南開大學
Nankai University

计算机学院

汇编语言与逆向技术实验报告

Lab7 Reverse Engineering Exercises –Advanced

姓名：杨冰雪

学号：2110508

专业：计算机科学与技术

2023 年 12 月 21 日

目录

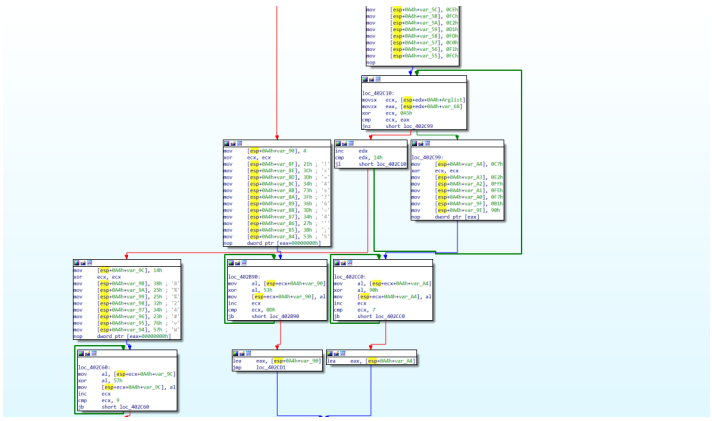
1 实验目的

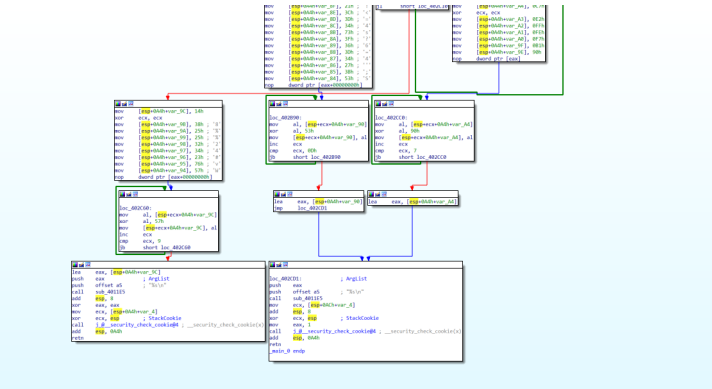
1. 进一步熟悉静态反汇编工具 IDA Freeware；
2. 熟悉将反汇编代码进行反编译的过程；
3. 掌握对于反编译伪代码的逆向分析；
4. 运用熟悉的编程语言，实现简单的脚本编写

