南邮 逆向 部分题解

转载

木槿华年 于 2018-08-24 23:08:28 发布 2603 收藏 1
分类专栏： ctf 文章标签： 南邮逆向wp


ctf 专栏收录该内容
2 篇文章 0 订阅
订阅专栏

转自： https://blog.csdn.net/xiangshangbashaonian/article/details/78878876

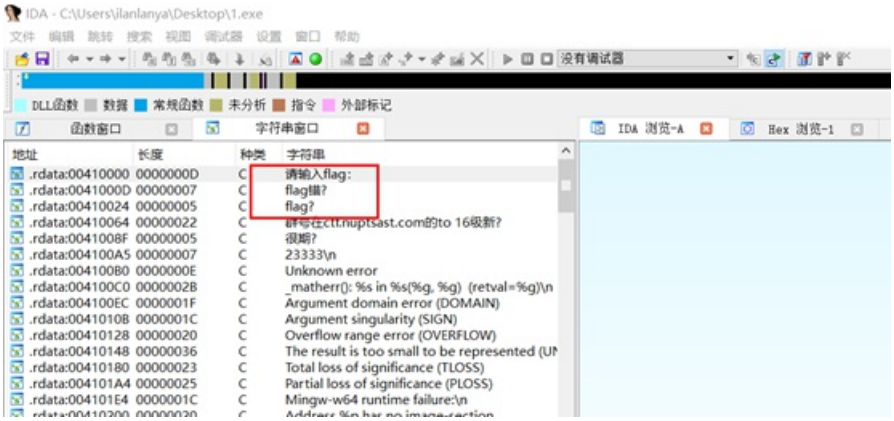
Hello,RE!

首先可以看到提示如下

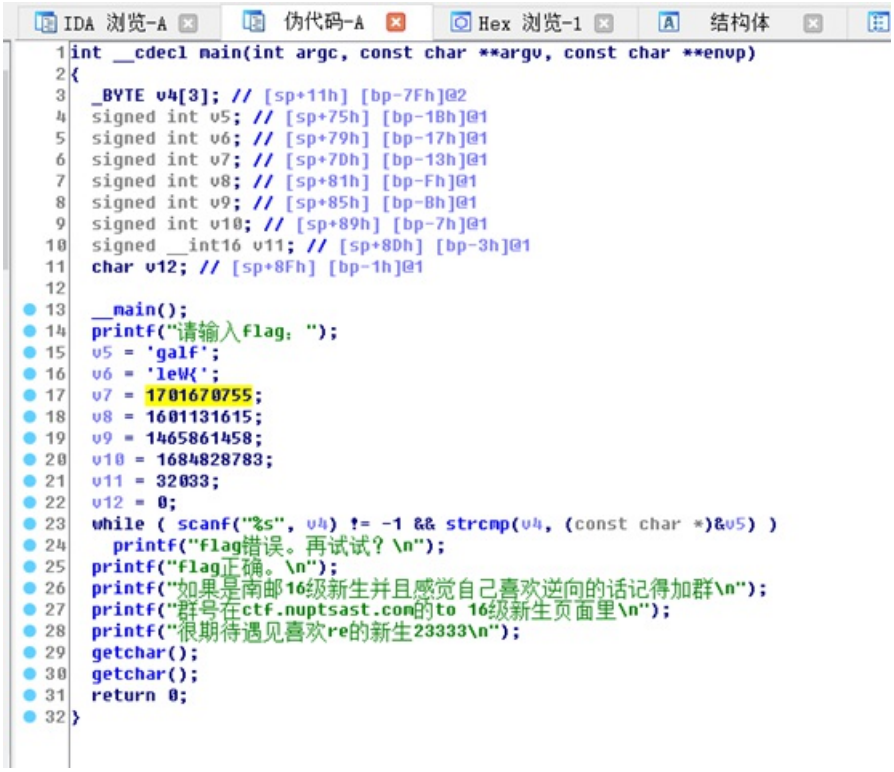
我还是查了一下 无壳

提示用IDA

那我们就载入 **shift+f12查找字符串**



在ida浏览A界面选定要查看的函数，**按f5看到了伪代码，** 于是**点击字符串按R转化为其实际值**




```
12
13  __main();
14  printf("请输入flag：");
15  v5 = 'galf';
16  v6 = 'leW{';
17  v7 = 'emoc';
18  v8 = '_oT_';
19  v9 = 'W_ER';
20  v10 = 'dlro';
21  v11 = '}!';
22  v12 = 0;
```

