尹浚宇 161130118

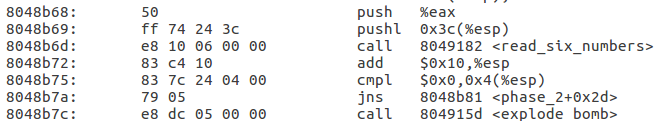
**阶段1：字符串比较**



根据该段汇编代码及函数名称可确定答案字符串存于0x804a098处, 用gdb得



**阶段2：循环**



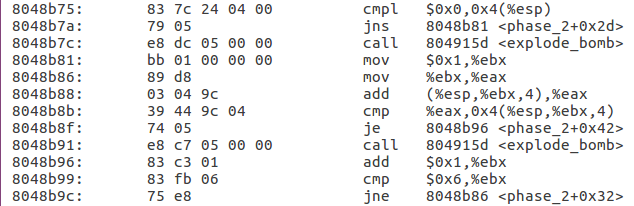
根据read\_six\_numbers可知这题需要读入六个数字.

在read\_six\_numbers中有



用gdb查看0x804a263处的字符串, 得到输入格式为



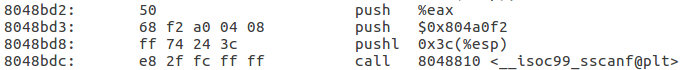


根据这一段代码可知, 需要输入的整数数组需要满足条件:

a[i + 1] = a[i] + i + 1

a[0] >= 0.

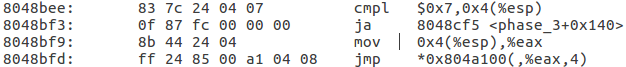
**阶段3：条件/分支**



同样用gdb查看0x804a0f2处的字符串, 得到输入格式为



三个参数分别在esp + 3, esp + 4, esp + 8位置.



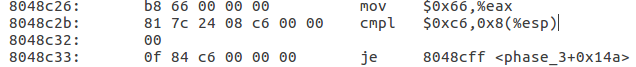
根据上图, 可知第一个参数范围在0 – 7之间.

且查看内存得知, 0x804a100指向8048c04, 即该条指令的下一条指令, 于是我们可以知道eax取值的不同使该函数跳向不同的片段.



根据末尾的这一段得知, 输入的字符需要满足一定要求.

取一个片段做分析:



可知不同的跳转片段需要输入不同的字符和不同的另一个整数来拆弹.

总结如下表

|  |  |  |
| --- | --- | --- |
| 0 | o | 199 |
| 1 | f | 198 |
| 2 | o | 266 |
| 3 | e | 131 |
| 4 | k | 290 |
| 5 | d | 450 |
| 6 | n | 377 |
| 7 | o | 411 |

**阶段4：递归调用和栈**

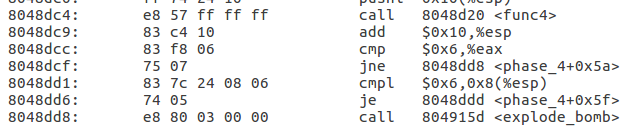


同样用gdb查看0x804a26f处的字符串, 得到输入格式为





根据此段得知参数一是在0 – 14之间的整数.



根据此段得知,函数 func4的返回值和我们输入的第二个参数都应该是6.



根据上图得知, func4有三个参数, 这里的调用形式是 func4(p1, 0, 14), p1为我们输入的第一个参数.

根据func4函数的汇编码, 可以得出它的C语言近似形式:

int f(int c, int b, int a)

{

if ((a + b) / 2 == c)

return 0;

if ((a + b) / 2 < c)

return 2 \* f(c, (a + b) / 2 + 1, a) + 1;

else

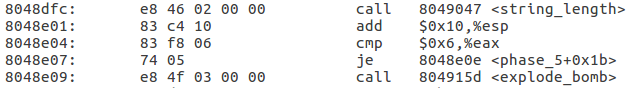
return 2 \* f(c, b, (a +b ) /2 -1);

}

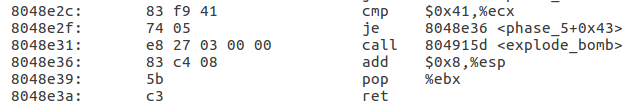
将f(p1, 0, 14) = 6 代入, 解得 p1 = 6.

