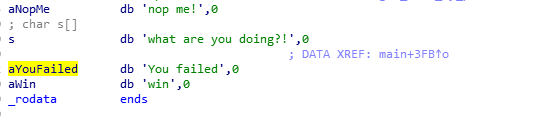
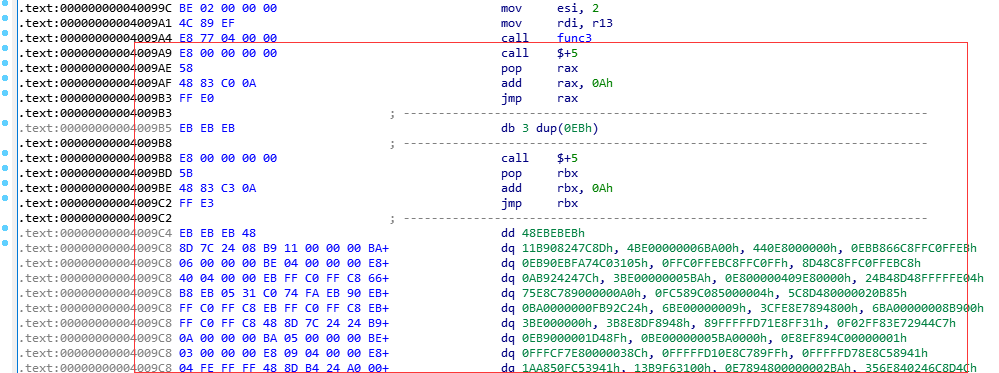
首先拖进ida后可以发现成功失败字符串没有引用地址。



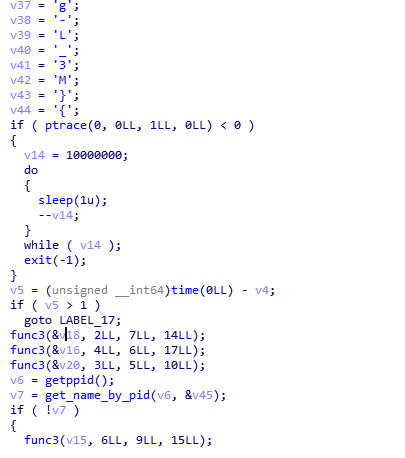
然后分析main函数可以确定是因为花指令的原因。

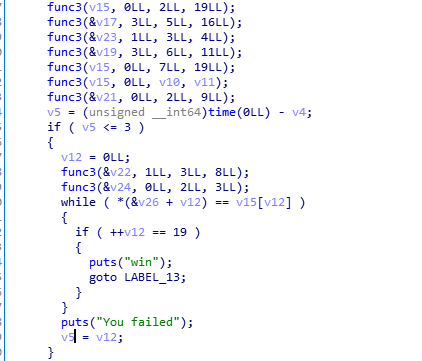


首先写出IDAPython脚本去花指令。

#nop花指令  
arr = [[0x4009A9,0x4009C7],[0x4009E0,0x4009FE],[0x400ACA,0x400AE8],[0x400B1A,0x400B38],[0x400B65,0x400B92],[0x400BEF,0x400C00]]  
for i in range(len(arr)):  
 for j in range(arr[i][0],arr[i][1]):  
 PatchByte(j,0x90)

然后可以得到清晰的反编译代码。





现在已经可以看到调用失败成功字符串的地方。

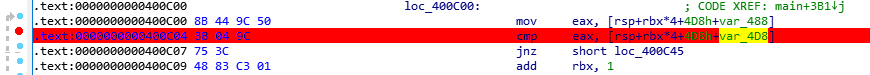
并且开始的一堆字符看起来很像字符串，输入经过很多处理后，和这些字符串对比。

猜测这些处理是将输入进行打乱。

我们进行动态调试验证是不是打乱输入，在ida调试elf文件之前，先使用IdaPython给nop掉反调试。

#nop反调试  
arr = [[0x400972,0x400978], [0x400987,0x40098D],[0x400A2F,0x400A35],[0x400A77,0x400A7D],[0x400AAB,0x400AB1],[0x400BBC,0x400BC2]]  
for i in range(len(arr)):  
 for j in range(arr[i][0],arr[i][1]):  
 PatchByte(j,0x90)

然后在这里设断点即可。



动态调试发现是将输入进行了打乱，输入“0123456789abcdefghi”后，断点处是“8f6c90e1dg237abh5i4”，然后写出脚本即可还原。

str = "0123456789abcdefghi"  
str1 = "8f6c90e1dg237abh5i4"  
str2 = "S@yRtfTl0+ag-L\_3M}{"  
flag = ""  
for i in range(len(str2)):  
 flag += str2[str1.find(str[i])]  
print flag

得到flag为  **flag{My-StL\_R0T@+3}**