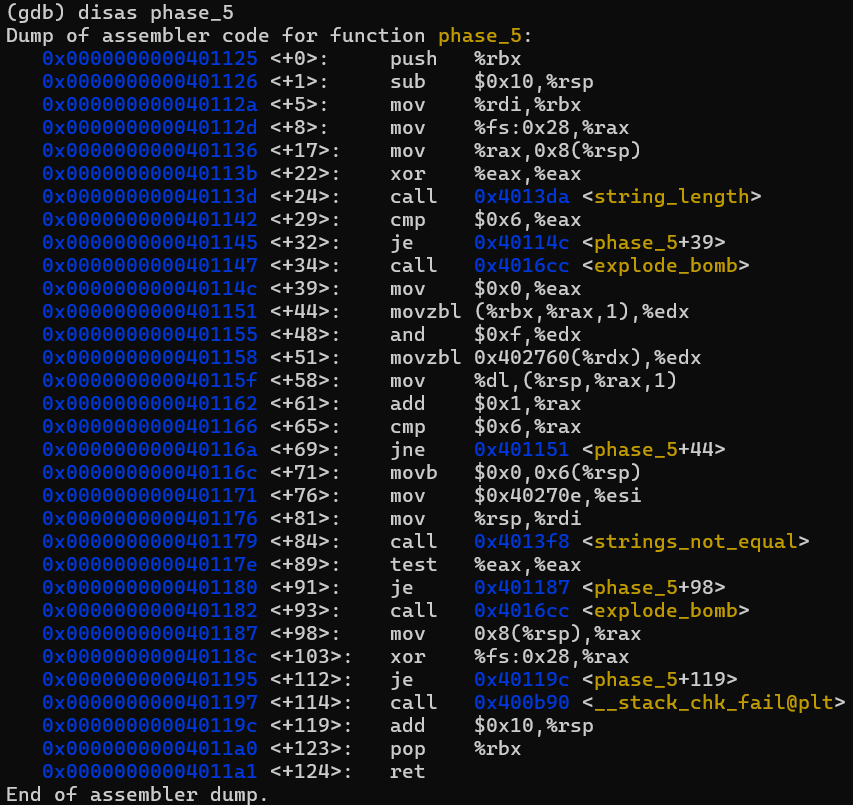
将%esi=0，1，2，3，4，%edi=6带入该函数执行得

|  |  |
| --- | --- |
| %esi（第二个数） | %eax（第一个数） |
| 0 | 0 |
| 1 | 20 |
| 2 | 40 |
| 3 | 60 |
| 4 | 80 |