flag:

flag{Welcome_To_RE_World!}

ReadAsm2

ReadAsm2

150

读汇编是逆向基本功。

给出的文件是func函数的汇编
main函数如下
输出的结果即为flag，格式为 `flag{**********}` ，请连flag{}一起提交

编译环境为linux gcc x86-64
调用约定为System V AMD64 ABI
请不要利用汇编器，IDA等工具。。这里考的就是读汇编与推算汇编结果的能力

```
int main(int argc, char const *argv[])
{
  char input[] = {0x0,  0x67, 0x6e, 0x62, 0x63, 0x7e, 0x74, 0
x62, 0x69, 0x6d,
                  0x55, 0x6a, 0x7f, 0x60, 0x51, 0x66, 0x63, 0
x4e, 0x66, 0x7b,
                  0x71, 0x4a, 0x74, 0x76, 0x6b, 0x70, 0x79, 0
x66 , 0x1c};
  func(input, 28);
  printf("%s\n",input+1);
  return 0;
}
```

参考资料:
https://github.com/veficos/reverse-engineering-for-beginners
《汇编语言》王爽
《C反汇编与逆向分析技术揭秘》

2.asm

Key

SUBMIT

分析main函数 可以发现主要功能还是在于func函数中 于是我们仔细看func函数

```
00000000004004e6 <func>:                                ;4004e6一列表示该指令对应的虚拟内存地址  55一列为该指令对应的计算机指令
  4004e6: 55                   push   rbp                ;入栈，将寄存器的值压入调用 bp 栈中
  4004e7: 48 89 e5             mov    rbp,rsp            ;建立新栈帧，别掉函数栈帧栈底地址放入寄存器
  4004ea: 48 89 7d e8          mov    QWORD PTR [rbp-0x18],rdi   ;对应main中input[]     这时i=0                    //[rbp-0x18] = input[0]
  4004ee: 89 75 e4             mov    DWORD PTR [rbp-0x1c],esi   ;放入28                                          //[rbp-0x1c] = 28
  4004f1: c7 45 fc 01 00 00 00 mov    DWORD PTR [rbp-0x4],0x1    ;首先将0x1赋值给[rbp-0x4]                        //i = 1
  4004f8: eb 28                jmp    400522 <func+0x3c>         ;接着跳转到400522的位置                          //for(i=1;i<=28;i++)     下面以第一次过程为例
  4004fa: 8b 45 fc             mov    eax,DWORD PTR [rbp-0x4]    ;将[rbp-0x4]的值赋给eax寄存器                    //即令eax=i    =1
  4004fd: 48 63 d0             movsxd rdx,eax                    ;将eax的值带符号扩展，并传送至rdx中              //即令rdx=eax    =i =1
  400500: 48 8b 45 e8          mov    rax,QWORD PTR [rbp-0x18]   ;将rax的值给input[0]                             //即令rax = input[0]     =[rbp-0x18]
  400504: 48 01 d0             add    rax,rdx                    ;将rdx的值加上rax再赋值给rax                     //即  rax=input[1]  =i+input[0] =rdx+rax
  400507: 8b 55 fc             mov    edx,DWORD PTR [rbp-0x4]    ;将[rbp-0x4]的值给edx                            //即令edx=i    =1
  40050a: 48 63 ca             movsxd rcx,edx                    ;将edx的值带符号扩展，并传送至rcx中              //即令rcx=i    =1
  40050d: 48 8b 55 e8          mov    rdx,QWORD PTR [rbp-0x18]   ;将[rbp-0x18]的值给rdx                           //即令rdx=[rbp-0x18]      =input[0]
  400511: 48 01 ca             add    rdx,rcx                    ;将rcx的值加上rdx再赋值给rdx                     //即i++  rdx=input[1]
  400514: 0f b6 0a             movzx  ecx,BYTE PTR [rdx]         ;将rdx无符号扩展，并传送至ecx                    //即ecx=chr(rdx)     =chr(input[0])
  400517: 8b 55 fc             mov    edx,DWORD PTR [rbp-0x4]    ;edx = [rbp-0x4]                                 //即edx=i    =1
  40051a: 31 ca                xor    edx,ecx                    ;将edx与ecx异或                                  //i^input[0]
  40051c: 88 10                mov    BYTE PTR [rax],dl          ;rax = dl
  40051e: 83 45 fc 01          add    DWORD PTR [rbp-0x4],0x1    ;[rbp-0x4]++                                     //i++
  400522: 8b 45 fc             mov    eax,DWORD PTR [rbp-0x4]    ;将[rbp-0x4]的值赋给eax寄存器                    //eax = i
  400525: 3b 45 e4             cmp    eax,DWORD PTR [rbp-0x1c]   ;将[rbp-0x1c]中的值与eax值比较 第一次就是28
  400528: 7e d0                jle    4004fa <func+0x14>         ;如果<=  那么就跳到4004fa                        //if eax即i <=28  跳到4004fa继续循环
  40052a: 90                   nop                               ;空指令
  40052b: 5d                   pop    rbp                        ;出栈
  40052c: c3                   ret                               ;ret相当于return
                                                                                                                  http://blog.csdn.net/xiangshangbashaonian
```