2 实验原理

2.1 task3 反汇编代码

1. 通过 IDA FreeWare 得到 task3.exe 的反汇编图形化显示如下：





2. 通过 IDA FreeWare 得到 task3.exe 的反汇编代码显示如下：

```
.text:00402A70
.text:00402A70 ; int __cdecl main_0(int argc, const char **argv, const char **envp)
.text:00402A70 _main_0          proc near               ; CODE XREF: _mainfj
.text:00402A70
.text:00402A70 var_A4           = byte ptr -0A4h
.text:00402A70 var_A3           = byte ptr -0A3h
.text:00402A70 var_A2           = byte ptr -0A2h
.text:00402A70 var_A1           = byte ptr -0A1h
.text:00402A70 var_A0           = byte ptr -0A0h
.text:00402A70 var_9F           = byte ptr -9Fh
.text:00402A70 var_9E           = byte ptr -9Eh
.text:00402A70 var_9C           = byte ptr -9Ch
.text:00402A70 var_9B           = byte ptr -9Bh
.text:00402A70 var_9A           = byte ptr -9Ah
.text:00402A70 var_99           = byte ptr -99h
.text:00402A70 var_98           = byte ptr -98h
.text:00402A70 var_97           = byte ptr -97h
.text:00402A70 var_96           = byte ptr -96h
.text:00402A70 var_95           = byte ptr -95h
.text:00402A70 var_94           = byte ptr -94h
.text:00402A70 var_90           = byte ptr -90h
.text:00402A70 var_8F           = byte ptr -8Fh
.text:00402A70 var_8E           = byte ptr -8Eh
.text:00402A70 var_8D           = byte ptr -8Dh
.text:00402A70 var_8C           = byte ptr -8Ch
.text:00402A70 var_8B           = byte ptr -8Bh
.text:00402A70 var_8A           = byte ptr -8Ah
.text:00402A70 var_89           = byte ptr -89h
.text:00402A70 var_88           = byte ptr -88h
.text:00402A70 var_87           = byte ptr -87h
.text:00402A70 var_86           = byte ptr -86h
.text:00402A70 var_85           = byte ptr -85h
```

```
.text:00402A70 var_69           = byte ptr -69h
.text:00402A70 var_68           = byte ptr -68h
.text:00402A70 var_67           = byte ptr -67h
.text:00402A70 var_66           = byte ptr -66h
.text:00402A70 var_65           = byte ptr -65h
.text:00402A70 var_64           = byte ptr -64h
.text:00402A70 var_63           = byte ptr -63h
.text:00402A70 var_62           = byte ptr -62h
.text:00402A70 var_61           = byte ptr -61h
.text:00402A70 var_60           = byte ptr -60h
.text:00402A70 var_5F           = byte ptr -5Fh
.text:00402A70 var_5E           = byte ptr -5Eh
.text:00402A70 var_5D           = byte ptr -5Dh
.text:00402A70 var_5C           = byte ptr -5Ch
.text:00402A70 var_5B           = byte ptr -5Bh
.text:00402A70 var_5A           = byte ptr -5Ah
.text:00402A70 var_59           = byte ptr -59h
.text:00402A70 var_58           = byte ptr -58h
.text:00402A70 var_57           = byte ptr -57h
.text:00402A70 var_56           = byte ptr -56h
.text:00402A70 var_55           = byte ptr -55h
.text:00402A70 Arglist          = byte ptr -54h
.text:00402A70 var_4            = dword ptr -4
.text:00402A70 argc             = dword ptr  4
.text:00402A70 argv             = dword ptr  8
.text:00402A70 envp             = dword ptr  0Ch
.text:00402A70
.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+ArgList], 42h ; 'B'
```

```
.text:00402B06                mov     [esp+ecx+0A4h+ArgList], al
.text:00402B0A                inc     ecx
.text:00402B0B                cmp     ecx, 18h
.text:00402B0E                jb      short loc_402B00
.text:00402B10                lea     eax, [esp+0A4h+ArgList]
.text:00402B14                push    eax             ; ArgList
.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 ; '?'
```

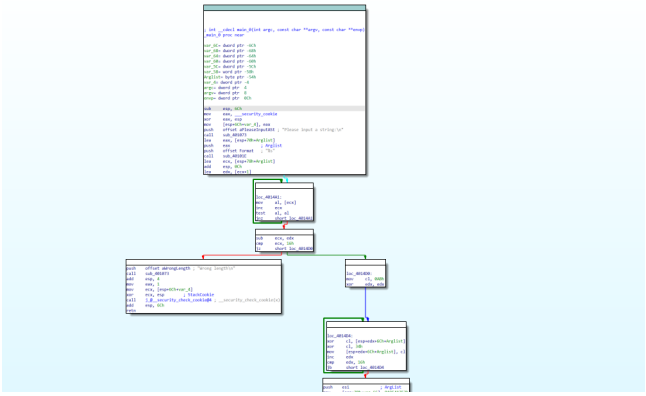
3. 通过 IDA FreeWare 得到 task3.exe 的反编译伪代码显示如下：

