

**实验报告**



**题目： 拆解二进制炸弹**

**班 级： 2020211302**

**学 号： 2020211262**

**姓 名： 鄭毓恒**

**学 院： 计算机学院**

**2021年 11 月11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. SecureCRT（10.120.11.12）
3. Linux
4. Objdump命令反汇编
5. GDB调试工具
6. 积分榜（http://10.120.11.13:19210/scoreboard）

**报告邮寄（x86版最迟时间：2021年11月15日晚23：59；Arm版最迟时间：2021年11月22日晚23:59）：**

**大一班（1-4班）：**clavicle@bupt.edu.cn

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

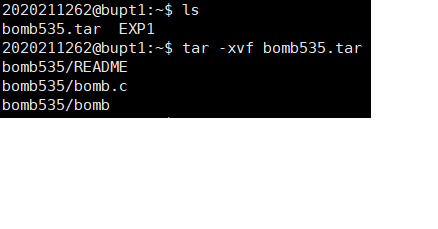
建议按照：准备工作、阶段1、阶段2、…等来组织内容

各阶段需要有操作步骤、运行截图、分析过程的内容

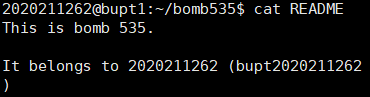
准备工作：



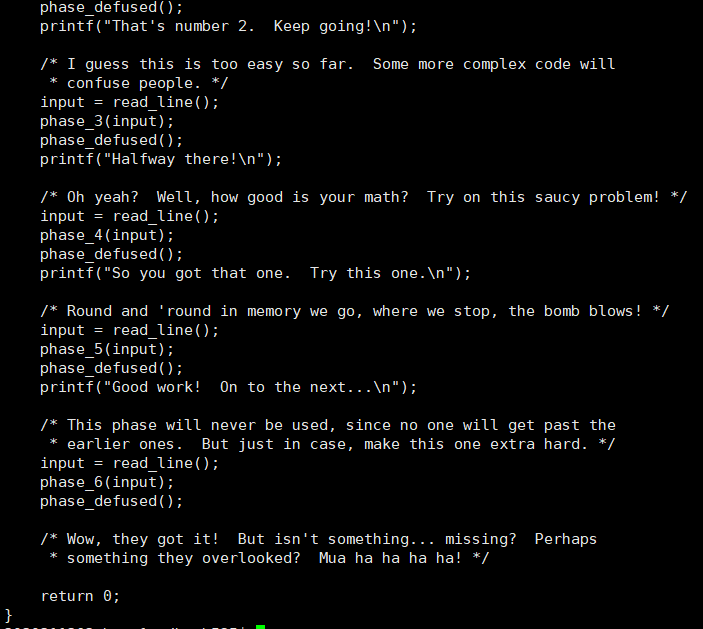
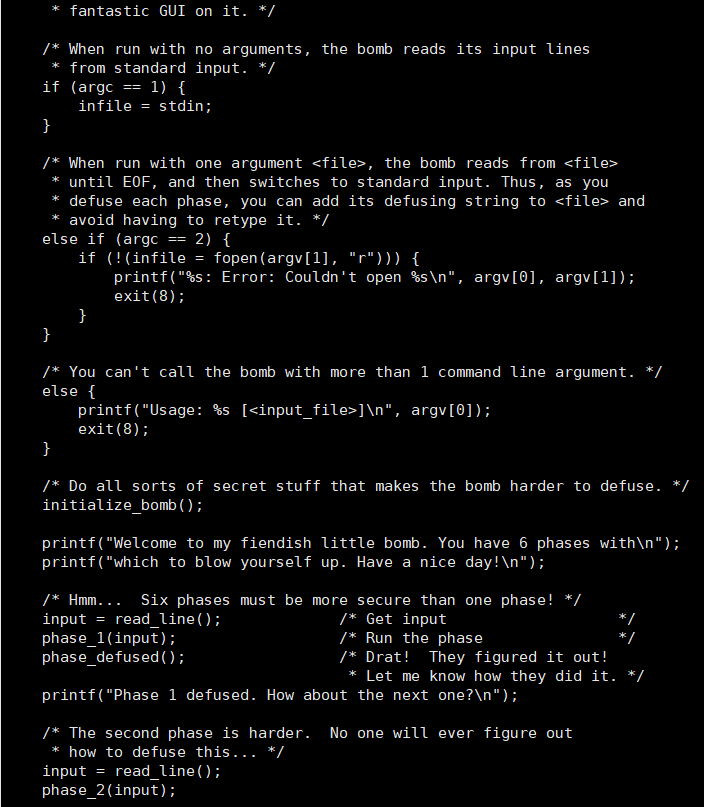
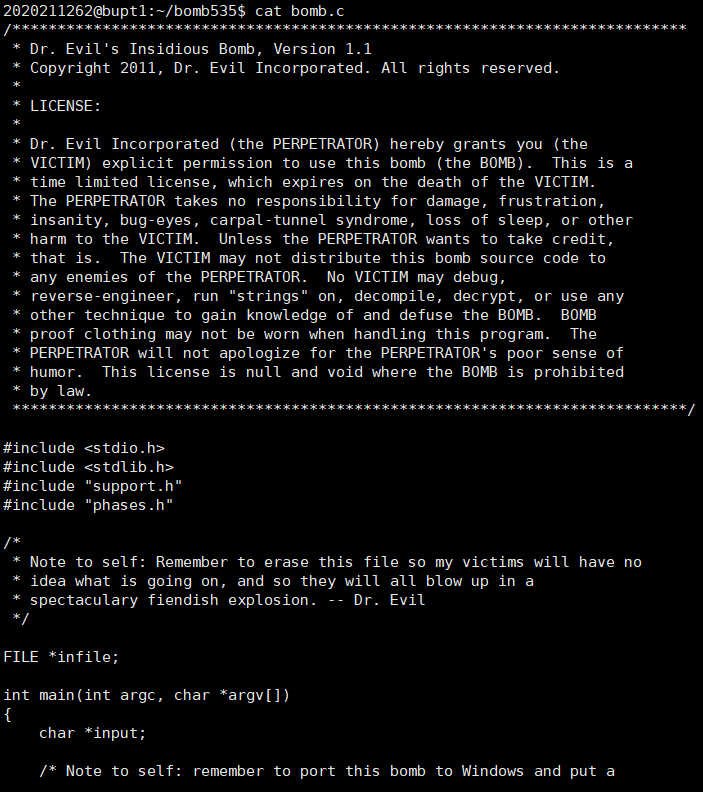
首先登录bupt1服务器，在home目录找到Dr. Evil设置的二进制炸弹bomb535.tar，535是我专属的炸弹编号。



输入指令tar -xvf bomb535.tar，生成一个名为bomb535的目录。目录当中有bomb、bomb535、README三个文件。README文件标识了炸弹和它的拥有者，也就是我。bomb.c和bomb分别是炸弹的源代码和可执行文件。

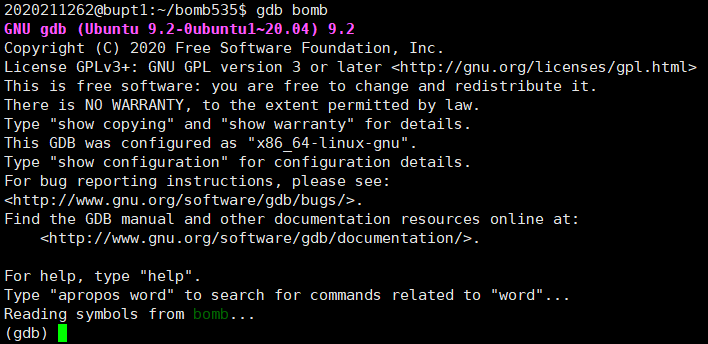


README文件标识了这是炸弹535，属于学号2020211262的我。

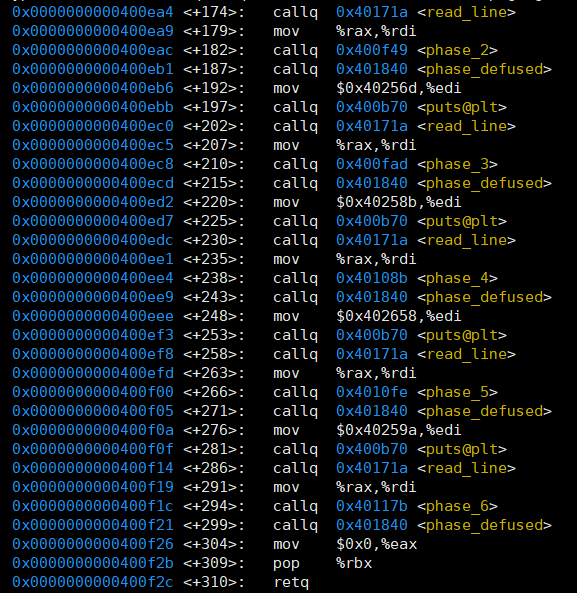
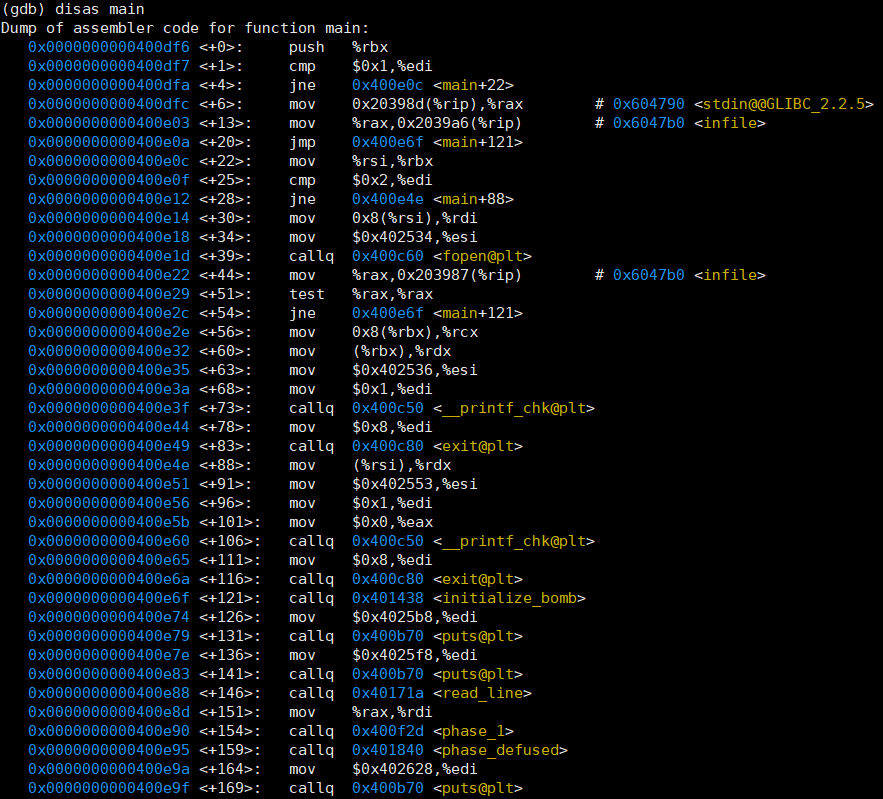


阅读bomb.c文件，可以发现当程序开始时或一个炸弹被拆除时会打印几条字符串。并且，在注释中，有一些Dr. Evil给的关于炸弹的提示。程序中目前观察中存在phase1至phase6六个炸弹，每次先输入一行字符串再调用炸弹函数，然后再调用phase\_defused函数。

源代码在开头包含了support.h和phases.h头文件，函数phase1-phase6等存在于这些头文件当中。我没有这些头文件的资料，因此这些函数的C语言源代码不可见，也不能利用-g参数编译再用反汇编显示源代码。因此，利用已给的可执行文件bomb和gdb调试，反汇编显示程序汇编代码来拆除炸弹。



