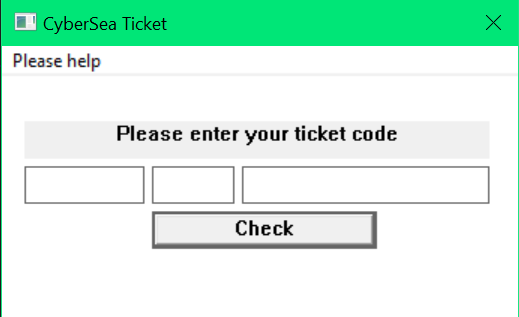
# **Reverse Engineering**

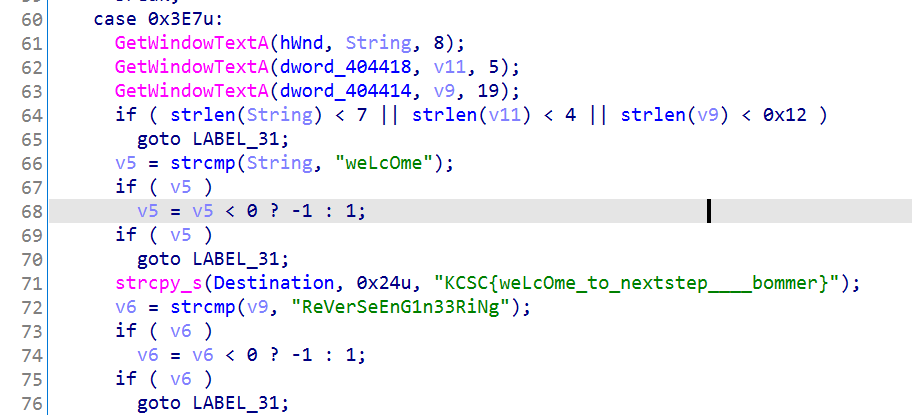
## **B1. KCSC\_Welcome**

Chạy chương trình ta thấy nó có 3 ô box cần nhập vào, nhập đúng thì check sẽ ra flag

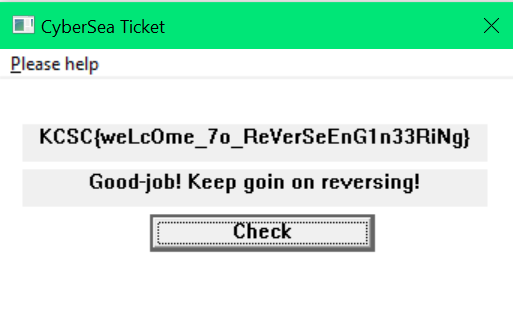


### **Solution:**

Dùng IDA để dịch ngược chương trình, ta thấy được đoạn mã giả C sau:



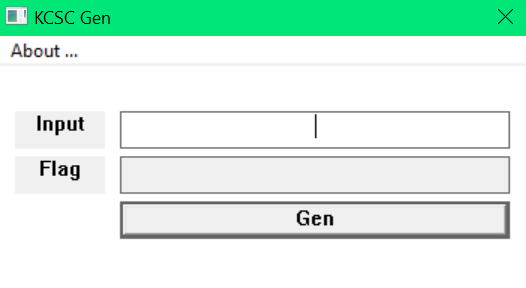
Thấy có strcmp và strcpy-> đoán là ticket cần nhập vào, ta nhập lần lượt "weLcOme", “KCSC{weLcOme\_to\_nextstep\_\_\_\_bommer”, 'ReVerSeEnG1n33RiNg' vào chương trình sẽ được flag.