00000000004004e6**<**func**>:**;4004e6一列表示该指令对应的虚拟内存地址 55一列为该指令对应的计算机指令

4004e6:55**push** rbp ;入栈，将寄存器的值压入调用 bp栈中

4004e7:4889 e5 **mov** rbp,rsp;建立新栈帧，别掉函数栈帧栈底地址放入寄存器

4004ea:48897d e8 **movQWORD**PTR**[**rbp-0x18**],**rdi;对应main中input[]这时i=0 //[rbp-0x18] = input[0]

4004ee:8975 e4 **movDWORD**PTR**[**rbp-0x1c**],esi**;放入28 //[rbp-0x1c] = 28

4004f1: c745 fc 01000000**movDWORD**PTR**[**rbp-0x4**],**0x1;首先将0x1赋值给[rbp-0x4] //i = 1

4004f8: eb28**jmp**400522**<**func**+**0x3c**>**;接着跳转到400522的位置 //for(i=1;i<=28;i++) 下面以第一次过程为例

4004fa:8b45 fc **moveax,DWORD**PTR**[**rbp-0x4**]**;将[rbp-0x4]的值赋给eax寄存器 //即令eax=i =1

4004fd:4863 d0 **movsxd** rdx,**eax**;将eax的值带符号扩展，并传送至rdx中 //即令rdx=eax =i =1

400500:488b45 e8 **mov** rax,**QWORD**PTR**[**rbp-0x18**]**;将rax的值给input[0] //即令rax = input[0] =[rbp-0x18]

400504:4801 d0 **add** rax,rdx;将rdx的值加上rax再赋值给rax //即 rax=input[1] =i+input[0] =rdx+rax

400507:8b55 fc **movedx,DWORD**PTR**[**rbp-0x4**]**;将[rbp-0x4]的值给edx //即令edx=i =1

40050a:4863 ca **movsxd** rcx,**edx**;将edx的值带符号扩展，并传送至rcx中 //即令rcx=i =1

40050d:488b55 e8 **mov** rdx,**QWORD**PTR**[**rbp-0x18**]**;将[rbp-0x18]的值给rdx //即令rdx=[rbp-0x18] =input[0]

400511:4801 ca **add** rdx,rcx;将rcx的值加上rdx再赋值给rdx //即i++ rdx=input[1]

400514:0f b6 0a**movzxecx,BYTE**PTR**[**rdx**]**;将rdx无符号扩展，并传送至ecx //即ecx=chr(rdx) =chr(input[0])

400517:8b55 fc **movedx,DWORD**PTR**[**rbp-0x4**]**;edx = [rbp-0x4] //即edx=i =1

40051a:31 ca **xoredx,ecx**;将edx与ecx异或 //i^input[0]

40051c:8810**movBYTE**PTR**[**rax**],dl**;rax = dl

40051e:8345 fc 01**addDWORD**PTR**[**rbp-0x4**],**0x1;[rbp-0x4]++ //i++

400522:8b45 fc **moveax,DWORD**PTR**[**rbp-0x4**]**;将[rbp-0x4]的值赋给eax寄存器 //eax = i