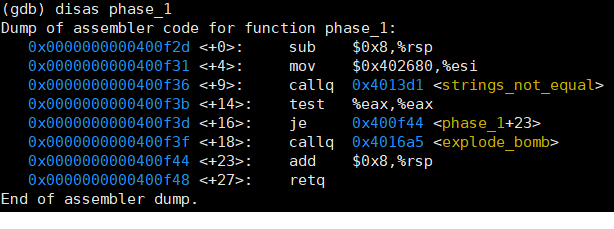
进入bomb可执行文件的gdb调试。



用gdb中的disas指令反汇编main函数，得到main函数的汇编代码，熟悉程序大体结构。

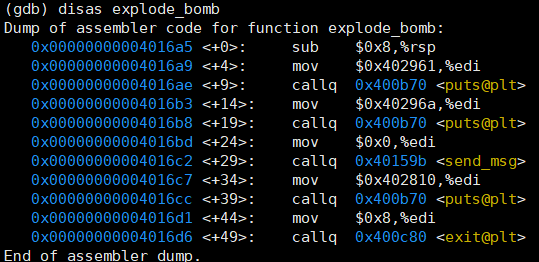
开始阶段一，拆除phase1炸弹。

阶段1：

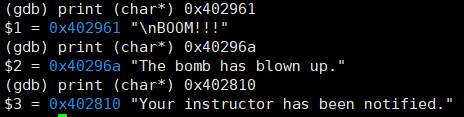


用gdb的disas指令反汇编后，阅读第一个炸弹phase\_1函数的汇编代码。

可见该函数先把地址0x402680传递给寄存器esi，然后调用strings\_not\_equal函数。如果该函数的返回值等于0，phase\_1函数返回主函数，否则调用explode\_bomb函数。



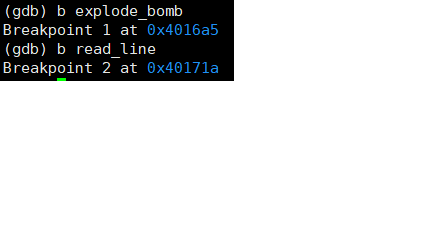
反汇编explode\_bomb函数。该函数会打印地址在0x402961,0x40296a,0x402810的三个字符串。



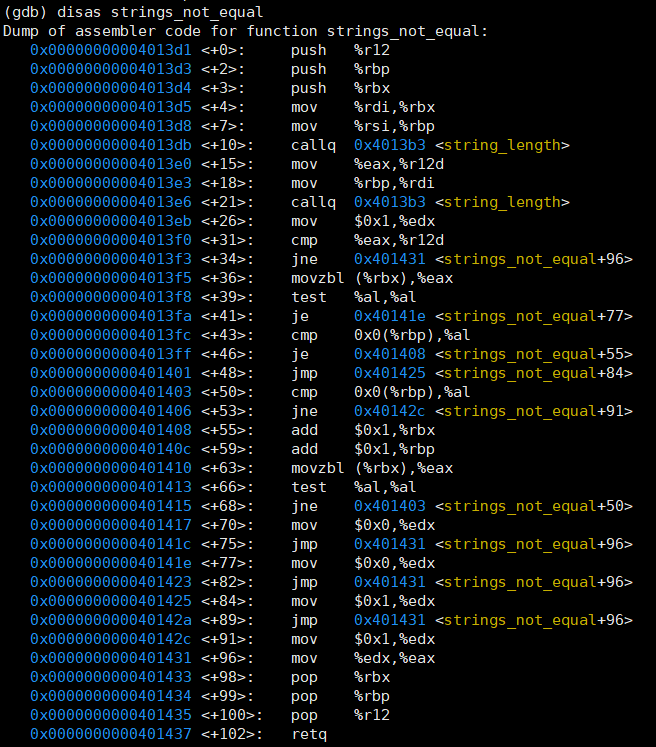
打印出这三个字符串，发现是炸弹爆炸的提示。

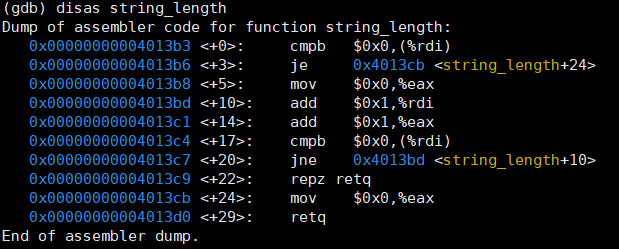
函数在打印完这三句后，调用exit函数使程序退出。因此，一旦调用explode\_bomb，炸弹就会爆炸，程序就会终止，如它的函数名所述。

以下的过程中，要防止程序调用该函数。



利用gdb的break指令在explode\_bomb函数设置断点。这样，在每次程序调用该函数之前程序都会暂停，防止炸弹爆炸，我们可以安全试错。

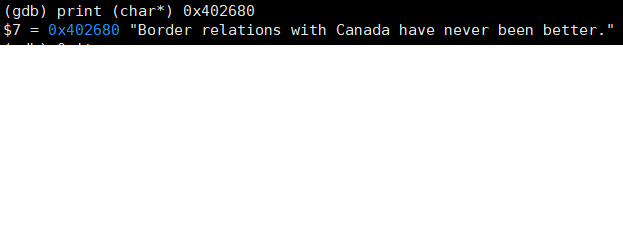




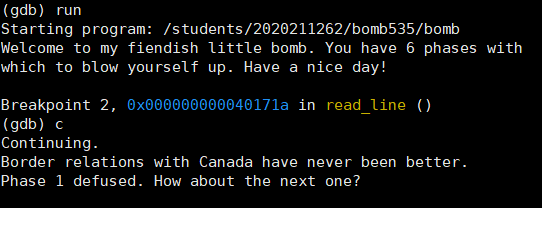
接下来看strings\_not\_equal函数。发现它会调用string\_length函数，因此还要看看string\_len函数。

两个函数的功能很清晰，如它们的函数名所述，string\_length函数返回一个字符串的长度，strings\_not\_equal函数判断两个字符串是否不相等，不相等则返回1，相等则返回0.

综上，phase\_1函数中，如果输入的字符串与地址0x402680的字符串不相等，炸弹爆炸。

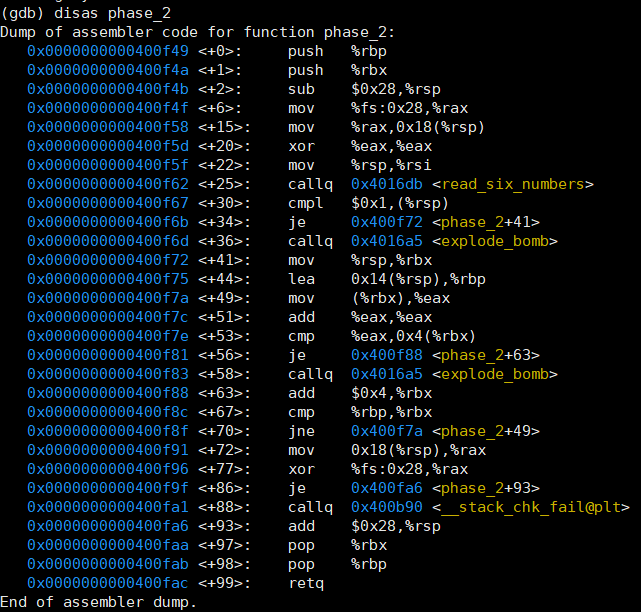


将0x402680地址的字符串打印出来，这应该就是拆除第一个炸弹所需要输入的字符串。

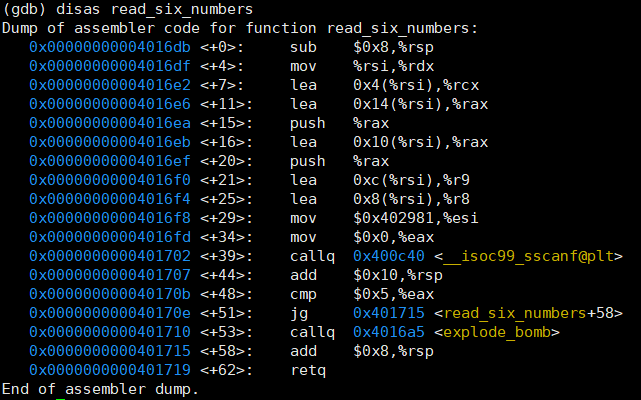


当输入同样的字符串时，程序提示“Phase 1 defused”。炸弹一成功拆除。开始阶段二，拆除第二个炸弹phase\_2。

阶段2：



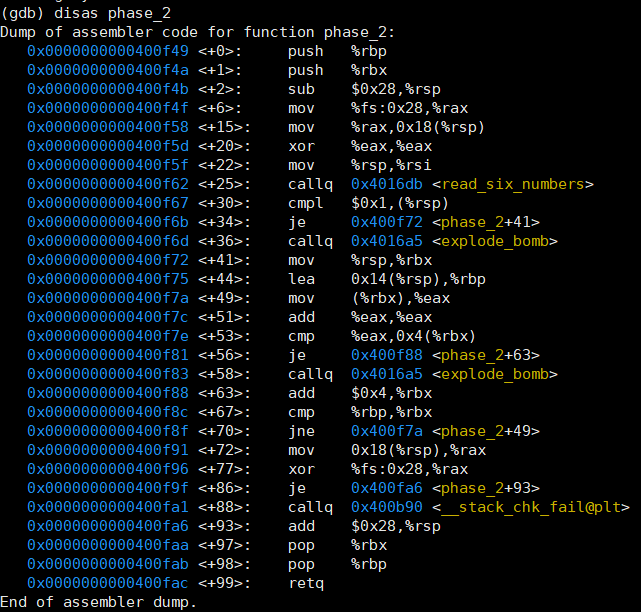
看phase\_2函数的汇编代码。该函数首先调用了read\_six\_numbers函数。



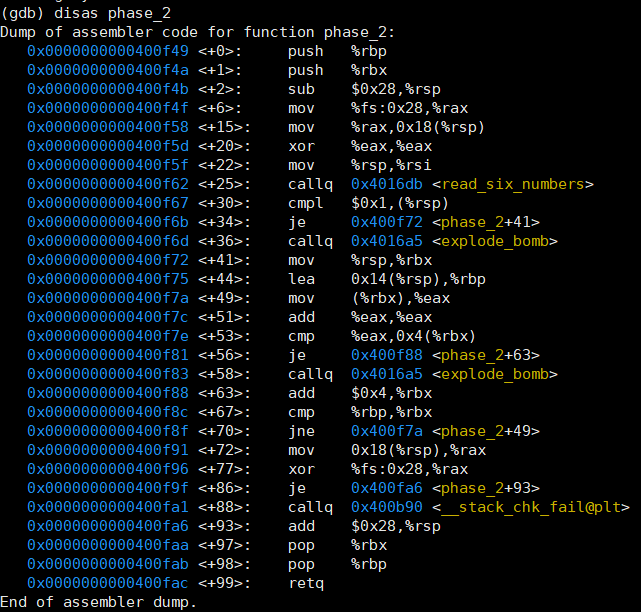


这个函数先将地址0x402981的字符串传给寄存器esi，再调用sscanf函数。地址0x402981的字符串打印出来为六个以空格分隔的%d，以函数名和调用了sscanf推断，这个函数将我输入的字符串以六个整数的形式读入，也就是说我在第二个炸弹这里需要输入六个整数。并且，函数之后将寄存器eax，也就是sscanf函数的返回值，与5比较。若返回值不大于5，也就是读入的整数的数量不大于5，调用explode\_bomb函数，炸弹爆炸。

已知拆除炸弹的密码是六个整数，回到phase\_2函数。

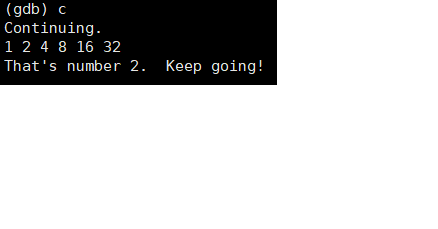


寄存器rsp存储的是输入的六个整数的首地址。函数将该地址存储的整数，也就是输入的第一个整数与1比较(cmpl $0x1, (%rsp))，如果不等于则调用explode\_bomb函数，爆炸。因此，第一个输入的整数必须为1.