故阶段四的答案从以上5种情况选一种即可。这次取60 3，将其输入answer.txt文件

5、第五阶段

使用disas phase\_5 指令查看其反汇编代码



根据阶段一的分析得知该函数首先将%rdx的值设置为%rdi（即输入的字符串），之后判断输入的字符串长度是否为6，若不为6则炸弹爆炸。

<+39>之后将%eax初始化为0；

<+44>将%edx设置为%eax+%rbx（即输入的字符串中的第%eax个字符）；

<+48>将%edx的ascii码和15进行与的运算，即将%edx的第四位保留下来；

<+51>接着将%edx设为地址0x402760+%rdx（即0x402760处储存的第%rdx个字符）；

<+58>将%rsp+%rax（即以%rsp为字符串首地址的第%rax个字符）设置为%dl（即%edx的低8位）；

重复执行<+44>到<65>操作6次。

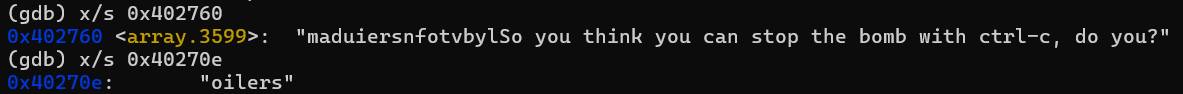
<+71>之后将字符串的第6位设为空（‘\0’）；

<+76>将%esi设为地址0x40270e处储存的字符串；

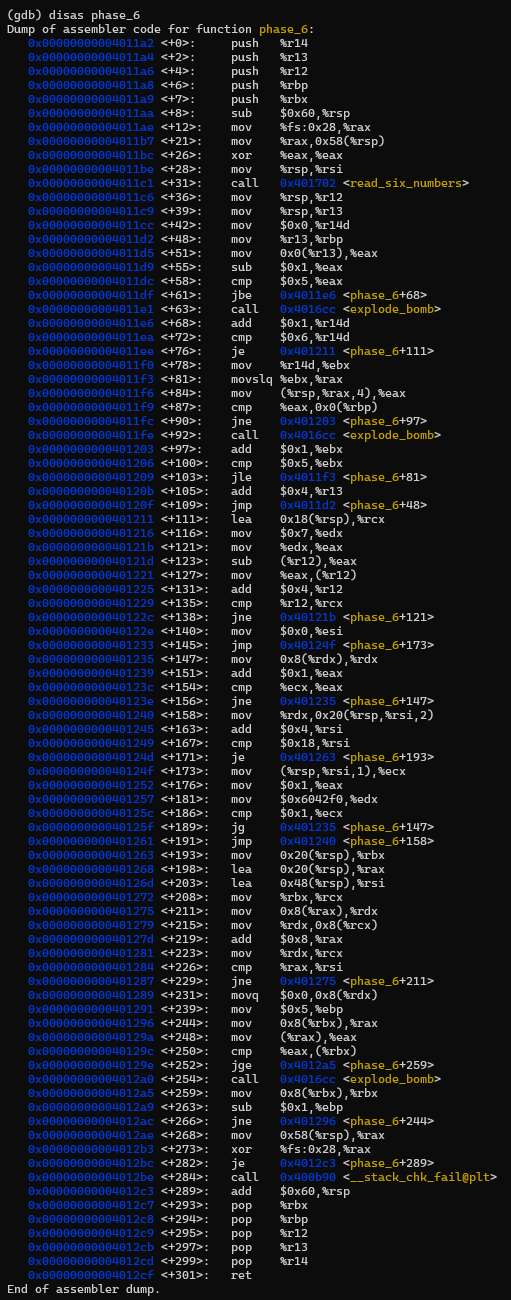
<+81>将%rdi设为%rsp；

<+84>调用strings\_not\_equal函数，根据阶段一的分析可知该函数判断%esi与%edi两个字符串是否相同。

使用x/s 0x402760 指令和x/s 0x40270e 指令查看该地址储存的值



故我们需要从0x402760的字符串中选择6个字符组成“oilers”。

所以<+51>行处的六个数为10 4 15 5 6 7，又其前一步进行了保留低4位的操作，故需要将6个数统一加上大于15的数，同时要保证能简单地在键盘中打出来。故统一加上64，得到6个输入的字符的ascii码为74 68 79 69 70 71，字符为JDOEFG，其为阶段五的答案，将其输入answer.txt文件

