# 南开大学

RE Challenge3

（汇编语言与逆向技术实验 9）

姓名：申宗尚

学号：2213924

专业：信息安全

## 一. 实验目的

1、 进一步熟悉静态反汇编工具 IDA Freeware；

2、 熟悉将反汇编代码进行反编译的过程；

3、 掌握对于反编译伪代码的逆向分析；

4、 运用熟悉的编程语言，实现简单的脚本编写

## 二. 实验环境

1、 ida

2、 task3.exe

3、 task4.exe

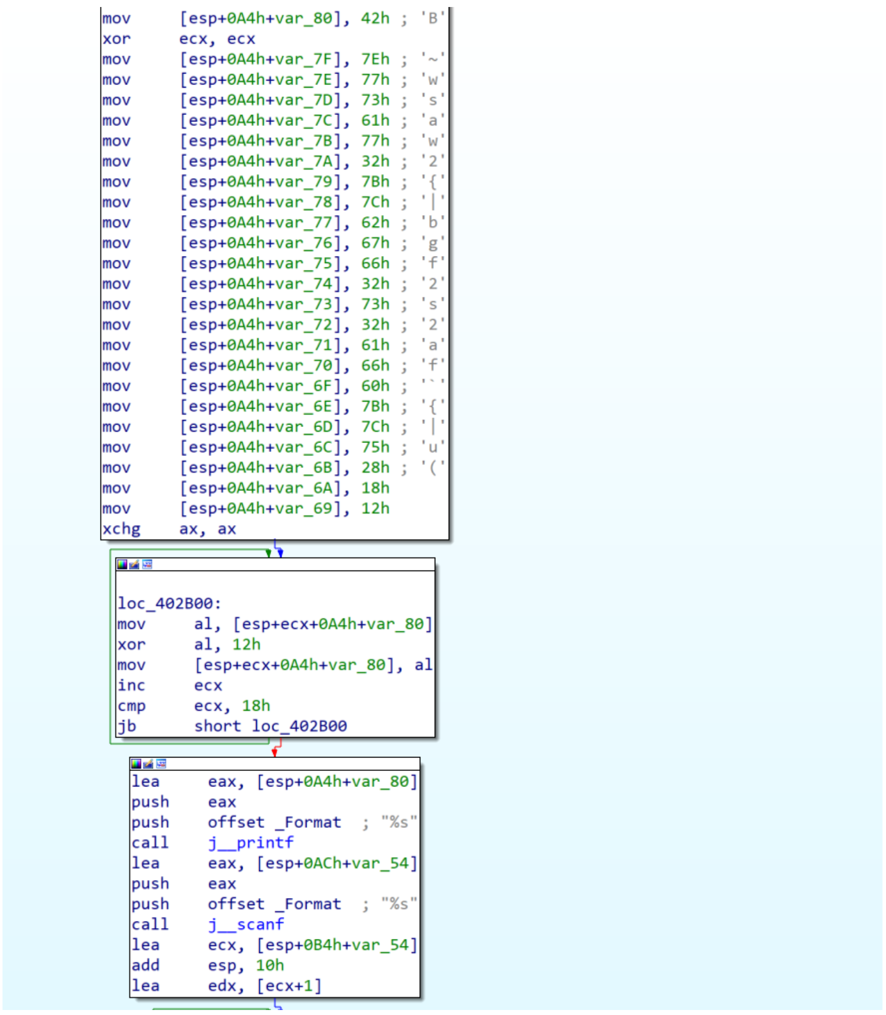
## 三. 实验原理

### (一) task3

1. 通过 IDA Freeware 得到 task3.exe 的反汇编代码，如图 1 和图 2 所示。

```
.text:00402A70                 sub     esp, 0A4h
.text:00402A76                 mov     eax, ___security_cookie
.text:00402A7B                 xor     eax, esp
.text:00402A7D                 mov     [esp+0A4h+var_4], eax
.text:00402A84                 mov     [esp+0A4h+var_80], 42h ; 'B'
.text:00402A89                 xor     ecx, ecx
.text:00402A8B                 mov     [esp+0A4h+var_7F], 7Eh ; '~'
.text:00402A90                 mov     [esp+0A4h+var_7E], 77h ; 'w'
.text:00402A95                 mov     [esp+0A4h+var_7D], 73h ; 's'
.text:00402A9A                 mov     [esp+0A4h+var_7C], 61h ; 'a'
.text:00402A9F                 mov     [esp+0A4h+var_7B], 77h ; 'w'
.text:00402AA4                 mov     [esp+0A4h+var_7A], 32h ; '2'
.text:00402AA9                 mov     [esp+0A4h+var_79], 7Bh ; '{'
.text:00402AAE                 mov     [esp+0A4h+var_78], 7Ch ; '|'
.text:00402AB3                 mov     [esp+0A4h+var_77], 62h ; 'b'
.text:00402AB8                 mov     [esp+0A4h+var_76], 67h ; 'g'
.text:00402ABD                 mov     [esp+0A4h+var_75], 66h ; 'f'
.text:00402AC2                 mov     [esp+0A4h+var_74], 32h ; '2'
.text:00402AC7                 mov     [esp+0A4h+var_73], 73h ; 's'
.text:00402ACC                 mov     [esp+0A4h+var_72], 32h ; '2'
.text:00402AD1                 mov     [esp+0A4h+var_71], 61h ; 'a'
.text:00402AD6                 mov     [esp+0A4h+var_70], 66h ; 'f'
.text:00402ADB                 mov     [esp+0A4h+var_6F], 60h ; '`'
.text:00402AE0                 mov     [esp+0A4h+var_6E], 7Bh ; '{'
.text:00402AE5                 mov     [esp+0A4h+var_6D], 7Ch ; '|'
.text:00402AEA                 mov     [esp+0A4h+var_6C], 75h ; 'u'
.text:00402AEF                 mov     [esp+0A4h+var_6B], 28h ; '('
.text:00402AF4                 mov     [esp+0A4h+var_6A], 18h
.text:00402AF9                 mov     [esp+0A4h+var_69], 12h
.text:00402AFE                 xchg    ax, ax
```

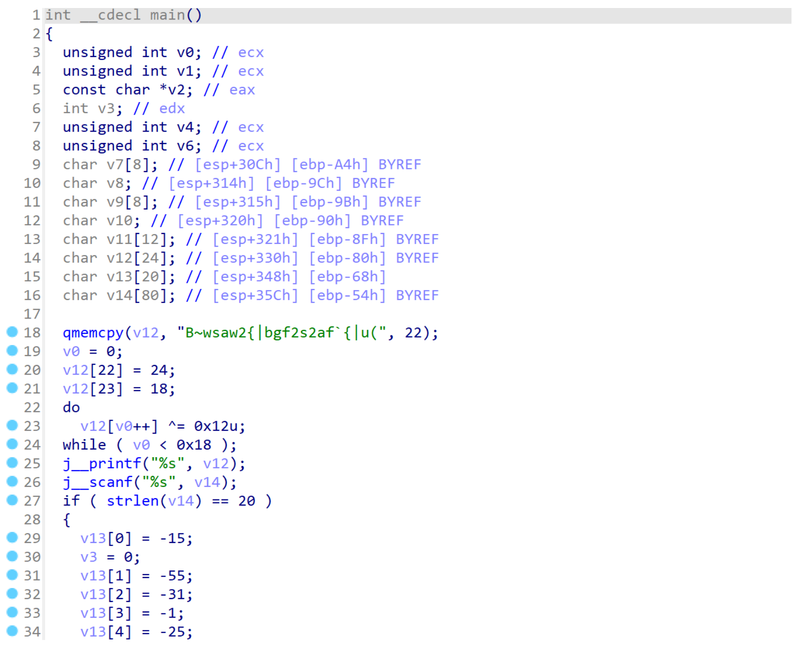
图 1 task3.exe 的反汇编代码



```
mov     [esp+0A4h+var_80], 42h ; 'B'
xor     ecx, ecx
mov     [esp+0A4h+var_7F], 7Eh ; '~'
mov     [esp+0A4h+var_7E], 77h ; 'w'
mov     [esp+0A4h+var_7D], 73h ; 's'
mov     [esp+0A4h+var_7C], 61h ; 'a'
mov     [esp+0A4h+var_7B], 77h ; 'w'
mov     [esp+0A4h+var_7A], 32h ; '2'
mov     [esp+0A4h+var_79], 7Bh ; '{'
mov     [esp+0A4h+var_78], 7Ch ; '|'
mov     [esp+0A4h+var_77], 62h ; 'b'
mov     [esp+0A4h+var_76], 67h ; 'g'
mov     [esp+0A4h+var_75], 66h ; 'f'
mov     [esp+0A4h+var_74], 32h ; '2'
mov     [esp+0A4h+var_73], 73h ; 's'
mov     [esp+0A4h+var_72], 32h ; '2'
mov     [esp+0A4h+var_71], 61h ; 'a'
mov     [esp+0A4h+var_70], 66h ; 'f'
mov     [esp+0A4h+var_6F], 60h ; '`'
mov     [esp+0A4h+var_6E], 7Bh ; '{'
mov     [esp+0A4h+var_6D], 7Ch ; '|'
mov     [esp+0A4h+var_6C], 75h ; 'u'
mov     [esp+0A4h+var_6B], 28h ; '('
mov     [esp+0A4h+var_6A], 18h
mov     [esp+0A4h+var_69], 12h
xchg    ax, ax

loc_402B00:
mov     al, [esp+ecx+0A4h+var_80]
xor     al, 12h
mov     [esp+ecx+0A4h+var_80], al
inc     ecx
cmp     ecx, 18h
jb      short loc_402B00

lea     eax, [esp+0A4h+var_80]
push    eax
push    offset _Format  ; "%s"
call    j__printf
lea     eax, [esp+0ACh+var_54]
push    eax
push    offset _Format  ; "%s"
call    j__scanf
lea     ecx, [esp+0B4h+var_54]
add     esp, 10h
lea     edx, [ecx+1]
```

