

**实验报告**



**题目： 拆解二进制炸弹ARM版**

**班 级： 2020211302**

**学 号： 2020211262**

**姓 名： 鄭毓恒**

**学 院： 计算机学院**

**2021年 11 月17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ARM AArch64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. XShell（10.99.0.230）
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kunpeng1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。

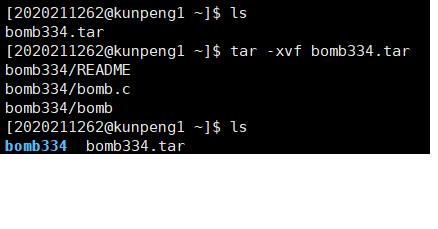
为完成二进制炸弹拆除任务，需要使用gdb调试器和objdump来反汇编bomb文件，可以单步跟踪调试每一阶段的机器代码，也可以阅读反汇编代码，从中理解每一汇编语言代码的行为或作用，进而设法推断拆除炸弹所需的目标字符串。实验2的具体内容见实验2说明。

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

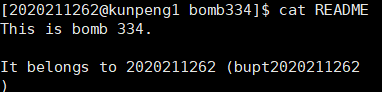
各阶段需要有操作步骤、运行截图、分析过程的内容

准备工作：

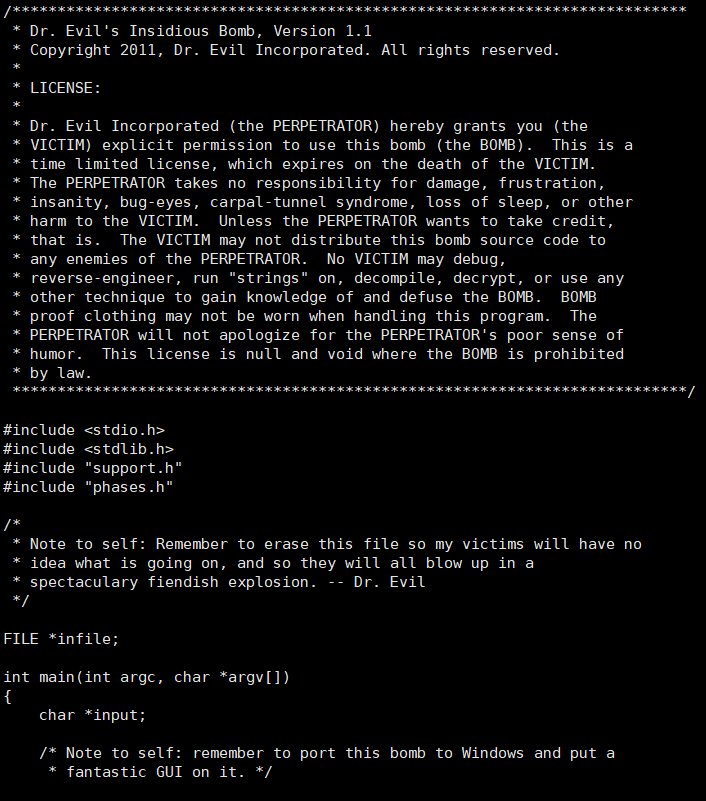


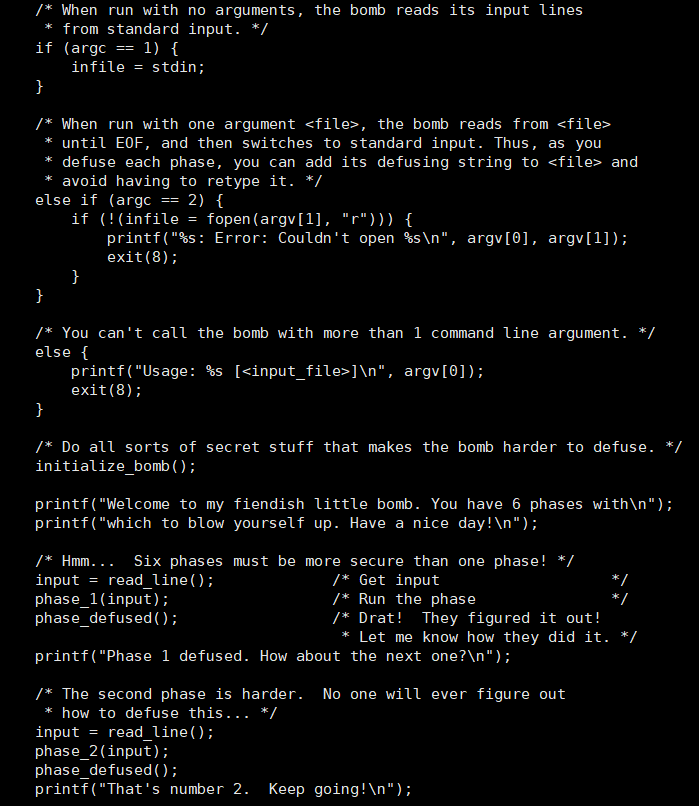
首先登录kunpeng1服务器，在home目录找到Dr. Evil设置的二进制炸弹bomb334.tar，334是我专属的炸弹编号。

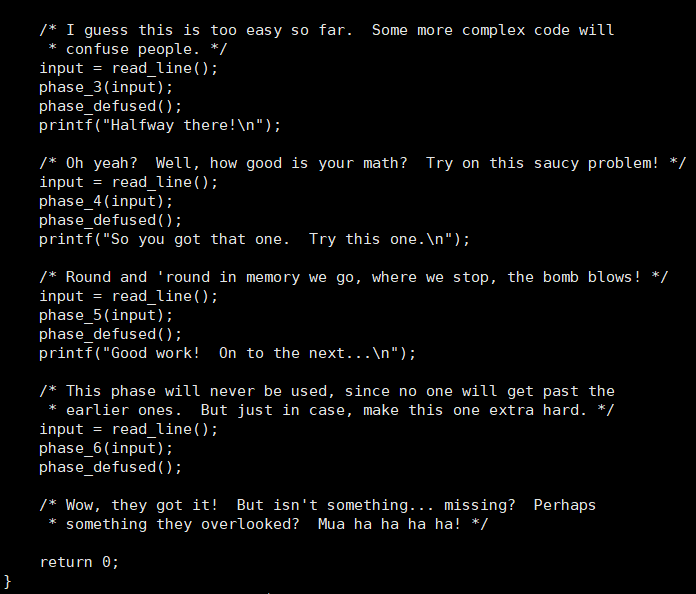
输入指令tar -xvf bomb334.tar，生成一个名为bomb334的目录。目录当中有bomb、bomb334、README三个文件。README文件标识了炸弹和它的拥有者，也就是我。bomb.c和bomb分别是炸弹的源代码和可执行文件。



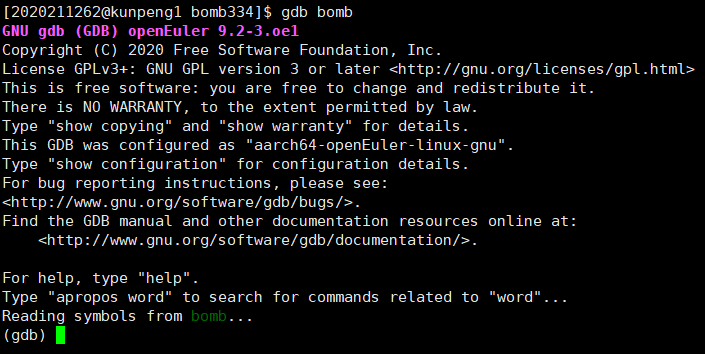
README文件标识了这是炸弹334，属于学号2020211262的我。



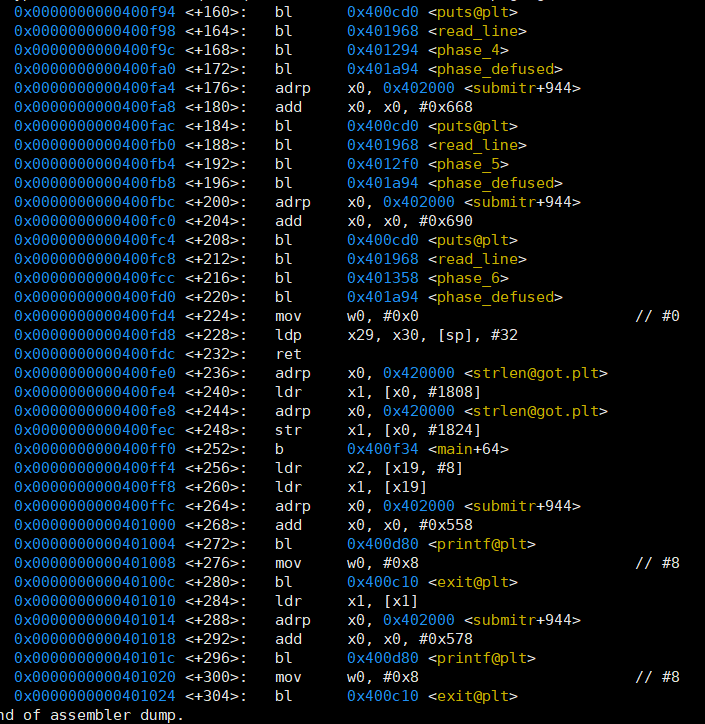
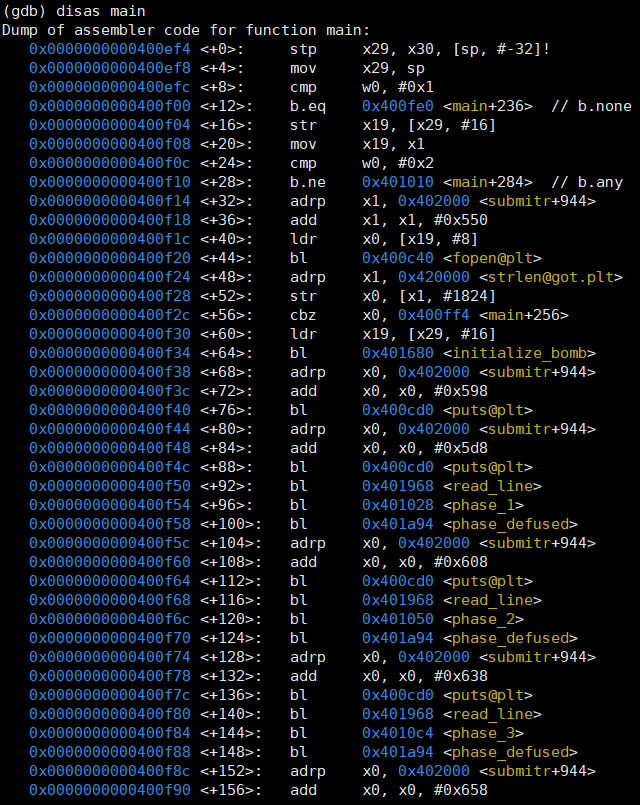