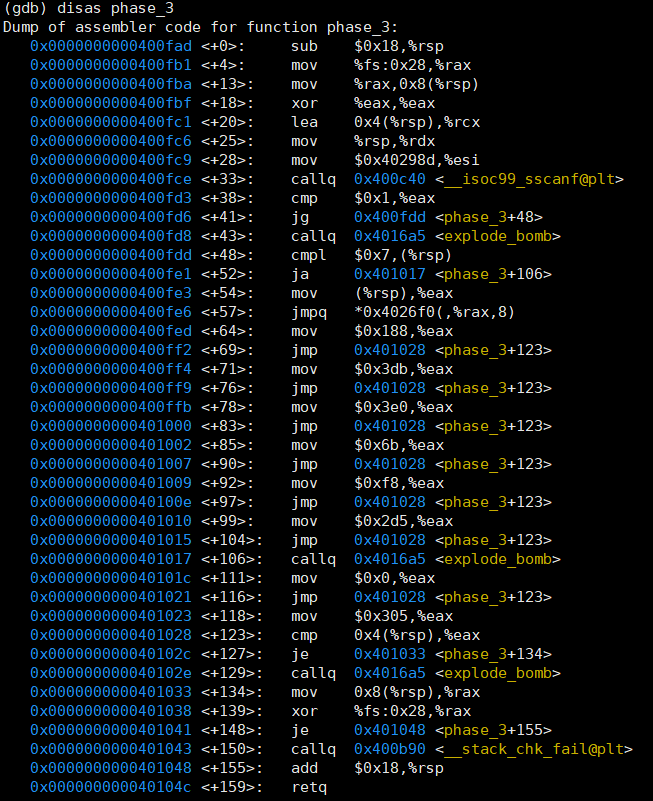
接着往下读，发现到<+70>处出现一句条件跳转指令，跳转至<+49>，此处存在一个循环。进入循环前，首先将s寄存器rsp存储的值，也就是输入的六个整数所在的数组的首地址，赋值给rbx。然后，将rsp+0x14的地址赋值给rbp，0x14=20，也就是从数组首地址往后移到第五个整数的地址。从<+49>开始循环，循环首先将寄存器rbx所存储的地址上的值，也就是当前移动到的整数，赋值给eax。然后，eax加上自己，也就是乘以二。之后，将eax与存储在rsp加上0x4（四个字节）的地址比较，如果不等于则爆炸。如果等于，rbx加上0x4（四个字节），移动到下一个整数的地址，然后在与rbp比较，如果等于则代表第五个整数已经和第六个整数比较完了，退出循环。因此，输入的六个整数第一个必须是1，并且下一个数是前一个的两倍。

函数phase\_2往后的汇编代码不会再跳转和调用explode\_bomb函数，即与拆除炸弹无关。可以尝试输入1, 2, 4, 8, 16, 32六个整数拆除炸弹。

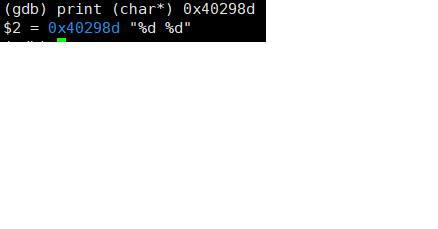


第二个炸弹拆除成功。开始阶段三，拆除第三个炸弹phase\_3。

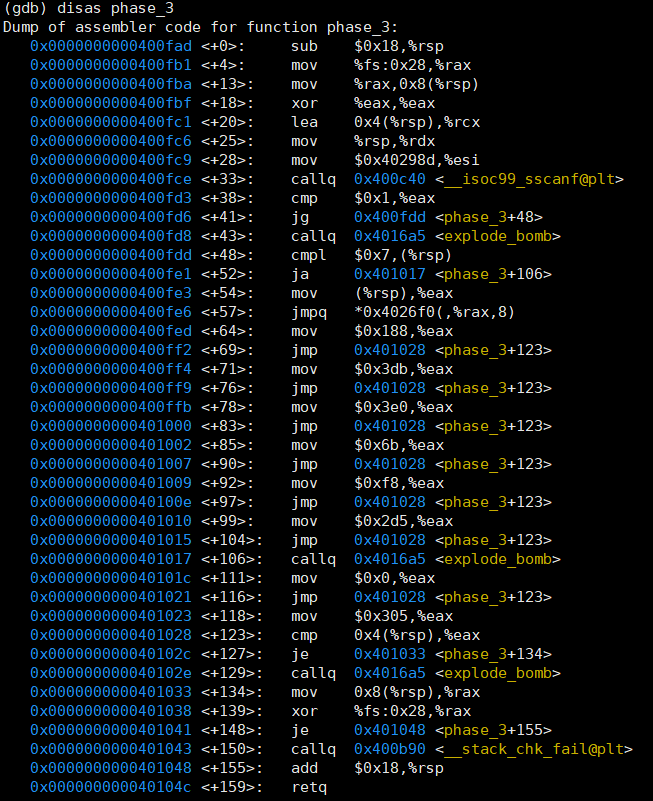
阶段3：



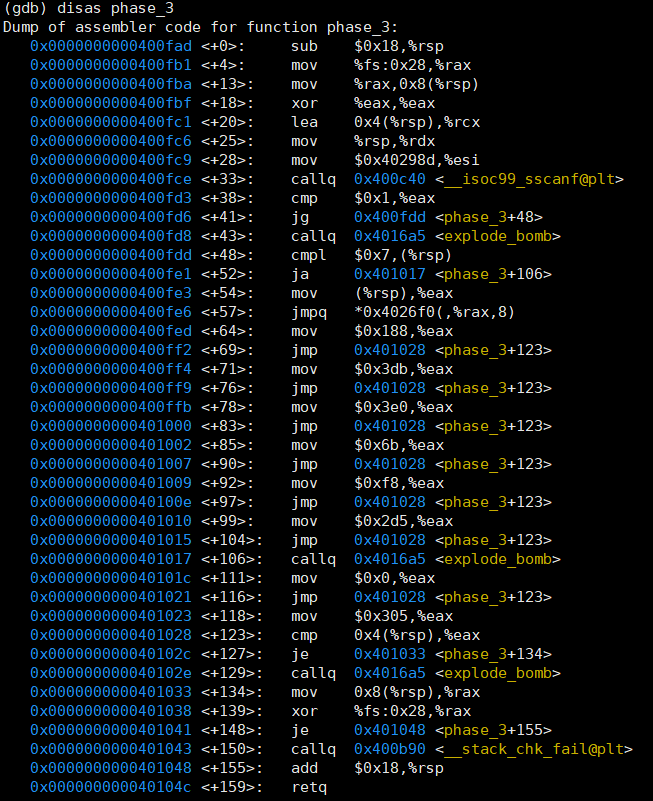
观察phase\_3函数的汇编代码，发现函数将地址0x40298d传递给寄存器esi，然后调用sscanf.



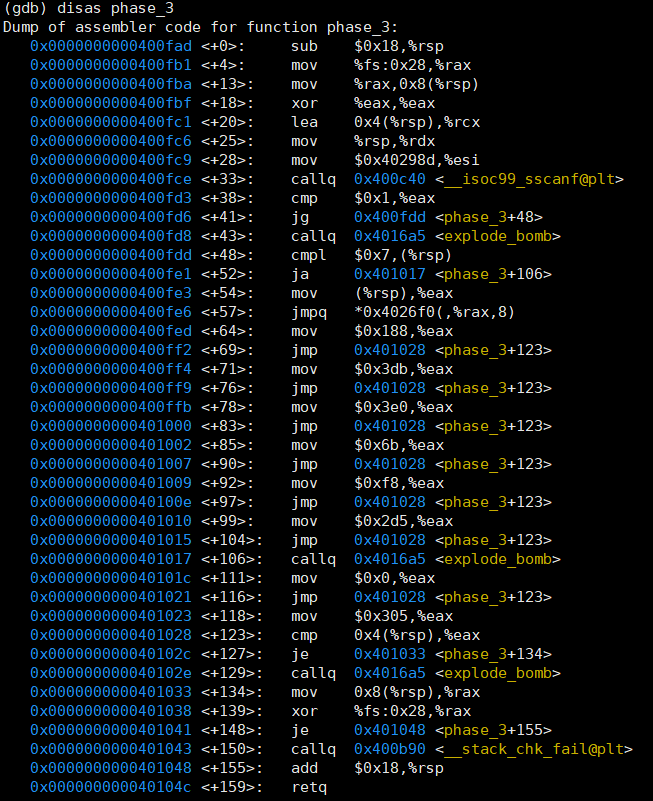
将地址为0x40298d的字符串打印出来，结合sscanf函数，这里应该是需要输入两个整数。



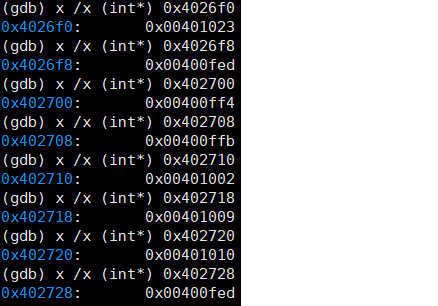
这里将sscanf函数的返回值与1比较，不大于1就爆炸。所以，确实是输入两个整数。



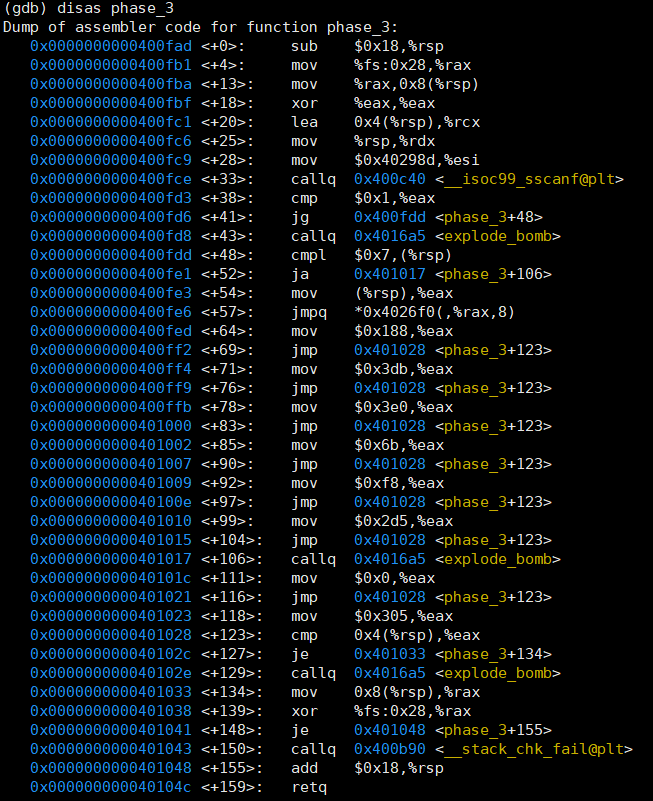
($rsp)，也就是输入的首地址所存储的值，输入的第一个整数。将其与7比较，如果不大于7则跳转到<+106>，爆炸。所以，输入的第一个整数必须小于或等于7. 同时，这里用的跳转指令是ja，因此输入应该是unsigned类型，输入为0-7.



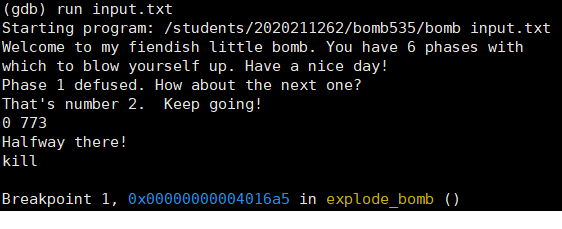
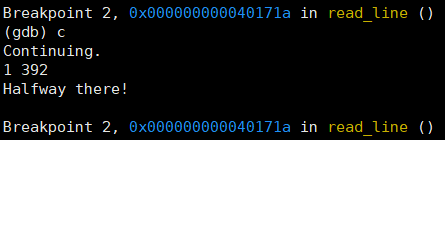
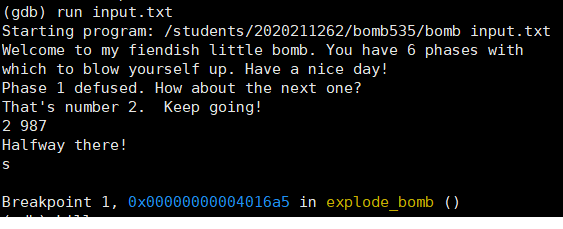
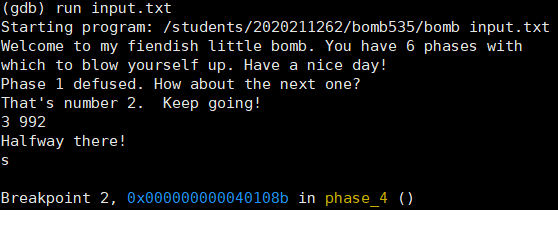
这里首先将(%rsp)，也就是输入的第一个整数，赋值给eax，然后将0x4026f0加上8乘以eax，得到一个地址，跳转至该地址所存储的地址。