6、第六阶段

使用disas phase\_6 指令查看其反汇编代码

因为该阶段的代码较长，将其分成几个部分分析

阶段一 <+0>至<+31>：

根据阶段二的分析可知该阶段输入6个整数，%rsp为这个一维数组的首元素的地址，%r12=%r13=%ebp为这个一维数组的末元素的地址

阶段二 <+32>至<+109>：

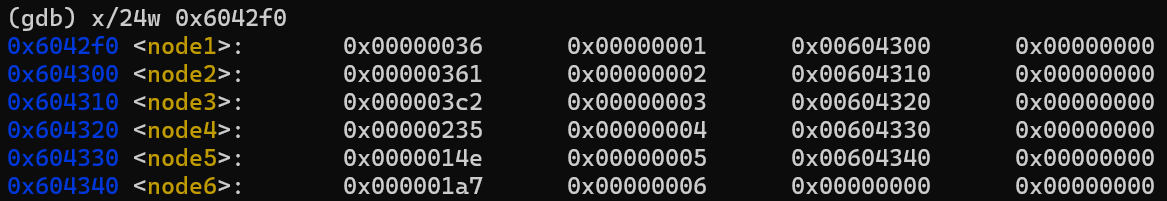
首先根据将%r14d先初始化为0后逐次加1与6比较可知进入了次数为6的循环，之后分别设置%rax=%rbx=%r14d，%rax=%rsp+4\*%eax（即%rax为第%eax个数），将其与%r13比较，若相同则炸弹爆炸，不同则继续执行；之后将%ebx加以后与 5比较可知进入了次数为从5开始每次次数减一的内层循环，每次将%r13+4（即下一个数）。由此可知该阶段遍历了6个数并要求这6个数都要小于等于6，且必须两两不相同否则炸弹爆炸。

阶段三<+111>至<+138>：

首先将%rcx设置为%rsp+24（即第6个数），随后设置%eax=%edx=7，再将第一个数设置为7减去第一个数，之后设置为下一个数并与%rcx即第6个数比较，若不同则重回循环，若相同则跳出循环。由此可知该阶段将每个输入的数都设置为7减去这个数。

阶段四<+140>至<+191>：

从阶段三跳出后将%esi设置为后执行到<+173>行，在该行将%ecx设置为%rsp+%rsi（即第%rsi mod 4个数），将%eax设置为1，将%edx设置为0x6042f0地址处的内容，将%ecx和1比较，若大于1则跳转到<+147>，否则跳转到<+158>。使用x/24w查看0x6042f0指令查看该地址的内容



由图可知该地址处的内容为一个6个节点的链表。分析<+147>行处发先将%rdx设置到下一个节点，将%eax加1后和%ecx比较，若不同则再回到循环头循环，若相同则继续执行到<+158>。故该循环将%rdx设置为0x6042f0地址处的链表中的第x个节点，x为处理后的输入的数的值，如若输入的数为1则处理后为6，%edx为链表的第6个节点。

继续从<+158>行处分析，首先将%rsp+2\*%rsi+32的地址处的值设置为%edx，之后将%rsi的值加上4后与24比较，若不同则继续循环，若相同则跳出阶段。由此可知该阶段将第每个处理后的数字的值个节点依次储存在一段线性储存地址中。