阅读bomb.c文件，可以发现当程序开始时或一个炸弹被拆除时会打印几条字符串。并且，在注释中，有一些Dr. Evil给的关于炸弹的提示。程序中目前观察中存在phase1至phase6六个炸弹，每次先输入一行字符串再调用炸弹函数，然后再调用phase\_defused函数。

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。

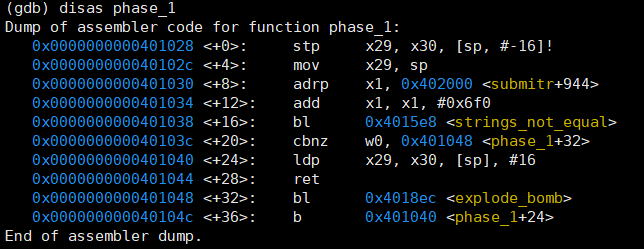
进入bomb可执行文件的gdb调试。



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：

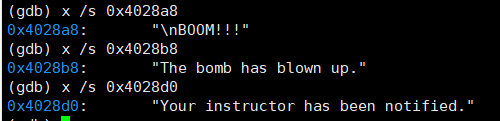


用gdb的disas指令反汇编后，阅读第一个炸弹phase\_1函数的汇编代码。

可见该函数先把地址0x4026f0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0x4028a8,0x4028b8,0x4028d0的三个字符串。



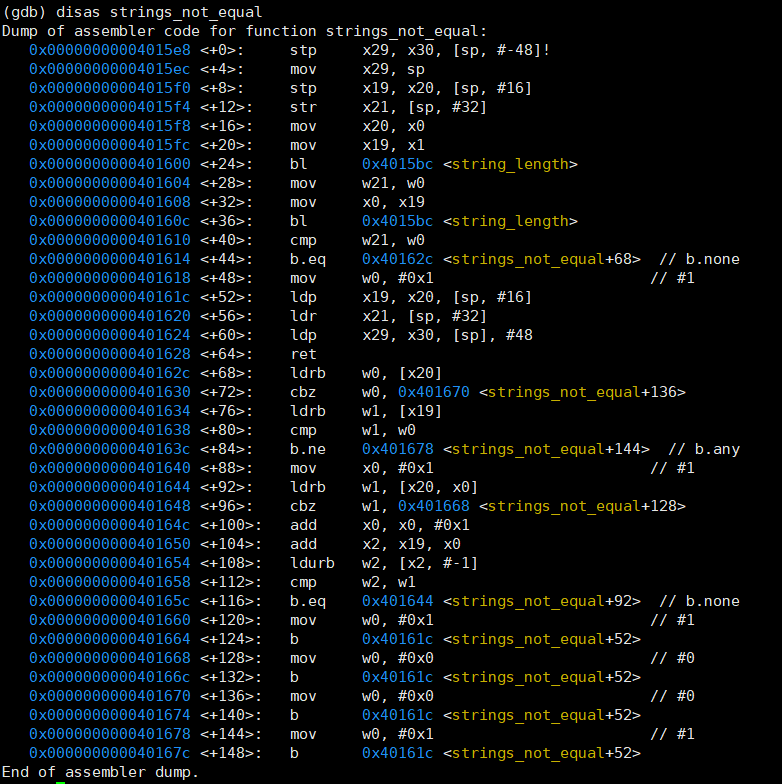
打印出这三个字符串，发现是炸弹爆炸的提示。

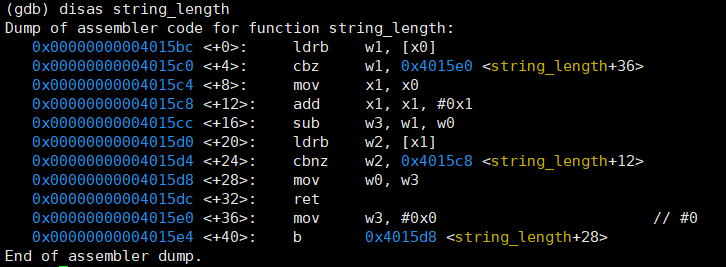
函数在打印完这三句后，调用exit函数使程序退出。因此，一旦调用explode\_bomb，炸弹就会爆炸，程序就会终止，如它的函数名所述。

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。

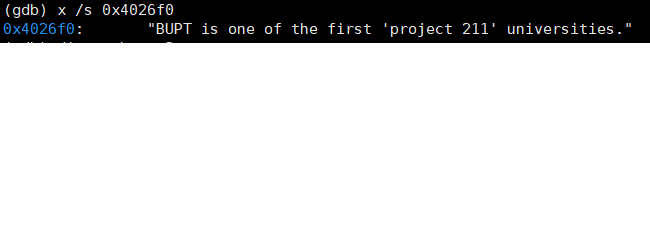




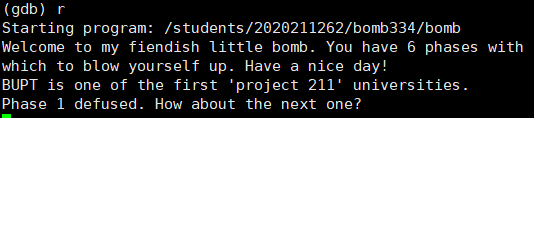
接下来看strings\_not\_equal函数。发现它会调用string\_length函数，因此还要看看string\_len函数。