### **Flag: KCSC{weLcOme\_7o\_ReVerSeEnG1n33RiNg}**

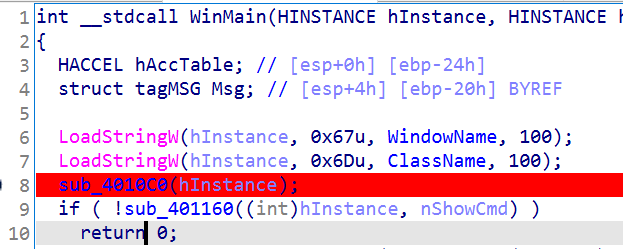
## **B2. KCSC\_NextGen**

Bài này chúng ta nhập đúng được input thì sẽ sinh ra đúng flag

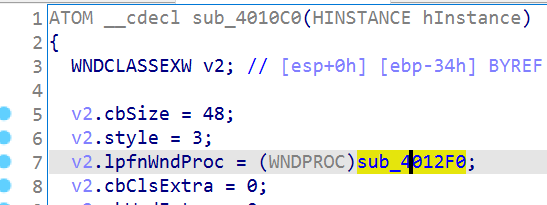


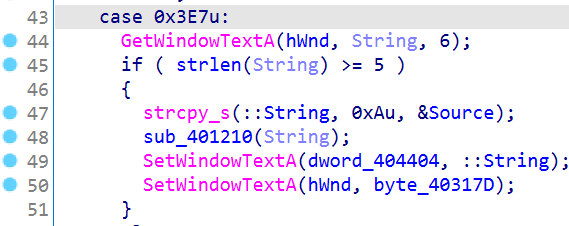
### **Solution:**

Dùng IDA để dịch ngược chương trình, từ hàm WinMain ta chạy vào hàm sub\_4010C0,

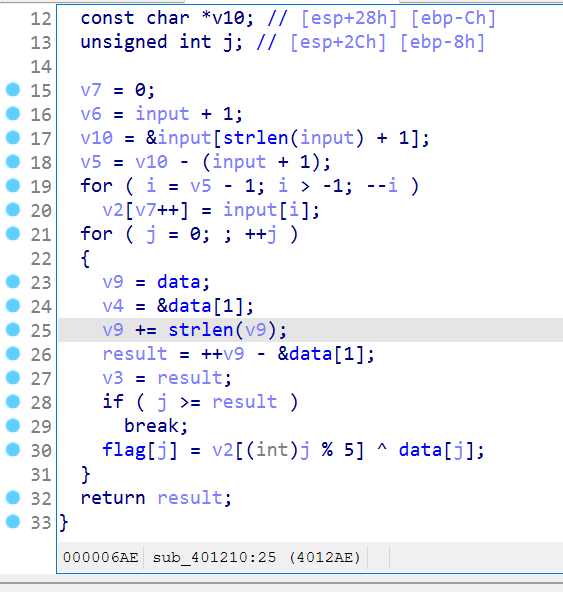


Tiếp tục vào hàm sub\_4012F0





=> độ dài Input string mình nhập vào phải >=5, vào tiếp sub\_401210, đây là hàm check input để sinh ra flag:



unsigned char data[] =

{

32, 45, 60, 55, 8, 30, 49, 27, 28, 26,

5, 5, 48, 1, 44, 62, 32, 11, 71, 1,

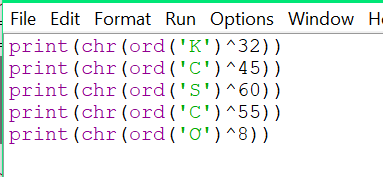
24, 26, 46, 26, 23, 52, 54, 95, 38, 14

};

flag [j] = v2[ j] ^data [j]

Mà định dạng của flag 5 chữ cái đầu là KCSC{

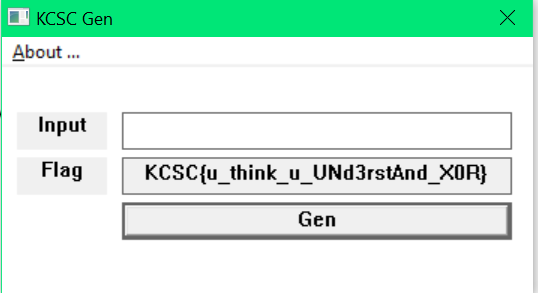
=> Tìm được v2.



v2[] = {k,n, o,t , ƨ}

input là chuỗi đảo ngược thứ tự của v2.

