源代码:

```
#include <windows.h>
#include <stdio.h>
#include <winternl.h>
char t[] = { 66, 0, 68, 70, 2, 2, 1, 72, 22, 12, 24, 27, 7, 58, 57, 7, 10, 14,
58, 57, 10, 13, 6, 88, 87, 9, 9, 9, 9, 21, 125,0 };
#pragma comment(linker, "/INCLUDE:__tls_used")
void NTAPI TLS_CALLBACK1(PVOID DLLHandle,DWORD Reason,PVOID Reserved) {
#ifdef _WIN64
    PPEB peb = (PPEB)__readgsqword(0x60);
#else
    PPEB peb = (PPEB)__readfsdword(0x30);
#endif
    if (IsDebuggerPresent()) {
        MessageBoxA(0i64, "Watch OUT! Hacker!", "TLS Callback", 0);
        ExitProcess(0xFFFFFFFF);
    }
    if (peb->BeingDebugged) {
        MessageBoxA(0i64, "Maybe you know something about anti-debug", "TLS
Callback",0);
        ExitProcess(0xFFFFFFFF);
    }
}
void NTAPI TLS_CALLBACK0(PVOID DLLHandle, DWORD Reason, PVOID Reserved) {
    t[8] = 8;
    t[9] = 67;
    t[10] = 93;
    t[11] = 94;
    t[12] = 71;
    t[13] = 28;
    t[14] = 89;
    t[15] = 95;
    t[16] = 87;
    t[17] = 102;
    t[18] = 61;
    t[19] = 81;
    t[20] = 85;
    t[21] = 86;
    t[22] = 66;
    t[23] = 65;
    t[24] = 108;
    t[25] = 50;
    t[26] = 89;
    t[27] = 5;
    t[28] = 95;
    t[29] = 19;
}
#pragma data_seg(".CRT$XLX")
PIMAGE_TLS_CALLBACK _tls_callback[] = { TLS_CALLBACK0 ,TLS_CALLBACK1,0 };
#pragma data_seg()
int main()
{
    printf("Please input your flag:");
    char s[32] = {0};
```

```
    scanf("%s", &s);
    for (int i = 0; i < 30; i++) {
        s[i] ^= s[i + 1];
    }
    if (!memcmp(s, t,31))printf("Good!The flag is your input");
    else {
        printf("Try harder!");
        exit(0);
    }
}
```

exp如下:

```
ans="}"
l =
[66,0,68,70,2,2,1,72,8,67,93,94,71,28,89,95,87,102,61,81,85,86,66,65,108,50,89,5
,95,19,125]
for i in range(len(l)-1,0,-1):
    ans+=chr(l[i]^l[i-1])
    l[i-1] = l[i]^l[i-1]
print(ans[::-1])
```

下面是详细版wp:

进入ida查看主函数逻辑，配合上文源代码食用更佳。

已知Buf1存储了用户的输入，经过逐位异或完成了加密，之后与用于校验的数组逐位比较，若相同即代表flag正确。

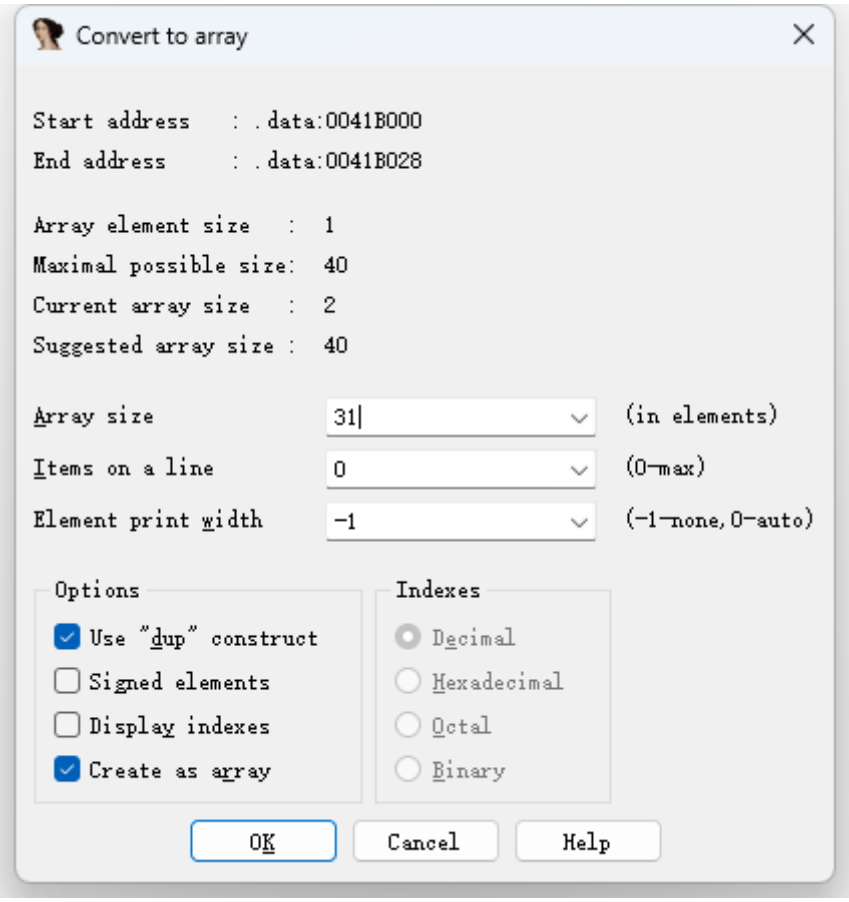
```
int __cdecl main_0(int argc, const char **argv, const char **envp)
{
  int i; // [esp+D0h] [ebp-34h]
  char Buf1[32]; // [esp+DCh] [ebp-28h] BYREF 根据下文发现栈上的Buf存储的是输入，使用y修改Buf1的类型为char Buf1[31]

  __CheckForDebuggerJustMyCode(&unk_41D0A3);    // 程序Canary保护
  puts("Please input your flag:");              // 根据输出猜测为puts函数，使用n键修改函数名
  memset(Buf1, 0, sizeof(Buf1));
  scanf("%s", Buf1);                            // 根据格式化字符串猜测这里是一个scanf函数，同样按n键重命名
  for ( i = 0; i < 30; ++i )
    Buf1[i] ^= Buf1[i + 1];
  if ( j_memcmp(Buf1, check, 31u) )             // 程序校验关键部分，如果进入该分支则程序退出，这里的校验为逐字节比较Buf1的内容和用于校验的数组的内容。使用y将用于校验的数组设定为char[31]
  {
    puts("Try harder!");
    exit(0);
  }
  puts("Good!The flag is your input");
  return 0;
}
```