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0x4026f0的字符串不相等，炸弹爆炸。

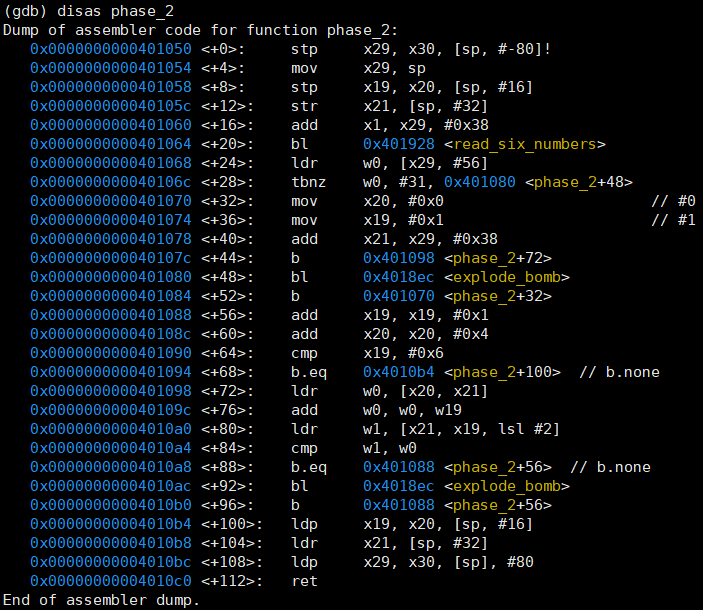


将0x4026f0地址的字符串打印出来，这应该就是拆除第一个炸弹所需要输入的字符串。

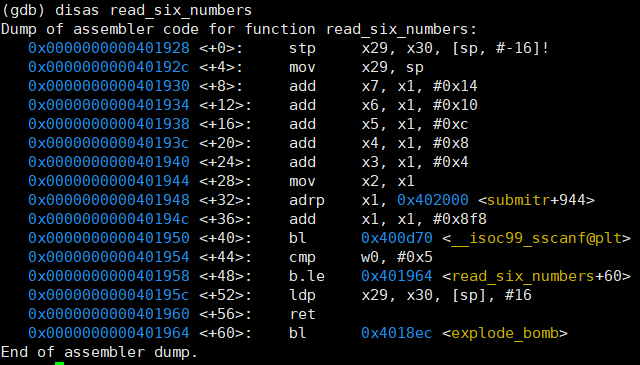


当输入同样的字符串时，程序提示“Phase 1 defused”。炸弹一成功拆除。开始阶段二，拆除第二个炸弹phase\_2。

阶段2：



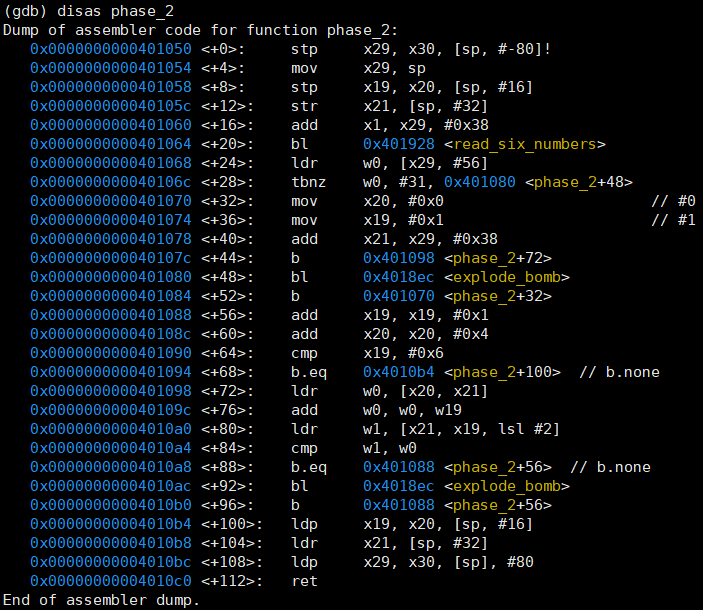
看phase\_2函数的汇编代码。该函数首先调用了read\_six\_numbers函数。



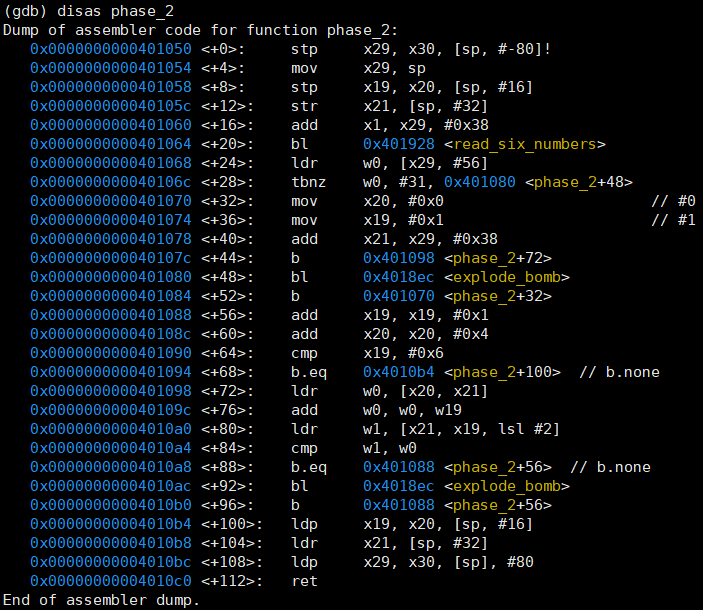


这个函数先将地址0x4028f8的字符串传给寄存器x1，再调用sscanf函数。地址0x4028f8的字符串打印出来为六个以空格分隔的%d，以函数名和调用了sscanf推断，这个函数将我输入的字符串以六个整数的形式读入，也就是说我在第二个炸弹这里需要输入六个整数。并且，函数之后将寄存器w0，也就是sscanf函数的返回值，与5比较。若返回值不大于5，也就是读入的整数的数量不大于5，调用explode\_bomb函数，炸弹爆炸。

已知拆除炸弹的密码是六个整数，回到phase\_2函数。

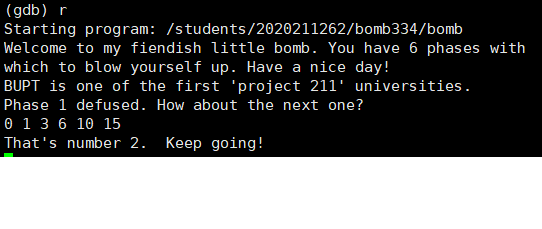


寄存器x29加#56字节的位置存储的是输入的六个整数的首地址。函数将该地址存储的整数，也就是输入的第一个整数与0比较，如果不等于则调用explode\_bomb函数，爆炸。因此，第一个输入的整数必须为0.



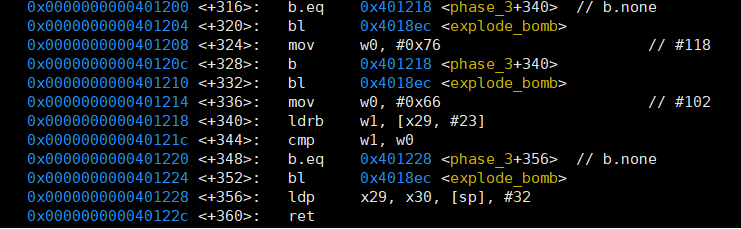
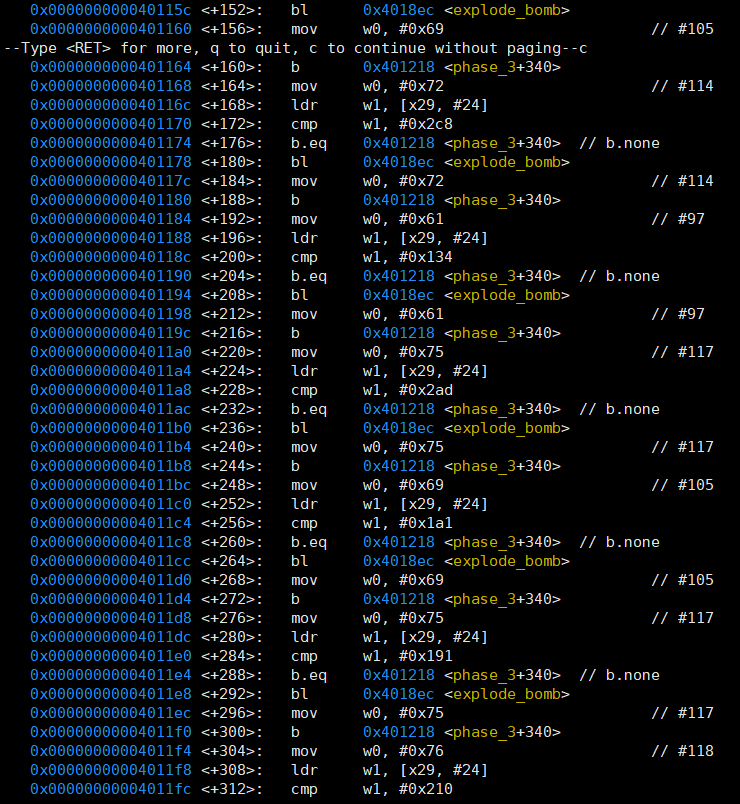
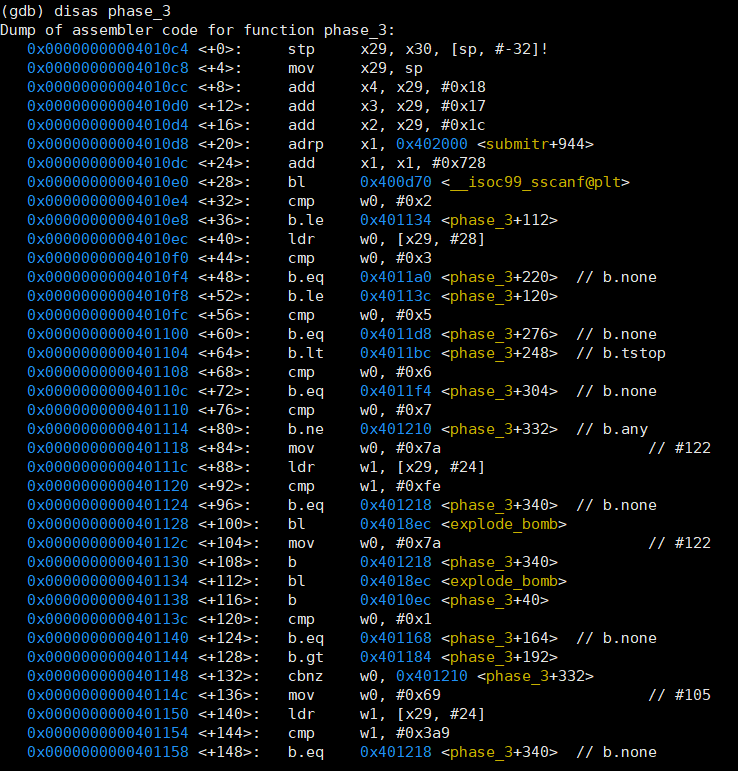
接着往下读，发现到<+88>处出现一句条件跳转指令，跳转至<+56>，此处存在一个循环。进入循环前，首先将x20赋值为0，x19赋值为1，寄存器x29加上#0x38处存储的值，也就是输入的六个整数所在的数组的首地址，赋值给x21。从<+72>开始循环，循环首先将寄存器x21的地址加上x20个字节位置所存储的地址上的值，也就是当前移动到的整数，赋值给w0。然后，w0加上w19，也就是加上自己的下标，第一个数字的下标是1。之后，将w0与存储在x21加上x19 乘以 lsl#2（四个字节）的地址比较，也就是和下一个整数比较，如果不等于则爆炸。如果等于，跳转至<+56>，x19加1，x20加上0x4（四个字节），然后x19在与6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0，并且下一个数是前一个的数加上自己下标的值。