阶段五<+193>至<+229>：

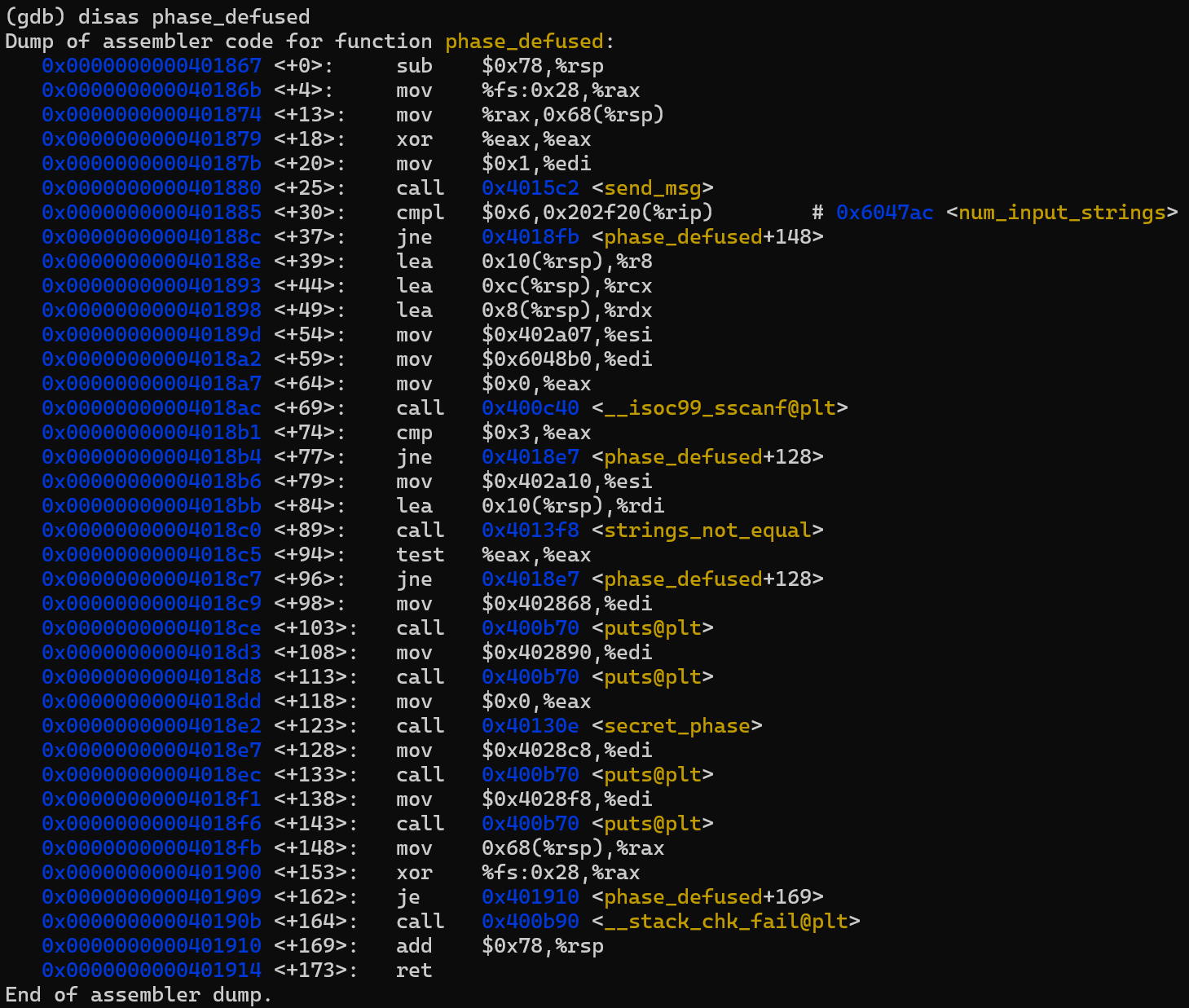
首先初始化%rax=%rbx=%rcx=%rsp+32（即储存头节点的的地址），%rsi=%rsp+72（即储存尾节点的地址），之后进入循环：%rdx=%rax+8（即下一个节点的地址），%rcx+8=%rdx，%rax=%rax+8，%rcx=%rdx，判断%rsi与%rax是否相同，若不同则回到循环，不同则跳出循环.故该阶段将节点储存的值放在以%rbx为首地址的一维数组中。

阶段六<+231>至<+301>：

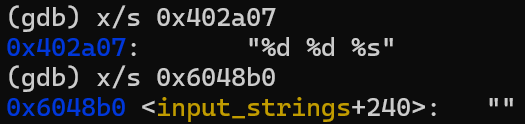
首先初始化%rdx+8=0，%ebp=5，之后进入循环：将%rax设置为%rbx下一个值，然后把它和%rax比较（即比较相邻的两个数），若前一个数大于等于后一个数则继续执行，否则爆炸，最后将%rbx指向下一个数同时将%ebp的值减一，若%ebp不为0则回到循环，等于0就跳出。故该阶段确保六个节点中的数为从大到小的排列顺序。

分析完所有过程后回到链表具体储存的数进行分析。因为阶段六要求六个节点要为倒序，故六个链表应该从1-2-3-4-5-6的顺序变为3-2-4-6-5-1的顺序，这六个数正是处理过的数。在用7减去每个数，得到最开始输入的6个数字为4-5-3-1-2-6，其为阶段6的答案，将其输入answer.txt文件

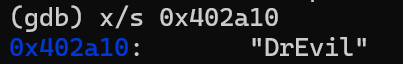
7、隐藏阶段

在分析完六个阶段的函数后，回到phase\_dufused函数进行分析。使用disas phase\_defused 指令查看其反汇编代码

在<+123>行处发现了secret\_phase函数，而想调用它必须使<+77>行与<+96>行的跳转失效。从<+77>行逆向分析，发现其需要输入一个三个内容的内容，使用x/s 0x402a07和x/s 0x6048b0指令查看内容

