

汇编语言与逆向技术课程实验报告

实验八：Reverse Engineering Exercises(Simple)



学院： 密码与网络空间安全学院

专业： 信息安全

学号： 2412950

姓名： 路浩斌

班级： 信安一班

一、实验目的

- 1、熟悉静态反汇编工具 Binary Ninja;
- 2、熟悉反汇编代码的逆向分析过程;
- 3、掌握反汇编语言中的数学计算、数据结构、条件判断、分支结构的识别和逆向分析。

二、实验原理

(1)task1

1. 通过 Binary Ninja 得到 task1.exe 的反汇编代码，如图 1 和图 2 所示。

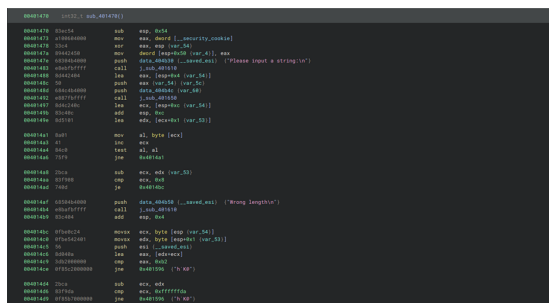


图 1: task1.exe 的反汇编代码



图 2: task1.exe 反汇编代码的图形化显示

2. 对反汇编代码和计算过程、条件判断、分支结构等信息进行分析，逆向推出程序的正确输入数据，完成逆向分析挑战。

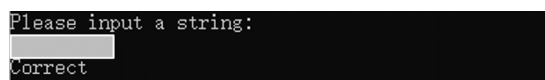


图 3: 逆向分析，完成 task1 练习

(2)task2

1. 通过 Binary Ninja 得到 task2.exe 的反汇编代码，如图 4 和图 5 所示。

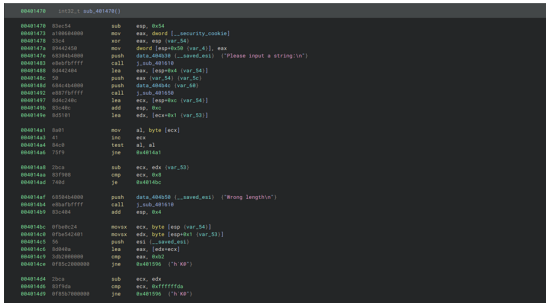


图 4: task2.exe 的反汇编代码



图 5: task2.exe 反汇编代码的图形化显示

2. 对反汇编代码的计算过程、条件判断、分支结构等信息进行分析，逆向推出程序的正确输入数据，完成逆向分析挑战。

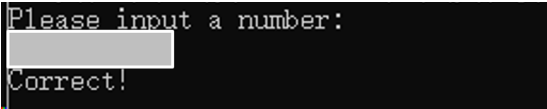


图 6: 逆向分析，完成 task2 练习

三、实验报告内容

3.1. task1

3.1.1 反汇编代码

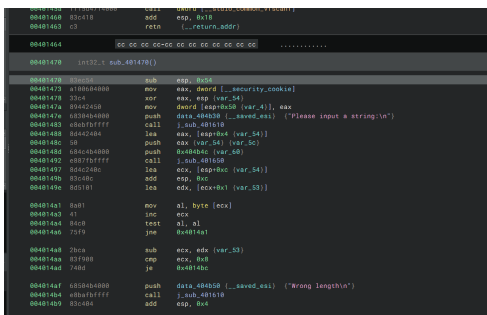


图 7: task1 反汇编代码

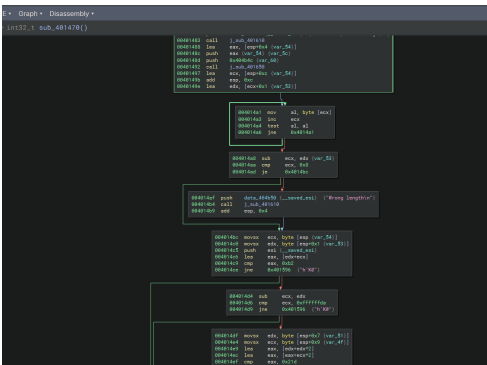


图 8: task1 图形化显示

3.1.2 逆向分析

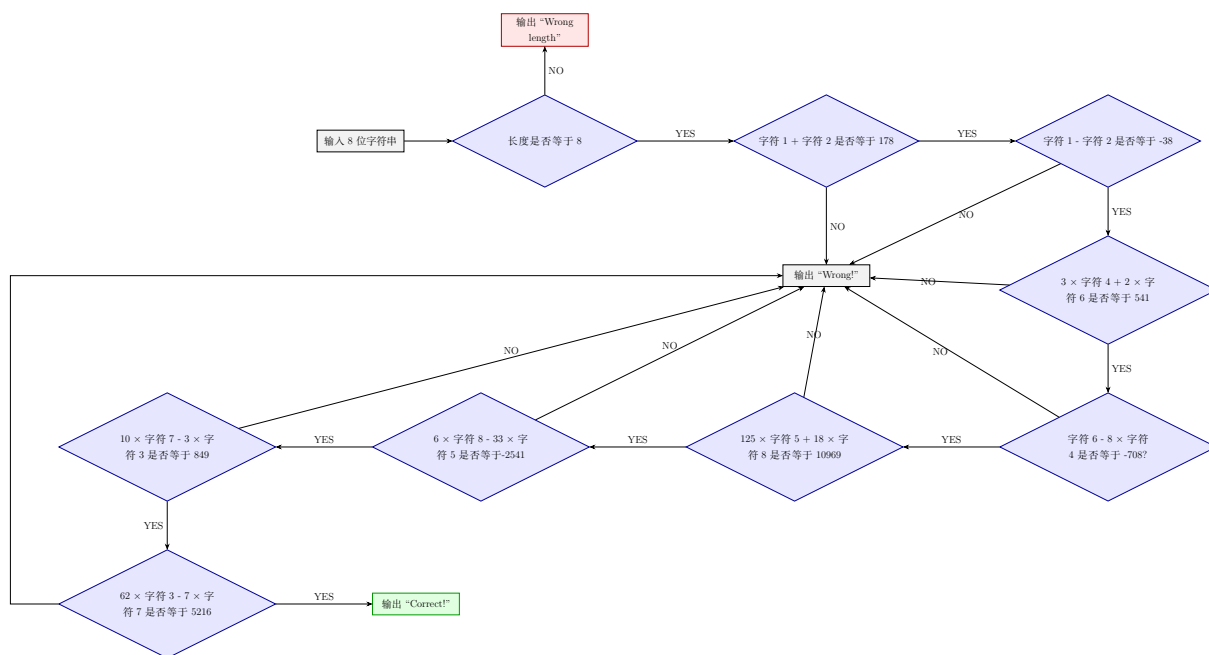


图 9: 程序逻辑流程图

```

sub_401470:
00401470 sub     esp, 0x54
00401473 mov     eax, dword [__security_cookie]
00401478 xor     eax, esp {var_54}
0040147a mov     dword [esp+0x50 {var_4}], eax
0040147e push    data_404b30 {__saved_esi} {"Please input a string:\n"}
00401483 call   j_sub_401610
00401488 lea     eax, [esp+0x4 {var_54}]
0040148c push    eax {var_54} {var_5c}
0040148d push    0x404b4c {var_60}
00401492 call   j_sub_401650
00401497 lea     ecx, [esp+0xc {var_54}]
0040149b add     esp, 0xc
0040149e lea     edx, [ecx+0x1 {var_53}]