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0, 1, 3, 6, 10, 15六个整数拆除炸弹。

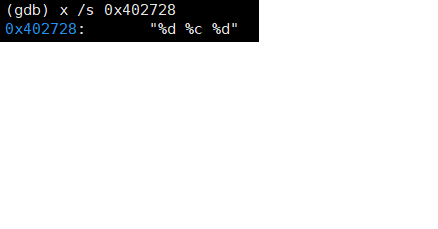


第二个炸弹拆除成功。开始阶段三，拆除第三个炸弹phase\_3。

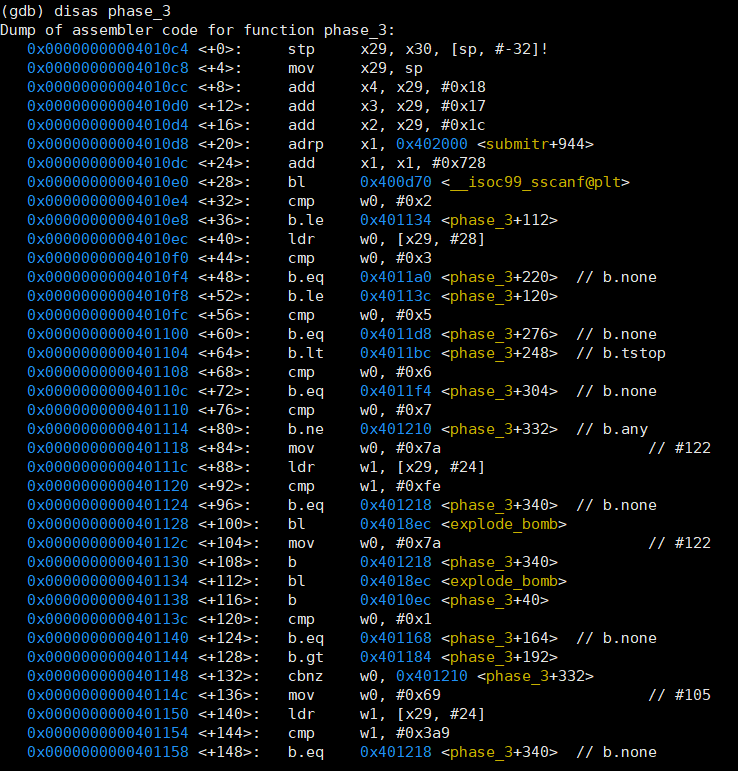
阶段3：



观察phase\_3函数的汇编代码，发现函数将地址0x402728传递给寄存器x1，然后调用sscanf.



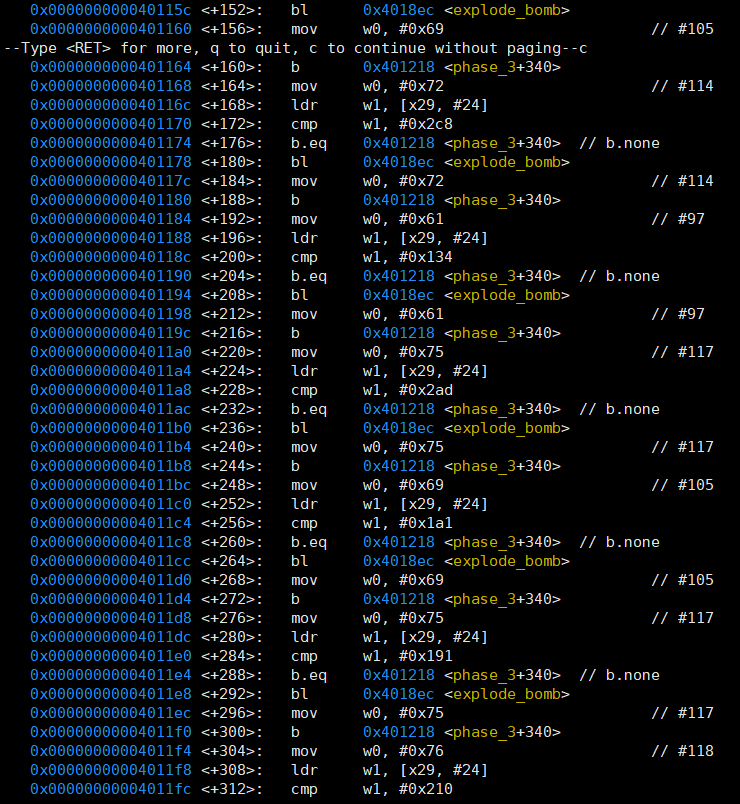
将地址为0x402728的字符串打印出来，结合sscanf函数，这里应该是需要输入两个整数，中间有一个字符。

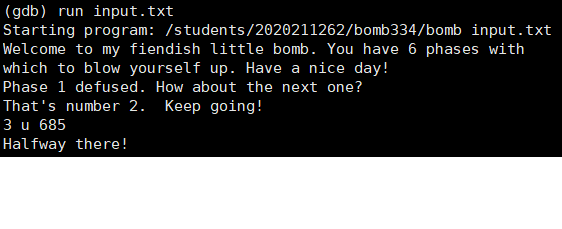


这里将sscanf函数的返回值与2比较，不大于2就爆炸。所以，确实是输入两个整数和一个字符。

[x29, #28]，也就是输入的首地址所存储的值，输入的第一个整数。程序代码会根据第一个整数的值进行跳转并分别进行不同的操作，因此这里是一个类似switch语句的结构。而且，第一个整数只能等于0-7其中之一，否则将爆炸。

由于第一个数字有0-7八种可能，所以根据输入值跳转后，有八种可能的跳转地址，执行不同的指令，再跳转到一个统一的地址<+340>。这几个地址的所执行的指令均相似，都是将寄存器w0赋值为一个特定的值，然后将寄存器w1赋值为第二个输入的整数，在将其与一个特定的数进行比较，如果不等则爆炸，等于则跳转至<+340>。<+340>处，程序将寄存器w1赋值为输入的字符的ASCII码，再与w0的值进行比较，如果等于则通关，不等则爆炸。

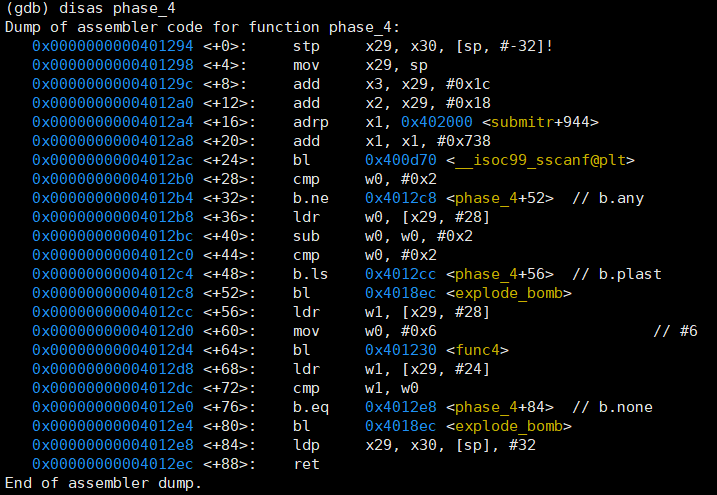




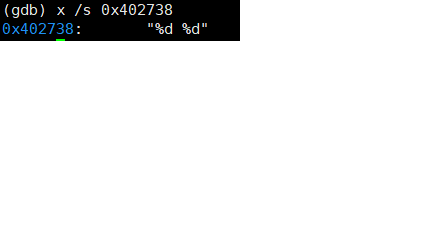
由于是switch语句，推测有8种解，输入其中一种进行测试。当第一个数字为3，字符的ASCII为117，也就是’u’，第二个数字是685.测试成功，第三个炸弹被拆除。

开始阶段四，拆除第四个炸弹phase\_4.