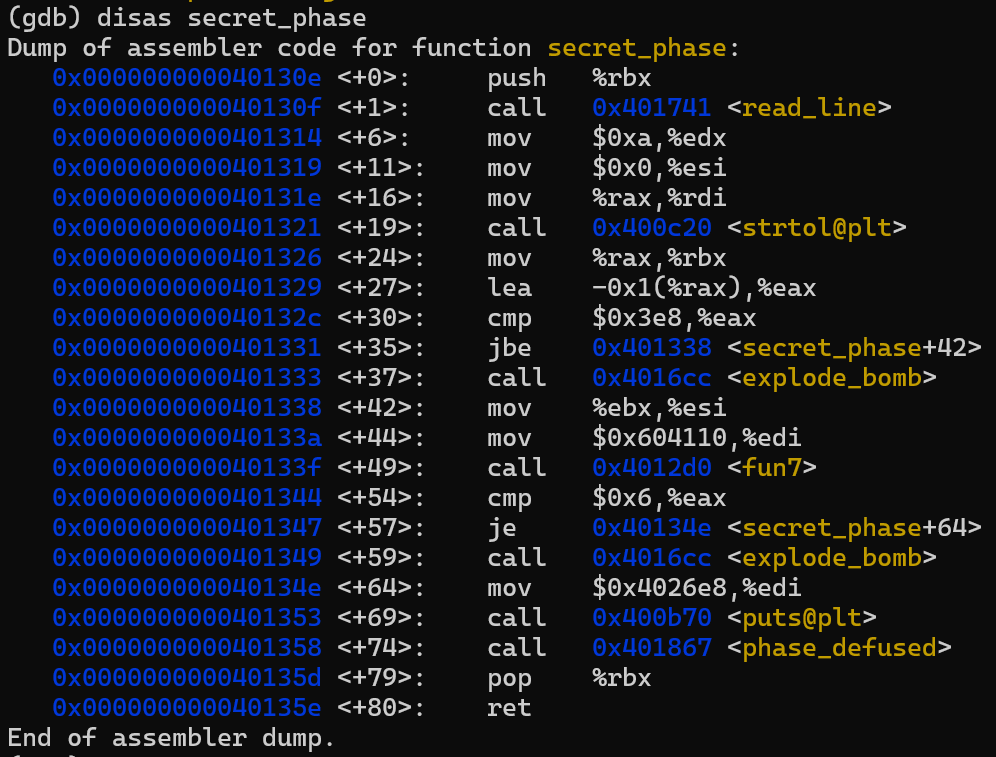


分析得需要输入两个整数和一个字符串，通过打断点的方式得知input\_strings+240处储存的值为第四阶段输入的值，故要进入隐藏阶段需要在第四阶段输入两个数字加上一个字符串。为确定字符串的内容，从<+96>行逆向分析，发现其再次调用了strings\_not\_equal函数，由之前的经验可知字符串的内容就是传入的参数%esi的值，即0x402a10处储存的值。使用x/s 0x402a10指令查看内容