图 2　task3.exe 反汇编代码的图形化显示

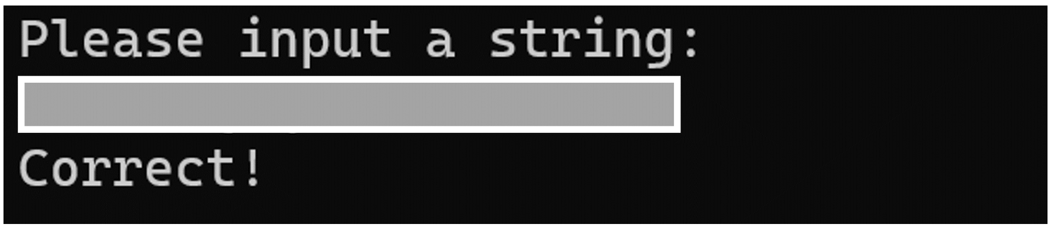
2. 使用 IDA 的反编译功能（F5 快捷键）得到伪代码，如图 3 所示。



```
int __cdecl main()
{
  unsigned int v0; // ecx
  unsigned int v1; // ecx
  const char *v2; // eax
  int v3; // edx
  unsigned int v4; // ecx
  unsigned int v6; // ecx
  char v7[8]; // [esp+30Ch] [ebp-A4h] BYREF
  char v8; // [esp+314h] [ebp-9Ch] BYREF
  char v9[8]; // [esp+315h] [ebp-9Bh] BYREF
  char v10; // [esp+320h] [ebp-90h] BYREF
  char v11[12]; // [esp+321h] [ebp-8Fh] BYREF
  char v12[24]; // [esp+330h] [ebp-80h] BYREF
  char v13[20]; // [esp+348h] [ebp-68h]
  char v14[80]; // [esp+35Ch] [ebp-54h] BYREF

  qmemcpy(v12, "B~wsaw2{|bgf2s2af`{|u(", 22);
  v0 = 0;
  v12[22] = 24;
  v12[23] = 18;
  do
    v12[v0++] ^= 0x12u;
  while ( v0 < 0x18 );
  j__printf("%s", v12);
  j__scanf("%s", v14);
  if ( strlen(v14) == 20 )
  {
    v13[0] = -15;
    v3 = 0;
    v13[1] = -55;
    v13[2] = -31;
    v13[3] = -1;
    v13[4] = -25;
```

图 3 task3.exe 的反编译伪代码

3. 通过对反汇编命令及反编译伪代码的分析，逆向推理出待输入字符串的计算公式
4. 使用熟悉的编程语言（C++、Java、Python 等）对待输入字符串进行计算，完成逆向分析挑战。

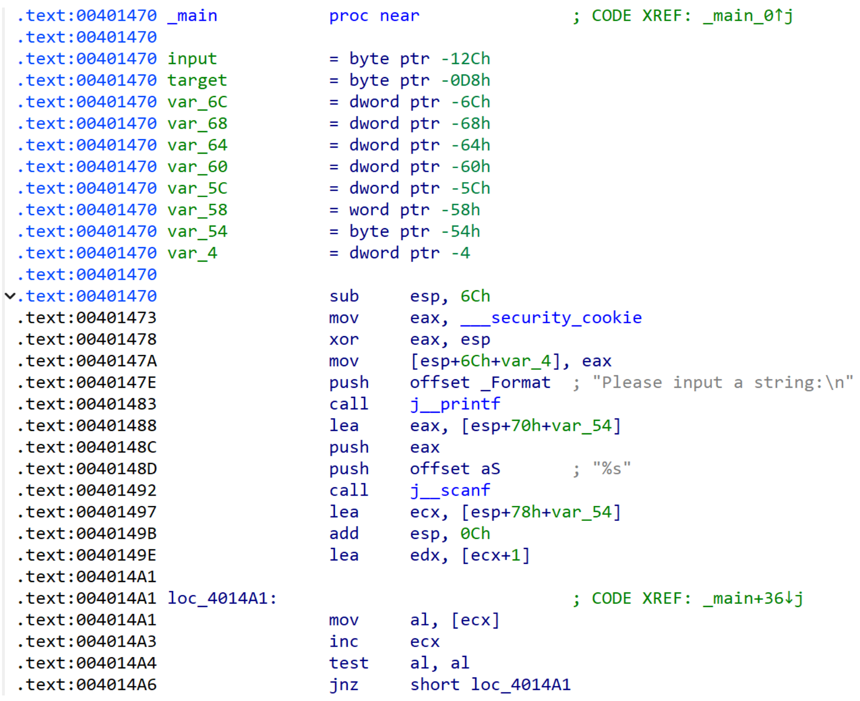


```
Please input a string:
[redacted]
Correct!
```

图 4 逆向分析，完成 task3 练习

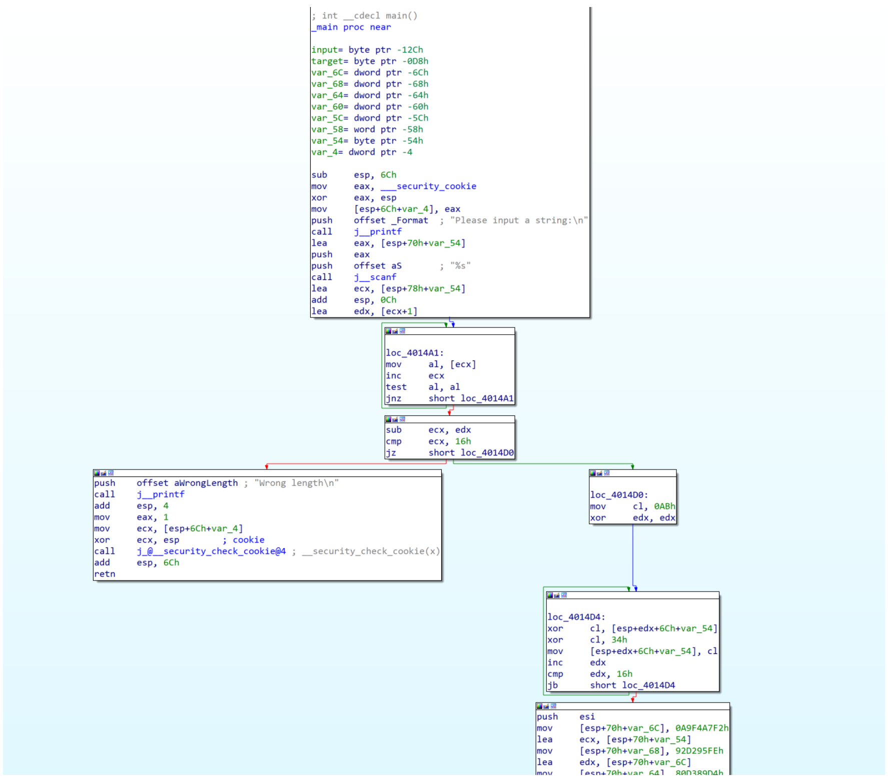
## (二) task4

1. 通过 IDA Freeware 得到 task4.exe 的反汇编代码，如图 5 和图 6 所示。



```
.text:00401470 _main           proc near               ; CODE XREF: _main_0↑j
.text:00401470
.text:00401470 input           = byte ptr -12Ch
.text:00401470 target          = byte ptr -0D8h
.text:00401470 var_6C          = dword ptr -6Ch
.text:00401470 var_68          = dword ptr -68h
.text:00401470 var_64          = dword ptr -64h
.text:00401470 var_60          = dword ptr -60h
.text:00401470 var_5C          = dword ptr -5Ch
.text:00401470 var_58          = word ptr -58h
.text:00401470 var_54          = byte ptr -54h
.text:00401470 var_4           = dword ptr -4
.text:00401470
.text:00401470                 sub     esp, 6Ch
.text:00401473                 mov     eax, ___security_cookie
.text:00401478                 xor     eax, esp
.text:0040147A                 mov     [esp+6Ch+var_4], eax
.text:0040147E                 push    offset _Format  ; "Please input a string:\n"
.text:00401483                 call    j__printf
.text:00401488                 lea     eax, [esp+70h+var_54]
.text:0040148C                 push    eax
.text:0040148D                 push    offset aS       ; "%s"
.text:00401492                 call    j__scanf
.text:00401497                 lea     ecx, [esp+78h+var_54]
.text:0040149B                 add     esp, 0Ch
.text:0040149E                 lea     edx, [ecx+1]
.text:004014A1
.text:004014A1 loc_4014A1:                             ; CODE XREF: _main+36↓j
.text:004014A1                 mov     al, [ecx]
.text:004014A3                 inc     ecx
.text:004014A4                 test    al, al
.text:004014A6                 jnz     short loc_4014A1
```