阶段4：



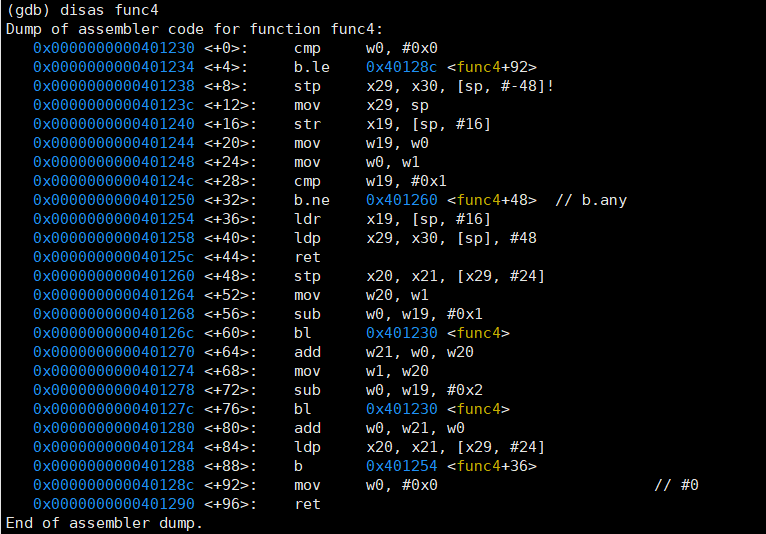
观察phase\_4函数的汇编代码，发现这里也是将一个地址0x402738传递给x1，再调用sscanf函数。

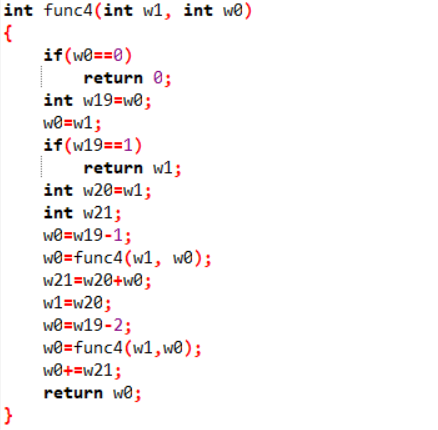


把该地址的字符串打印出来，可见phase\_4需要输入两个整数。并且，调用sscanf函数后，程序又将返回值与2比较，如果不等于则爆炸。因此，必须输入刚好两个整数。

看下几句汇编代码，将输入的第二个数赋值给w0，然后w0减2，与2比较，如果大于2则爆炸。并且，这里应是unsigned类型。输入的第二个整数的值在2-4之间。

程序后续调用了func4函数，并且该函数的参数在w1以及w0存储，w0=6,w1等于输入的第二个数。





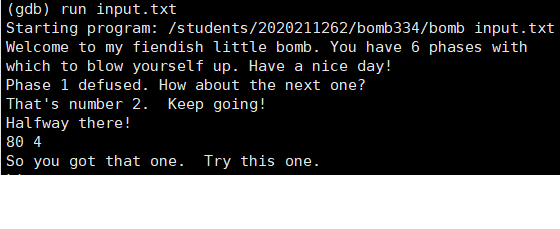
看func4函数的汇编代码。一步步看，下面可见函数根据条件，改变参数的值，调用自己。因此，这里存在函数递归的运用。将func4函数的汇编代码一一解读，最终可知func4函数可以写成以上的C语言代码函数，实现类似功能。

在调用func4函数后，程序将w1赋值为输入的第一个数，将w0与w1比较，如果不等于则爆炸。因此func4函数的返回值以及输入的第一个数必须相等.

因为之前已经推出func4函数的类似C语言代码，经过计算，这里根据输入的第二数字（2-4）应该存在三种解，用C语言程序进行测试。

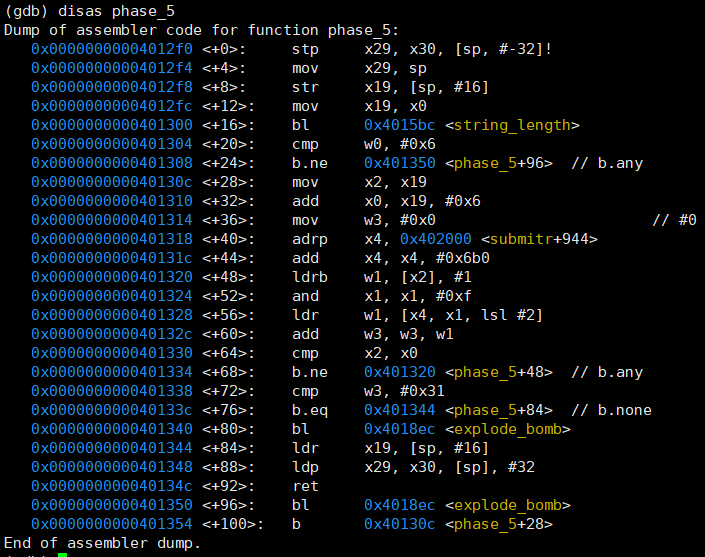
  

程序打印了func4函数的返回值。当输入为2, 3或4时，func4函数分别返回40，60 和80.



将80 4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：

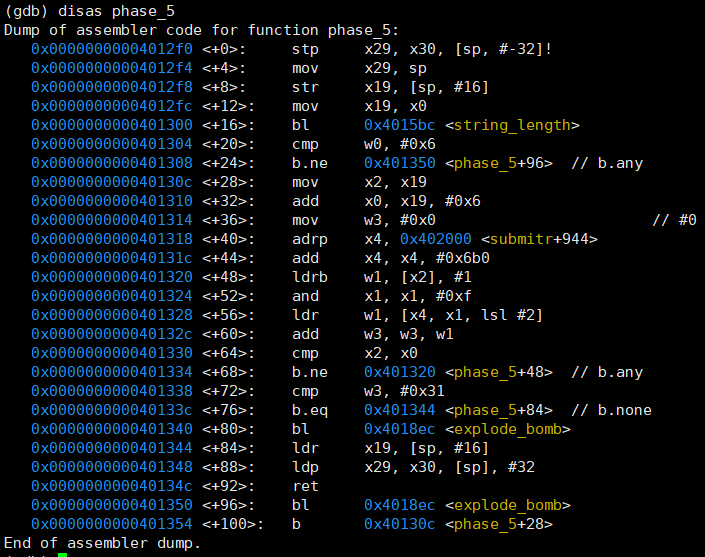


看phase\_5的汇编代码，可见函数调用了string\_length函数，因此这里需要输入的应该是一段字符串。

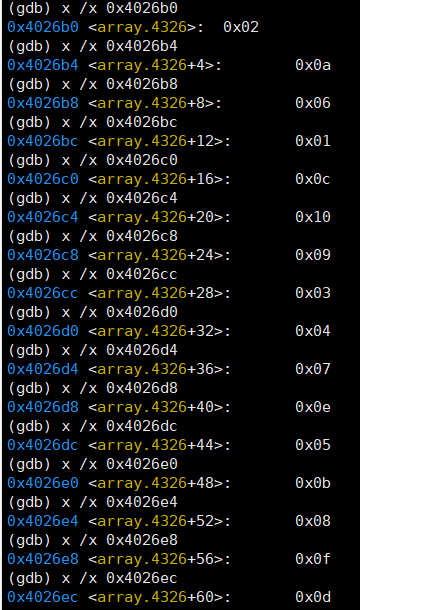
函数string\_length返回了输入的字符串的长度，长度必须等于6，否则爆炸，因此需要输入6个字符。

执行到<+68>处，有一个条件跳转语句，可能跳转到<+48>。因此这里存在一个循环。

循环开始前，x2被赋值为x19所存的地址，x0为x19加上6个字节的地址，即分别是输入的字符串的开头和结尾。寄存器x2在执行完<+48>处的语句会加1，在<+64>处，如果x2不等于x0，即未遍历到字符串结尾，跳转至<+48>处进行下一次循环。循环的结束条件是寄存器w3等于0x31（49）.



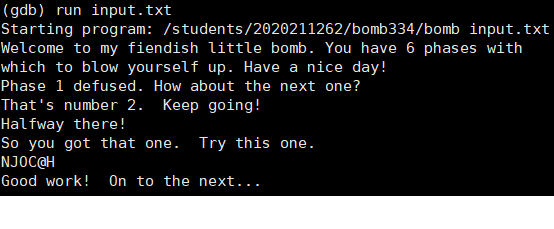
循环开始前，程序将寄存器x4赋值未0x4026b0。在循环内，寄存器x1赋值为当前遍历到字符，与0xf进行于运算，即取最低四位。然后，寄存器w1等于x4加上4等以x1处地址所存储的值，将w3加上w1.