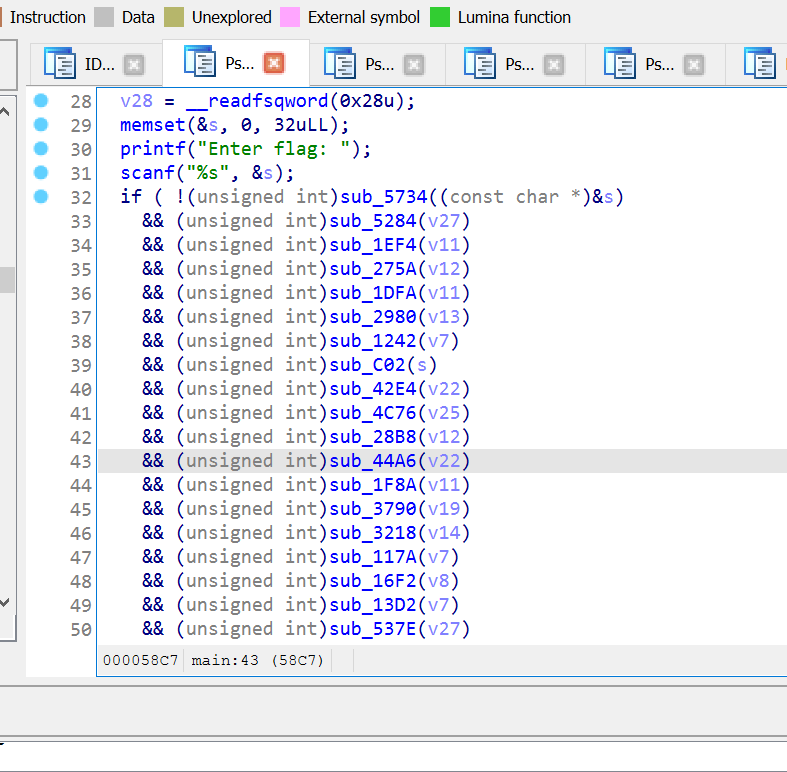
input = ‘stonk’



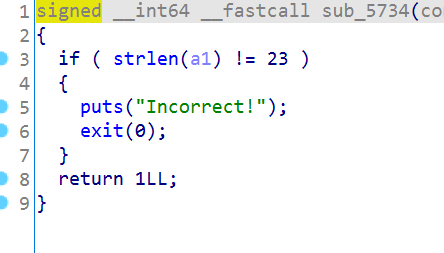
### **Flag: KCSC{u\_think\_u\_UNd3rstAnd\_X0R}**

## **B3. Chal\_3 Check condition!**

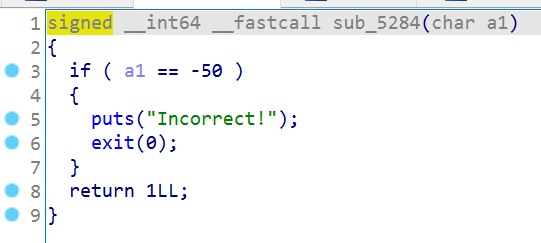
Chạy vào trong hàm main ta sẽ thấy rất nhiều hàm check ký tự của flag,



Tại hàm check đầu tiên, check độ dài của flag, flag dài 23 ký tự

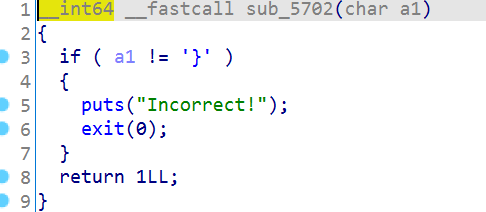


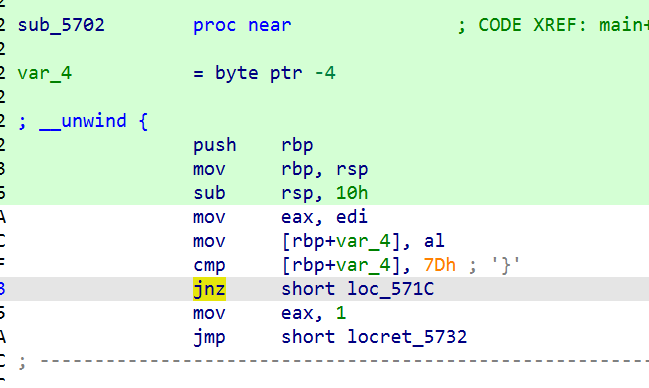
Tại các hàm check còn lại ta đều thấy nó check từng ký tự đó khác giá trị bao nhiêu...