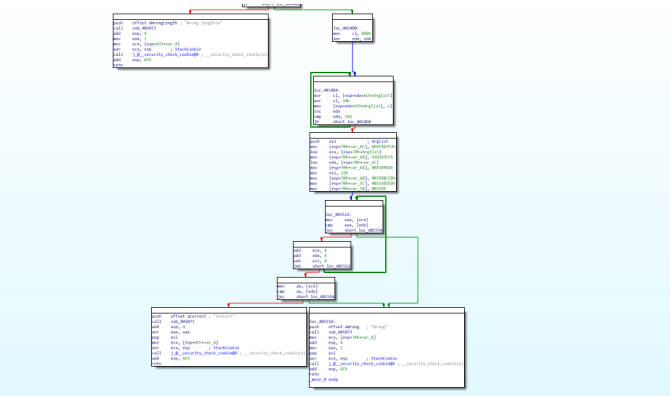
```
int __cdecl main_0(int argc, const char **argv, const char **envp)
{
  unsigned int v3; // ecx
  unsigned int v4; // ecx
  char *v5; // eax
  int v6; // edx
  unsigned int v7; // ecx
  unsigned int v9; // ecx
  char v10[8]; // [esp+0h] [ebp-A4h] BYREF
  char v11; // [esp+8h] [ebp-9Ch] BYREF
  char v12[8]; // [esp+9h] [ebp-98h] BYREF
  char v13; // [esp+14h] [ebp-90h] BYREF
  char v14[12]; // [esp+15h] [ebp-8Fh] BYREF
  char ArgList[24]; // [esp+24h] [ebp-80h] BYREF
  char v16[20]; // [esp+3Ch] [ebp-68h]
  char Arglist[80]; // [esp+50h] [ebp-54h] BYREF

  qmemcpy(ArgList, "B~wsaw2{|bgf2s2af`{|u(", 22);
  v3 = 0;
  ArgList[22] = 24;
  ArgList[23] = 18;
  do
    ArgList[v3++] ^= 0x12u;
  while ( v3 < 0x18 );
  sub_4011E5("%s", (char)ArgList);
  sub_4011EF("%s", (char)Arglist);
  if ( strlen(Arglist) == 20 )
  {
    v16[0] = -15;
    v6 = 0;
    v16[1] = -55;
    v16[2] = -31;
    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;
      do
        v10[v9++] ^= 0x90u;
      while ( v9 < 7 );
      v5 = v10;
    }
    else
    {
      v13 = 4;
      v4 = 0;
      qmemcpy(v14, "!<=4s?6=4';S", sizeof(v14));
      do
        v14[v4++ - 1] ^= 0x53u;
      while ( v4 < 0xD );
      v5 = &v13;
    }
    sub_4011E5("%s\n", (char)v5);
    return 1;
  }
```

2.2 task4 反汇编代码

1. 通过 IDA FreeWare 得到 task4.exe 的反汇编图形化显示如下：





2. 通过 IDA FreeWare 得到 task4.exe 的反汇编代码显示如下：

```
.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:004014A8                sub     ecx, edx
.text:004014AA                cmp     ecx, 16h
.text:004014AD                jz      short loc_4014D0
.text:004014AF                push    offset aWrongLength ; "Wrong length\n"
.text:004014B4                call    sub_401073
.text:004014B9                add     esp, 4
.text:004014BC                mov     eax, 1
.text:004014C1                mov     ecx, [esp+6Ch+var_4]
.text:004014C5                xor     ecx, esp        ; StackCookie
.text:004014C7                call    j_@__security_check_cookie@4 ; __security_check_cookie(x)
.text:004014CC                add     esp, 6Ch
.text:004014CF                retn
.text:004014D0 ; ---------------------------------------------------------------------------
.text:004014D0
.text:004014D0 loc_4014D0:                            ; CODE XREF: _main_0+3D↑j
.text:004014D0                mov     cl, 0ABh
.text:004014D2                xor     edx, edx
.text:004014D4
.text:004014D4 loc_4014D4:                            ; CODE XREF: _main_0+73↓j
.text:004014D4                xor     cl, [esp+edx+6Ch+Arglist]
.text:004014D8                xor     cl, 34h
.text:004014DB                mov     [esp+edx+6Ch+Arglist], cl
.text:004014DF                inc     edx
.text:004014E0                cmp     edx, 16h
.text:004014E3                jb      short loc_4014D4
.text:004014E5                push    esi             ; ArgList
.text:004014E6                mov     [esp+70h+var_6C], 0A9F4A7F2h
.text:004014EE                lea     ecx, [esp+70h+Arglist]
.text:004014F2                mov     [esp+70h+var_68], 92D295FEh
.text:004014FA                lea     edx, [esp+70h+var_6C]
.text:004014FE                mov     [esp+70h+var_64], 80D389D4h
.text:00401506                mov     esi, 12h
```

```
.text:00401522 loc_401522:                            ; CODE XREF: _main_0+C1↓j
.text:00401522                mov     eax, [ecx]
.text:00401524                cmp     eax, [edx]
.text:00401526                jnz     short loc_40155A
.text:00401528                add     ecx, 4
.text:0040152B                add     edx, 4
.text:0040152E                sub     esi, 4
.text:00401531                jnb     short loc_401522
.text:00401533                mov     ax, [ecx]
.text:00401536                cmp     ax, [edx]
.text:00401539                jnz     short loc_40155A
.text:0040153B                push    offset aCorrect ; "Correct"
.text:00401540                call    sub_401073
.text:00401545                add     esp, 4
.text:00401548                xor     eax, eax
.text:0040154A                pop     esi
.text:0040154B                mov     ecx, [esp+6Ch+var_4]
.text:0040154F                xor     ecx, esp        ; StackCookie
.text:00401551                call    j_@__security_check_cookie@4 ; __security_check_cookie(x)
.text:00401556                add     esp, 6Ch
.text:00401559                retn
.text:0040155A ; ---------------------------------------------------------------------------
.text:0040155A
.text:0040155A loc_40155A:                            ; CODE XREF: _main_0+B6↑j
.text:0040155A                                        ; _main_0+C9↑j
.text:0040155A                push    offset aWrong   ; "Wrong"
.text:0040155F                call    sub_401073
.text:00401564                mov     ecx, [esp+74h+var_4]
.text:00401568                add     esp, 4
.text:0040156B                mov     eax, 1
.text:00401570                pop     esi
.text:00401571                xor     ecx, esp        ; StackCookie
```