双击check字段进入反汇编窗口，发现有很多的交叉引用..真的很多。

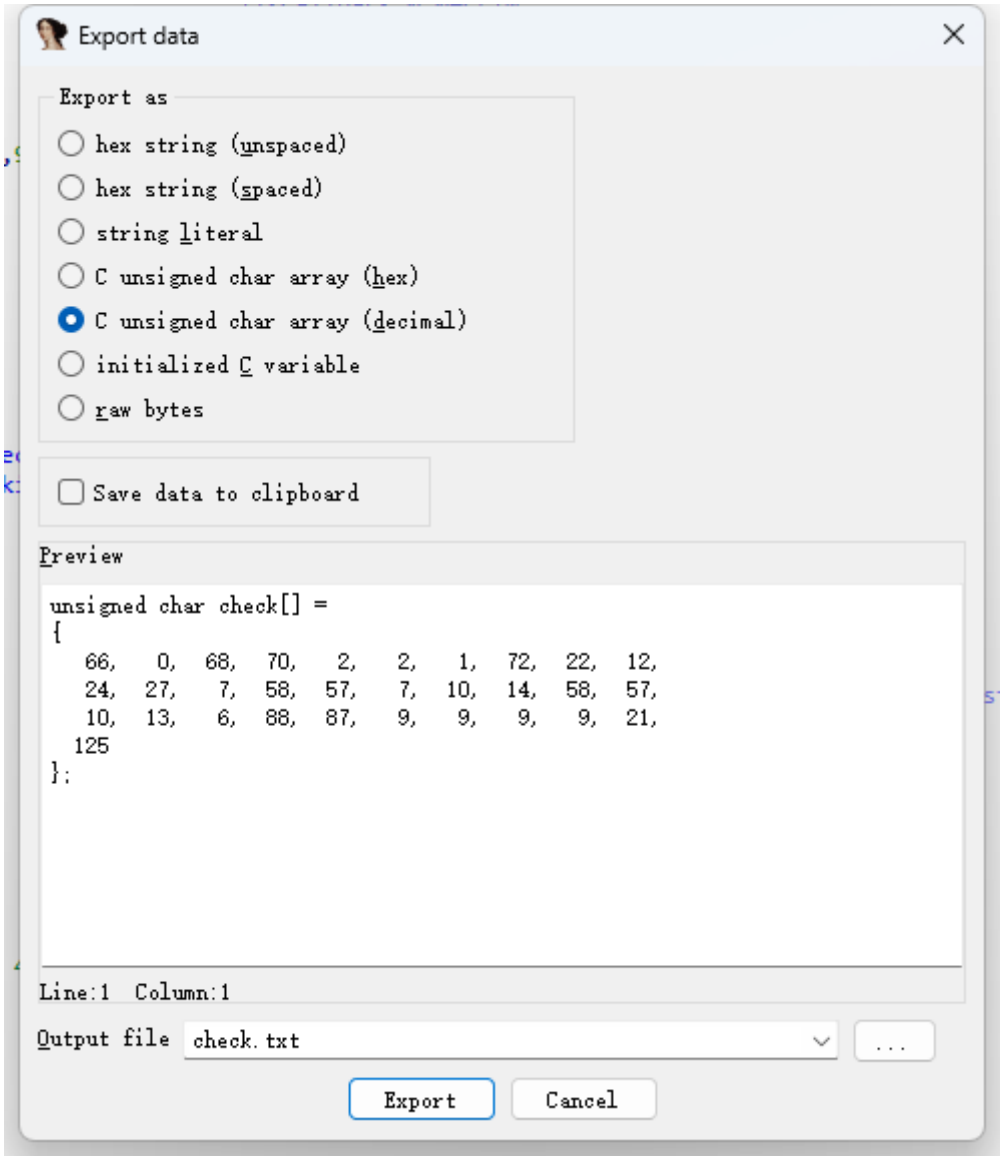
```
; char check[31]
check db 42h                    ; DATA XREF: TlsCallback_0_0+29↑w
                                ; TlsCallback_0_0+38↑w
                                ; TlsCallback_0_0+47↑w
                                ; TlsCallback_0_0+56↑w
                                ; TlsCallback_0_0+65↑w
                                ; TlsCallback_0_0+74↑w
                                ; TlsCallback_0_0+83↑w
                                ; TlsCallback_0_0+92↑w
                                ; TlsCallback_0_0+A1↑w
                                ; TlsCallback_0_0+B0↑w
                                ; TlsCallback_0_0+BF↑w
                                ; TlsCallback_0_0+CE↑w
                                ; TlsCallback_0_0+DD↑w
                                ; TlsCallback_0_0+EC↑w
                                ; TlsCallback_0_0+FB↑w ...
align 2
db  44h ; D
db  46h ; F
db    2
db    2
db    1
db  48h ; H
db  16h
db  0Ch
db  18h
db  1Bh
db    7
db  3Ah ; :
db  39h ; 9
db    7
db  0Ah
db  0Eh
db  3Ah ; :
db  39h ; 9
db  0Ah
db  0Dh
db    6
db  58h ; X
db  57h ; W
db    9
db    9
db    9
db    9
db  15h
db  7Dh ; }
db    0
db    0
db    0
db    0
db    0
db    0
db    0
db    0
db    0
```

