首先是pyc的反编译问题，uncompyle6版本最好是最新的，或者试下在线反编译，因为比赛期间有同学说反编译的代码装了unicorn库也运行不了，然后看了下是反编译出的bytescode不对劲。

然后是关于unicorn，要了解的话跟着大佬的文章学习就ok了，这方面不多说，放上学习链接：

<https://www.anquanke.com/post/id/95199#h2-18>

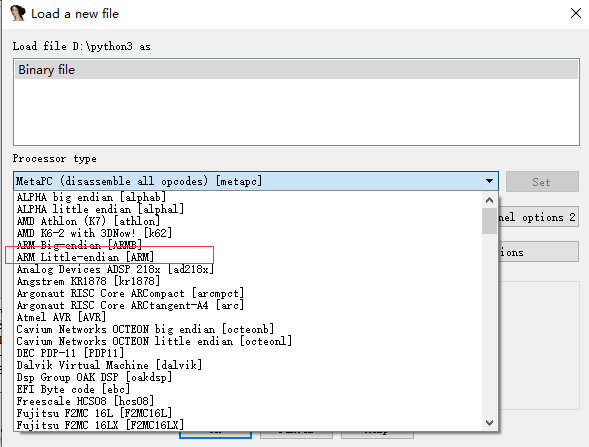
<https://bbs.pediy.com/thread-253868.htm>

然后简单说下题目的代码，bytescode是arm架构的一个函数的字节码，然后通过unicorn引擎模拟执行了，就是arm汇编的字节码，我们就可以用ida进行反汇编和反编译。

首先将bytescode写入到文件中。

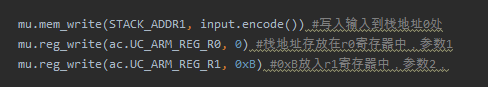
def write(path,bytes):  
 file = open(path,'wb')  
 file.write(bytes)  
 file.close()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 bytescode = b'\x08\xb0-\xe5\x04\xe0\x8d\xe5\x04\xb0\x8d\xe2\x10\xd0M\xe2\x10\x00\x0b\xe5\x14\x10\x0b\xe5\x000\xa0\xe3\x080\x0b\xe5\x000\xa0\xe3\x080\x0b\xe5\x1b\x00\x00\xea\x080\x1b\xe5\x010\x03\xe2\x00\x00S\xe3\n\x00\x00\n\x080\x1b\xe5\x10 \x1b\xe5\x030\x82\xe0\x08 \x1b\xe5\x10\x10\x1b\xe5\x02 \x81\xe0\x00 \xd2\xe5\x07 \x82\xe2r \xef\xe6\x00 \xc3\xe5\t\x00\x00\xea\x080\x1b\xe5\x10 \x1b\xe5\x030\x82\xe0\x08 \x1b\xe5\x10\x10\x1b\xe5\x02 \x81\xe0\x00 \xd2\xe5\x04 \x82\xe2r \xef\xe6\x00 \xc3\xe5\x080\x1b\xe5\x010\x83\xe2\x080\x0b\xe5\x080\x1b\xe5\x1e\x00S\xe3\xe0\xff\xff\xda\x000\xa0\xe3\x080\x0b\xe5\x17\x00\x00\xea\x080\x1b\xe5\x10 \x1b\xe5\x030\x82\xe0\x000\xd3\xe5\x0c0\x0b\xe5\x080\x1b\xe5\x10 \x1b\xe5\x030\x82\xe0\x08 \x1b\xe5\x10 \x82\xe2\x10\x10\x1b\xe5\x02 \x81\xe0\x00 \xd2\xe5\x00 \xc3\xe5\x080\x1b\xe5\x100\x83\xe2\x10 \x1b\xe5\x030\x82\xe0\x0c \x1b\xe5r \xef\xe6\x00 \xc3\xe5\x080\x1b\xe5\x010\x83\xe2\x080\x0b\xe5\x080\x1b\xe5\x0e\x00S\xe3\xe4\xff\xff\xda\x000\xa0\xe3\x080\x0b\xe5\x17\x00\x00\xea\x080\x1b\xe5\x10 \x1b\xe5\x030\x82\xe0\x000\xd3\xe5\x0c0\x0b\xe5\x080\x1b\xe5\x10 \x1b\xe5\x030\x82\xe0\x08 \x1b\xe5\x01 \x82\xe2\x10\x10\x1b\xe5\x02 \x81\xe0\x00 \xd2\xe5\x00 \xc3\xe5\x080\x1b\xe5\x010\x83\xe2\x10 \x1b\xe5\x030\x82\xe0\x0c \x1b\xe5r \xef\xe6\x00 \xc3\xe5\x080\x1b\xe5\x020\x83\xe2\x080\x0b\xe5\x080\x1b\xe5\x1d\x00S\xe3\xe4\xff\xff\xda\x140\x1b\xe5\x00\x00S\xe3\x01\x00\x00\x1a\x010\xa0\xe3\x04\x00\x00\xea\x140\x1b\xe5\x010C\xe2\x03\x10\xa0\xe1\x10\x00\x1b\xe5\x8f\xff\xff\xeb\x03\x00\xa0\xe1\x04\xd0K\xe2\x00\xb0\x9d\xe5\x04\xd0\x8d\xe2\x04\xf0\x9d\xe4'  
 write("D:/python3",bytescode)