400525:3b45 e4 **cmpeax,DWORD**PTR**[**rbp-0x1c**]**;将[rbp-0x1c]中的值与eax值比较第一次就是28

400528:7e d0 **jle**4004fa**<**func**+**0x14**>**;如果<=那么就跳到4004fa //if eax即i <=28跳到4004fa继续循环

40052a:90**nop**;空指令

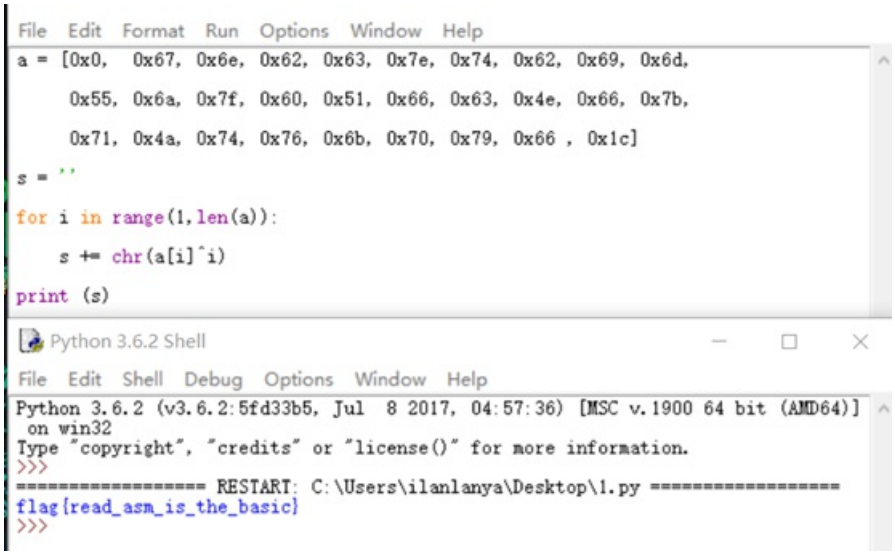
40052b:5d**pop** rbp ;出栈

40052c: c3**ret**;ret相当于return

分析可得Python3代码：

```
a = [0x0, 0x67, 0x6e, 0x62, 0x63, 0x7e, 0x74, 0x62, 0x69, 0x6d,
0x55, 0x6a, 0x7f, 0x60, 0x51, 0x66, 0x63, 0x4e, 0x66, 0x7b,
0x71, 0x4a, 0x74, 0x76, 0x6b, 0x70, 0x79, 0x66 , 0x1c]
s = ''
for i in range(1,len(a)):
s += chr(a[i]^i)
print (s)
```

运行结果如下：



相应的汇编知识：

转自：https://www.cnblogs.com/Chesky/p/nuptzj_re_writeup.html

> 入栈，将寄存器的值压入调用 bp 栈中
> 建立新栈帧，别掉函数栈帧栈底地址放入寄存器

实现

```
push   rbp
mov    rbp,rsp
```

寄存器类型：

```
ax(accumulator): 可用于存放函数返回值
bp(base pointer): 用于存放执行中的函数对应的栈帧的栈底地址
sp(stack poinger): 用于存放执行中的函数对应的栈帧的栈顶地址
ip(instruction pointer): 指向当前执行指令的下一条指令
```

前缀加上 r 表示 64 位， e 表示 32 位，使用时表示该寄存器存储 xx 位的数据。

- -word 表示字
  q 四字 d 双字
  dword qword

```
dword 2*16 =32 位
qword 4*16 = 64 位
```

PTR 指针（pointer）
没有寄存器名时， X ptr 指明内存单元的长度，X 在汇编指令中可以为word 或 byte 。

- 内存地址

```
[rbp-0x18]
```

- 涉及指令

```
1.movsxd 指令为扩展至零
将32位的寄存器和内存操作数符号扩展到64位的寄存器
2.逻辑异或运算指令 XOR
XOR OPRD1,OPRD2
实现两个操作数按位‘异或’(异为真,相同为假)运算,结果送至目的操作数中.
OPRD1<--OPRD1 XOR OPRD2
3.JLE
 小于等于时转移
```

flag:

flag{read_asm_is_the_basic}

MAZE

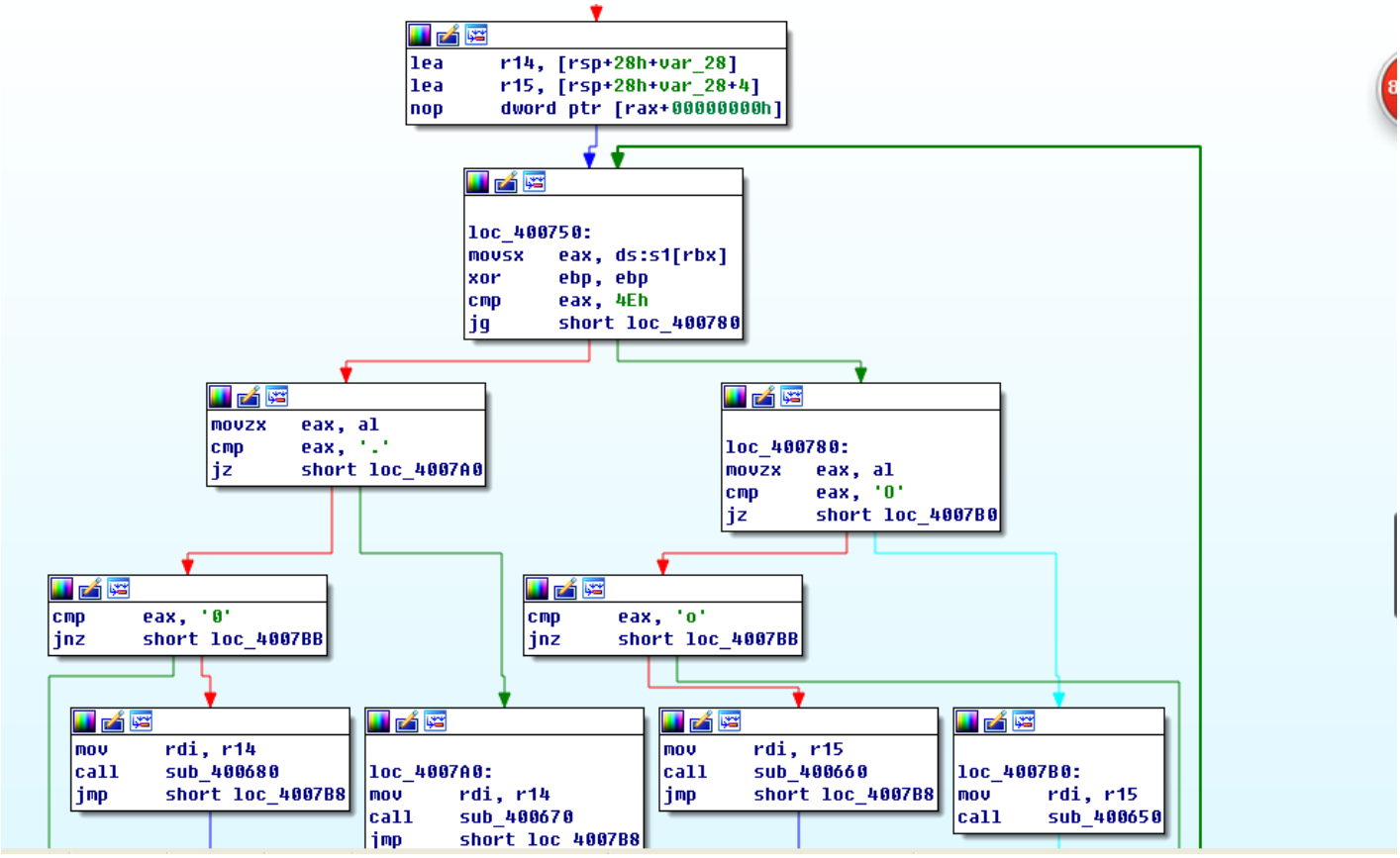
MAZE拖进winhex发现是elf文件，ida64打开，调出main函数，反编译：

```
__int64 __fastcall main(__int64 a1, char **a2, char **a3)
{
  signed __int64 v3; // rbx@4
  signed int v4; // eax@5
  bool v5; // bp@5
  bool v6; // al@8
  const char *v7; // rdi@19
  __int64 v9; // [sp+0h] [bp-28h]@1

  v9 = 0LL;
  puts("Input flag:");
  scanf("%s", &s1, 0LL);
  if ( strlen(&s1) != 24 || strncmp(&s1, "nctf{", 5uLL) || *(&byte_6010BF + 24) != 125 )
  {
LABEL_22:
    puts("Wrong flag!");
    exit(-1);
  }
  v3 = 5LL;
  if ( strlen(&s1) - 1 > 5 )
  {
    while ( 1 )
    {
      v4 = *(&s1 + v3);
      v5 = 0;
      if ( v4 > 78 )
      {
        v4 = (unsigned __int8)v4;
        if ( (unsigned __int8)v4 == 'O' )
        {
          v6 = sub_400650((_DWORD *)&v9 + 1);    //v6为v9的下一个字节
          goto LABEL_14;
        }
```

```
        if ( v4 == 'o' )
        {
          v6 = sub_400660((int *)&v9 + 1);    //v6为v9的下一个字节
          goto LABEL_14;
        }
      }
      else
      {
        v4 = (unsigned __int8)v4;
        if ( (unsigned __int8)v4 == '.' )
        {
          v6 = sub_400670(&v9);         //v6为v9的本字节
          goto LABEL_14;
        }
        if ( v4 == '0' )
        {
          v6 = sub_400680((int *)&v9);    //v6为v9
LABEL_14:
          v5 = v6;
          goto LABEL_15;
        }
      }
LABEL_15:
      if ( !(unsigned __int8)sub_400690((__int64)asc_601060, SHIDWORD(v9), v9) )
        goto LABEL_22;
      if ( ++v3 >= strlen(&s1) - 1 )
      {
        if ( v5 )
          break;
LABEL_20:
        v7 = "Wrong flag!";
        goto LABEL_21;
      }
    }
  }
  if ( *(&asc_601060[8 * (signed int)v9] + SHIDWORD(v9)) != 35 )          ///v9为行数
    goto LABEL_20;
  v7 = "Congratulations!";
LABEL_21:
  puts(v7);
  return 0LL;
}
```