先提取出check数组的数据：

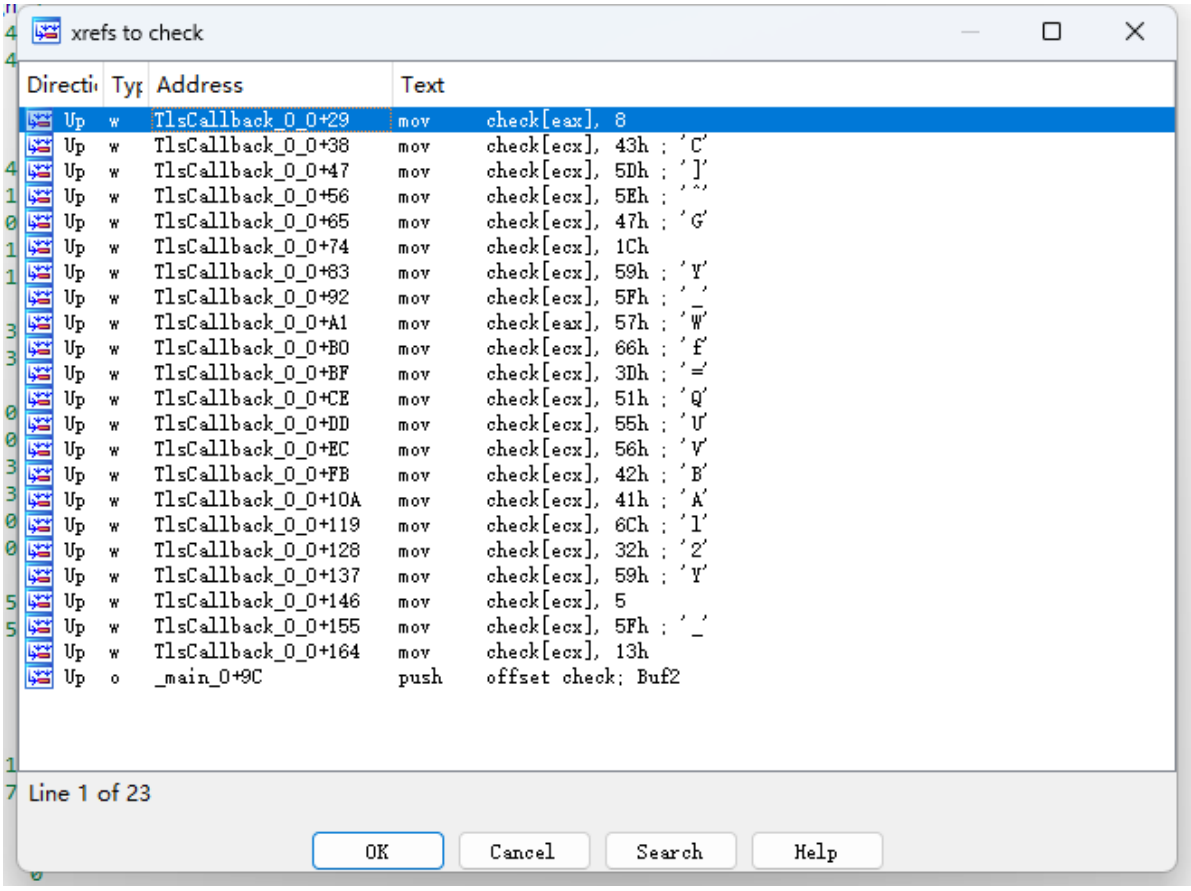
右键点击check，在弹出的菜单中选择array，让ida将其识别为array，根据刚才分析出的长度，填入array的长度为31


Convert to array
Start address : .data:0041B000
End address : .data:0041B028
Array element size : 1
Maximal possible size: 40
Current array size : 2
Suggested array size : 40
Array size
31
(in elements)
Items on a line
0
(0-max)
Element print width
-1
(-1-none, 0-auto)
Options
Use "dup" construct
Signed elements
Display indexes
Create as array
Indexes
Decimal
Hexadecimal
Octal
Binary
OK
Cancel
Help

按shift+e提取：



按x查看都在什么地方引用了这个check数组，可以发现在一个TlsCallback_0_0的函数中对其进行了修改。



双击进入该函数，发现无非就是修改了check数组的对应位置。

```
int __stdcall TlsCallback_0_0(int a1, int a2, int a3)
{
  int result; // eax

  __CheckForDebuggerJustMyCode(&unk_41D0A3);
  check[8] = 8;
  check[9] = 67;
  check[10] = 93;
  check[11] = 94;
  check[12] = 71;
  check[13] = 28;
  check[14] = 89;
  check[15] = 95;
  check[16] = 87;
  check[17] = 102;
  check[18] = 61;
  check[19] = 81;
  check[20] = 85;
  check[21] = 86;
  check[22] = 66;
  check[23] = 65;
  check[24] = 108;
  check[25] = 50;
  check[26] = 89;
  check[27] = 5;
  check[28] = 95;
  result = 1;
  check[29] = 19;
  return result;
}
```

最后的check数组为
[66,0,68,70,2,2,1,72,8,67,93,94,71,28,89,95,87,102,61,81,85,86,66,65,108,50,89,5,95,19,125]

编写exp即可。