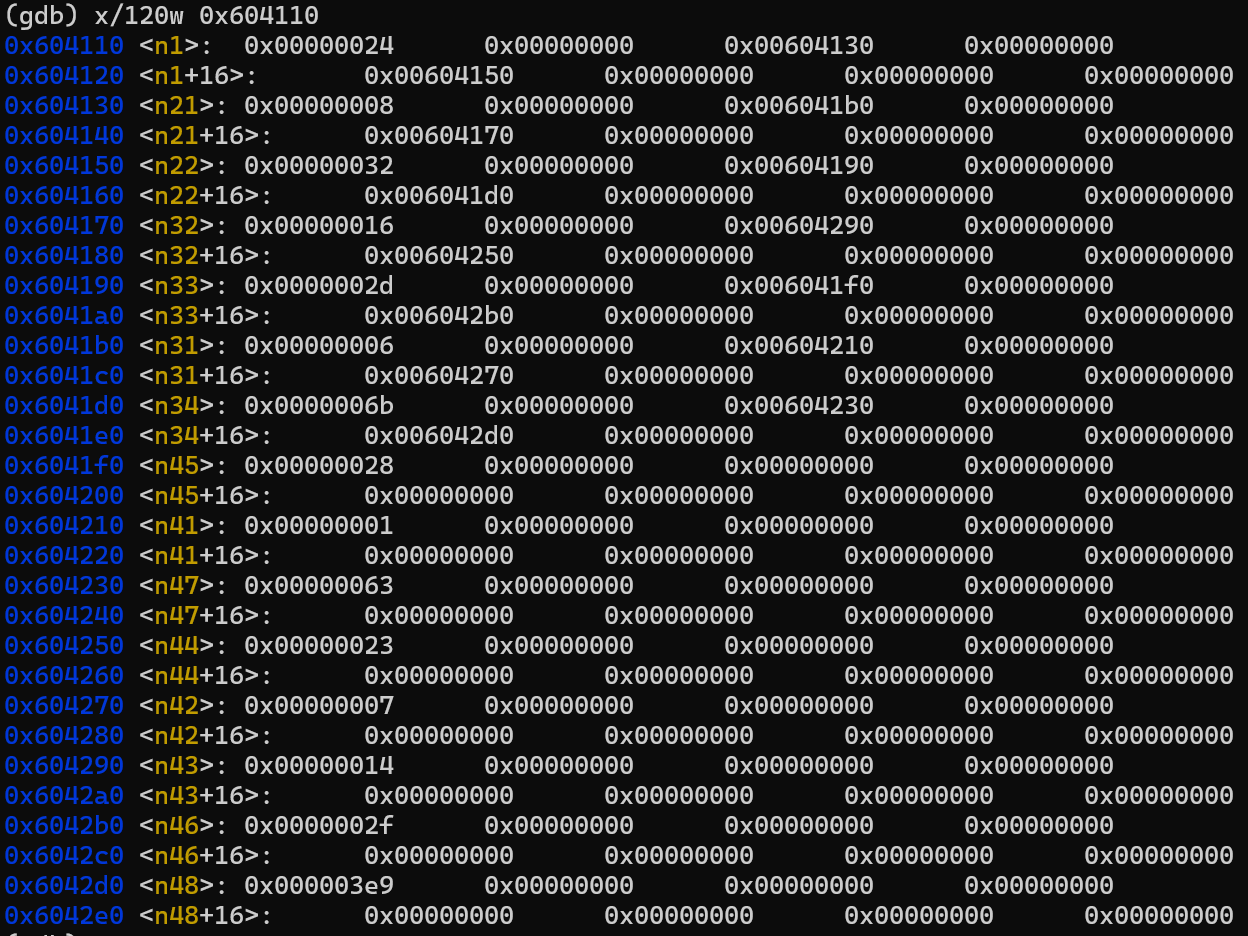


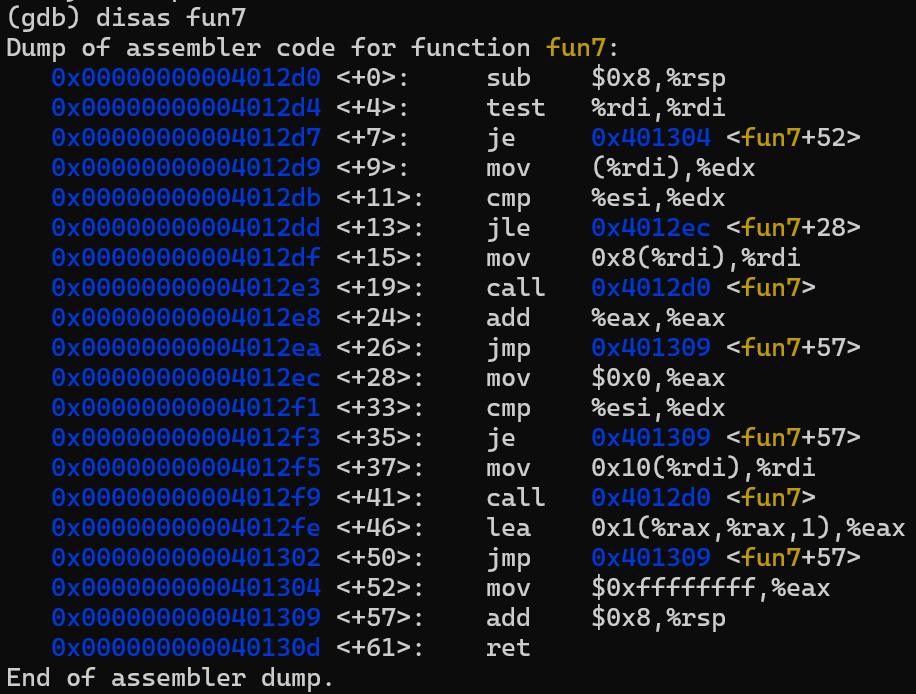
参数%esi的故在第四阶段的两个整数后加上DrEvil即可进入隐藏阶段。

输入disas secret\_phase指令查看其反汇编代码