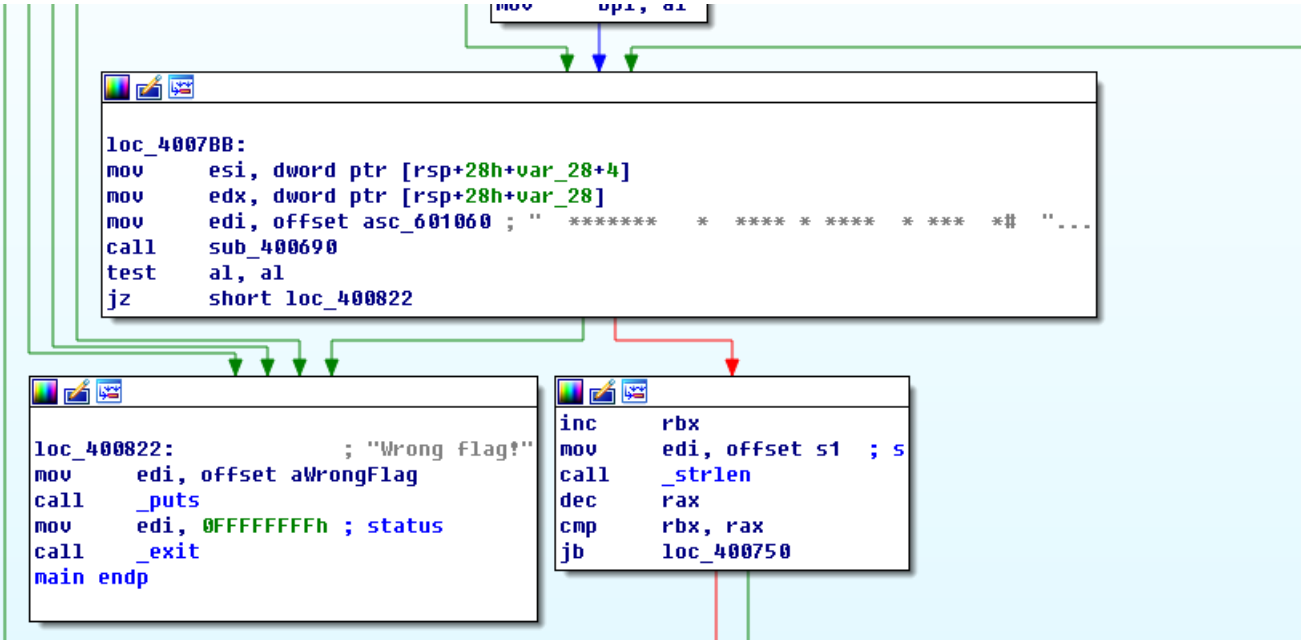
/SHIDWORD///查了一下IDA的宏定义 #define HIDWORD(x) (*((_DWORD*)&(x)+1))