图 5 task4.exe 的反汇编代码



```
; int __cdecl main()
_main proc near

input= byte ptr -12Ch
target= byte ptr -0D8h
var_6C= dword ptr -6Ch
var_68= dword ptr -68h
var_64= dword ptr -64h
var_60= dword ptr -60h
var_5C= dword ptr -5Ch
var_58= word ptr -58h
var_54= byte ptr -54h
var_4= dword ptr -4

sub     esp, 6Ch
mov     eax, ___security_cookie
xor     eax, esp
mov     [esp+6Ch+var_4], eax
push    offset _Format  ; "Please input a string:\n"
call    j__printf
lea     eax, [esp+70h+var_54]
push    eax
push    offset aS       ; "%s"
call    j__scanf
lea     ecx, [esp+78h+var_54]
add     esp, 0Ch
lea     edx, [ecx+1]
```

```
loc_4014A1:
mov     al, [ecx]
inc     ecx
test    al, al
jnz     short loc_4014A1
```

```
sub     ecx, edx
cmp     ecx, 16h
jz      short loc_4014D0
```

```
push    offset aWrongLength ; "Wrong length\n"
call    j__printf
add     esp, 4
mov     eax, 1
mov     ecx, [esp+6Ch+var_4]
xor     ecx, esp        ; cookie
call    j_@__security_check_cookie@4 ; __security_check_cookie(x)
add     esp, 6Ch
retn
```

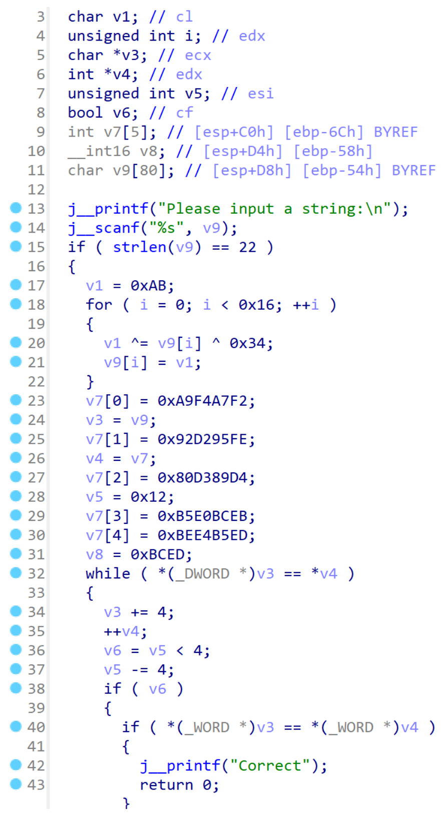
```
loc_4014D0:
mov     cl, 0ABh
xor     edx, edx
```

```
loc_4014D4:
xor     cl, [esp+edx+6Ch+var_54]
xor     cl, 34h
mov     [esp+edx+6Ch+var_54], cl
inc     edx
cmp     edx, 16h
jb      short loc_4014D4
```

```
push    esi
mov     [esp+70h+var_6C], 0A9F4A7F2h
lea     ecx, [esp+70h+var_54]
mov     [esp+70h+var_68], 92D295FEh
lea     edx, [esp+70h+var_6C]
```

图 6 task4.exe 反汇编代码的图形化显示

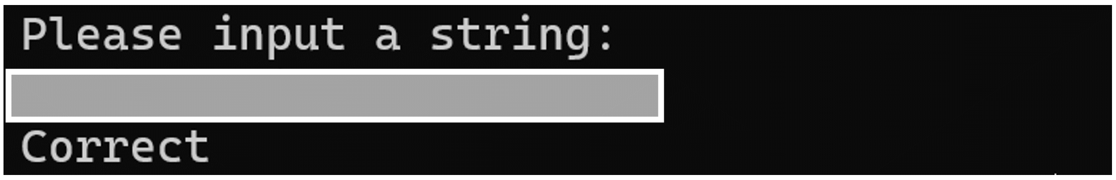
2. 使用 IDA 的反编译功能（F5 快捷键）得到伪代码，如图 7 所示。右键点击数字对象可实现数制转换。



```
char v1; // cl
unsigned int i; // edx
char *v3; // ecx
int *v4; // edx
unsigned int v5; // esi
bool v6; // cf
int v7[5]; // [esp+C0h] [ebp-6Ch] BYREF
__int16 v8; // [esp+D4h] [ebp-58h]
char v9[80]; // [esp+D8h] [ebp-54h] BYREF

j__printf("Please input a string:\n");
j__scanf("%s", v9);
if ( strlen(v9) == 22 )
{
  v1 = 0xAB;
  for ( i = 0; i < 0x16; ++i )
  {
    v1 ^= v9[i] ^ 0x34;
    v9[i] = v1;
  }
  v7[0] = 0xA9F4A7F2;
  v3 = v9;
  v7[1] = 0x92D295FE;
  v4 = v7;
  v7[2] = 0x80D389D4;
  v5 = 0x12;
  v7[3] = 0xB5E0BCEB;
  v7[4] = 0xBEE4B5ED;
  v8 = 0xBCED;
  while ( *(_DWORD *)v3 == *v4 )
  {
    v3 += 4;
    ++v4;
    v6 = v5 < 4;
    v5 -= 4;
    if ( v6 )
    {
      if ( *(_WORD *)v3 == *(_WORD *)v4 )
      {
        j__printf("Correct");
        return 0;
      }
```

图 7 task4.exe 的反编译伪代码

3. 通过对反汇编命令及反编译伪代码的分析，逆向推理出待输入字符串的计算公式
4. 使用熟悉的编程语言（C++、Java、Python 等）对待输入字符串进行计算，完成逆向分析挑战。



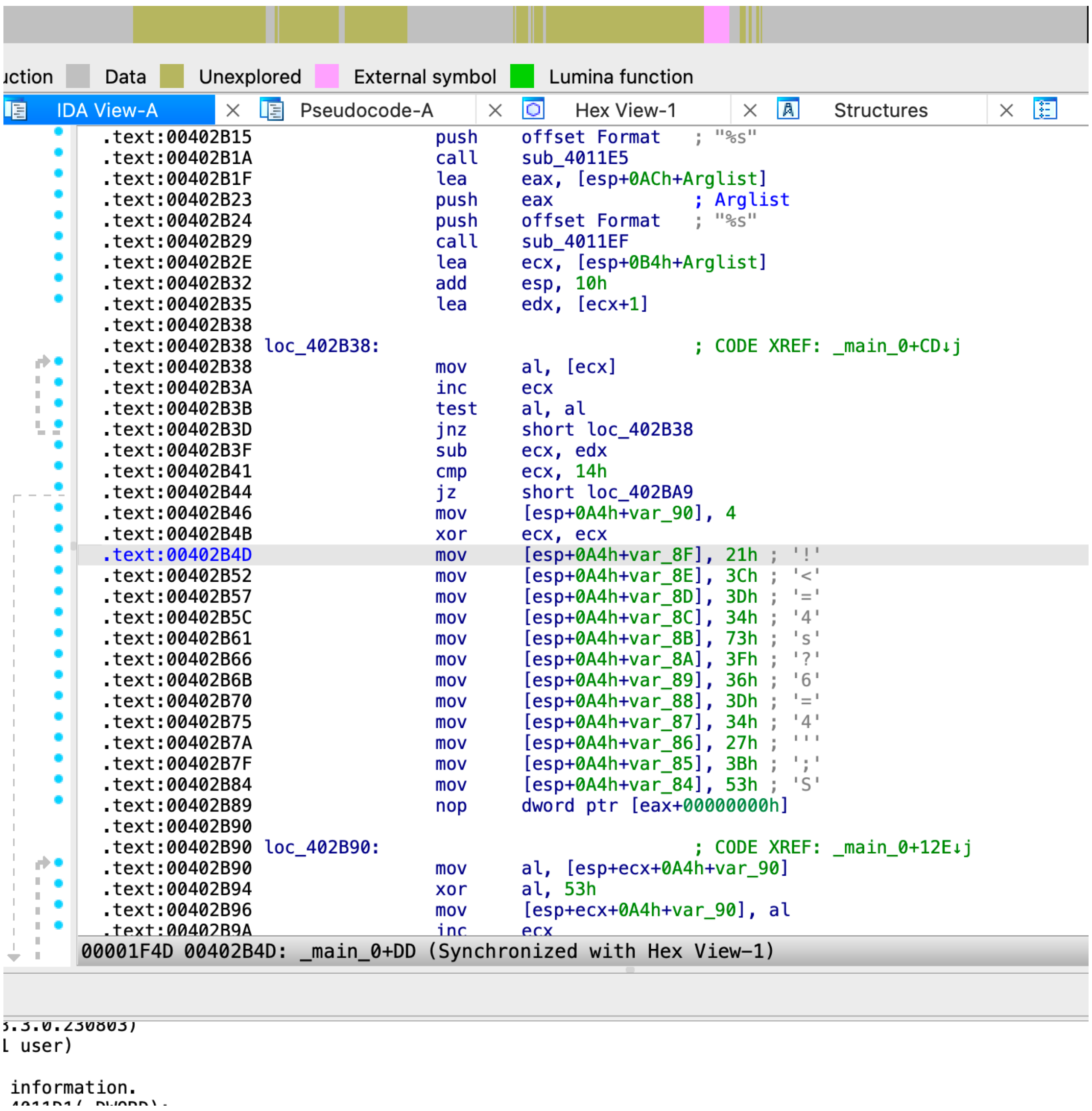
```
Please input a string:
[redacted]
Correct
```