3. 通过 IDA FreeWare 得到 task4.exe 的反编译伪代码显示如下：

```
int __cdecl main_0(int argc, const char **argv, const char **envp)
{
  char v3; // si
  char v5; // cl
  unsigned int i; // edx
  char *v7; // ecx
  int *v8; // edx
  unsigned int v9; // esi
  bool v10; // cf
  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[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;
  }
}
sub_401073("Wrong", v11);
return 1;
}
else
{
  sub_401073("Wrong length\n", v12[0]);
  return 1;
}
```

3 实验过程

3.1 task3 逆向分析

- ArgList 数组被赋值为字符 B wsaw2|bgf2s2af‘|u(，并将数组的第 23，24 位分别赋值为 24，18，将该数组中的每一个元素都与 0x12 异或并输出，其结果为 “Please input a string:”
- 进行输入，并将输入的字符串储存在 Arglist 中
- 然后输入字符串，如果输入的字符串长度不是 20，则!<=4s?6=4’;S 与 0x53 异或，其结果为 Wrong Length，结束程序
- 如果长度为 20，先给 v16 数组进行赋值，再将输入字符串的每一位与 0xA5 异或，若得到的字符串的每一位都和 v16 中的元素相同，则将 8%%24#vW 与 0x57 异或，输出 Correct!
- 如果不相同，则将 v10 赋值，并与 0x90 相异或，输出 Wrong

