Phase_1:

利用 gdb 调试时，使用 disas phase_1 反汇编，根据<string_not_equal>可以知道这个函数可能是字符输入检测程序。调用函数前，给%rsi 赋值，中间要进行是否相等判断，如果不相等就会执行 explode_bomb 爆炸。根据<+11>旁边注释猜测字符可能存在于地址 0x2ab0 中。

利用 x /s 0x2ab0 来调用这个地址的内容，刚好是一串字符串 0x2ab0: "He is evil and fits easily into most overhead storage bins."

Phase_2:

反汇编后，根据<read_six_numbers>知道这个函数要调用另一个读取六个数字的函数，然后判断%rsp 中的数字是否为 1，不相等就会跳转引发爆炸，由此可以判断这个数字一定是 1。

在 read_six_numbers 函数中，先给这个函数分配栈帧，然后又入栈了六个内存空间。在<+29>时，查看地址 0x2d8，发现是%d %d %d %d %d %d，也就是输入了六个整数。后面又调用函数，判断 eax 中是否大于 5，大于就会引发爆炸。

把函数输入的第一个数进行自加，也就是×2 后，再与下一个数字进行比较，不同就会引发爆炸，也会是说，下一个数字一定是前一个数字的 2 倍。在被执行数等于栈底数字时，会跳出循环，return 结束。

所以，第一个数必须是 1，*2 后是 2，*2 后是 4，依次为 8、16、32。根据输入格式，可知答案是 1 2 4 8 16 32

Phase_3:

先给主函数分配空间，访问地址 0x2b16 得到%d %c %d 可以知道输入是两个整数一个字符。然后调用循环，判断 eax 是否大于等于 2，满足就跳转爆炸：所以 eax<=1。然后又判断输入的第一个数是否比 7 大，满足就爆炸：所以第一个数<=7。然后把第一个数放到 eax 中。

再访问第二个地址 0x2b20，得到一串数字，再根据后面的代码，可知后面紧接着 swich 语句，来判断第三个数的值，也就是第三个数取值决定于第一个数取值。第一个数取 0-7 时根据(%rdx,%rax,4),%rax 来跳转地址。swich 语句里第三

个数字的值分别可能等于 0x3c9（969）、0x2ab（683）、0x1eb（491）、0x336（822）、0xdf（223）、0x101（257）、0x14b（331）、0x27f（639）。正常执行就不会引发爆炸，执行完跳转到<+341>。

判断第二个字母是否等于%al 中的内容，不等就会爆炸。最后退出、还原。

第一个数字 0-7 任意一个，我选定 3，然后利用 gdb 监测%al 里的值来确定中间字母是多少，当输入执行后，第一个为 3 跳到第四种情况，所以第三个数为 822，跳出循环，判断字母时发现%al 中存入了 107，查找 ASCII 码得字母为 k。所以答案是 3 k 822。

Phase_4:

准备工作做完之后，查看输入字符的格式，访问 0x2d95 得到%d %d，可以知道输入两个整数。然后调用循环，当 eax 不等于 2 时爆炸，所以 eax=2，也就是输入两个数。然后第二个参数-2>=2 时跳转，否则爆炸。第一个参数给 esi，edi 存 9。

调用 func4，eax 存 0。检查 edi 是否为 0，为 0 就 return。把 esi 里的第二个参数放到 eax 里，比较 edi 和 1，不等就跳转，相等就会 return。然后分配空间，把 esi 里第二个参数给 r12d，edi 给 ebx；edi 自减 1，开始递归调用；ebp=rax+r12，rbx 自减 2 放到 edi 中，r12d 放到 rsi 里，递归调用。最后把 eax+=ebp。

调用完，比较返回值与第一个数，不等爆炸，相等正常退出，还原。

我令第二个数等于 4，通过检测 eax 寄存器的内容得到最后退出循环时，里面放置的是 352，也就是第一个参数必须等于 352 才能避免爆炸，再结合输入格式，答案是 352 4

Phase_5:

反汇编后，访问 0x2d95 这个地址可知输入”%d %d”，输入两个整数。假设输入个数<=1 会跳转爆炸。然后把第一个参数给 eax，并和 0xf 与运算，也就是看第一个参数的后四位是不是 1111，如果是就爆炸。然后对 ecx、edx 都赋值为 0，rsi 存了地址，访问是\n。

edx 自加 1,把 e 开头都扩展为 r 开头后,是一个地址换算:eax=*(rsi+4rax),

对应数组的各个元素位置。后执行加法 ecx+=eax。再次比较 eax 的后四位是否为 1111，不是就跳转到本段开头，重新执行。所以这部分是一个循环，每次取出数组的一个数加到 ecx，直到最后取出 15。加上一开始参数一加了一次，总共取了 16 个值相加。

所以这里我取二进制 1010，也就是 5，它不是 1111，第二个数通过检测 ecx 的数值，可以知道，加到最后循环跳出时，总和为 115。所以答案是 5 115

Phase_6:

前面是准备工作，通过<+34>read_six_number 可知输入为六个整数，由跳转后的 eax-1<5 就爆炸可知第一个参数要<6，正常执行后，比较 r14d+1 是否等于 6，要是相等就跳转将参数分配到寄存器里，要是不等就会重新执行前面的步骤，循环往复下去，直到六个数全部满足小于 6，并且互不相等。这可以推出这些数是作为链表的节点。

后面的一系列比较、跳转、移动代表将输入的参数按照一定的顺序存储在栈帧中，由<+153>到<+156>可以知道第一个节点值>=第二个节点值从而依次类推。同时从<+117>到<+144>得出节点发生线性变化，也就是 x=7-x。

当六个节点信息全部被排列完成后，可以看到有一个地址存储着 node1 的信息，输入 x &0x204230，再用 x/3x 来访问每四个地址里存储的数值，得到 0x0000022a、0x00000221、0x00000188、0x000001e0、0x00000252 五个数值，由 node5 的指针指向 0x00204110 得出 node6 的值：0x00000102。紧接着执行后面一连串的 mov 指令，组成一个完整链表，恢复这个链表信息。

后面<+275>到<+283>，这一段循环来判断我们输入的六个数对应的链表中的数是不是降序排列的，否则就会爆炸。然后就会恢复栈帧，返回，还原。

由上面访问的六个节点降序排列应该是 5 1 2 4 3 6，变换之后是 2 6 5 3 4 1

Secret_phase:

在上面六道解题过程中，用 gdb 进行逐步观测时，发现每次成功后都会调用 phase_defused()函数，再解第 5 题时，我不小心 si 进了 phase_defused，disas

访问最后多出来的地址发现有两行字 Curses, you've found the secret phase!But finding it and solving it are quite different...。其中发现第四个 phase 读取了三个数，其中最后一个数作为字符串读取。读取地址发现 DrEvil，也就是要在第四次中增加 DrEvil 才能进入 secret_phase。

进入汇编 secret_phase 的部分，其中还有调用函数<strtol@plt>可以将字符串变成 long int，前提是数值范围 0-100。调用 fun7 后返回值为 3 就是成功。

反汇编 fun7，发现是个递归函数，有两种情况，一种是乘 2，一种是乘 2+1，因为最后必须要等于 0 退出递归，那么就只能两次后面这种情况退出。x/64xw 0x204150 之后可以得到这个树形结构，只需要找两层节点右边值就可以，也就是到 0x0000006b，即 107

得分截图

13 bomb33	Fri Oct 27 21:51	7	7	67	valid