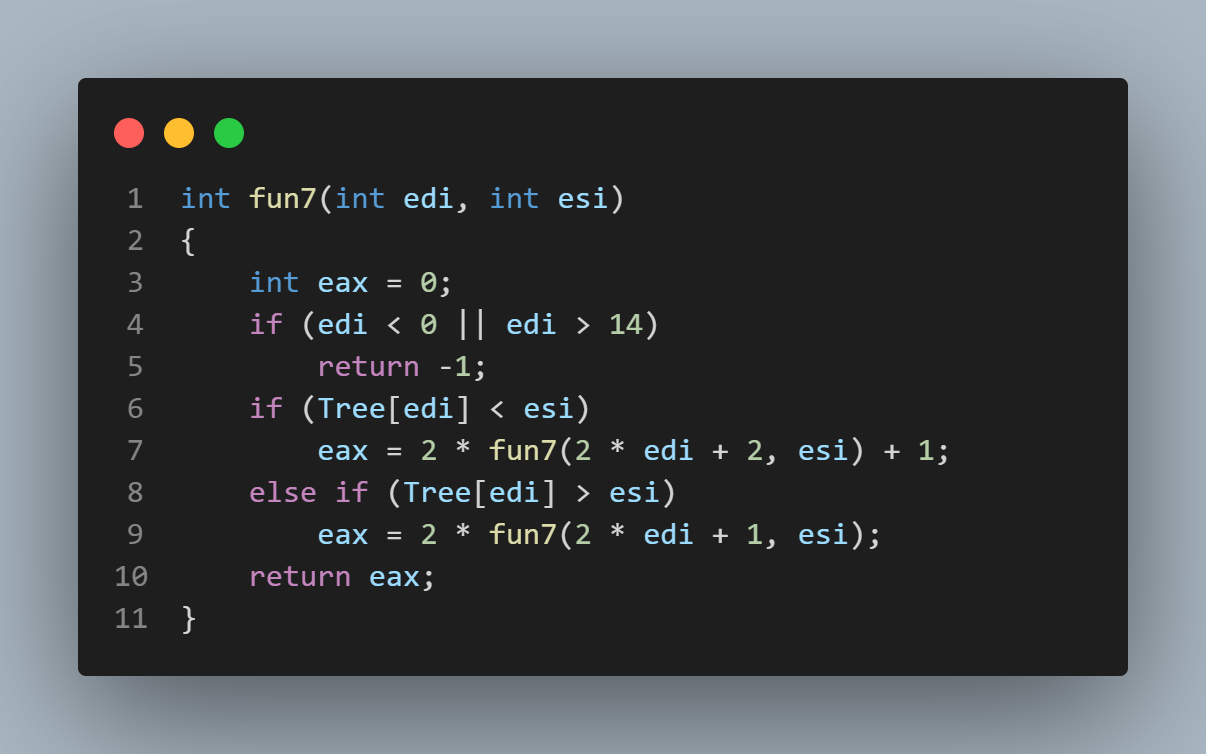
分析得函数先设置%edx为10，%esi为0，%rdi为%rax之后输入一个整数存在%rax中，%rbx为%rax，%eax的值减1，后判断%eax是否小于等于1000，若大于则炸弹爆炸，否则继续执行。使用x/120w 0x604110指令查看内容