太长了，简单看了一下，对输入的字符串限制了24的长度，保证以“nctf{”开头，“}”结尾。换成graph view：



可以看到他根据判断是否为'.'，'0'，'o'，'O'来决定进行什么操作

```
bool __fastcall sub_400650(_DWORD *a1)//(_DWORD *)&v9 + 1
{
//为O（0x4f）的时候

  int v1; // eax@1

  v1 = (*a1)--;
  return v1 > 0;
}

bool __fastcall sub_400660(int *a1)//(int *)&v9 + 1
{
///为o的时候
  int v1; // eax@1

  v1 = *a1 + 1;
  *a1 = v1;
  return v1 < 8;
}

bool __fastcall sub_400670(_DWORD *a1)//&v9
{
//为.的时候
  int v1; // eax@1

  v1 = (*a1)--;
  return v1 > 0;
}

bool __fastcall sub_400680(int *a1)//(int *)&v9
{
//为0(0x30)的时候
  int v1; // eax@1

  v1 = *a1 + 1;
  *a1 = v1;
  return v1 < 8;
}
```



判断O为左移一位，o为右移一位，.为上移一位，0为下移一位。根据下图，判断出偏移为601060的地方放着我们要比较的东西，他依次把数据交给寄存器EDI，调用函数：

```
sub_400690 proc near
movsxd  rax, esi
add     rax, rdi
movsxd  rcx, edx
movzx   eax, byte ptr [rax+rcx*8]
cmp     eax, 20h
setz    cl
cmp     eax, 23h
setz    al
or      al, cl
retn
sub_400690 endp
```

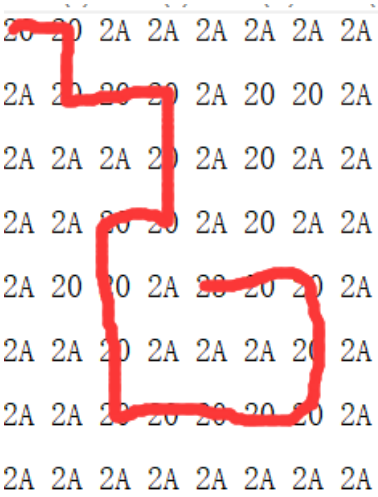
MAZE这个题目就是迷宫了，上面的函数说明了在将EDI中的数与20h和23h比较，不符合就GG，来看看我们需要走的那个迷宫，就是地址为0000000000601060的HEX：

```
0000000000601060  20 20 2A 2A 2A 2A 2A 2A  2A 20 20 20 2A 20 20 2A   ··******* ···* ··*
0000000000601070  2A 2A 2A 20 2A 20 2A 2A  2A 2A 20 20 2A 20 2A 2A   *** *· ***** ·· *·*
0000000000601080  2A 20 20 2A 23 20 20 2A  2A 2A 20 2A 2A 2A 20 2A   *··*#··*** *··· *·
0000000000601090  2A 2A 20 20 20 20 20 2A  2A 2A 2A 2A 2A 2A 2A 2A   **·····*********
00000000006010A0  00                                                ·
```

整理一下得到8X8的矩阵：

```
20 20 2A 2A 2A 2A 2A 2A
2A 20 20 20 2A 20 20 2A
2A 2A 2A 20 2A 20 2A 2A
2A 2A 20 20 2A 20 2A 2A
2A 20 20 2A 23 20 20 2A
2A 2A 20 2A 2A 2A 20 2A
2A 2A 20 20 20 20 20 2A
2A 2A 2A 2A 2A 2A 2A 2A
```

按照函数规则走一下



将走的线路换成上诉的四个字符，得到o0oo00O000oooo..OO即为flag

flag:

nctf{o0oo00O000oooo..OO}