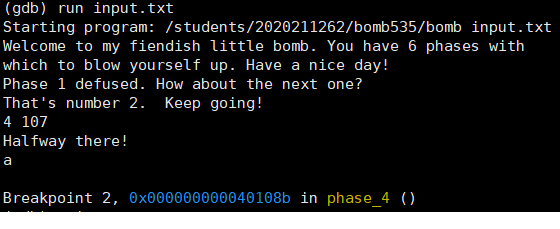
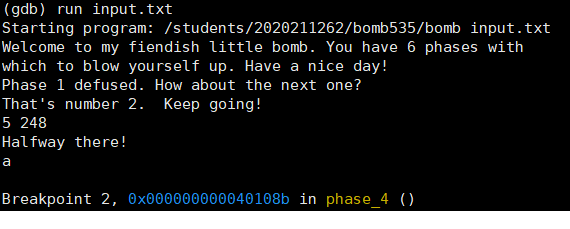
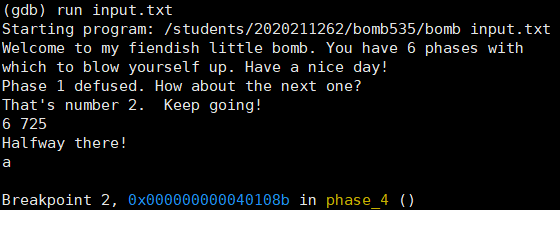
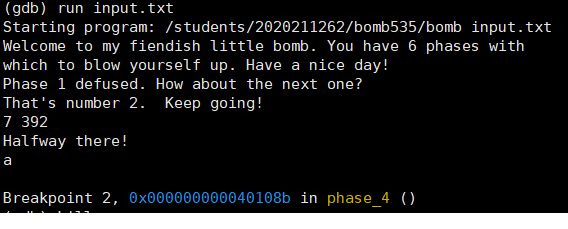


由于第一个数字有0-7八种可能，所以上一步运算后，根据输入值，有八种可能的跳转地址。将其一一打印出来。这些地址全都存在于phase\_3函数当中，跳往每个地址后，执行不同的指令，再跳转到一个统一的地址<+123>，结构类似于switch语句。



根据第一个输入的值，程序会赋值给寄存器eax不同的值，然后再与输入的第二个整数比较。如果不等于则爆炸。因此，可知拆除第三个炸弹，需要输入两个整数，第一个在0-7之间，第二个为switch语句根据第一个整数赋值给eax的值。

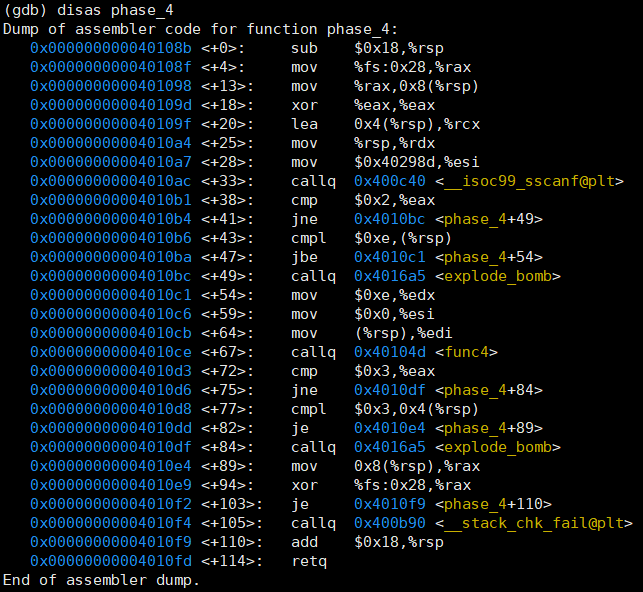
   

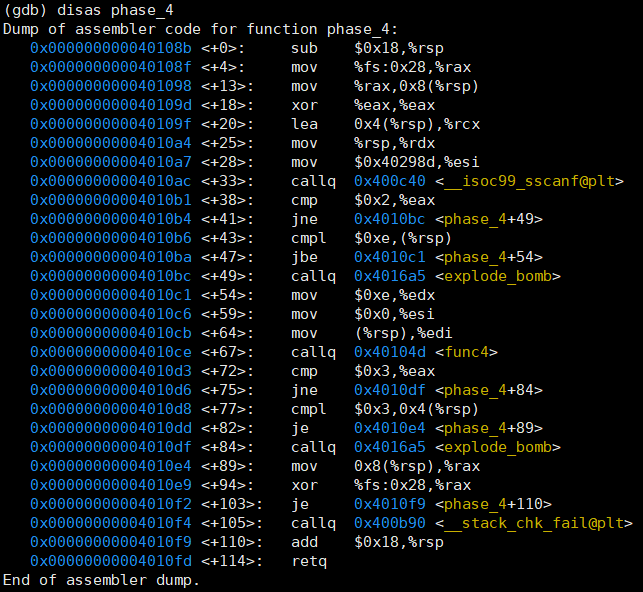
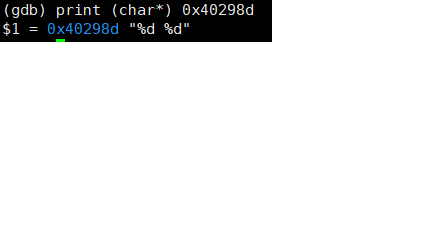
由于是switch语句，推测有8种解。一一输入测试，结果都成功，第三个炸弹被拆除。

开始阶段四，拆除第四个炸弹phase\_4.

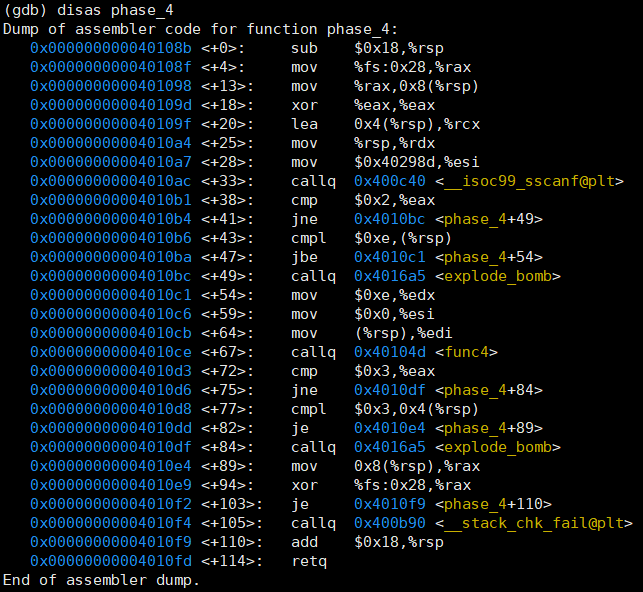
阶段4：



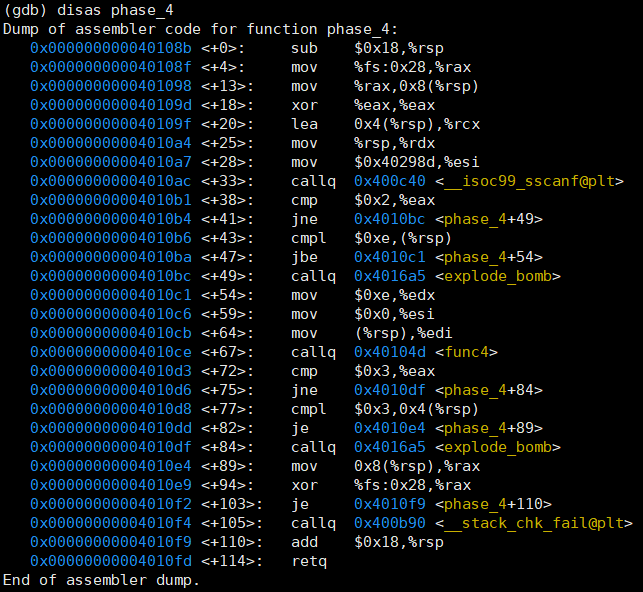
观察phase\_4函数的汇编代码，发现这里也是将一个地址0x40298d传递给esi，再调用sscanf函数。



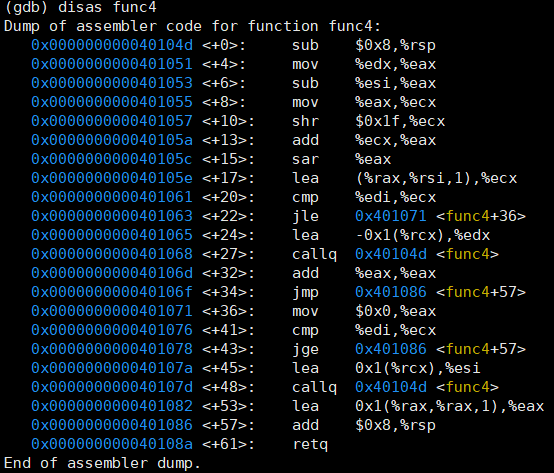
把该地址的字符串打印出来，可见phase\_4需要输入两个整数。并且，调用sscanf函数后，程序又将返回值与2比较，如果不等于则爆炸。因此，必须输入刚好两个整数。



看下几句汇编代码，将输入的第一个数与0xe(14)比较，如果大于14则爆炸。并且，这里用的是jbe指令，所以输入的是应是unsigned类型。输入的第一个整数的值在0-14之间。

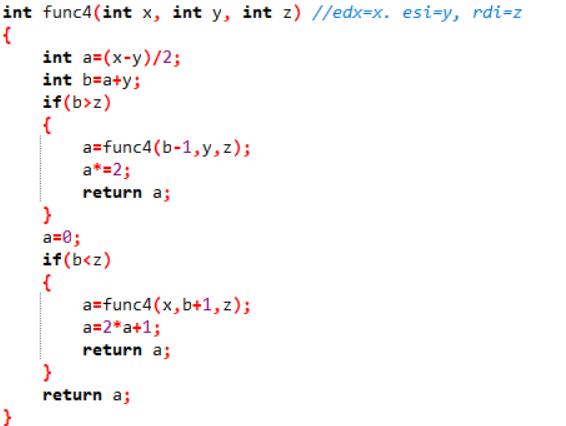


程序后续调用了func4函数，并且该函数的参数在edx,esi以及edi存储，edx=14,esi=0,edi等于输入的第一个数。

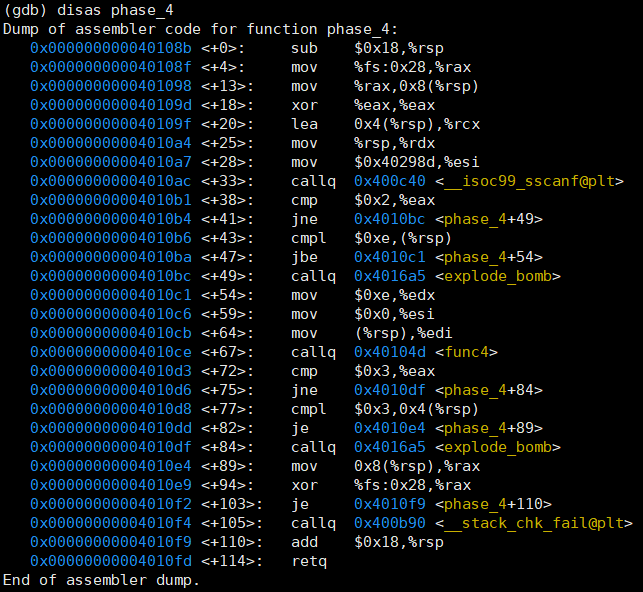


看func4函数的汇编代码。一步步看，eax=edx，eax-=esi，eax=edx-esi。然后，ecx=eax，ecx逻辑右移31位，也就是取ecx的最高位，已知ecx=eax=edx-esi <= 14-0 ,所以最高位为0，ecx=0,eax+ecx=eax.接着，eax算术右移一位，也就是eax=(edx-esi)/2，然后ecx=eax+esi。上面这些可以简化为eax=(edx-esi)/2，ecx=eax+esi。

下面可见函数根据条件，改变参数的值，调用自己。因此，这里存在函数递归的运用。

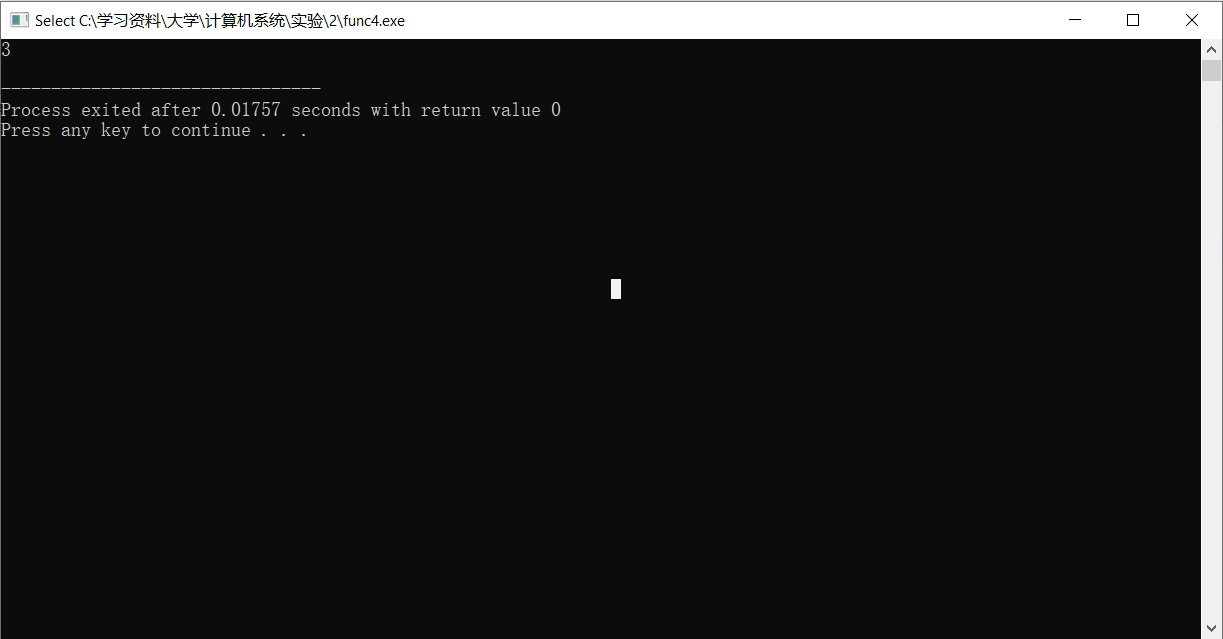


将func4函数的汇编代码一一解读，最终可知func4函数可以写成以上的C语言代码函数，实现类似功能。

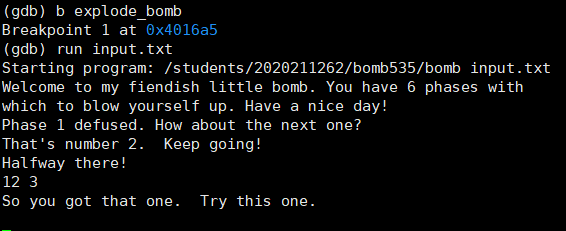


在调用func4函数后，程序将eax与3比较，如果不等于则爆炸。并且，将输入的第二个数与3比较，不等于的话同样爆炸。因此func4函数的返回值以及输入的第二个数必须为3.