****

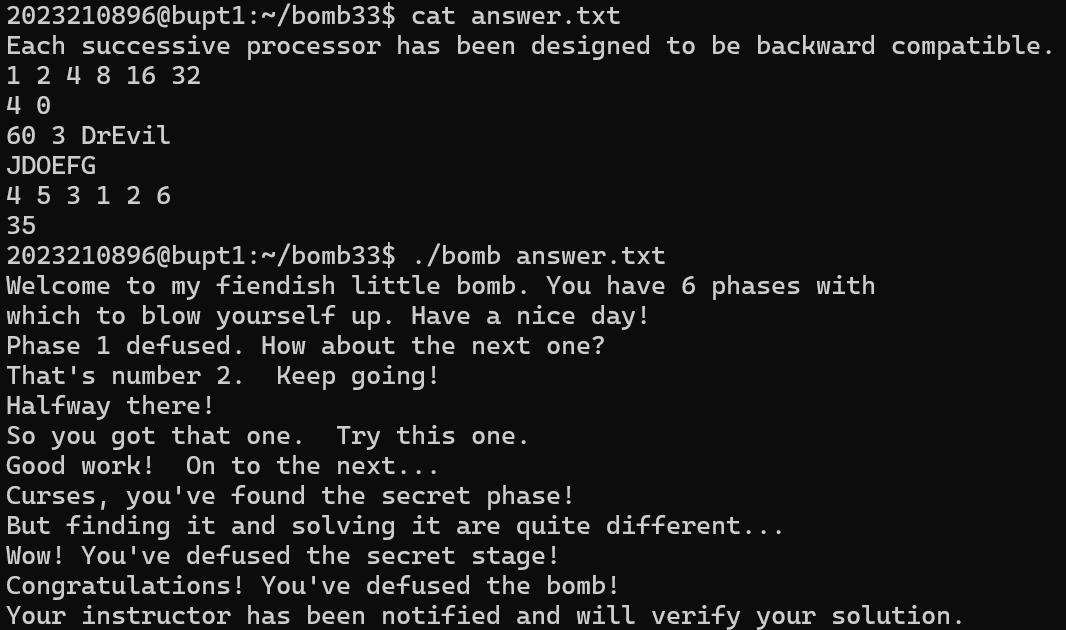
由图可知该地址处的内容为一个二叉树。随后将%esi设置为%ebx，%edi设置为该二叉树，后将%esi和%edi作为参数传入fun7函数。后判断fun7函数返回值是否为6，若不为6则炸弹爆炸，若为6则继续执行。输入disas fun7指令查看其反汇编代码

不难看出该函数出现了递归操作，故逐步分析汇编语言比较困难，不妨将其转化为C语言分析。

将汇编语言翻译成C语言为



将%esi=0，1，……1000，1001代入该函数，并令函数的返回值为6时输出%esi值，结果为35，其为阶段6的答案，将其输入answer.txt文件

五、答案汇总与实验截图

六、总结体会

在这次拆炸弹的实验中，因为担心动态调试中的调用炸弹爆炸的函数过多导致断点设少，从而误使炸弹爆炸，我全程使用静态调试的方式，即仅使用disas指令查看各个被调用的函数的反汇编代码，运用课内的知识一句一句将汇编语言翻译成C语言，虽然不如动态设置断点更加直观快捷，但是逐步分析反汇编代码让我的基本功更加扎实，加深了我对各种机器指令的理解。