

这道题是一切的开端，我想好好理解一下这个数据溢出的问题。

0xFF是怎么来的？

这是一个十六进制的表达式，十六进制的 F转换为十进制为15；

十六进制的FF转换为十进制为255，二进制为1111 1111

并且一个字节由八个比特组成，每个比特位只能说0或者1

PS：

1.在ASCII编码中一个英文字母占一个字节的空间，一个中文汉字占两个字节的空间

在UTF-8编码中一个英文字母占一个字节的空间，一个中文汉字占三到四个的空间

1. 按位与‘&’的功能是参与运算的两数各自对应的二进补码相与：同为1，结果为1；不同，结果为0

例如：9&5可写算式如下：0000 1001(9的二进制补码)&0000 0101(5的二进制补码)

0000 1001

0000 0101

对比2个数的补码可以得出结果为0000 0001也就是1的二进制补码

所以9&5=1

回到题目：

我们平时变成输入的字符如果要转换成十六进制时，一般会进行ASCII编码，而此时转换结果是一个单字节（byte）的表达式。但这道题我们需要对双字节（dword）数组list2进行运算，所以就必须让它也变成单字节，所以&xFF（二进制表示为1111 1111，如果和双字节的word比较就当做0000 0000 1111 1111来处理）处理来只取最后八位比特，高位就会全变为0，在python里输出就会将高位的0删去

比如0xFFFFFFFC4 & 0xFF = 196（0xC4）

这样就可以把list2中的数据数据变为和我们输入的字符的十六进制表达式相同的一字节

但最后我们得到的字符串flag对应列表为[-146, 867, -908, 1126, -389, -187, -147, 1378, -142, 1332, 1123, -667, -1185, -682, 1133, 95, 874, 816, 617, 622, -161, -686, 1587, 381]

这是因为python不会控制数据溢出导致的。

在C语言中由于在定义一个变量时会对变量进行声明，这个时候就会确定这个变量的内存空间和数据范围，与此同时C语言还被赋予控制数据溢出的功能

数据溢出转换规则：

数据范围的表示是一个环状结构。

如果数据增长到最大值，则数据从最小值方向开始依次递增

如果数据减到最小值，则数据从最大值方向开始依次递减

比如：char a，char对应的数据范围是-2^7~2^7-1(-128~127)

后续如果赋值a=400，那么赋值的结果其实是a=400-256-256=-122，这相当于一个256取模（取余）或者说是取最低八位的过程

这样就可以保证数据不会溢出，但是python中没有声明变量类型的步骤，自然也不会控制数据溢出，所以需要我们认为控制，这个时候只需要再对这个溢出的数据进行取模就行了。

再回到题目：

因为题目是由C原因写的，但是我是用python写的，所以还需要进行数据溢出的控制。

只需要对每个数据进行&0xFF（或者%256）就可以控制数据溢出就可以得到字符串flag每个字符字符真正对应的十六进制表达式