因为之前已经推出func4函数的类似C语言代码，而且知道除了edi以外其他参数的初始值，edi存储的是输入的第一个数的值的范围在0-14.经过计算，输入12应该能得到返回值为3，用C语言程序进行测试。

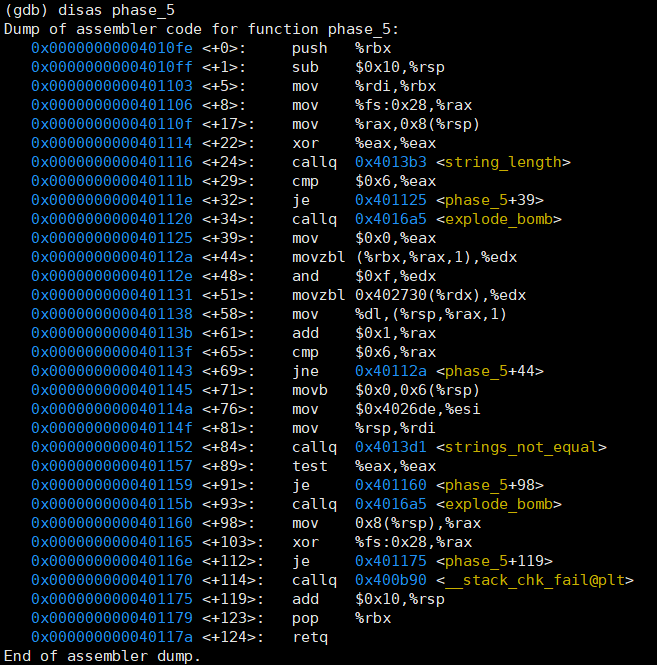


程序打印了func4函数的返回值。当输入为12时，返回3.因此，拆除第四个炸弹应该输入12， 3两个整数。

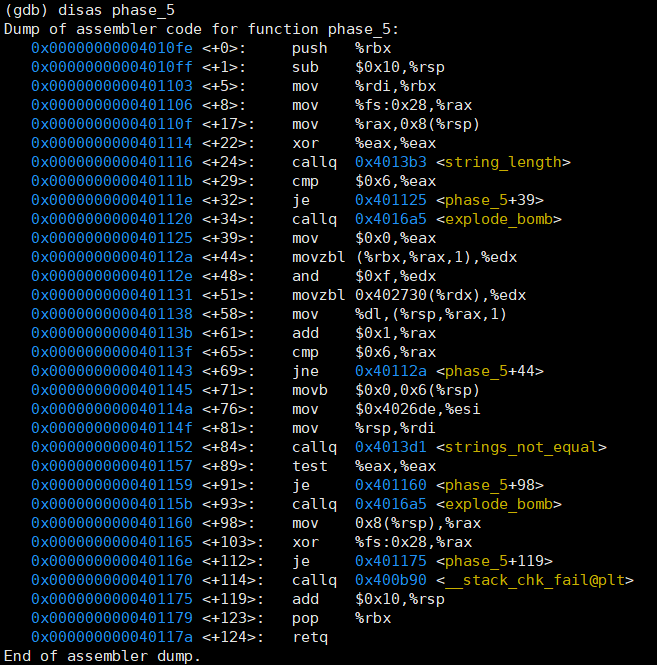


将12，3输入，phase\_4通过，第四个炸弹成功拆除。开始阶段五，拆除第五个炸弹phase\_5.

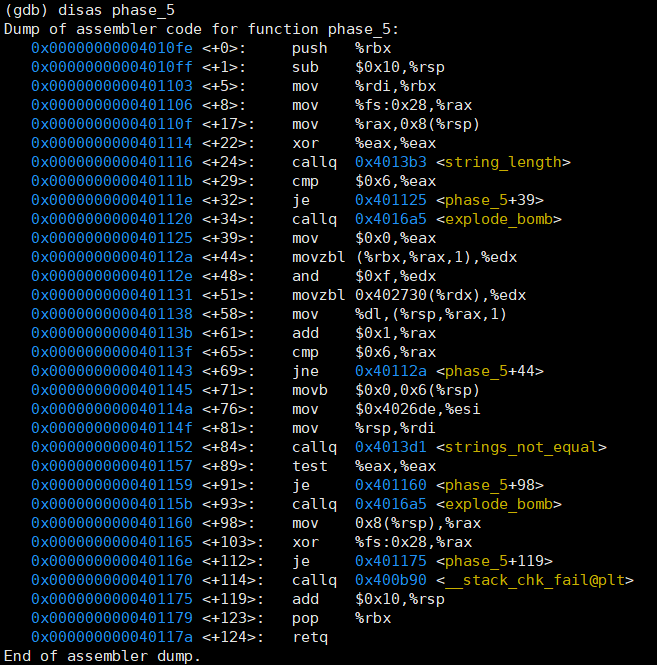
阶段5：



看phase\_5的汇编代码，可见函数调用了string\_length和strings\_not\_equal函数，因此这里需要输入的应该是一段字符串。

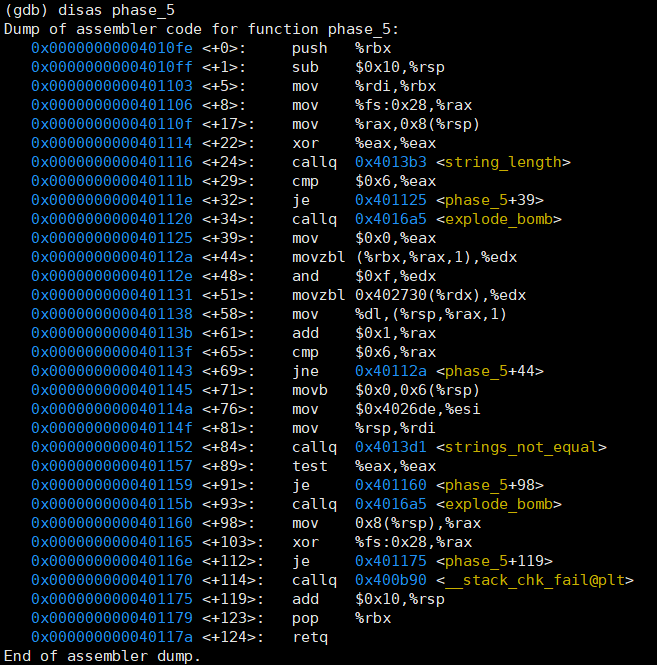


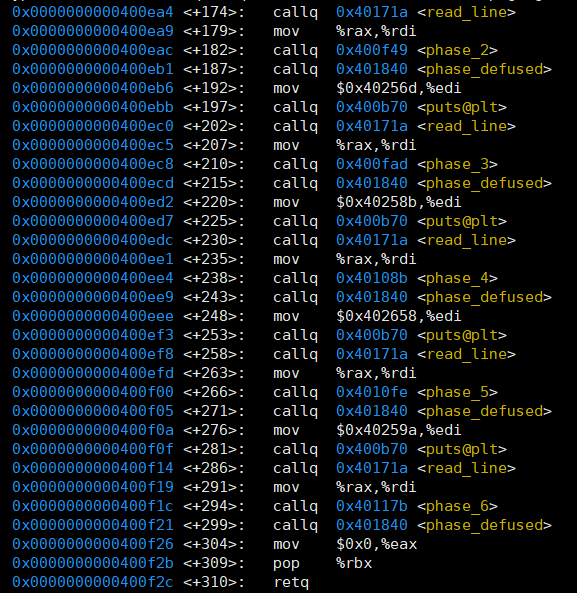
函数string\_length返回了输入的字符串的长度，长度必须等于6，否则爆炸，因此需要输入6个字符。



执行到<+69>处，有一个条件跳转语句，可能跳转到<+44>。因此这里存在一个循环。

循环开始前，eax被赋值为0。在循环的最后两句，rax加1，如果rax不等于6，跳转到<+49>.因此rax在这里相当于循环的计数变量i的角色。





函数phase\_5开始时将rbx赋值为rdi，见main函数的汇编代码，rdi为输入的字符串的首地址。

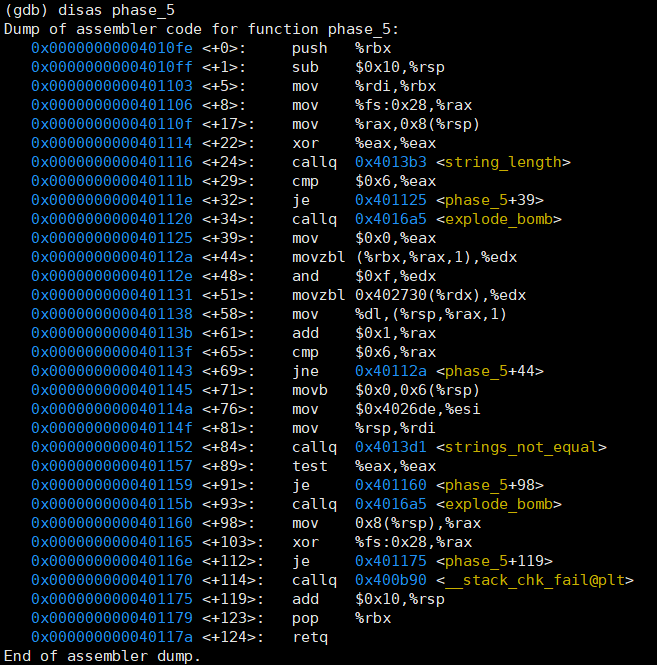
以下将rax称为变量i。

循环的第一句，将rbx+i处地址的一个字节的值赋值给edx。然后edx与0xf做and运算，相当于将edx除了最低四位以外都取0，最低四位不变。接着，将地址0x402730加上rdx个字节处的一个字节的值赋值给edx，再将dl也就是edx的最低8位的值存储在rsp+i处的地址。循环变量i++，继续下一个循环，直到i==6.