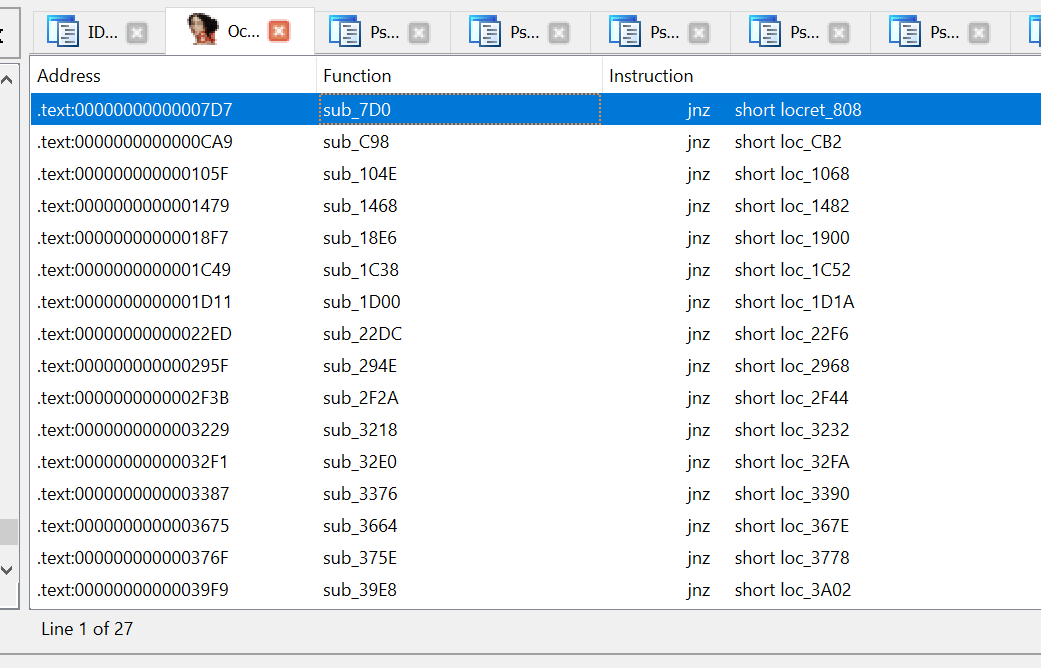
Nhưng check kĩ lại thì ta sẽ thấy cũng có những hàm đưa ta kí tự đó luôn:

VD: tại sub\_5702, a1 != ‘}’ thì incorrect => a1 = ‘!’

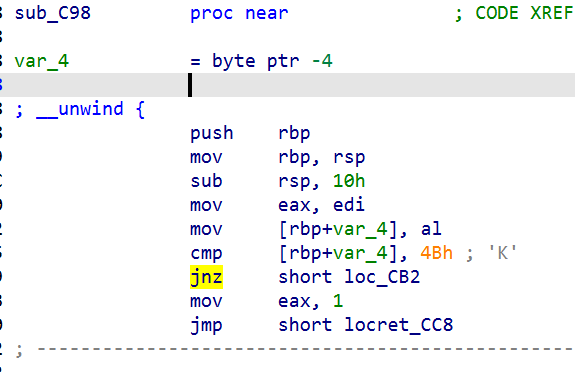




Vì thế 1 search lệnh jnz ra để check các hàm so sánh với đúng ký tự của flag:



Vào từng hàm 1 và mình lấy được flag dễ dàng ^^



### **Flag: KCSC{an9r\_can\_h3lp\_y0u}**

------

Ngoài ra các bạn có thể sd angr trong python để giải bài này nhé.