0x402b0处存储的是一个数字，并且它的往后每四个字节所存储的也是数字。所以，x4实际上是一个数组的首地址。

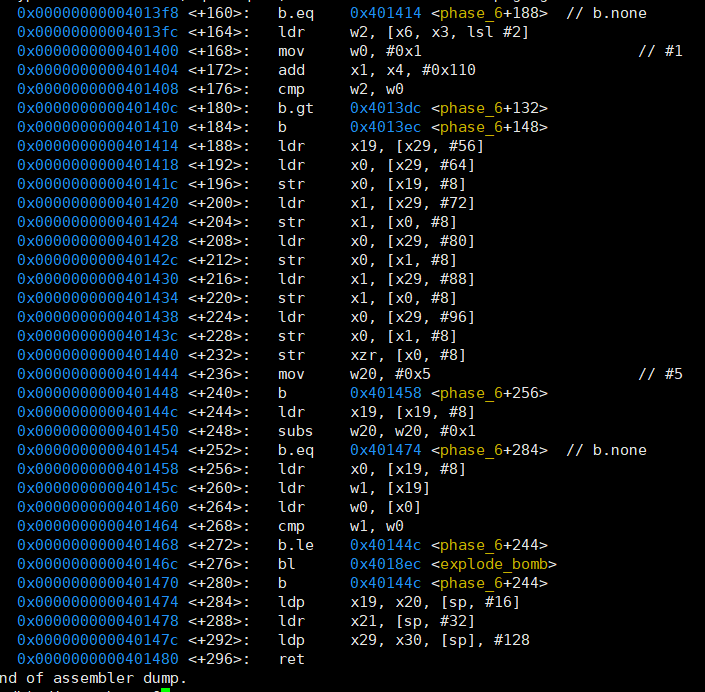
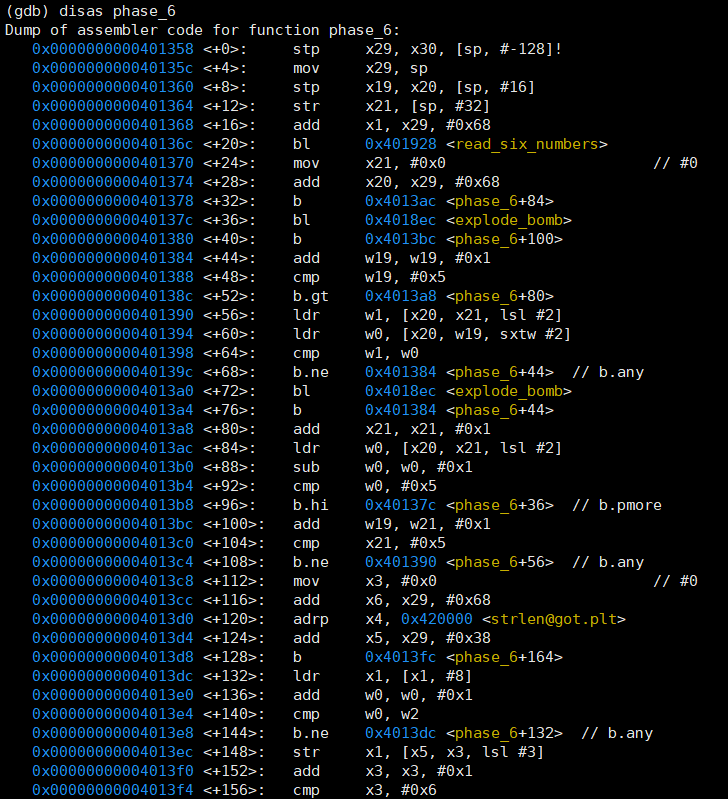
要通过这个炸弹，需要输入6个正确ASCII码的字符，根据它们的最低四位从数组从取出6个数，6个数字的和等于49.

49=15+14+13+1+2+4，则需要输入的6个字符的ASCII码的最低四位分别为14、10、15、3、0、8. 设字符的高四位为0100，所以输入的字符串的ASCII码为78、74、79、67、64、72，六个字符是NJOC@H。



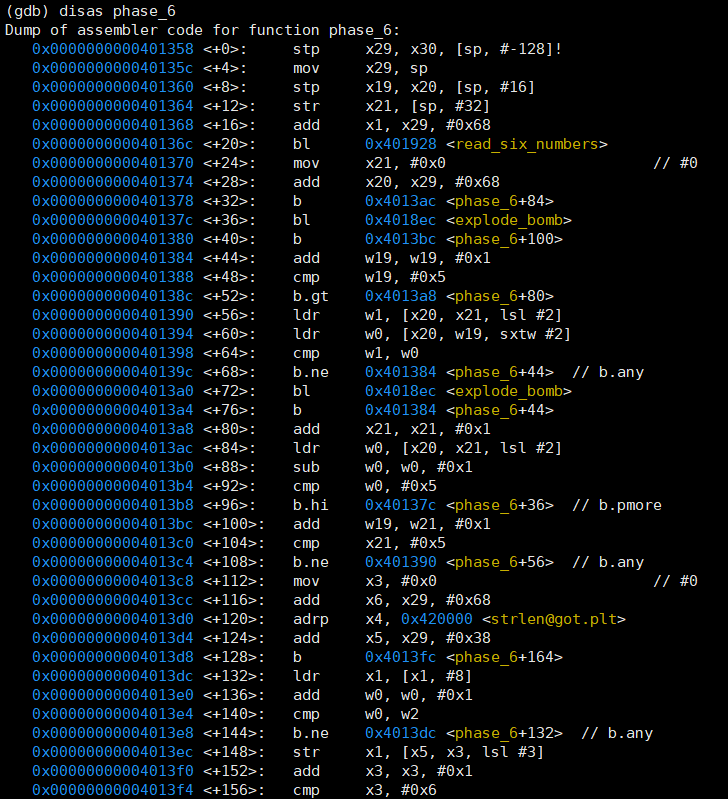
输入NJOC@H，第五个炸弹成功拆除。开始阶段六，拆除第六个炸弹phase\_6。

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。



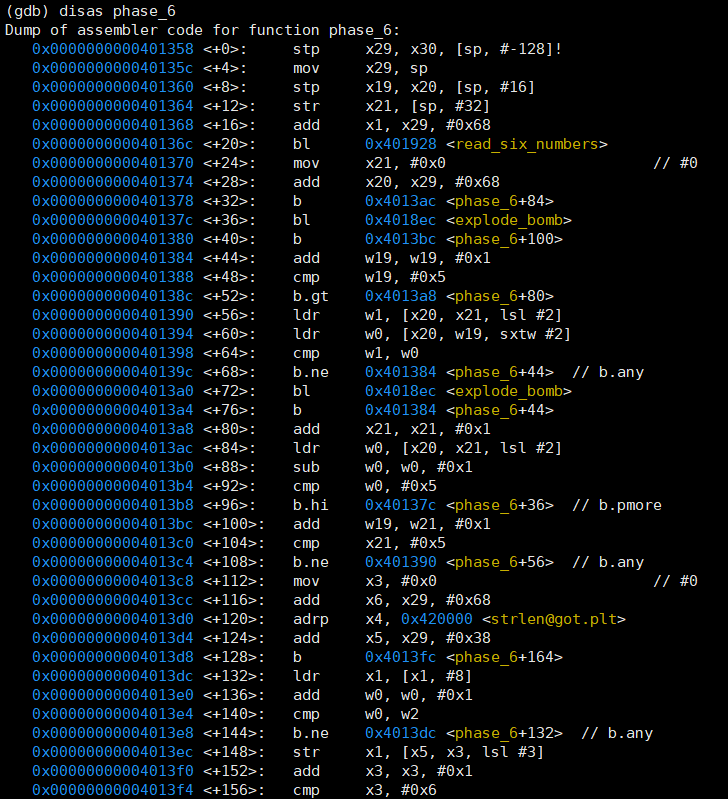
根据这部分存在的跳转指令，这里应该存在多重循环。循环在<+112>结束。

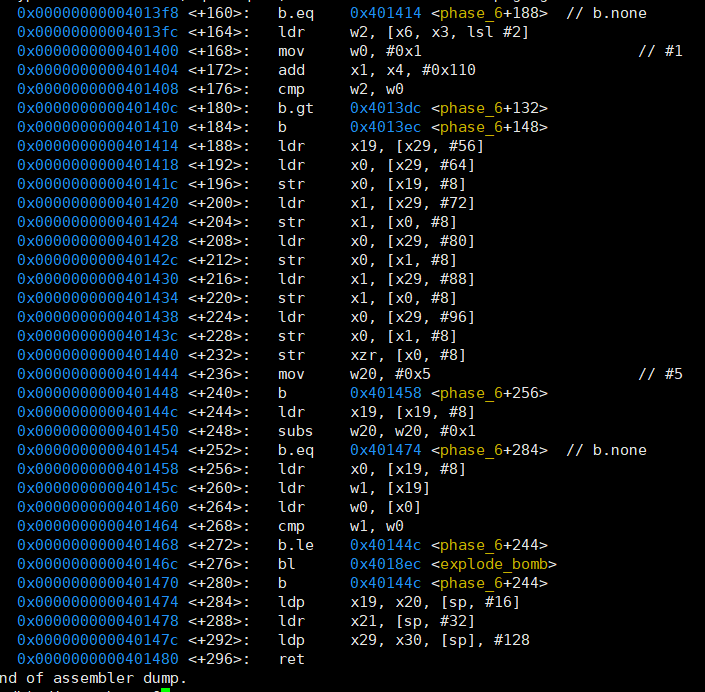
已知x29+#0x68是输入的六个整数的数组的首地址，设该数组名为input。

进入循环前，首先将x21赋值为0和r13赋值为input.进入<+84>开始循环。

首先，将x20加上4\*x21处的值赋值给w0，也就是当前遍历到的数组元素。将w0减1，如果w0大于5则爆炸。因此，以上操作的意思是数组的每个元素都必须在1-6之间。

寄存器w19=w21+1，跳转至<+56>，根据w21和w19的值分别从数组中提取元素赋值给w1和w0，也就是相邻的两个元素。当他们不等于，跳转至<+44>，w19加1后如果不大于5，跳转至<+80>，进行下一次循环。当他们等于则爆炸。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。



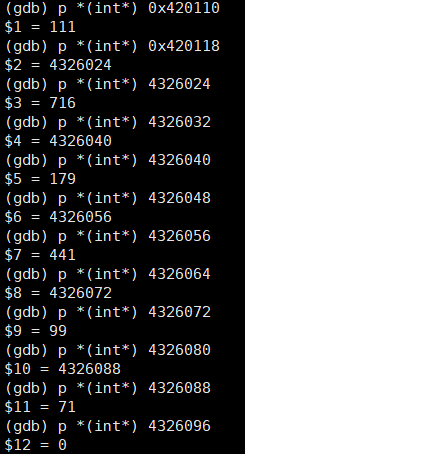


第二部分，从<+112>至<+184>同样是一个循环。进入循化前将&input赋值给x6，也就是数组结束的地址，x3赋值为0，x5赋值为x29+0x38处的地址，x4赋值为0x420000.