0x402730处的字符串为上。

设输入的字符串为input，循环是将input[i]赋值给edx。取edx的最低四位意味着取input[i]的ASCII码的二进制的最低四位。然后根据这四位的值x，从0x402730处的字符串c,取出c[x]，赋值给rsp+i处的地址，以rsp为首地址，产生新的字符串str。

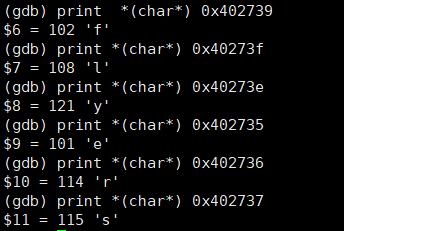


退出循环，函数将str[6]赋值为0，也就是‘/0’，使首地址rsp处存在完整的字符串。

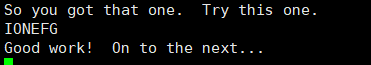
然后，将地址0x4026de赋值给寄存器esi，rsp赋值给rdi，调用strings\_not\_equal函数。如果返回值不等于0，则爆炸。因此，字符串str和首地址为0x4026de的字符串必须一样。



0x4026de处的字符串是“flyers”。因此，str必须同样是“flyers”。

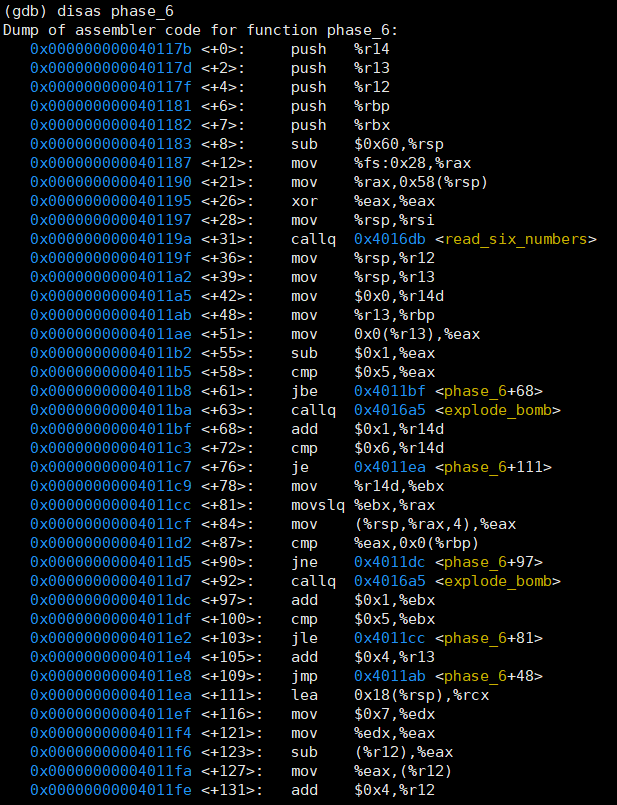


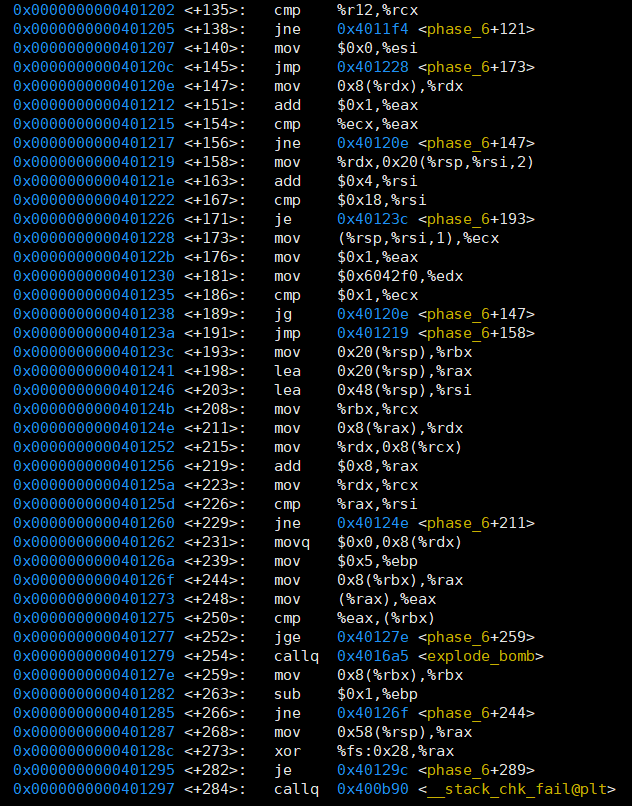
根据字符串x的首地址0x402730，“flyers”的六个字符的地址分别如上。因此，输入的六个字符串的ASCII码的最低四位分别为9，15，14，5，6，7. 不能直接输入ASCII码为这6个值得字符，因为它们都不是可显示字符，难以输入。可显示字符的ASCII码的范围在32-126，其中有大写A到Z二十六个字符。因此，输入ASCII码低四位符合要求的大写字母。这六个字母为IONEFG。

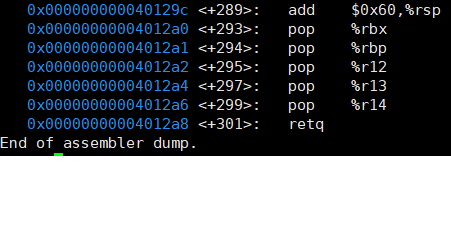


输入IONEFG，第五个炸弹成功拆除。开始阶段六，拆除第六个炸弹phase\_6。

阶段6：

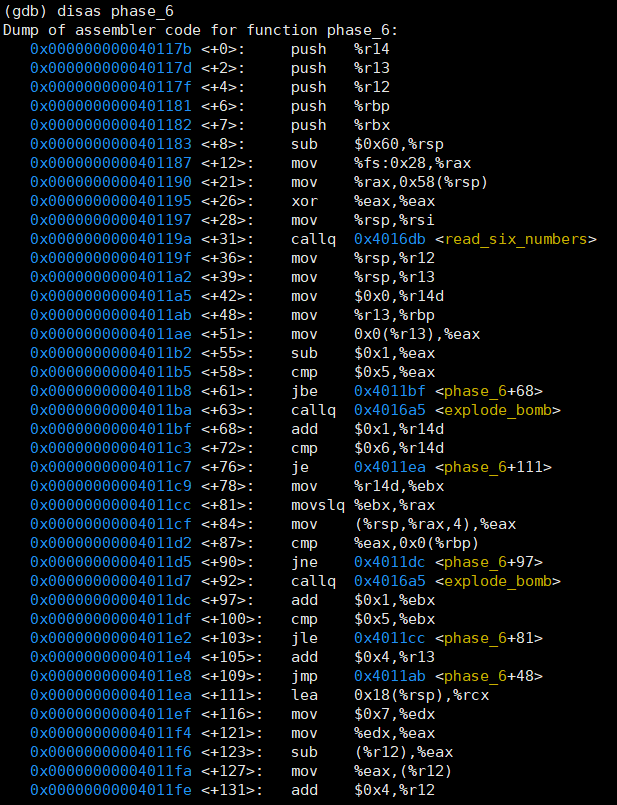






看phase\_6函数的汇编代码。函数开头调用了read\_six\_numbers函数，因此可知这里需要输入六个整数。

以下将phase\_6函数分开解读。



根据这部分存在的跳转指令，这里应该存在多重循环。循环结束在<+111>结束。

已知rsp是输入的六个整数的数组的首地址，设该数组名为input。

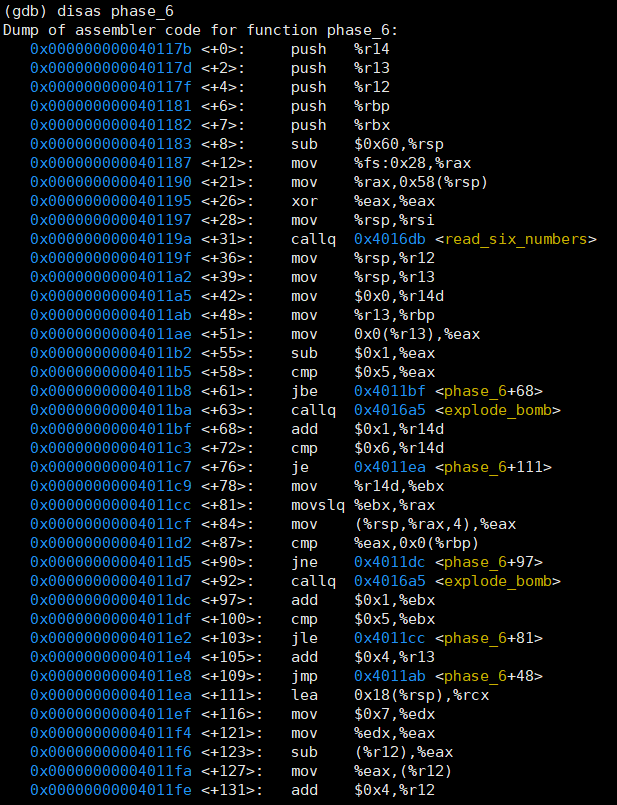
进入循环前，首先将r12和r13赋值为input，r14d赋值为0.进入循环。

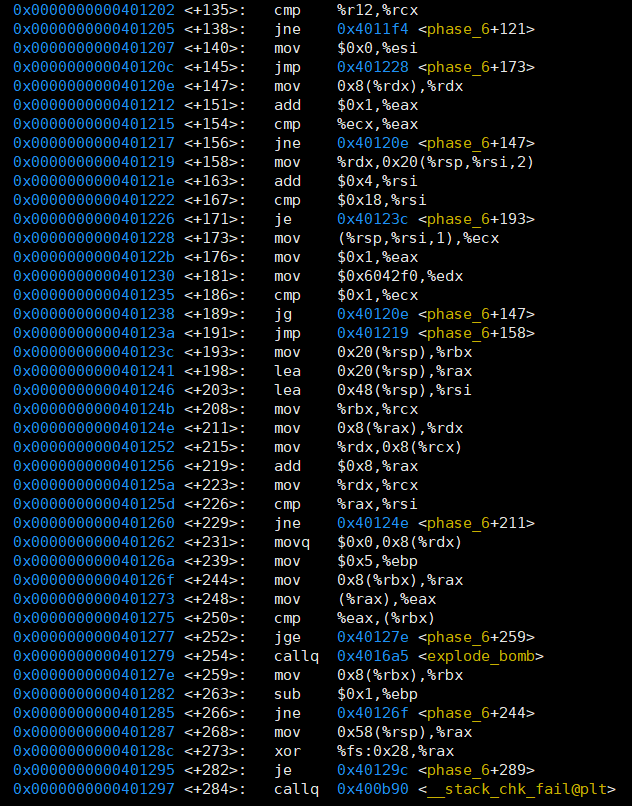
首先，将r13赋值给rbp，将r13存储的地址上的值赋值给eax，eax减一。如果eax大于5则爆炸。看循环的最后一句<+105>，每次循环结束r13+4，也就是移动到下一个数组元素。因此，以上操作的意思是数组的每个元素都必须小于或等于6，并且由于用的是jbe指令，所以必须大于0.

寄存器r14d加一，如果r14d等于6则结束循环，则r14d相当于循环变量i的作用，以下将其称为i。

寄存器ebx=i，进入内循环，这个循环的条件是ebx小于或等于5，每次循环后ebx加一，因此称ebx为循环变量j。

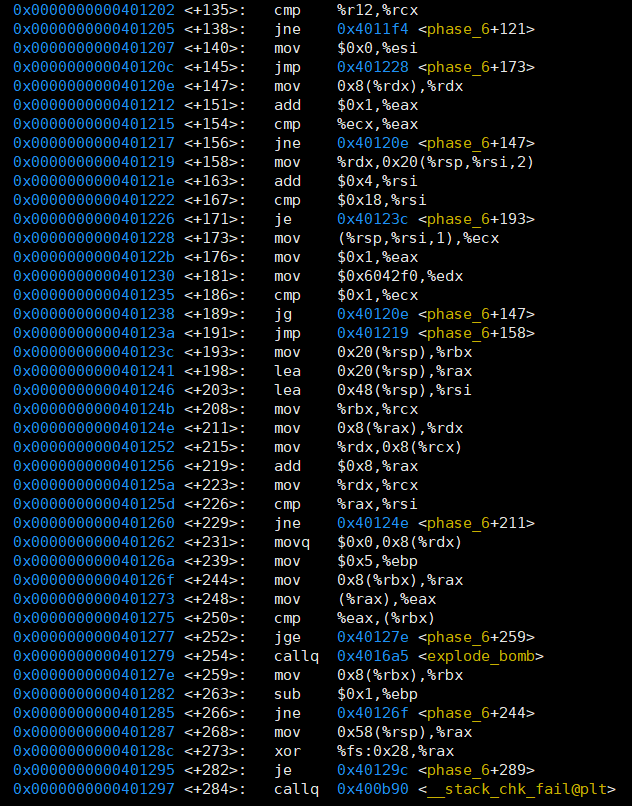
寄存器rax=j，eax=\*(input+4\*j)=input[j]，如果eax等于rbp处存储的值，也就是当前a[i]，炸弹爆炸。这个循环的意思是数组input的六个元素应该均不相等，否则爆炸。





第二部分，从<+121>至<+138>同样是一个循环。进入循化前将&input[7]赋值给rcx，也就是数组结束的地址，edx赋值为7.

