|  |  |
| --- | --- |
| 实验题目：bomblab | |
| 实验目的：利用工具反汇编，逆向分析程序 | |
| 实验环境：linux | |
| 实验内容及操作步骤：  objdump -d bomb > bomb.txt  将汇编代码重定向到bomb.txt 找到phase\_1的汇编并分析  下面我们将每句汇编的作用写在ida注释。  Phase\_1    将ebp+8处的值存到eax,即是输入的第一个参数，然后将eax放到phase\_1的栈顶处。调用strings\_not\_equal。strings\_not\_equal的函数做的事情主要有，先比较上面提到的第一个输入的数和0x804a26c处的值的长度，如果**长度不相等**，将%eax置1，调回phase\_1，然后phase\_1判断eax是否不为0，引爆炸弹。  如果**长度相等**，就一直比较答案与输入的字符，如果不相等。  综上分析，取出0x804a26c中的值即可得到第一关答案为He is evil and fits easily into most overhead storage bins.    Phase\_2    在call 8048840 <\_\_isoc99\_sscanf@plt>之后，进入了奇怪的函数之后，单步调试，看到sscanf输入的格式要为 %d %d %d %d %d %d,所以是要为6个单独的数字.  因为要反复跳转代码，所以下面我们同ida分析汇编代码。通过分析，我们发现要求输入的后一个数要比前一个数大ebx。而ebx是累加的。所以 我们输入1 2 4 7 11 16，结果正确。      Phase\_3    进入sscanf后，单步调试，发现要转换成的是两个整数。    之后需要判断输入的个数是否大于1，不大于1则爆炸，并且第一位数不能大于7。所以第一位数可以选0-7。这里选择0。  发现这一关就是一个switch的跳转表。第三个数要是对于case的值。  我们选择case0,那么数就是0x15D.  所以第二个char变成ascii码，就是k。    所以输入答案，正确。    Phase\_4    这是一个递归函数，要求返回值为6.    我们进入func4观察功能。就是首先对参数3进行一些变换，然后判断参数1和参数3的值。若相等，就停止递归。这是后序递归，不断对eax进行变化。停止递归时，eax=0.我们要达到6，就是0x2+1=1,1x2+1=3,3x2=6.所以我们实际要递归4层。 我们要设计参数1.      这里放一下计算x的手稿。就是通过每次递归，变换参数，不断缩小x的范围。最终，x=6.第二个参数要求为6.所以结果就是6 6    Phase\_5.  MOVSX说明：带符号扩展传送指令 signed extend  MOVZX 无符号扩展      通过查看逻辑，就知道是要输入字符串。然后取每个字节的低4位做偏移，取data段找固定字符，要与 devils匹配。 Devils的偏移分别是25c4f7，那么我们就找ascii表，找到相同低4位的字符，beldog。输入，正确。    Phase\_6.  Xchg指令，交换指令XCHG是两个寄存器，寄存器和[内存变量](https://baike.baidu.com/item/%E5%86%85%E5%AD%98%E5%8F%98%E9%87%8F/3389507)之间内容的交换指令，两个[操作数](https://baike.baidu.com/item/%E6%93%8D%E4%BD%9C%E6%95%B0/7658270)的数据类型要相同，可以是一个字节，也可以是一个字，也可以是双字    这里其实是一个嵌套for循环，外层循环做的事是确保每一位小于6,而内层循环则是多次确保各位不相等。 突破口是esi,ebx同时当计数器，而每次都是esi变满，ebx再变。 每次判断2个数释放一样，一样就退出。      这里就很简单。我们发现这是一个链表，每个节点有自己的int成员，还有下一个节点的地址。  程序要求前一个节点的值比后一个大。      这里一开始犯了个错误。Int是4字节，而且我们机器是小端，所以低位在前。一开始我只算了第一个字节，发现不对。这才发现每个node的第二个字节还有值。把第二个字节的值放在高位，重新排序，得到346125，正确。    Secret\_phase:    我认为全实验最有价值的就是发现隐藏关入口的过程。我们用ida发现phase\_defused函数会调用secret\_phase.然而我们之前都没有调用，说明触发隐藏关肯定有条件。我们看代码，发现会首先判断num\_input\_strings这个全局变量是否为6，然后看从0x804c4d0这个地方读的第三个字符串是否为DrEvil. 而这个num\_input\_strings是什么时候变换呢？0x804c4d0这个到底是什么地方？我们找到read\_line函数，    发现看不懂，那么我们就动态调试！我们用display设置好每次要显示的num\_input\_strings和0x804c4d0    发现初值为0，每输入一次，就++。而输入完第四关时，0x804c4d0就出现了我们刚输入的字符串。 那么我们就明白了，我们第四关在答案后面还要输入DrEvil，然后等num\_input\_strings=6时，就会判断第四关是否有DrEvil。若有，就进入隐藏关！  下面我们分析隐藏关。      我们发现和第四关的逻辑很像，都是递归函数。只是循环的内容变了而已。我们跟入递归函数，发现是个二叉树。通过判断结点的值，当相等时，就停止递归。逻辑和第四关一模一样。也是要递归三层，这里就不再赘述。最后答案得出47，正确。    实验结果及分析：  通过反汇编与动态分析的结合，成功拆除所有炸弹。且发现并破解隐藏关。  各关答案如下：  He is evil and fits easily into most overhead storage bins.  1 2 4 7 11 16  0 k 349  6 6 DrEvil  beldog  3 4 6 1 2 5  47  收获与体会：  1. 熟悉了一些之前陌生的指令  Xchg指令，交换指令XCHG是两个寄存器，寄存器和[内存变量](https://baike.baidu.com/item/%E5%86%85%E5%AD%98%E5%8F%98%E9%87%8F/3389507)之间内容的交换指令，两个[操作数](https://baike.baidu.com/item/%E6%93%8D%E4%BD%9C%E6%95%B0/7658270)的数据类型要相同，可以是一个字节，也可以是一个字，也可以是双字  MOVSX说明：带符号扩展传送指令 signed extend  MOVZX 无符号扩展  2. display 用于每次都展示变量/地址上的值 display 变量名/addr  与x/fmt 不同的是，display是每一次操作后都显示一下之前display设置过的东西。  同时注意，gdb中 \*就是寻址。 如x/s \*addr，那么就是取addr里存的内容当指针，根据指针打印字符串。 而x/s addr，就是把addr当指针，打印字符串。  动态调试是来找运行时一些变量，地址内容的变换。特别是一些静态难以得出的数据。 通常先静态分析，有不懂的点再专门动态去分析。  3.真实逆向都是ida+gdb+peda结合使用。Ida当之无愧是静态分析的神器。虽然可以看到伪代码，但是当程序加入花指令或反调试时，我们的汇编功底就显得十分重要。特别是通过汇编反推数据结构的能力，如链表，二叉树等。只有推出数据结构，才能更好的理解代码的目的。 | |
| 实  验成绩 |  |