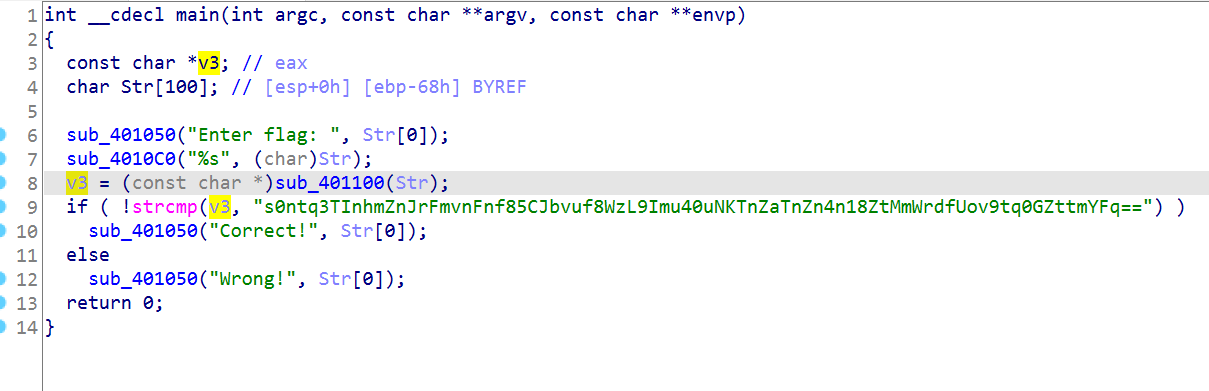
-----------

## **B4. Chal\_0 Base64**

Chạy chương trình nó yêu cầu nhập flag và sẽ check đúng sai cho mình.

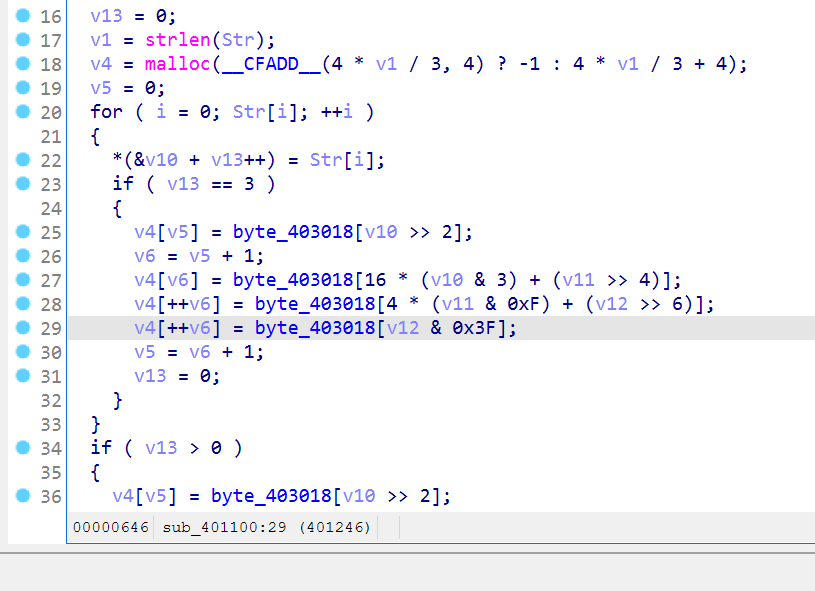
### **Solution:**

Mình tiếp tục dùng IDA để đọc mã giả của nó.