进入循环，eax=edx=7，eax=eax-\*(r12), \*(r12)=eax, r12+=4, 如果r12不等于rcx则继续循环。这个循环对input数组进行操作，input[i]=7-input[i]。



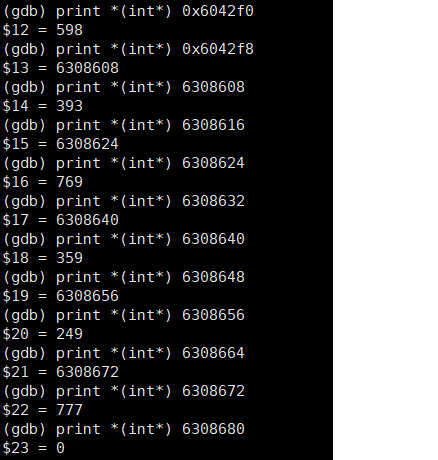
第三个部分，这里同样存在循环。在进入循环之前，函数先将esi赋值为0，并且，从<+163>和<+167>处可以看到，完成一次循环后，rsi+4，如果rsi等于0x18则退出循环。寄存器rsi的值相当于循环变量i，以下称它为i。

程序之后跳转至<+173>，将ecx赋值为input[i]，eax为1，edx赋值为地址0x6042f0. 如果ecx，也就是a[i]，就跳转至<+147>，否则跳转至<+158>。

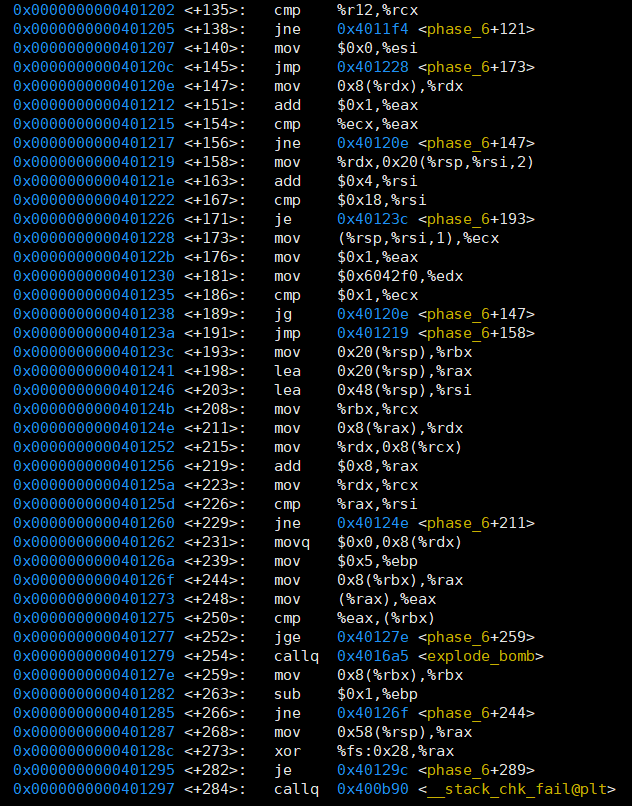
看<+147>处，rdx=rdx+8，也就是rdx的地址移动八个字节，eax加一。如果eax不等于a[i]，则返回<+147>，继续循环。结束循环也是跳到<+158>。

看<+158>，rsp+2\*rsi+0x20=rdx，也就是a[8+2\*i]=\*rdx.寄存器rsi加4，相当于i++，如果等于0x18，也就是i==6则跳转至<+193>，退出循环。否则，从<+173>继续下一次循环。

这里的意思是根据a[i],从0x6042f0开始，将相应的值赋值给a[10+2\*i]。



将0x6042f0处的整数输出，是598.但是，当加上8个字节，0x6042f8处的值是一个地址。打印出该地址，是393，该地址加上8个字节处的值依然是地址。因此，推断这里应该是一个拥有6个元素的链表。



看最后一个部分，这里有两个分开的循环，都较之前的简单。

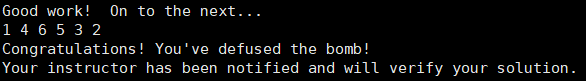
首先<+211>至<+229>处为第一个循环，进入循环前，rbx赋值为&a[10]，rax赋值为&a[10]，rsi=&a[20]，rcx=rbx=&a[10]。

看第一个循环，意思是执行了五次a[i]=a[i+2]，也就是从a[10]开始，a[18]结束，a[i]=a[i+2]。

进入第二个循环前，因为上一个循环将a[12]至a[20]，往前移动两位。因此，将a[20]置零。然后将ebp设为5.

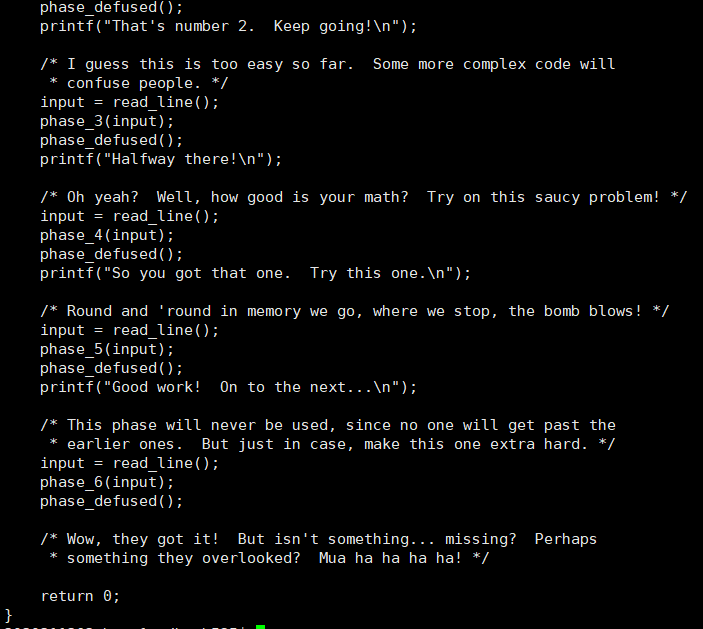
第二个循环里，ebp相当于循环变量i，每次循环后-1，直到它等于0. 这个循环将rbx+8赋值给rax，eax=\*rax，然后比较\*rbx和eax，如果\*rbx<eax则爆炸，否则rbx加八个字节，相当于移动到后第二个数组元素。因此，这里的意思是从a[10]开始，a[i]必须大于a[i+2].

因此，需要将输入六个正确的整数，根据它们各自和7的差，将0x6042f0开始的链表中的数据按倒序在a[10],a[12]…a[18],a[20]中存储。链表的结构为598->393->769->359->249->777，因此需要输入的六个数字为1 4 6 5 3 2。

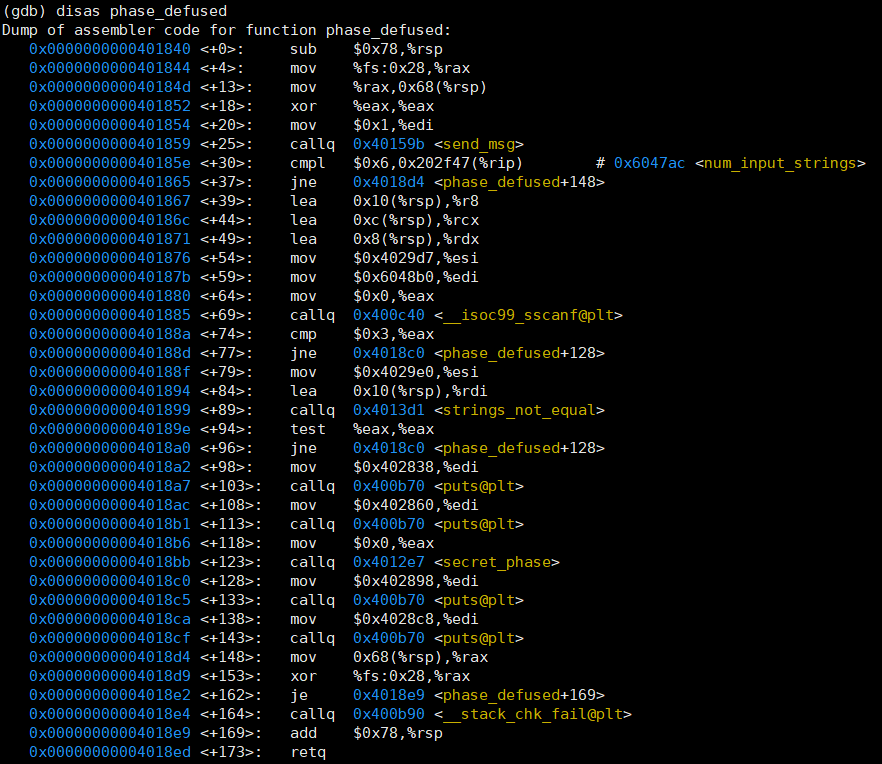


成功拆除第六个炸弹。

Secret phase:

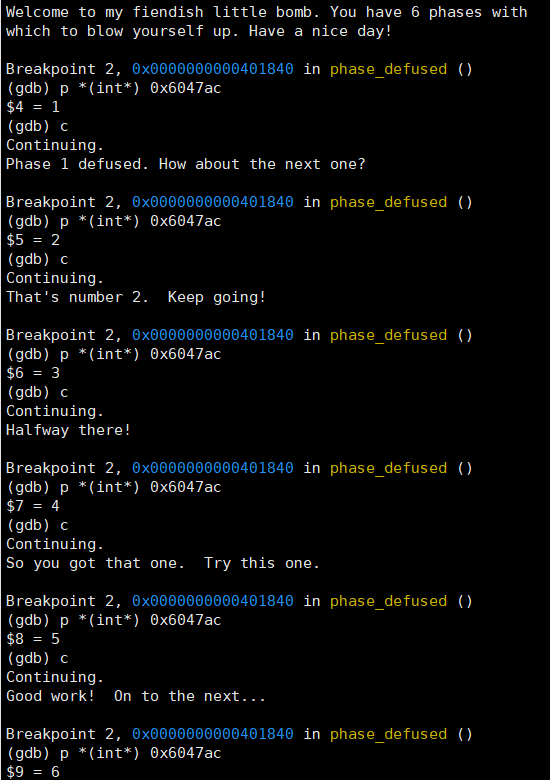


主函数的源代码中，拆除完第六个炸弹后还有一段注释，提示了第七个炸弹的存在。



将程序内出现的几个函数反汇编，寻找第七个炸弹。发现在函数phase\_defused中，调用了函数secret\_phase，也就是第七个炸弹存在于这个函数当中。

可以看到在<+30>处，判断0x6047ac的值是否等于6，不等于则直接跳过下面调用secret\_phase函数的部分.

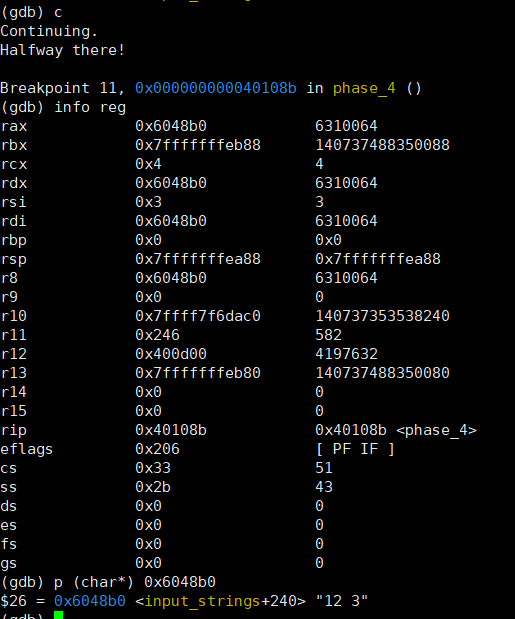


将这个地址的值打印出来，发现是指当前拆除的炸弹的编号。就是说只有拆除了六个炸弹才能进入secret\_phase拆除第七个炸弹。

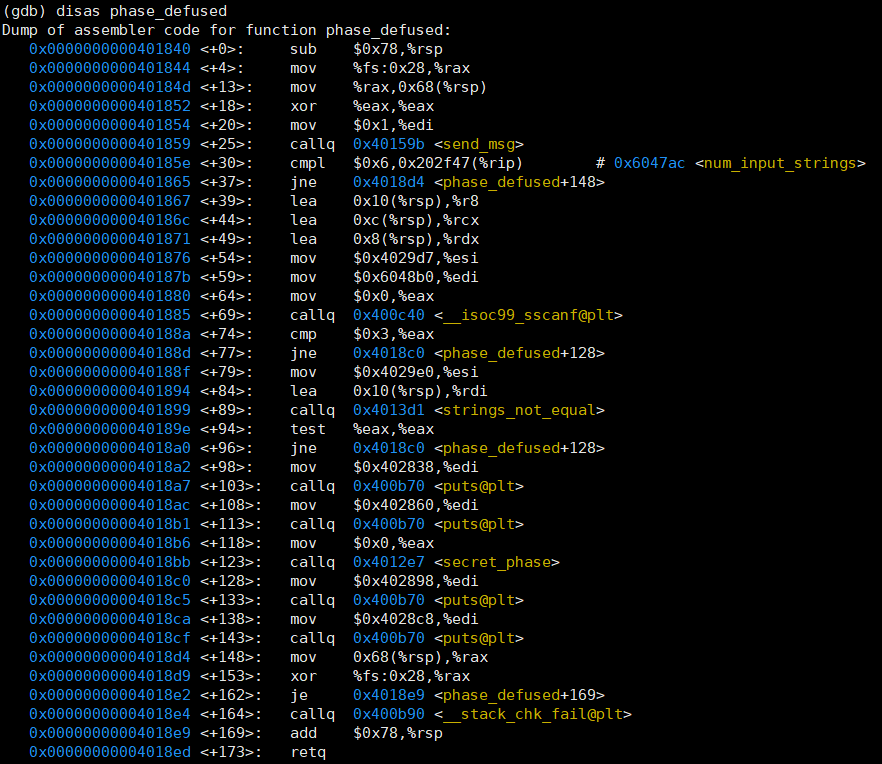
函数调用了sscanf函数，其中有参数esi=0x4029d7和edi=0x6048b0。