然后拖进ida时选择模式是arm little-endian。



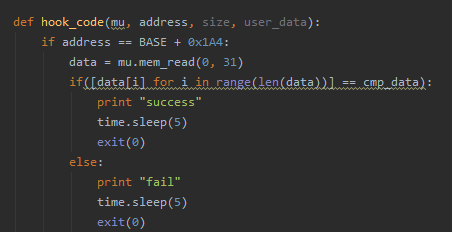
然后按下在按下C键即可反汇编，这时也能反编译了。

算法十分简单，主要是递归调用函数进处理，需要留意的是传入函数的两个参数是在python代码里面



也就是说传入的参数一个是输入，一个是递归调用的次数0xB。

然后就是判断地方的Hook\_code，当执行到0x1A4地址处，也就是模拟执行的函数已经执行完了，就会通过mu.mem\_read(0,31)，读出递归调用函数执行完后，改变了的输入的数据，然后进行比较。



现在逻辑很清晰了，传进了函数两个参数，一个是输入，一个是0xB，然后执行完后，和比较数据进行对比，算法逆向不再具体分析。

源码：

#-\*- coding:utf-8 -\*-  
from unicorn import \*  
from unicorn import arm\_const as ac  
import time  
  
def Unicorn(input):  
 bytescode = b'\x08\xb0-\xe5\x04\xe0\x8d\xe5\x04\xb0\x8d\xe2\x10\xd0M\xe2\x10\x00\x0b\xe5\x14\x10\x0b\xe5\x000\xa0\xe3\x080\x0b\xe5\x000\xa0\xe3\x080\x0b\xe5\x1b\x00\x00\xea\x080\x1b\xe5\x010\x03\xe2\x00\x00S\xe3\n\x00\x00\n\x080\x1b\xe5\x10 \x1b\xe5\x030\x82\xe0\x08 \x1b\xe5\x10\x10\x1b\xe5\x02 \x81\xe0\x00 \xd2\xe5\x07 \x82\xe2r \xef\xe6\x00 \xc3\xe5\t\x00\x00\xea\x080\x1b\xe5\x10 \x1b\xe5\x030\x82\xe0\x08 \x1b\xe5\x10\x10\x1b\xe5\x02 \x81\xe0\x00 \xd2\xe5\x04 \x82\xe2r \xef\xe6\x00 \xc3\xe5\x080\x1b\xe5\x010\x83\xe2\x080\x0b\xe5\x080\x1b\xe5\x1e\x00S\xe3\xe0\xff\xff\xda\x000\xa0\xe3\x080\x0b\xe5\x17\x00\x00\xea\x080\x1b\xe5\x10 \x1b\xe5\x030\x82\xe0\x000\xd3\xe5\x0c0\x0b\xe5\x080\x1b\xe5\x10 \x1b\xe5\x030\x82\xe0\x08 \x1b\xe5\x10 \x82\xe2\x10\x10\x1b\xe5\x02 \x81\xe0\x00 \xd2\xe5\x00 \xc3\xe5\x080\x1b\xe5\x100\x83\xe2\x10 \x1b\xe5\x030\x82\xe0\x0c \x1b\xe5r \xef\xe6\x00 \xc3\xe5\x080\x1b\xe5\x010\x83\xe2\x080\x0b\xe5\x080\x1b\xe5\x0e\x00S\xe3\xe4\xff\xff\xda\x000\xa0\xe3\x080\x0b\xe5\x17\x00\x00\xea\x080\x1b\xe5\x10 \x1b\xe5\x030\x82\xe0\x000\xd3\xe5\x0c0\x0b\xe5\x080\x1b\xe5\x10 \x1b\xe5\x030\x82\xe0\x08 \x1b\xe5\x01 \x82\xe2\x10\x10\x1b\xe5\x02 \x81\xe0\x00 \xd2\xe5\x00 \xc3\xe5\x080\x1b\xe5\x010\x83\xe2\x10 \x1b\xe5\x030\x82\xe0\x0c \x1b\xe5r \xef\xe6\x00 \xc3\xe5\x080\x1b\xe5\x020\x83\xe2\x080\x0b\xe5\x080\x1b\xe5\x1d\x00S\xe3\xe4\xff\xff\xda\x140\x1b\xe5\x00\x00S\xe3\x01\x00\x00\x1a\x010\xa0\xe3\x04\x00\x00\xea\x140\x1b\xe5\x010C\xe2\x03\x10\xa0\xe1\x10\x00\x1b\xe5\x8f\xff\xff\xeb\x03\x00\xa0\xe1\x04\xd0K\xe2\x00\xb0\x9d\xe5\x04\xd0\x8d\xe2\x04\xf0\x9d\xe4'  
 cmp\_data = [149,187,165,189,151,176,171,165,114,180,176,161,115,181,155,174,117,163,174,115,187,161,163,175,163,116,115,176,169,99,185]  
  