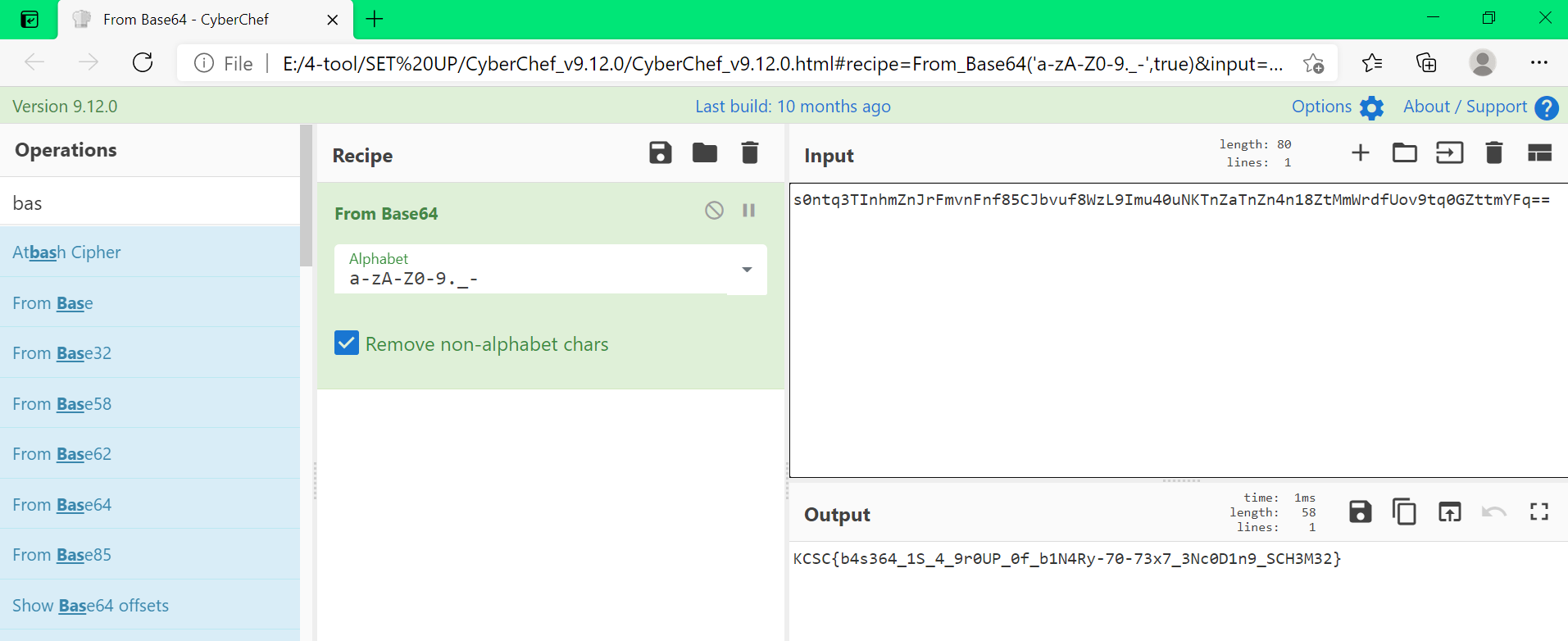
Input nhập vào sẽ được mã hóa base 64 qua hàm sub\_401100 và so sánh với chuỗi Str2 (s0ntq3TInhmZnJrFmvnFnf85CJbvuf8WzL9Imu40uNKTnZaTnZn4n18ZtMmWrdfUov9tq0GZttmYFq==)

Đây là đoạn code base64 trong sub\_401100



Check kỹ giá trị trong byte\_403018 thì ta thấy nó là từ a-z A-Z 0-9 + -, khác với giá trị map thông thường của base64 là A-Z a-z 0-9. (đoạn này sau khi có hint mình mới biết:<)

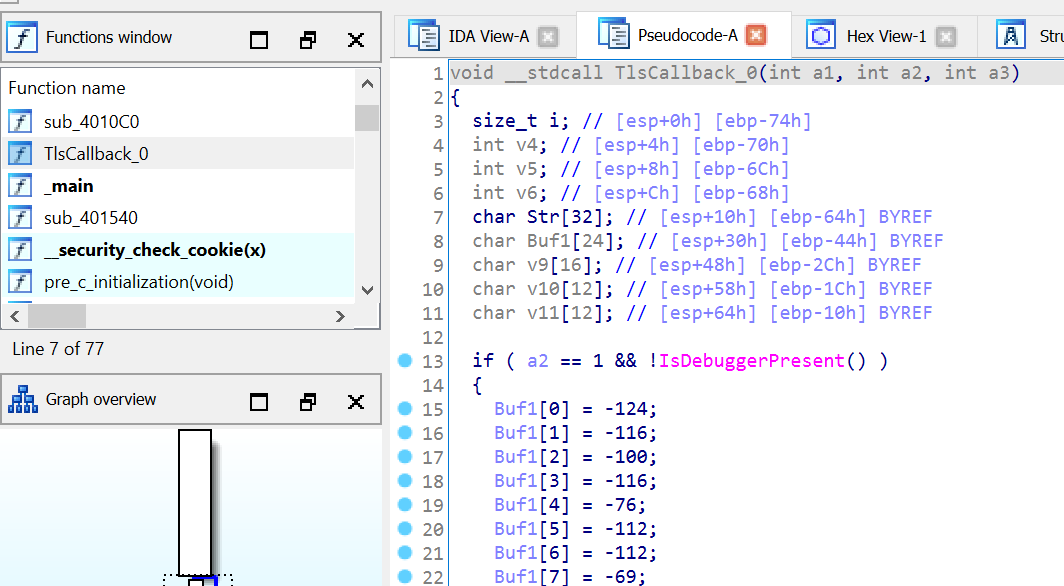
Giải: mình lên CyberChef và sửa đoạn From hex là a-zA-Z0-9.\_-, input thì nhập Str2 thì sẽ ra flag