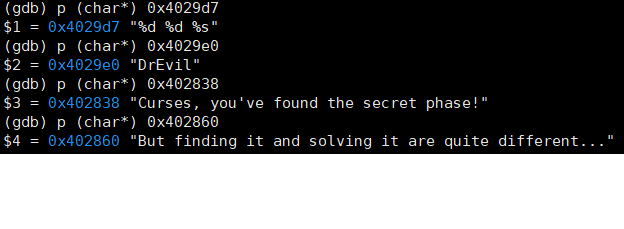
地址0x4029d7的字符串表示这里会读入两个整数和一个字符串。



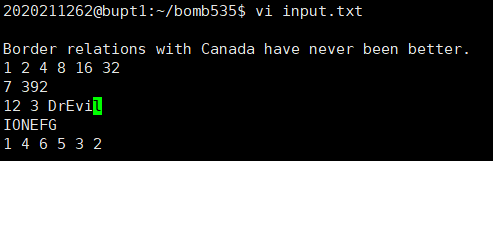
因为当前并没有任何可以再次输入的指令，因此推断这里的输入其实是拆除某次炸弹时的输入。借助gdb寻找哪次的输入被存储在了0x6048b0。发现是第四个炸弹的输入。也就是说，这里需要后面再输入一个正确的字符串，才能让defused\_phase函数调用secret\_phase函数。

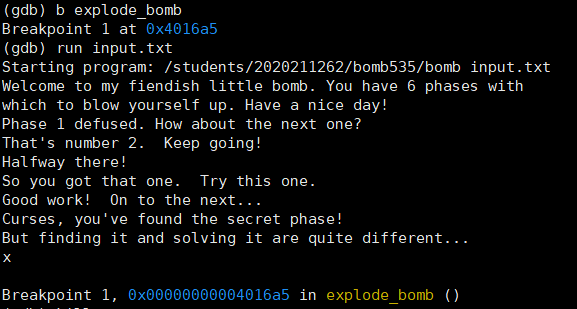


这里运用了strings\_not\_equal函数，如果输入的字符串与0x4029e0处的字符串不相等则不会进入第七个炸弹。

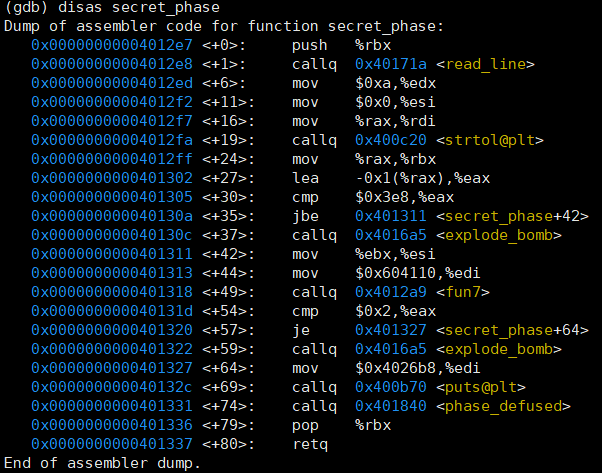


将0x4029e0处的字符串打印出来，是“DrEvil”。也就是说需要输入这个字符串才能进入第七个炸弹。





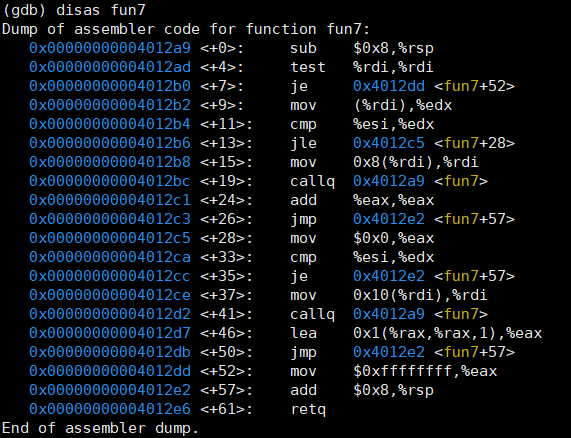
将字符串加入在第四个炸弹的输入的后面，发现在拆除了第六个炸弹之后确实进入了秘密炸弹。



看secret\_phase函数的汇编代码，首先调用read\_line函数，也就是输入。然后调用strtol函数，这个函数将读入的字符串转换为整数。

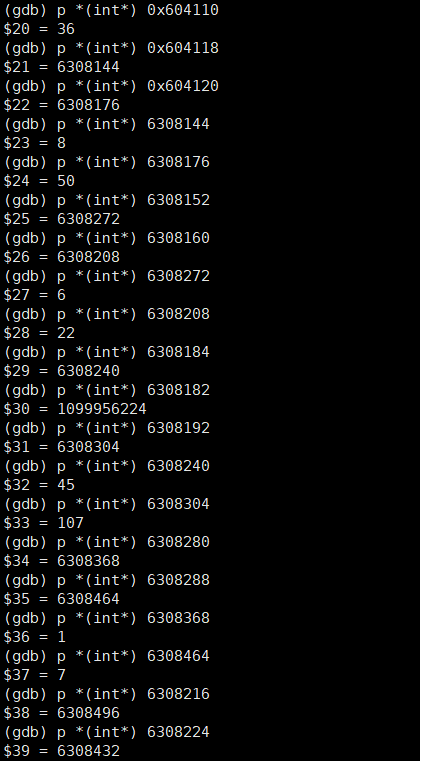
函数将strtol函数的返回值赋值给rbx，然后将（返回值-1）赋值给eax.如果eax大于0x3e8(999)，则爆炸。而且，这里用的是jbe指令，所以输入的数必须在1-999之间。

函数之后把ebx赋值给esi，也就是输入的数。然后把0x604110赋值给edi，调用fun7函数。如果fun7函数不等于2则爆炸，否则就成功拆除第七个炸弹。

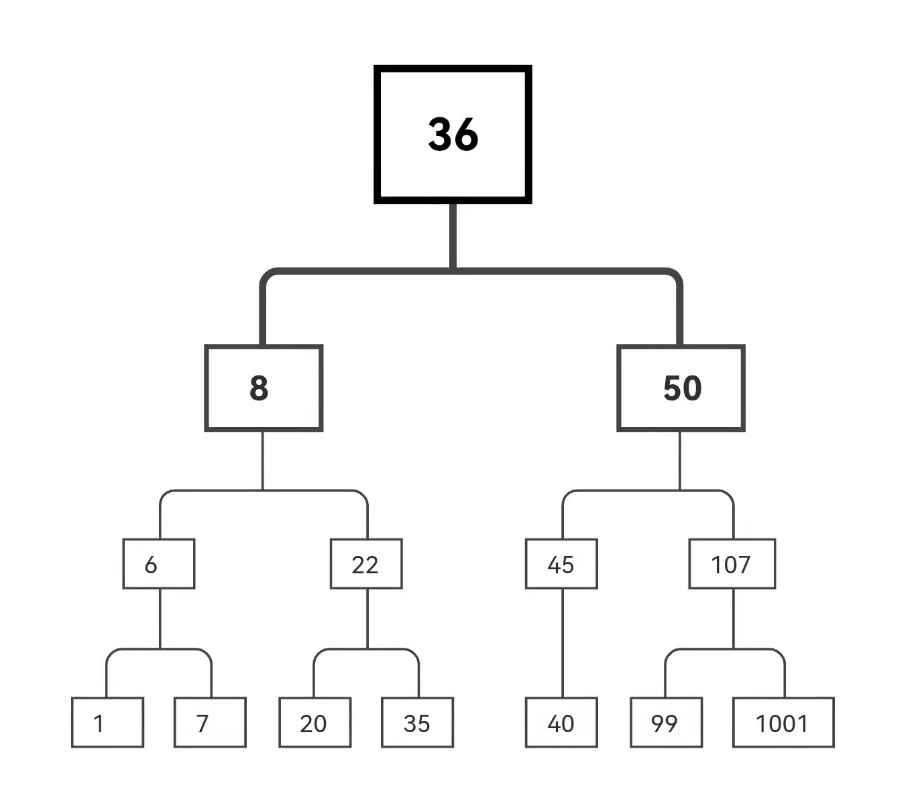


看fun7函数的汇编代码，逐一解读。

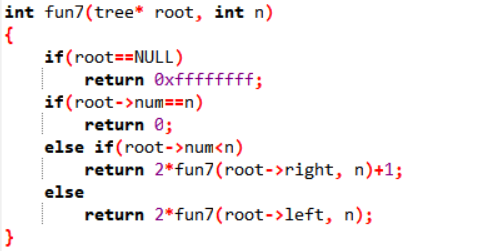
发现根据条件，rdi可能加8个字节，也可能加16个字节。



将地址所存的值打印出来，发现rdi+0x10和rdi+0x8是两个不同的地址。因此，rdi是一个二叉树的根结点地址。

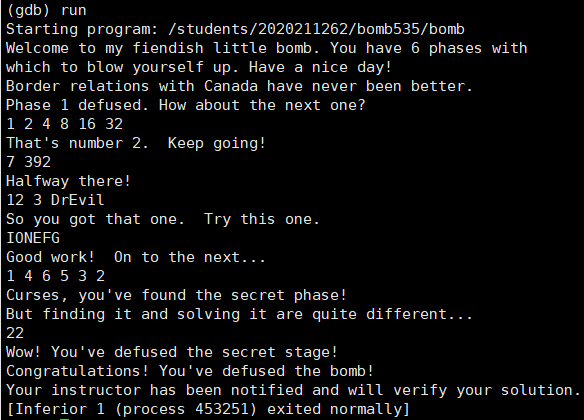


将+0x8的地址设为左子树，+0x10为右子树。将整个树的每个结点打印出来，该二叉树的结构如上图。



因此，fun7函数的功能相当于上图的C语言函数。

要使返回到secret\_phase函数的值为2，根据上图递归，输入需要等于根结点的左结点的右结点，也就是22.



输入22，成功拆除隐藏炸弹。

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）

实验刚开始，由于对汇编代码并不熟悉，有种无从下手的感觉。但只要试着拆除了一两个之后，就可以得知其中的一些规律，帮助拆除后续的炸弹，例如当程序传递0xXXXXXXXX给某个寄存器，这通常代表一个地址，可以根据输入的类型，判断这是个字符串还是整数。又例如程序的汇编代码存在许多与解题无关的代码，不用特地去解读。

后面几个炸弹的函数较长，并存在循环、递归等复杂结构。一开始，光是阅读汇编代码很难解读，所以后面试着将其写下来，一步步解读，尝试理解程序做了什么，可以的话，就用C语言实现，方便验证输入。

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、来自苏承烨的建议，如0xXXXXXXXX通常是个地址，可以把这里的值打印出来。并且，和苏承烨同学一起找出了进入第七个炸弹的方法。

此外，我还参考了以下资料：

1. [(12条消息) CSAPP二进制炸弹实验 bomb lab详细解析\_Eternitykc的博客-CSDN博客](https://blog.csdn.net/Eternitykc/article/details/110427039?ops_request_misc=%257B%2522request%255Fid%2522%253A%2522163670936716780271571985%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=163670936716780271571985&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~top_positive~default-1-110427039.first_rank_v2_pc_rank_v29&utm_term=bomblab&spm=1018.2226.3001.4187) 参考该博主

的方法，将某些复杂的汇编代码逐一解读，并尝试以C语言写出，方便理解和验证输入。

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）