进入循环，从<+164>开始，w2赋值为x6+4\*x3处地址的值，w0赋值为1，x1赋值为x4+0x110=0x420110.

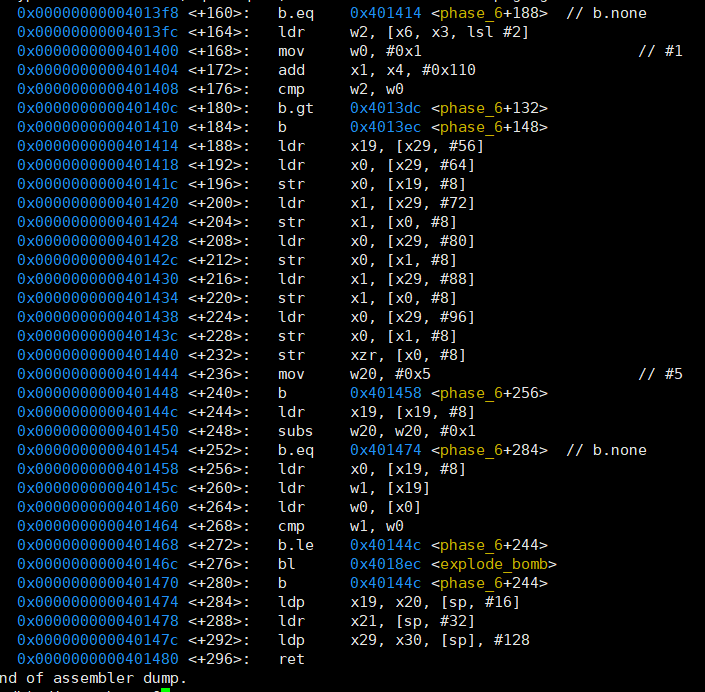
如果w2大于w0，也就是大于1，跳转至<+132>，否则跳转至<+148>。<+132>循环将x1所存的地址加上8个字节所存的值赋值给x1，w0加一，继续循环直至w0等于w2.

<+148>处，将x5+x3\*8处的值更改为x1，x3加1，x3不等于6则继续下一次循环。



将地址0x420110处打印出来，并将加上8个字节处的也打印出来，发现这是一个有六个元素的链表。

也就是说，第二个部分根据输入的六个数字，从链表中提取数字，存到x5所指向的数组A当中。

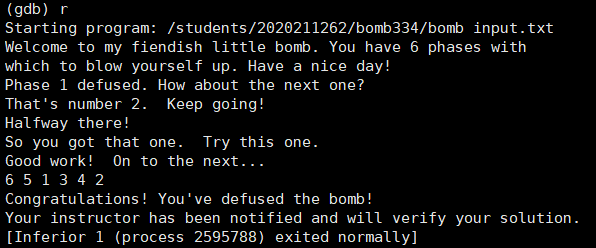


第三个部分，<+188>至<+232>处的功能对数组A并没有影响。

程序之后将w20赋值为5，跳转至<+256>，将x0赋值为\*(w19+8)，w1赋值为\*(w19)，如果w1小于或等于w0，跳转至<+244>，将x19加8，相当于移动到下一个元素，w20减一，如果w20不等于0则继续循环。

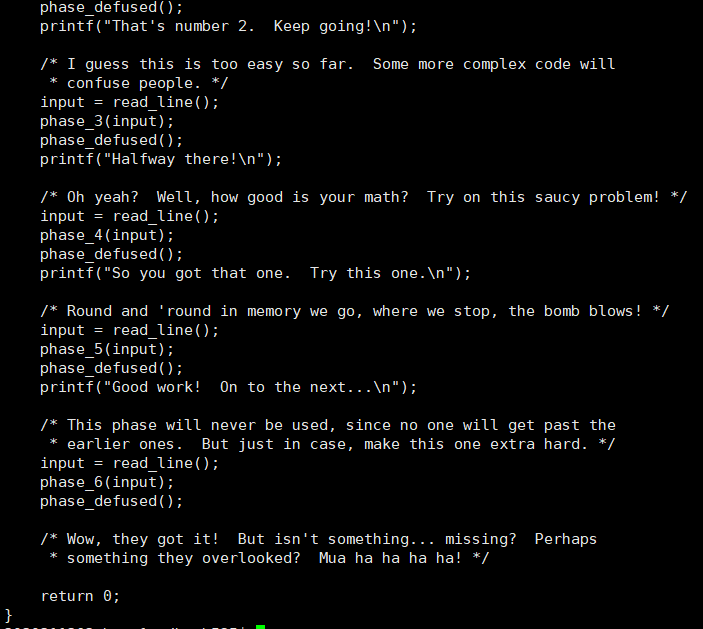
这代表了这里需要数组A为正序，而数组A的元素的次序是从链表中提取，由输入的六个数字决定。

链表中按顺序有111、716、179、441、99、71六个元素。要使它们正序排列，需要输入6 5 1 3 4 2。

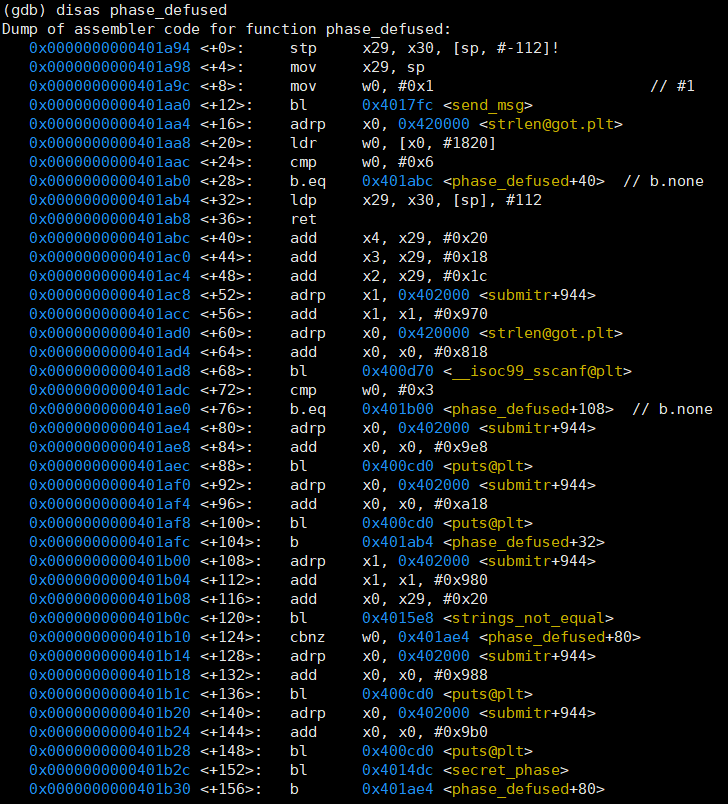


成功拆除第六个炸弹。

Secret phase:



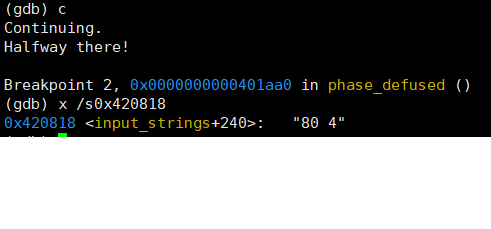
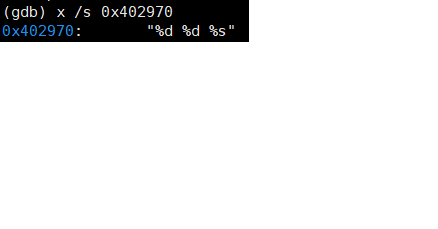
主函数的源代码中，拆除完第六个炸弹后还有一段注释，提示了第七个炸弹的存在。



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。

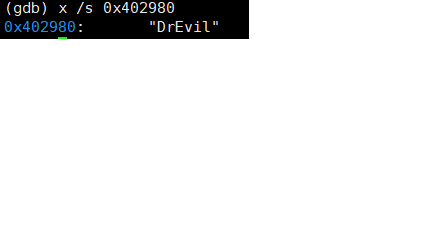
可以看到在<+28>处，判断w0=0x421820的值是否等于6，不等于则直接跳过下面调用secret\_phase函数的部分，也就是说只有拆除了六个炸弹才能进入secret\_phase。

函数调用了sscanf函数，其中有参数w1=0x402970和w0=0x420818。

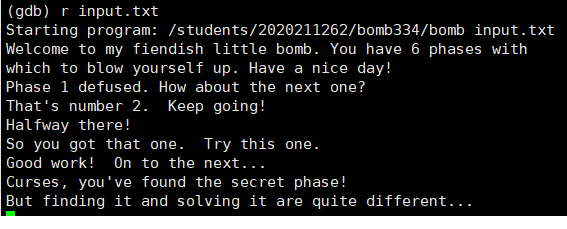


地址0x402970的字符串表示这里会读入两个整数和一个字符串。地址0x420818是拆除第四个炸弹时的输入，所以需要在这里后面加上某个字符串才能进入secret\_phase。