于是该题答案为6 6.

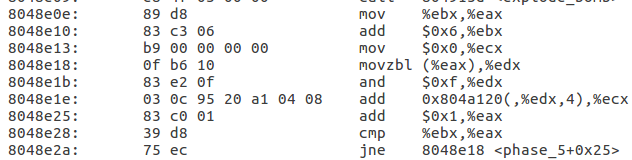
**阶段5：指针**



根据该段, 我们得知输入的字符串长度应该为6.



根据该段, 我们可知最后ecx中的值应该为0x41.



根据该段, 我们可知, 我们输入的每一个字符应该作为一个数组的下标, 且满足这些下标对应的数组中的值的和为0x41.

其中

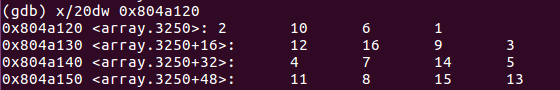


这两句的效果是将我们输入的字符串中的字符减去’0’得到的差作为下标.



这一行暗示我们, 数组起始位置在0x804a120处.

用gdb打印该处的值:



可将结果总结为下表:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ascii | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | : | ; | < | = | > | ? |
| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| value | 2 | 10 | 6 | 1 | 12 | 16 | 9 | 3 | 4 | 7 | 14 | 5 | 11 | 8 | 15 | 13 |

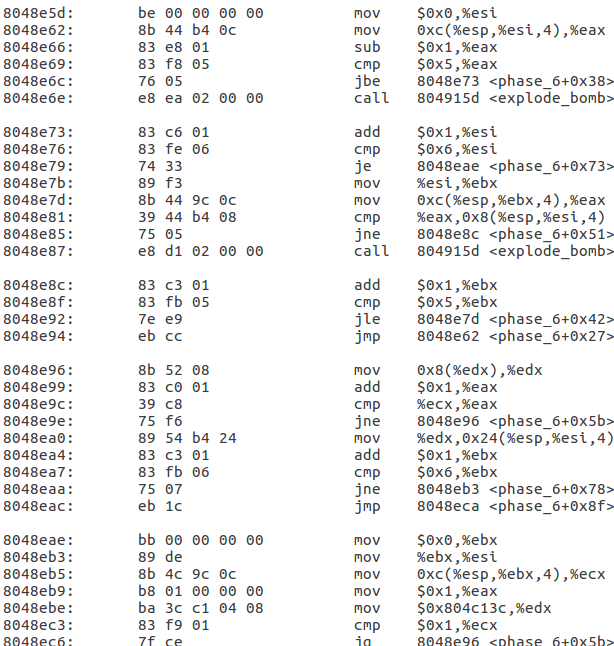
一个答案为578:>?

**阶段6：链表/指针/结构**

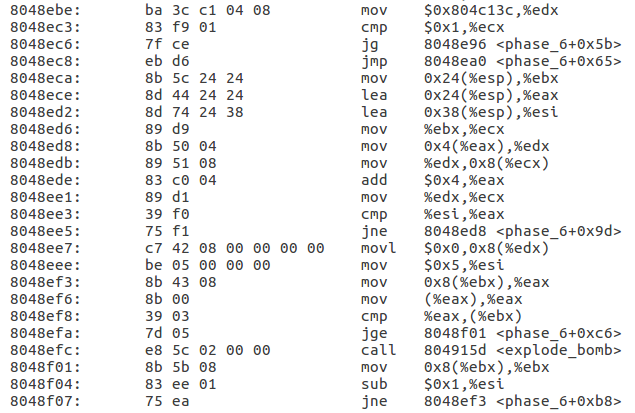
该题较长, 需要分段多次分析.



根据这段可知, 该题也需要读入六个整数.



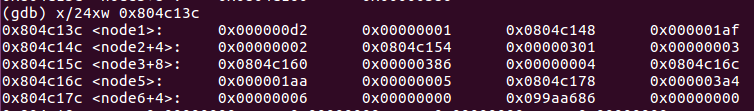
根据该段可知, 我们输入的六个参数必须两两各不相同, 且在范围1 – 6内.



根据该段可知, 我们需要对链表中的数据进行降序排序.



