Bomb Lab

- **phase1**
  - 观察汇编代码可以发现，输入值的地址作为参数传给了phase1函数，然后函数内部又给%rsi赋值，然后调用strings_not_equal函数，那么%rsi很有可能就是目标地址串的地址，通过gdb打印发现猜想正确，并且目标字符串如下
    `We have to stand with our North Korean allies.`
- **phase2**
  - 依据函数名和传入的参数是指向栈上的地址，猜测是读入6个整数，并且存储在栈中
  - 继续观察代码可以得到第一个数应该非负
  - 然后进入一个循环，每次比较%rbp指向的栈上的值加上某一个数和下一个数是否相等，不相等就会boom，所以输入的数后一个和前一个需要满足一些条件，使用gdb调试就可以发现，每次加的那个值分别是1，2，3，4，5
  - 综上，应该输入6个整数，第一个数非负，并且为二级等差数列，一种答案如下

    > 1 2 4 7 11 16

- **phase3**
  - 首先在调用读取函数之前，通过gdb打印出%rsi的值，发现是"%d %d"，说明应该输入两个整数，并且通过传给函数的第三个和第四个参数就可以知道，这两个整数读进来，会分别存储在0xc(%rsp)和0x8(%rsp)中
  - 第一步判断第一个数是否非负并且小于等于7，然后接下来的代码从格式上来看，似乎是在用第一个数的值作为偏移量结合一个跳转表去跳转，之后的过程可以理解为在计算%eax的值，最后和输入的第二个数比较，相等才能通过，并且还有一个判断第一个数是否小于5的
  - 那%eax的值怎么办呢，从代码上看好像不是很容易得知，但是调试一次可以发现，这个%eax的值只和第一个输入的数有关系，第一个数知道后，就可以通过gdb调试直到那个判断的地方，打印出%eax的值，这样就知道第二个数是什么了，然后试了一下，发现果然正确，由前面的分析可以得到第一个数在0到5，所以分别测试可以得到下面5组都是可以通过的答案

    > 5 -758
    > 4 0
    > 3 -758
    > 2 -157
    > 1 -690
    > 0 -19

- **phase4**
  - 和第三题一个套路，发现传入的是"%d %d"，所以输入的应该是两个整数，存储也都是一样的，记作x和y
  - 接着将第一个数作为参数传给fun4，一起传入的还有两个数，就记作a和b，然后内容就是先计算$b=b/2+a$，然后比较b和x的大小关系，利用gdb调试一遍就可以知道这个函数的一个递归调用了，如果$b/2+a>x$，那么调用$fun4(x,b-1,a)$，然后将eax的值变为之前两倍，返回，如果$b/2+a<x$，那么调用$fun4(x,b-1,a)$，然后$eax=2*eax+1$，返回，如果相等，那么直接返回
  - fun4函数结束之后，判断返回值和2是否相等，所以我们需要想到一个值，使得fun4的返回值可以是2，可以发现，如果要得到2，那么就应该是先小于，然后大于，接着等于(因为先给加1，然后翻倍)，那么第一次$14/2=7$，接着$(7-1)/2+0=3$，然后$3/2+(3+1)=5$，所以第一个数应该是5
  - 接着看调试发现，很明显可以发现第二个数需要和2相等，所以答案如下

    ```
    5 2
    ```

- **phase5**
  - 和之前还是相同的套路，输入的是两个整数，并且第一个数不能等于15

- 猜测后面的过程是将第一个数作为下标访问一个数组，然后将得到的值继续作为下标去访问，这样一直到访问到15结束，每访问一次，edx就会加1，ecx会加上得到的值，然后判断edx和15是否相等，以及第二个数和ecx是否相等
- 然后开始检验，先通过`x/60d $rsi`打印出那个"数组"，发现确实是数组，并且数组中的元素互不相同，更加证明猜想正确性(并且最后发现和预料的那样，数组刚好是循环的，从一值开始可以访问到这个值，并且遍历了其他的元素)，然后我第一次输入的就是5，发现这样循环访问，刚好15次可以访问到15，满足那个条件，并且此时ecx的值是115，赶紧试了一下，发现直接通过

  > 5 115

- **phase6**
  - 和2同理，输入的是6个整数
  - gdb调试发现，刚进去之后就进入一个循环，然后每次循环又有一个循环，仔细调试发现，是在判断6个数是否大于0小于7，并且互不相同，恰好我输入的就是`1 2 3 4 5 6`，继续调试
  - 接着似乎从内存中取了一个链表，然后依据输入的数将指向链表节点的指针放入栈中，比如当前值为4，就将第四个节点的指针压入栈中(反着压)，这个可以通过打印出被赋值后的rdx指向的值和栈里面的情况来验证，发现是正确的!

- 继续调试，发现把栈中压入的节点按顺序连接形成新的链表，然后似乎是判断整个链表是否是单调不减的，知道了这个那就好办了，根据前面打印出的每个节点的值很容易知道如何排，但是这个时候遇到了问题，就是那个node6在内存中的位置并不在node5之后，虽然我找到了那个的位置，但是我怎么打印都看不到我想要的那个值，于是我就先把前面5个排好顺序，然后就可以运行到第6个，看到它的大小，最后答案就直接出来了，测试一下，通过！

  > 5 1 2 6 4 3

- **secretphase**
  - 发现
    - 先直接在asm中查找，因为肯定是在什么地方call了一下，然后发现在phase_defused函数中call了一下，但是phase_defused函数一进入就进行判断，看某个地方的值和6是否相等，不相等直接返回，不是很理解，因为我也不知道那块地方存储的是什么东西，然后gdb调试发现在第6关通过之后，就可以进入，但是call那个还得判断一下输入，和之前的套路一样，打印出那个格式发现是"%d %d %s",说明需要在哪一关的答案后面加字符串，后面还有判断字符串是否相等，和phase1一个套路，直接打出来，发现是"DrEvil"，但是这个加在哪我没有分析出来，但是问题不大，就那么几行，我试一试就知道了，最后发现是在第4行加

- 进入之后，会先读取一行内容，然后调用一个将输入转为10进制的long类型的函数，并且后面继续给出范围，1~1001，然后给rdi赋一个值，似乎是一个字符串的地址，然后调用fun7
- 里面的套路和fun4很像，但是每次判断的数是从内存中取的，无法计算(尝试打印出来，但是未果)，但是这不影响得出答案，先确定好应该是先小于，再大于，然后又小于，最后等于，随便输一个就可以知道第一个是50，所以我重新输入40，发现第二个是36，符号，但是第三个是45，我又重新输入47，没想到第四次直接就是47，通过！

- **感想**
  - 一开始还感觉一头雾水，不知道从哪里开始，但是跟着引导做完1和2之后，再做后面的就轻松很多，并且感觉很好玩，而且对汇编代码的理解以及函数调用和参数传递的认识也加深了