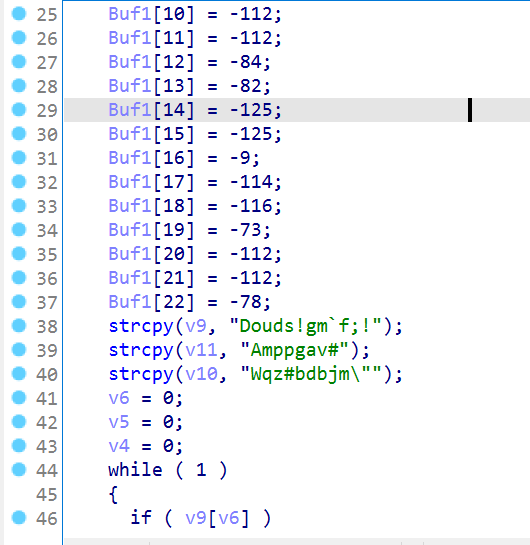


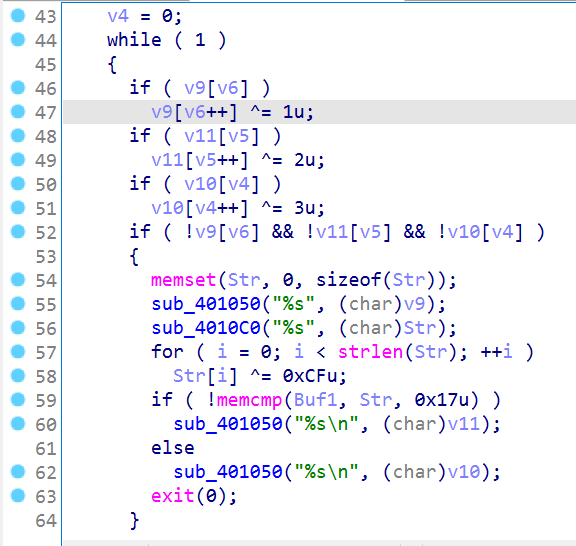
### **Flag: KCSC{b4s364\_1S\_4\_9r0UP\_0f\_b1N4Ry-70-73x7\_3Nc0D1n9\_SCH3M32}**

## **B5. Chal\_1**

Trong bài này cần để ý kỹ vì nó có gọi hàm Tlscallback\_0 trước khi vào hàm main chính. Trong hàm này thì lại có hàm check IsDebuggerPresent, nop qua hàm này thì chương trình sẽ chạy vào case đúng!

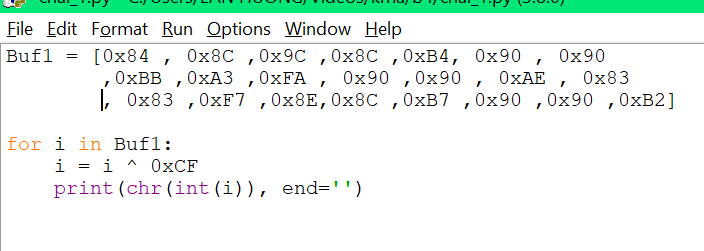






Giá trị của biến buf1 ^ 0xCF => input cần nhập!

code giải bằng python:



### **flag: KCSC{tl5\_\_aLL8ACx}**