图 8 逆向分析，完成 task4 练习

## 四. 实验内容

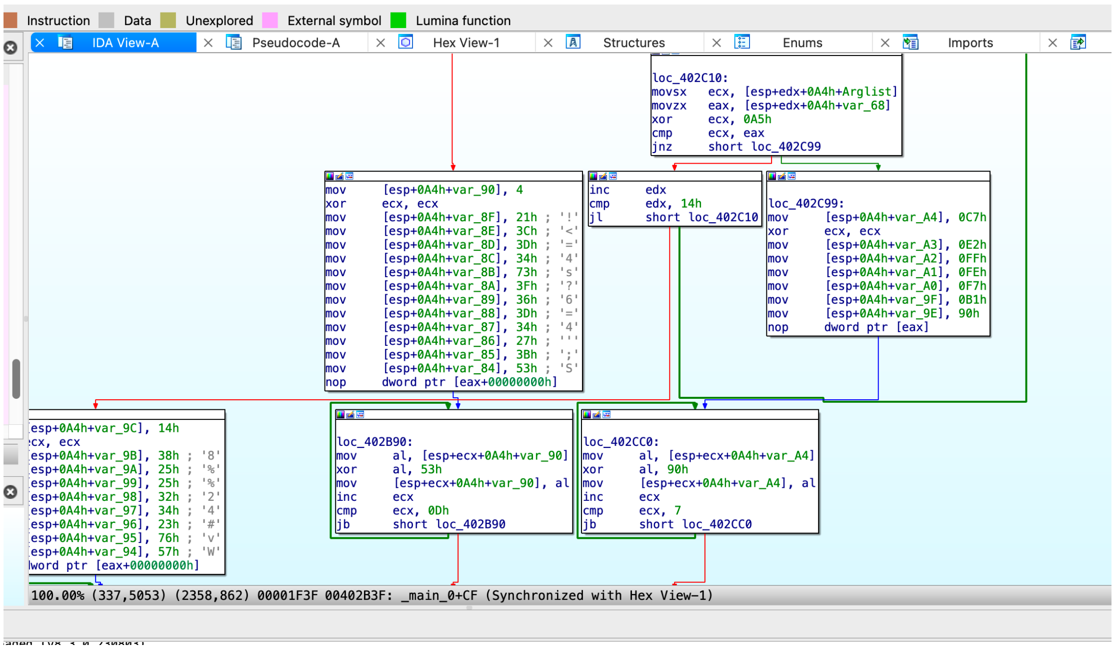
### （1）Task3

如图，将 task3.exe 导入 ida64 Freeware 中，可以得到二进制代码的反汇编代码



```
.text:00402B15                 push    offset Format   ; "%s"
.text:00402B1A                 call    sub_4011E5
.text:00402B1F                 lea     eax, [esp+0ACh+Arglist]
.text:00402B23                 push    eax             ; Arglist
.text:00402B24                 push    offset Format   ; "%s"
.text:00402B29                 call    sub_4011EF
.text:00402B2E                 lea     ecx, [esp+0B4h+Arglist]
.text:00402B32                 add     esp, 10h
.text:00402B35                 lea     edx, [ecx+1]
.text:00402B38
.text:00402B38 loc_402B38:                             ; CODE XREF: _main_0+CD↓j
.text:00402B38                 mov     al, [ecx]
.text:00402B3A                 inc     ecx
.text:00402B3B                 test    al, al
.text:00402B3D                 jnz     short loc_402B38
.text:00402B3F                 sub     ecx, edx
.text:00402B41                 cmp     ecx, 14h
.text:00402B44                 jz      short loc_402BA9
.text:00402B46                 mov     [esp+0A4h+var_90], 4
.text:00402B4B                 xor     ecx, ecx
.text:00402B4D                 mov     [esp+0A4h+var_8F], 21h ; '!'
.text:00402B52                 mov     [esp+0A4h+var_8E], 3Ch ; '<'
.text:00402B57                 mov     [esp+0A4h+var_8D], 3Dh ; '='
.text:00402B5C                 mov     [esp+0A4h+var_8C], 34h ; '4'
.text:00402B61                 mov     [esp+0A4h+var_8B], 73h ; 's'
.text:00402B66                 mov     [esp+0A4h+var_8A], 3Fh ; '?'
.text:00402B6B                 mov     [esp+0A4h+var_89], 36h ; '6'
.text:00402B70                 mov     [esp+0A4h+var_88], 3Dh ; '='
.text:00402B75                 mov     [esp+0A4h+var_87], 34h ; '4'
.text:00402B7A                 mov     [esp+0A4h+var_86], 27h ; '''
.text:00402B7F                 mov     [esp+0A4h+var_85], 3Bh ; ';'
.text:00402B84                 mov     [esp+0A4h+var_84], 53h ; 'S'
.text:00402B89                 nop     dword ptr [eax+00000000h]
.text:00402B90
.text:00402B90 loc_402B90:                             ; CODE XREF: _main_0+12E↓j
.text:00402B90                 mov     al, [esp+ecx+0A4h+var_90]
.text:00402B94                 xor     al, 53h
.text:00402B96                 mov     [esp+ecx+0A4h+var_90], al
.text:00402B9A                 inc     ecx
00001F4D 00402B4D: _main_0+DD (Synchronized with Hex View-1)
```

```
3.3.0.230803)
l user)

 information.
```



```
loc_402C10:
movsx   ecx, [esp+edx+0A4h+Arglist]
movzx   eax, [esp+edx+0A4h+var_68]
xor     ecx, 0A5h
cmp     ecx, eax
jnz     short loc_402C99
```

```
mov     [esp+0A4h+var_90], 4
xor     ecx, ecx
mov     [esp+0A4h+var_8F], 21h ; '!'
mov     [esp+0A4h+var_8E], 3Ch ; '<'
mov     [esp+0A4h+var_8D], 3Dh ; '='
mov     [esp+0A4h+var_8C], 34h ; '4'
mov     [esp+0A4h+var_8B], 73h ; 's'
mov     [esp+0A4h+var_8A], 3Fh ; '?'
mov     [esp+0A4h+var_89], 36h ; '6'
mov     [esp+0A4h+var_88], 3Dh ; '='
mov     [esp+0A4h+var_87], 34h ; '4'
mov     [esp+0A4h+var_86], 27h ; '''
mov     [esp+0A4h+var_85], 3Bh ; ';'
mov     [esp+0A4h+var_84], 53h ; 'S'
nop     dword ptr [eax+00000000h]
```

```
inc     edx
cmp     edx, 14h
jl      short loc_402C10
```

```
loc_402C99:
mov     [esp+0A4h+var_A4], 0C7h
xor     ecx, ecx
mov     [esp+0A4h+var_A3], 0E2h
mov     [esp+0A4h+var_A2], 0FFh
mov     [esp+0A4h+var_A1], 0FEh
mov     [esp+0A4h+var_A0], 0F7h
mov     [esp+0A4h+var_9F], 0B1h
mov     [esp+0A4h+var_9E], 90h
nop     dword ptr [eax]
```