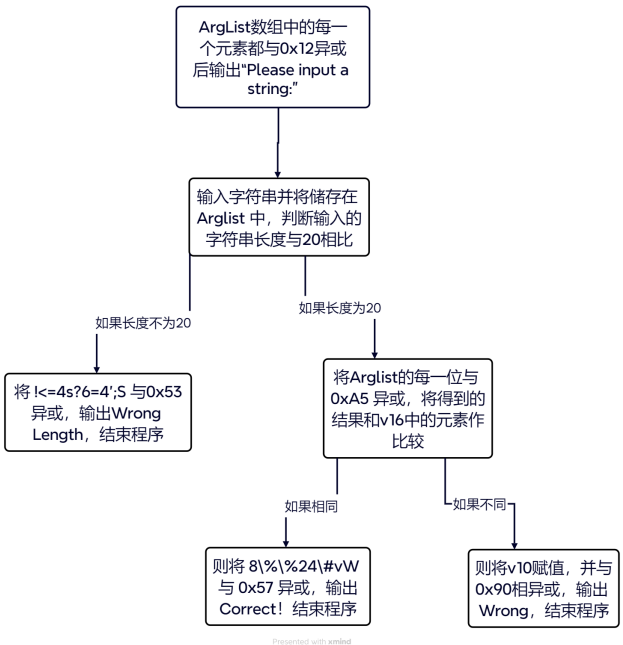


图 3.1: task3 流程图

根据上面分析，得出其计算公式：

$$a[i] = v16[i] \oplus 0x53$$

3.2 task4 逆向分析

- 首先程序输出提示：”Please input a string:”
- 然后输入的字符串，判断字符串长度是否为 22
- 如果长度不为 22，则输出 “Wrong length”，然后退出程序
- 如果长度为 22，对原来的字符数组 a[22] 进行循环异或操作得到 b[22]
- 给 v12 中的每一位赋值，共计 20 个字节，v13 在内存上与 v12 相邻，占四个字节
- 把 v9 与 4 的大小进行比较，从而比较 b[22] 中每个元素的字节内容和 v12，v13 中的字节内容比较，若完全相等，则可以得到 Correct! 的输出。

根据上面分析，得出公式如下：

$$b[i] = a[i] \oplus 0x34 \oplus b[i-1]$$

$$a[i] = b[i] \oplus b[i-1] \oplus 0x34$$

4 实验内容

4.1 task3 脚本编写

使用 c++ 对 task3.exe 编写脚本如下：

```cpp
#include<iostream>
using namespace std;
int main() {
    int num[20] = { -15,-55,-31,-1,-25,-109,-12,-17,-44,-24,-17,-64,-50,-4,-30,-47,-3,-64,-15,-4 };
    for (int i = 0;i < 20;i++) {
        char a = 0xA5 ^ (_int8)num[i];
        cout << a;
    }
}
```

图 4.2: task3 脚本

其脚本输出结果如下：

```
Microsoft Visual Studio 调试控制台
TlDZB6QJqMJekYGtXeTY
E:\c++_code\test\x64\Debug\test.exe (进程 28080)已退出，代码为 0。
按任意键关闭此窗口. . .
```

图 4.3: task3 脚本输出

4.2 task4 脚本编写

使用 c++ 对 task4.exe 编写脚本如下：

```cpp
using namespace std;
int main()
{

    int a[22] = { 0xf2, 0xa7, 0xf4, 0xa9, 0xfe, 0x95, 0xd2, 0x92, 0xd4, 0x89, 0xd3, 0x80,
                  0xeb, 0xbc, 0xe0, 0xb5, 0xed, 0xb5, 0xe4, 0xbe, 0xed, 0xbc};
    char str;
    for (int i = 0; i < 22; i++)
    {
        if (i == 0)
        {
            str = a[i] ^ 52 ^ (-85);
        }
        else
        {
            str = a[i] ^ a[i - 1] ^ 52;
        }
        cout << str;
    }
    return 0;
}
```

图 4.4: task4 脚本

其脚本输出结果如下：

```
Microsoft Visual Studio 调试控制台
magic_string_challenge
E:\c++_code\test\x64\Debug\test.exe (进程 38888)已退出，代码为 0。
按任意键关闭此窗口. . .
```

图 4.5: task4 脚本输出

5 实验结果

将脚本输出结果分别输入到 task3.exe 和 task4.exe 程序中，程序输出 correct，说明实验成功！

```
E:\IDA_code>task3.exe
Please input a string:
TlDZB6QJqMJekYGtXeTY
Correct!
```

图 5.6: task3 成功截图

```
E:\IDA_code>task4.exe
Please input a string:
magic_string_challenge
Correct
E:\IDA_code>
```

图 5.7: task4 成功截图

6 实验总结

通过本次实验，使我更加熟练掌握 IDA FreeWare 的使用和操作，能够通过该工具来查看可执行文件的反汇编代码和反编译伪代码，加强了对反编译伪代码逆向分析的能力，学会使用编程语言 c++ 来编写简单的脚本，强化了编写代码的能力。