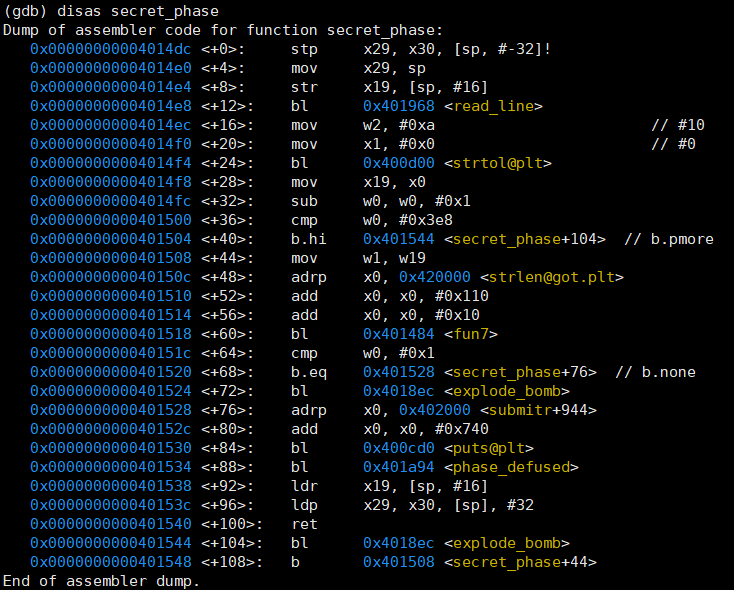
<+120>处运用了strings\_not\_equal函数，如果输入的字符串与0x402980处的字符串不相等则不会进入第七个炸弹。



将0x402980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。



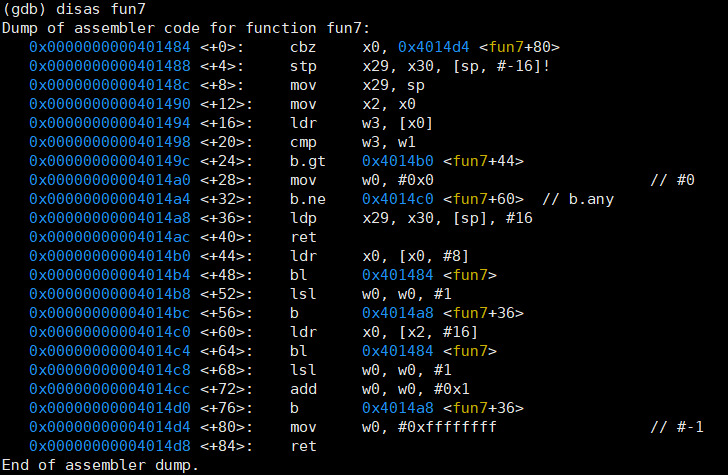
将字符串加入在第四个炸弹的输入的后面，发现在拆除了第六个炸弹之后确实进入了秘密炸弹。



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。

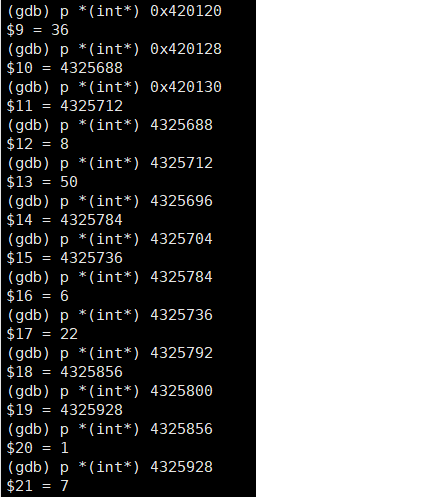
函数将strtol函数的返回值赋值给x19，然后将返回值w0坚毅.如果w0大于0x3e8(999)，则爆炸，所以输入的数必须在1-1000之间。

函数之后把w19赋值给w1，也就是输入的数。然后把0x420120赋值给x0，调用fun7函数。如果fun7函数返回值不等于1则爆炸，否则就成功拆除第七个炸弹。

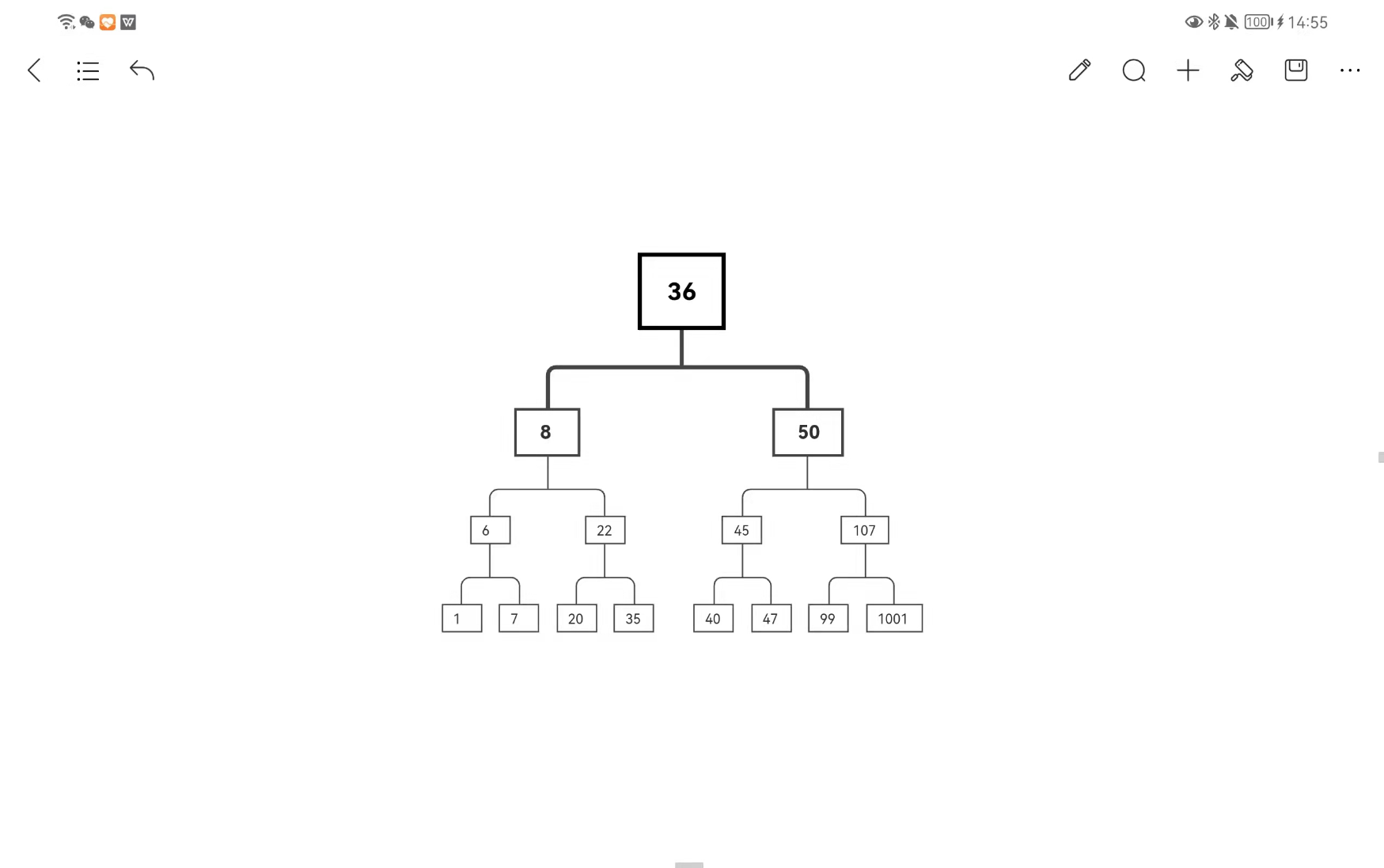


看fun7函数的汇编代码，逐一解读。

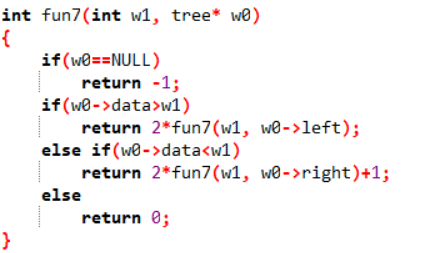
发现根据条件，x0可能加8个字节，也可能加16个字节。



将地址所存的值打印出来，发现x0+0x10和x0+0x8是两个不同的地址。因此，x0原是一个二叉树的根结点地址。

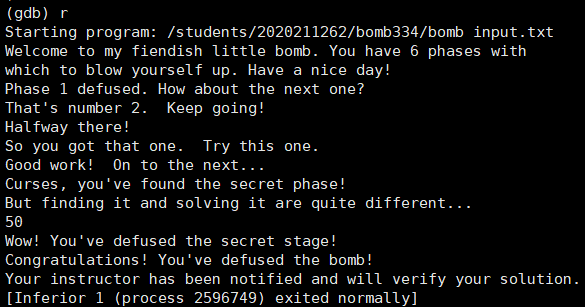


将+0x8的地址设为左子树，+0x10为右子树。将整个树的每个结点打印出来，该二叉树的结构如上图。



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1，根据上图递归，输入需要等于根结点的右结点，也就是50.



输入50，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ARM AArch64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，并借助于ARM指令集的资料，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. ARMv8-A A64 ISA Overview presented by Matteo Franchin

对ARM指令进行学习

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）