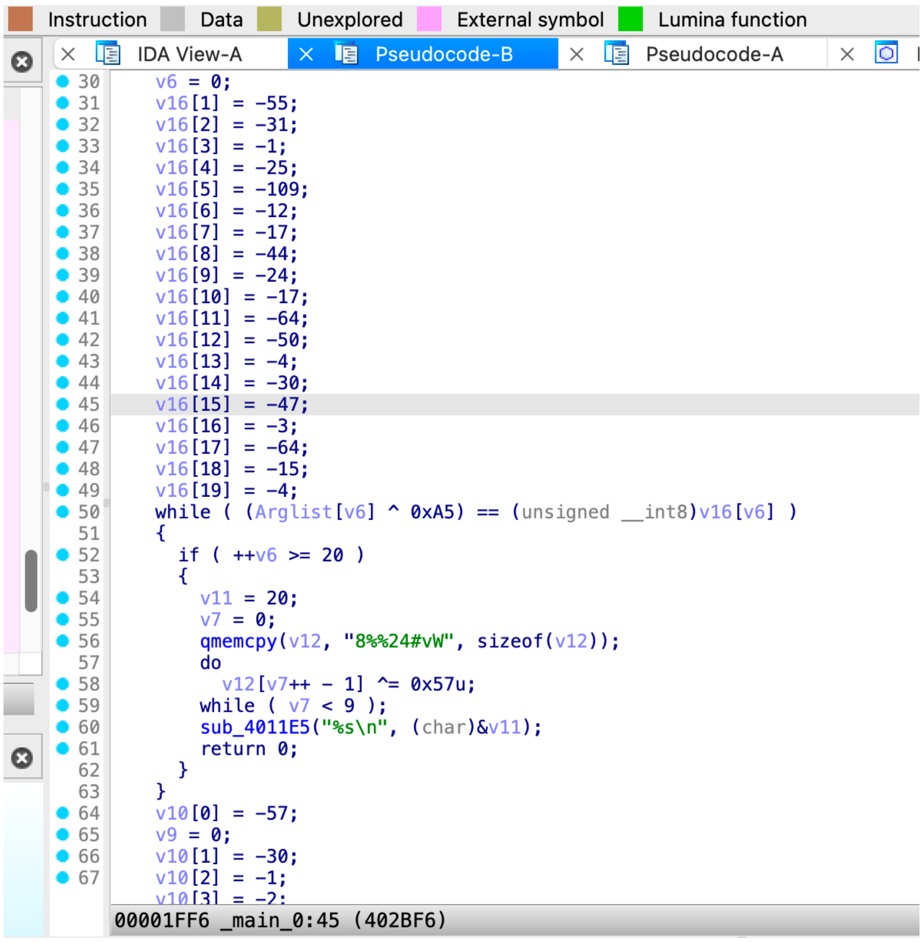
```
esp+0A4h+var_9C], 14h
cx, ecx
esp+0A4h+var_9B], 38h ; '8'
esp+0A4h+var_9A], 25h ; '%'
esp+0A4h+var_99], 25h ; '%'
esp+0A4h+var_98], 32h ; '2'
esp+0A4h+var_97], 34h ; '4'
esp+0A4h+var_96], 23h ; '#'
esp+0A4h+var_95], 76h ; 'v'
esp+0A4h+var_94], 57h ; 'W'
word ptr [eax+00000000h]
```

```
loc_402B90:
mov     al, [esp+ecx+0A4h+var_90]
xor     al, 53h
mov     [esp+ecx+0A4h+var_90], al
inc     ecx
cmp     ecx, 0Dh
jb      short loc_402B90
```

```
loc_402CC0:
mov     al, [esp+ecx+0A4h+var_A4]
xor     al, 90h
mov     [esp+ecx+0A4h+var_A4], al
inc     ecx
cmp     ecx, 7
jb      short loc_402CC0
```

```
100.00% (337,5053) (2358,862) 00001F3F 00402B3F: _main_0+CF (Synchronized with Hex View-1)
```

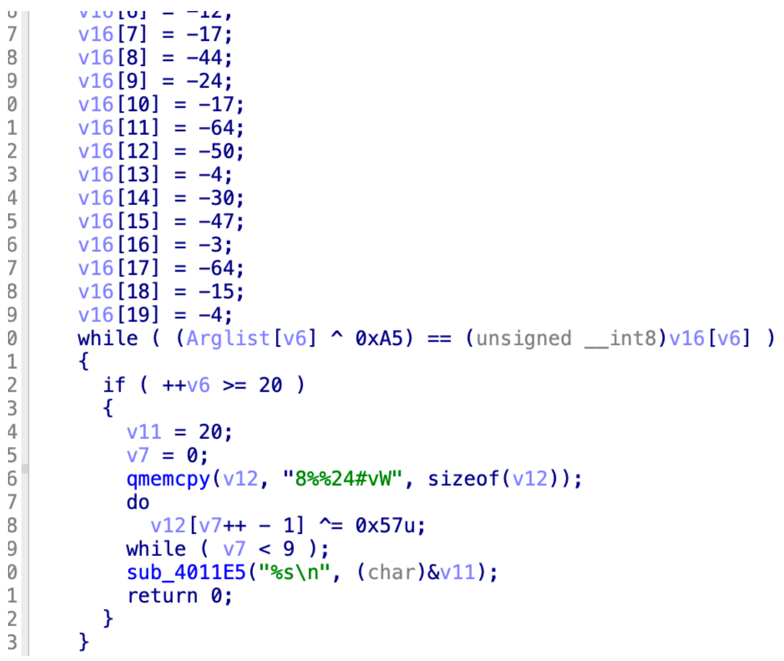
通过 ida 的反编译功能，得到伪代码：



```
v6 = 0;
v16[1] = -55;
v16[2] = -31;
v16[3] = -1;
v16[4] = -25;
v16[5] = -109;
v16[6] = -12;
v16[7] = -17;
v16[8] = -44;
v16[9] = -24;
v16[10] = -17;
v16[11] = -64;
v16[12] = -50;
v16[13] = -4;
v16[14] = -30;
v16[15] = -47;
v16[16] = -3;
v16[17] = -64;
v16[18] = -15;
v16[19] = -4;
while ( (Arglist[v6] ^ 0xA5) == (unsigned __int8)v16[v6] )
{
  if ( ++v6 >= 20 )
  {
    v11 = 20;
    v7 = 0;
    qmemcpy(v12, "8%%24#vW", sizeof(v12));
    do
      v12[v7++ - 1] ^= 0x57u;
    while ( v7 < 9 );
    sub_4011E5("%s\n", (char)&v11);
    return 0;
  }
}
v10[0] = -57;
v9 = 0;
v10[1] = -30;
v10[2] = -1;
v10[3] = -2;
```

```
00001FF6 _main_0:45 (402BF6)
```

从而，我们可以通过对该反汇编代码的计算过程、数据结构、条件判断、分支结构等信息进行分析，逆向推理出程序的正确输入数据，完成逆向分析挑战。

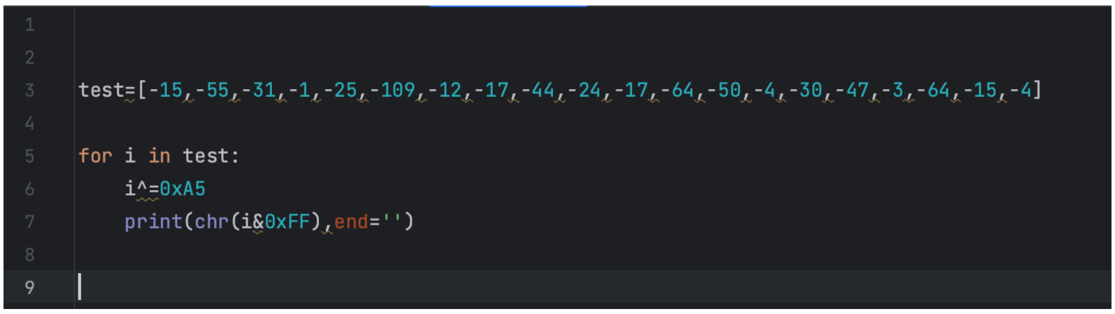


```
v16[6] = -12;
v16[7] = -17;
v16[8] = -44;
v16[9] = -24;
v16[10] = -17;
v16[11] = -64;
v16[12] = -50;
v16[13] = -4;
v16[14] = -30;
v16[15] = -47;
v16[16] = -3;
v16[17] = -64;
v16[18] = -15;
v16[19] = -4;
while ( (Arglist[v6] ^ 0xA5) == (unsigned __int8)v16[v6] )
{
  if ( ++v6 >= 20 )
  {
    v11 = 20;
    v7 = 0;
    qmemcpy(v12, "8%%24#vW", sizeof(v12));
    do
      v12[v7++ - 1] ^= 0x57u;
    while ( v7 < 9 );
    sub_4011E5("%s\n", (char)&v11);
    return 0;
  }
}
```