由该行可知, 链表中的数据可能位于0x804c13c处, 使用gdb查看:

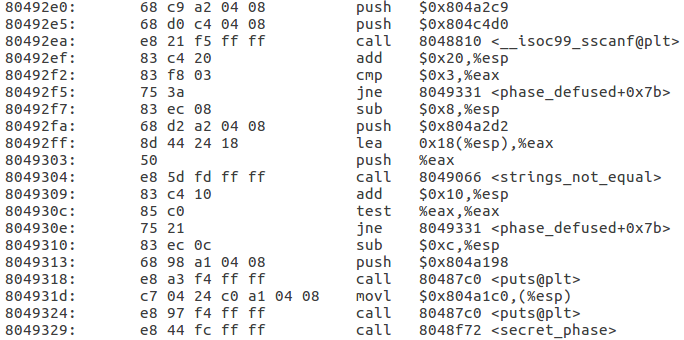


可总结如下表:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 原顺序 | 1 | 2 | 3 | 4 | 5 | 6 |
| 降序 | 6 | 4 | 3 | 2 | 5 | 1 |
| 值(十进制) | 210 | 431 | 769 | 902 | 426 | 932 |

答案即为6 4 3 2 5 1.

**隐藏阶段:**

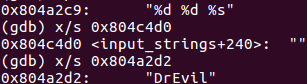


根据phase\_defused中的该段可知, 存在一个secret\_phase.

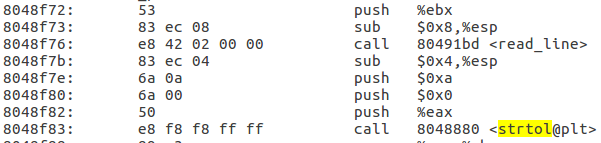
且触发条件为某一次输入的参数个数为三个.

且后面调用了strings\_not\_equal函数比较, 我们可猜想需要在某一步输入一段正确的字符串以开启隐藏关.

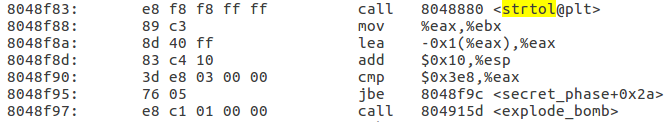
根据之前的经验, 我们分别打印0x804a2c9, 0x804c4d0, 0x804a2d2处的字符串,



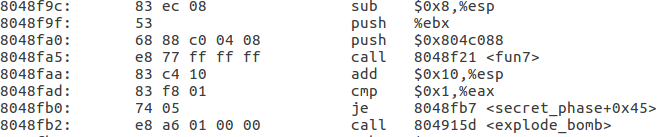
回想之前的关卡, 发现第四关输入的参数后面可以多加一个字符串, 于是在第四关的答案后加上 DrEvil, 成功开启隐藏关.



观察隐藏关的片段汇编码, 得知这关我们需要输入一行数字.



继续观察可知, 输入的数字必须在0 – 0x3e8之间.

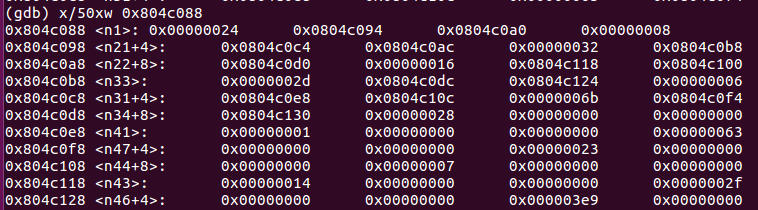


而后又该段得知, secret\_phase函数调用了fun7函数, 且fun7函数的返回值要求为1.



由该段得知, fun7函数有两个参数, 一个是我们输入的数字, 一个是一个地址.

于是我们使用gdb查看该地址



根据该片段我们可以推测, 该指针指向一种自定数据结构, 该结构由两个指向这种结构的指针和一个值组成.

我们再来观察fun7函数, 发现这是一个递归函数, 我们可以写出它的C语言近似版本.

struct node

{

int val;

node \*Lc;

node \*Rc;

};

int f(long x, node \*p)

{

if (p == NULL)

return 0xffffffff;

else

{

if(p->val == x)

return 0;

else if (p->val < x)

return 2\*f(p->Rc, x) + 1;

else if (p->val > x)

return 2\*f(p->Lc, x);

}

}

由f(x, p) = 1, 和该数据结构的组织形式, 解得x = 50.

于是答案为50.