 def hook\_code(mu, address, size, user\_data):  
 if address == BASE + 0x1A4:  
 data = mu.mem\_read(0, 31)  
 if([data[i] for i in range(len(data))] == cmp\_data):  
 print "success"  
 time.sleep(5)  
 exit(0)  
 else:  
 print "fail"  
 time.sleep(5)  
 exit(0)  
  
 mu = Uc(UC\_ARCH\_ARM, UC\_MODE\_ARM)  
  
 BASE = 0x400000  
 STACK\_ADDR1 = 0x0  
 STACK\_ADDR2 = 1024  
 STACK\_SIZE = 1024\*1024  
  
 mu.mem\_map(BASE, 1024\*1024)  
 mu.mem\_map(STACK\_ADDR1, STACK\_SIZE)  
  
 mu.mem\_write(STACK\_ADDR1, input.encode()) #写入输入到栈地址0处  
 mu.reg\_write(ac.UC\_ARM\_REG\_R0, 0) #栈地址存放在r0寄存器中，参数1  
 mu.reg\_write(ac.UC\_ARM\_REG\_R1, 0xB) #0xB放入r1寄存器中，参数2，  
  
 mu.reg\_write(ac.UC\_ARM\_REG\_SP, STACK\_ADDR2-1)  
  
 mu.mem\_write(BASE, bytescode)  
  
 mu.hook\_add(UC\_HOOK\_CODE, hook\_code)  
 mu.emu\_start(BASE,BASE + 0x1A8)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 input = raw\_input("plz input your flag:")  
 Unicorn(input)

脚本：

def print\_flag(str,n):  
 new\_str = ""  
 new\_str1 = ""  
 new\_str2 = ""  
 for i in range(0,len(str)-1,2):  
 new\_str += str[i+1]  
 new\_str += str[i]  
 new\_str += str[30]  
  
 new\_str1 += new\_str[16:] + new\_str[15] + new\_str[:15]  
  
 for i in range(len(new\_str1)):  
 if(i % 2):  
 new\_str2 += chr(ord(new\_str1[i]) - 7)  
 else:  
 new\_str2 += chr(ord(new\_str1[i]) - 4)  
 if(n == 0):  
 print(new\_str2)  
 return  
 print\_flag(new\_str2, n - 1)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
  
 arr = [149,187,165,189,151,176,171,165,114,180,176,161,115,181,155,174,117,163,174,115,187,161,163,175,163,116,115,176,169,99,185]  
 arr1 = "".join(map(chr,arr))  
 print\_flag(arr1, 0xB)

Flag：Syc{Unic0rn\_1s\_r3al1y\_ama21ng!}