由这段代码可以推出，该程序将输入的字符串逐位与 0xA5 进行异或运算后，与定义的 v16 数组逐位进行数据的对比，如果全部正确，则输出“Correct”，否则输出“Wrong”（还有长度错误判断，如果输入的字符串长度不对，也提示错误）

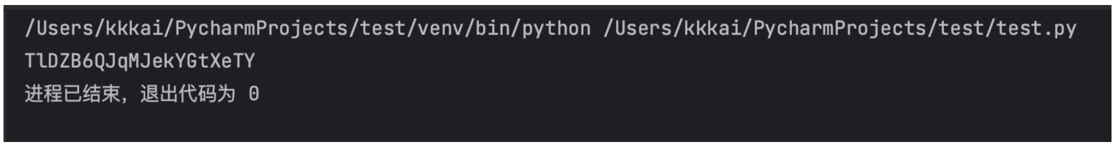
则应该有，输入字符串 str[i]^0xA5=test[i]

即 str[i]=test[i]^0xA5



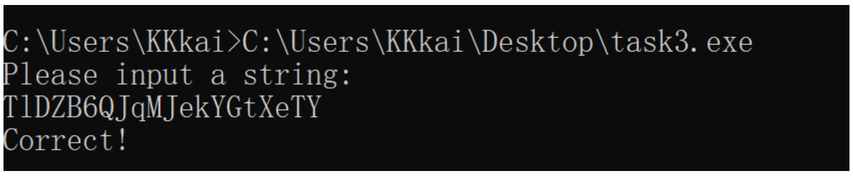
```python


test=[-15,-55,-31,-1,-25,-109,-12,-17,-44,-24,-17,-64,-50,-4,-30,-47,-3,-64,-15,-4]

for i in test:
    i^=0xA5
    print(chr(i&0xFF),end='')

```

从而，编写 python 代码如图，即可得出正确的输入字符串：



```
/Users/kkkai/PycharmProjects/test/venv/bin/python /Users/kkkai/PycharmProjects/test/test.py
TlDZB6QJqMJekYGtXeTY
进程已结束，退出代码为 0
```

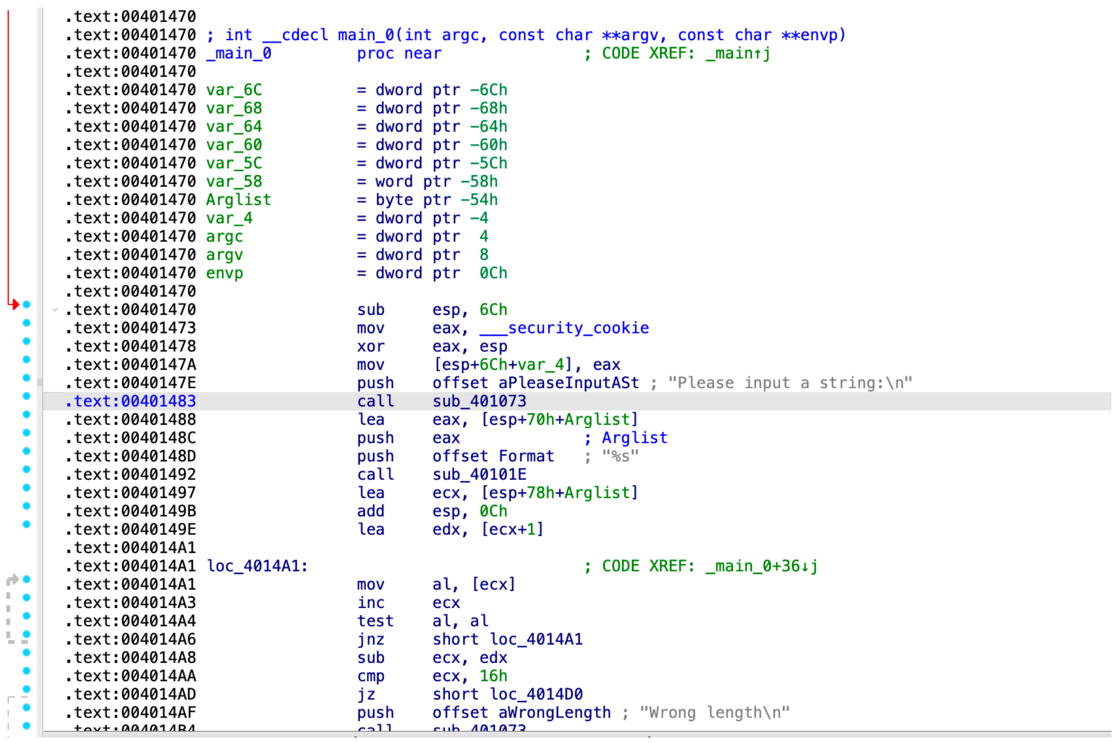
上图为程序运行结果，将结果字符串 TlDZB6QJqMJekYGtXeTY 输入程序中，得到正确答案：



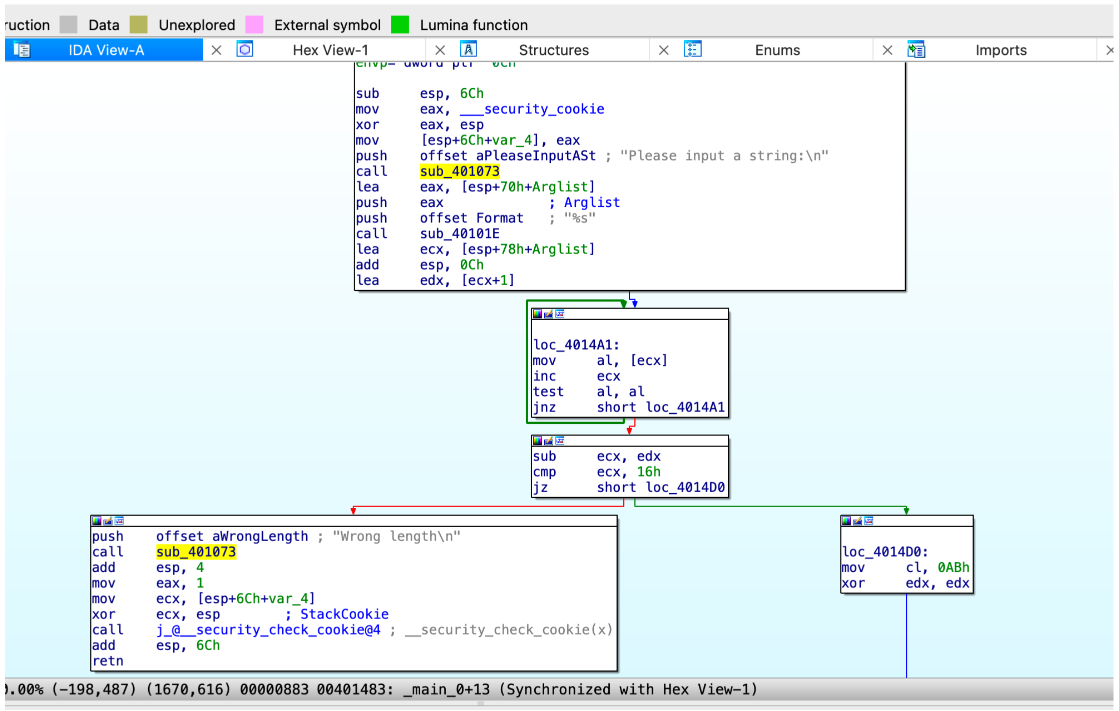
```
C:\Users\KKkai>C:\Users\KKkai\Desktop\task3.exe
Please input a string:
TlDZB6QJqMJekYGtXeTY
Correct!
```

## （1）Task4

如图，将 task4.exe 导入 ida64 Freeware 中，可以得到二进制代码的反汇编代码



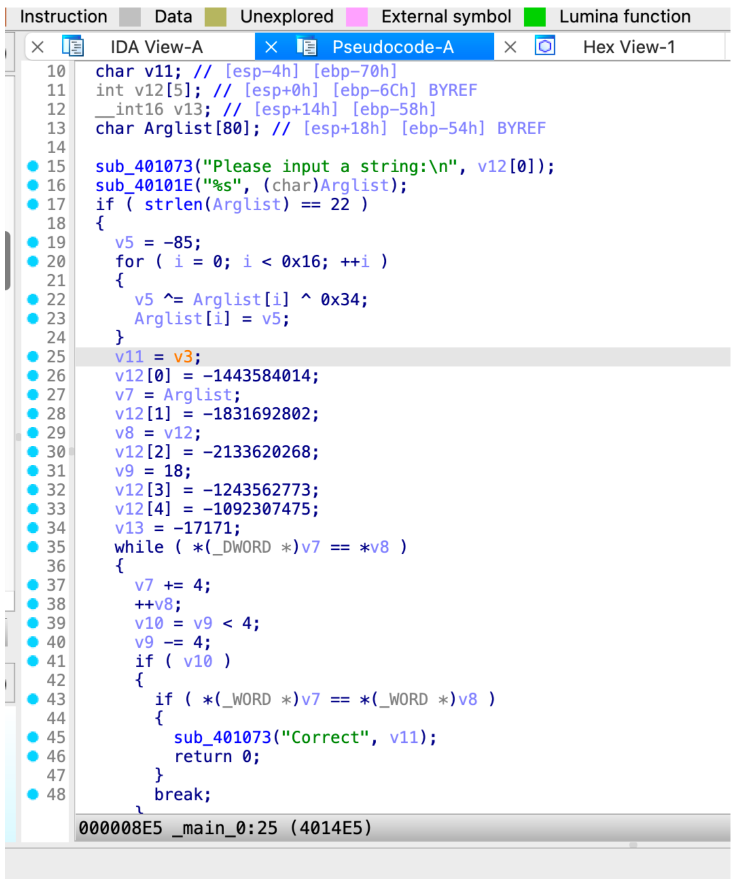
```
.text:00401470
.text:00401470 ; int __cdecl main_0(int argc, const char **argv, const char **envp)
.text:00401470 _main_0         proc near               ; CODE XREF: _main↑j
.text:00401470
.text:00401470 var_6C          = dword ptr -6Ch
.text:00401470 var_68          = dword ptr -68h
.text:00401470 var_64          = dword ptr -64h
.text:00401470 var_60          = dword ptr -60h
.text:00401470 var_5C          = dword ptr -5Ch
.text:00401470 var_58          = word ptr -58h
.text:00401470 Arglist         = byte ptr -54h
.text:00401470 var_4           = dword ptr -4
.text:00401470 argc            = dword ptr  4
.text:00401470 argv            = dword ptr  8
.text:00401470 envp            = dword ptr  0Ch
.text:00401470
.text:00401470                 sub     esp, 6Ch
.text:00401473                 mov     eax, ___security_cookie
.text:00401478                 xor     eax, esp
.text:0040147A                 mov     [esp+6Ch+var_4], eax
.text:0040147E                 push    offset aPleaseInputASt ; "Please input a string:\n"
.text:00401483                 call    sub_401073
.text:00401488                 lea     eax, [esp+70h+Arglist]
.text:0040148C                 push    eax             ; Arglist
.text:0040148D                 push    offset Format   ; "%s"
.text:00401492                 call    sub_40101E
.text:00401497                 lea     ecx, [esp+78h+Arglist]
.text:0040149B                 add     esp, 0Ch
.text:0040149E                 lea     edx, [ecx+1]
.text:004014A1
.text:004014A1 loc_4014A1:                             ; CODE XREF: _main_0+36↓j
.text:004014A1                 mov     al, [ecx]
.text:004014A3                 inc     ecx
.text:004014A4                 test    al, al
.text:004014A6                 jnz     short loc_4014A1
.text:004014A8                 sub     ecx, edx
.text:004014AA                 cmp     ecx, 16h
.text:004014AD                 jz      short loc_4014D0
.text:004014AF                 push    offset aWrongLength ; "Wrong length\n"
```



```
sub     esp, 6Ch
mov     eax, ___security_cookie
xor     eax, esp
mov     [esp+6Ch+var_4], eax
push    offset aPleaseInputASt ; "Please input a string:\n"
call    sub_401073
lea     eax, [esp+70h+Arglist]
push    eax             ; Arglist
push    offset Format   ; "%s"
call    sub_40101E
lea     ecx, [esp+78h+Arglist]
add     esp, 0Ch
lea     edx, [ecx+1]

loc_4014A1:
mov     al, [ecx]
inc     ecx
test    al, al
jnz     short loc_4014A1

sub     ecx, edx
cmp     ecx, 16h
jz      short loc_4014D0

push    offset aWrongLength ; "Wrong length\n"
call    sub_401073
add     esp, 4
mov     eax, 1
mov     ecx, [esp+6Ch+var_4]
xor     ecx, esp        ; StackCookie
call    j_@__security_check_cookie@4 ; __security_check_cookie(x)
add     esp, 6Ch
retn

loc_4014D0:
mov     cl, 0ABh
xor     edx, edx
```

```
).00% (-198,487) (1670,616) 00000883 00401483: _main_0+13 (Synchronized with Hex View-1)
```

使用反汇编功能，将其转为 C++代码

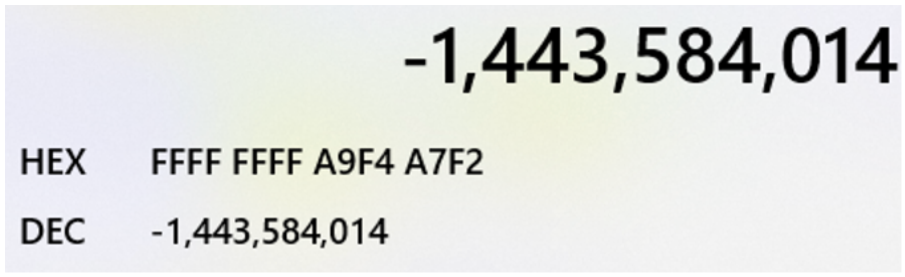


```
char v11; // [esp-4h] [ebp-70h]
int v12[5]; // [esp+0h] [ebp-6Ch] BYREF
__int16 v13; // [esp+14h] [ebp-58h]
char Arglist[80]; // [esp+18h] [ebp-54h] BYREF

sub_401073("Please input a string:\n", v12[0]);
sub_40101E("%s", (char)Arglist);
if ( strlen(Arglist) == 22 )
{
  v5 = -85;
  for ( i = 0; i < 0x16; ++i )
  {
    v5 ^= Arglist[i] ^ 0x34;
    Arglist[i] = v5;
  }
  v11 = v3;
  v12[0] = -1443584014;
  v7 = Arglist;
  v12[1] = -1831692802;
  v8 = v12;
  v12[2] = -2133620268;
  v9 = 18;
  v12[3] = -1243562773;
  v12[4] = -1092307475;
  v13 = -17171;
  while ( *(_DWORD *)v7 == *v8 )
  {
    v7 += 4;
    ++v8;
    v10 = v9 < 4;
    v9 -= 4;
    if ( v10 )
    {
      if ( *(_WORD *)v7 == *(_WORD *)v8 )
      {
        sub_401073("Correct", v11);
        return 0;
      }
      break;
    }

000008E5 _main_0:25 (4014E5)
```

分析代码可知，该程序提示用户输入一个字符串，然后对字符串每一位进行异或加密（先初始化了一个 v5=-85，随后在每次循环中对每一位 Arr[i]=Arr[i]^v5^0x34 对异或处理，再更新 v5 的值为 Arr[i]，然后 i+1 进行下一位的加密）。

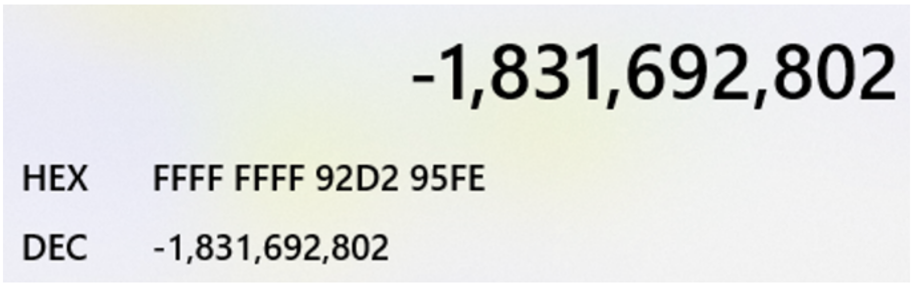
随后对加密后的字符串与 v12 进行比对，如果全部匹配，则输出 Correct，否则输出 Wrong。

则由上述分析，先将 v12 中储存的数字分析得：



```
-1,443,584,014

HEX    FFFF FFFF A9F4 A7F2
DEC    -1,443,584,014
```

V12[0]存储为 F2 A7 F4 A9 （小端字节序），且 v12 定义为 DWORD（4 字节），则可舍去前 4 字节的 FFFF FFFF。同理，可得

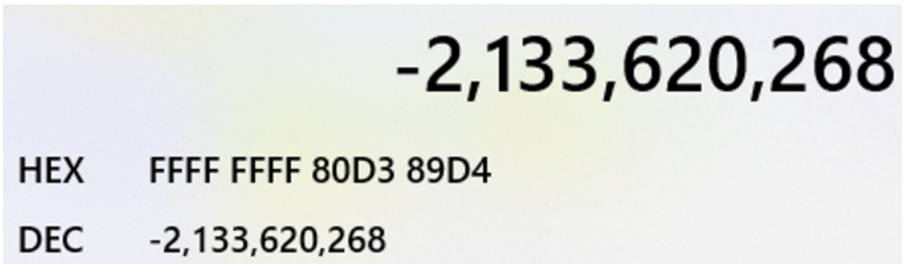


-1,831,692,802

HEX FFFF FFFF 92D2 95FE

DEC -1,831,692,802

V12[1]存储为 FE 95 D2 92

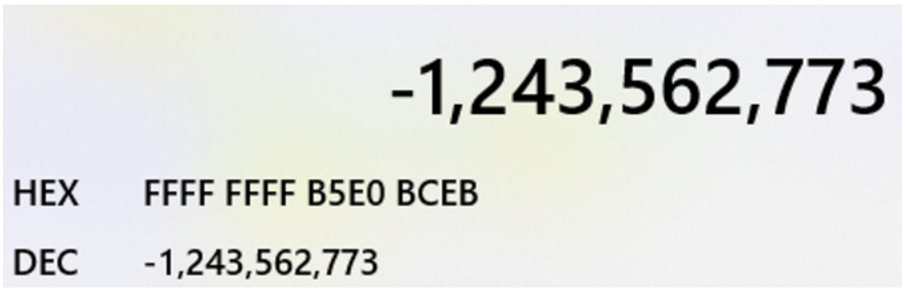


-2,133,620,268

HEX FFFF FFFF 80D3 89D4

DEC -2,133,620,268

V12[2]存储为 D4 89 D3 80

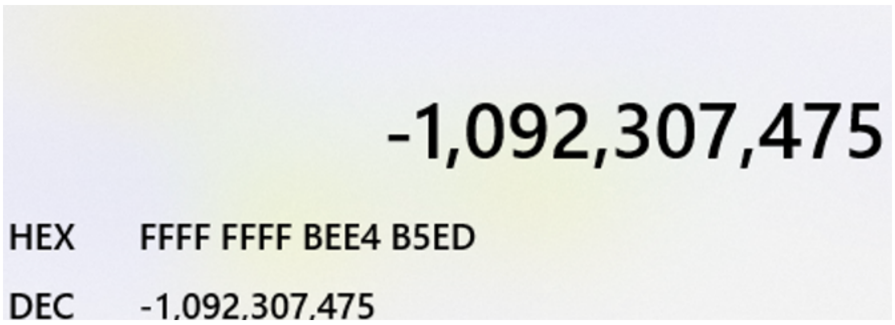


-1,243,562,773

HEX FFFF FFFF B5E0 BCEB

DEC -1,243,562,773

V12[3]存储为 EB BC E0 B5

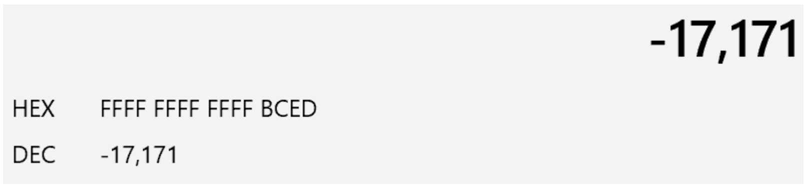


-1,092,307,475

HEX FFFF FFFF BEE4 B5ED

DEC -1,092,307,475

V12[4]存储为 ED B5 E4 BE



-17,171

HEX FFFF FFFF FFFF BCED

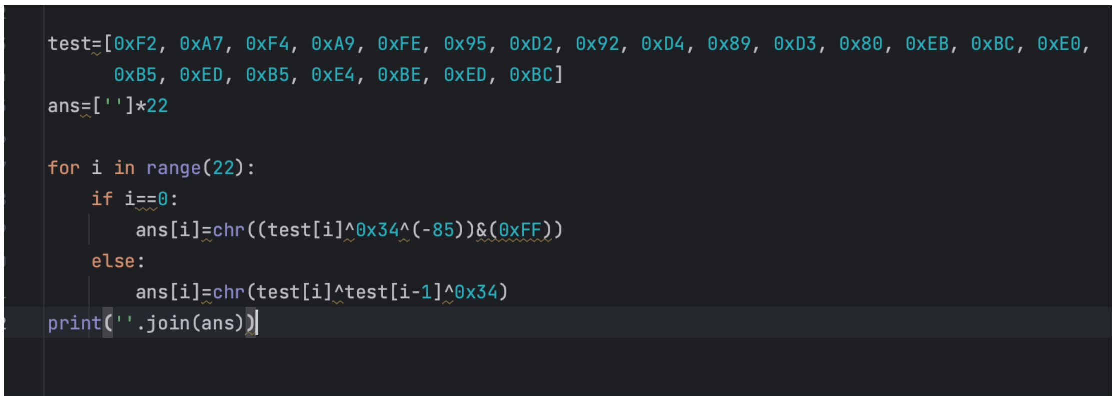
DEC -17,171

对应的最后一个 WORD 类型存储为 ED BC

则可知，作为 DWORD 存储的最后数据应该为：

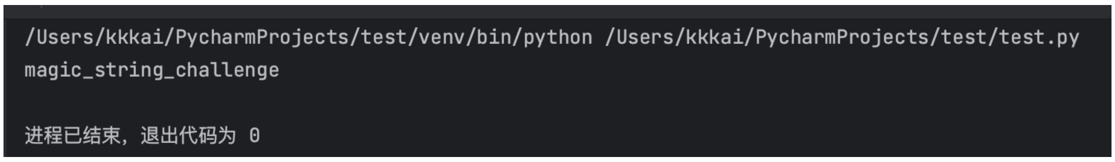
F2 A7 F4 A9 FE 95 D2 92 D4 89 D3 80 EB BC E0 B5 ED B5 E4 BE ED BC

以 F2 为例，由 Arr[0]^v5^0x34=F2 得，Arr[0]=F2^0x34^v5,再更新 v5=Arr[0]，通过相同的式子计算后面的 Arr 部分，python 代码实现如下：



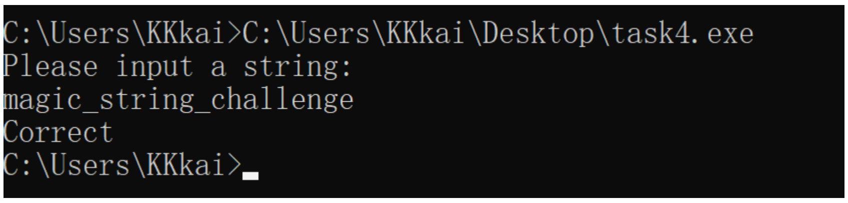
```python
test=[0xF2, 0xA7, 0xF4, 0xA9, 0xFE, 0x95, 0xD2, 0x92, 0xD4, 0x89, 0xD3, 0x80, 0xEB, 0xBC, 0xE0,
      0xB5, 0xED, 0xB5, 0xE4, 0xBE, 0xED, 0xBC]
ans=['']*22

for i in range(22):
    if i==0:
        ans[i]=chr((test[i]^0x34^(-85))&(0xFF))
    else:
        ans[i]=chr(test[i]^test[i-1]^0x34)
print(''.join(ans))
```

运行程序，得出答案



```
/Users/kkkai/PycharmProjects/test/venv/bin/python /Users/kkkai/PycharmProjects/test/test.py
magic_string_challenge

进程已结束，退出代码为 0
```

运行 task4.exe 程序，输入得到最终结果



```
C:\Users\KKkai>C:\Users\KKkai\Desktop\task4.exe
Please input a string:
magic_string_challenge
Correct
C:\Users\KKkai>_
```

## 五. 附件（python 代码）

### Task1:

```python
test=[-15,-55,-31,-1,-25,-109,-12,-17,-44,-24,-17,-64,-50,-4,-30,-
47,-3,-64,-15,-4]

for i in test:
    i^=0xA5
    print(chr(i&0xFF),end='')
```

### Task4:

```python
test=[0xF2, 0xA7, 0xF4, 0xA9, 0xFE, 0x95, 0xD2, 0x92, 0xD4, 0x89,
0xD3, 0x80, 0xEB, 0xBC, 0xE0,
      0xB5, 0xED, 0xB5, 0xE4, 0xBE, 0xED, 0xBC]
ans=['']*22
```

```python
for i in range(22):
    if i==0:
        ans[i]=chr((test[i]^0x34^(-85))&(0xFF))
    else:
        ans[i]=chr(test[i]^test[i-1]^0x34)
print(''.join(ans))
```