004014a1 mov     al, byte [ecx]
004014a3 inc     ecx
004014a4 test    al, al
004014a6 jne     0x4014a1

004014a8 sub     ecx, edx {var_53}
004014aa cmp     ecx, 0x8
004014ad je     0x4014bc

004014af push    data_404b50 {__saved_esi} {"Wrong length\n"}
004014b4 call   j_sub_401610
004014b9 add     esp, 0x4
  
```

首先输出提示信息 “Please input a string:”，用户输入一个字符串后，判断输入字符串的长度是否为 8，若不是，输出 “Wrong length”。若是，进入程序的核心逻辑部分。

```

004014bc movsx ecx, byte [esp {var_54}]
004014c0 movsx edx, byte [esp+0x1 {var_53}]
004014c5 push esi {__saved_e si}
004014c6 lea eax, [edx+ecx]
004014c9 cmp eax, 0xb2
004014ce jne 0x401596 {"h`K@"}

```

```

004014d4 sub ecx, edx
004014d6 cmp ecx, 0xfffffda
004014d9 jne 0x401596 {"h`K@"}

```

该部分的功能是将第一个字符存入 ecx 中，将第二个字符存入 edx 中，将两个数相加判断其是否与 178 相等，如果不相等直接跳转到最后的错误输出中，如果相等则继续判断；将上面两个数相减判断其是否等于-38，相等则继续判断；

```

004014df movsx edx, byte [esp+0x7 {var_51}]
004014e4 movsx ecx, byte [esp+0x9 {var_4f}]
004014e9 lea eax, [edx+edx*2]
004014ec lea eax, [eax+ecx*2]
004014ef cmp eax, 0x21d
004014f4 jne 0x401596 {"h`K@"}

```

```

004014fa lea eax, [edx*8]
00401501 sub ecx, eax
00401503 cmp ecx, 0xfffffd3c
00401509 jne 0x401596 {"h`K@"}

```

该部分的功能是将第四个字符放入 edx 中，第六个字符放入 ecx 中，判断” 第四个字符 *3+ 第六个字符 *2 的和” 是否与 541 相等，相等则继续判断” 第六个字符-第四个字符的八倍” 是否与 -708 相等；

```

0040150f movsx esi, byte [esp+0x8 {var_50}]
00401514 movsx edx, byte [esp+0xb {var_4d}]
00401519 imul ecx, esi, 0x7d
0040151c lea eax, [edx+edx*8]
0040151f lea eax, [ecx+eax*2]
00401522 cmp eax, 0x2ad9
00401527 jne 0x401596 {"h`K@"}

```

```

00401529 mov eax, esi
0040152b lea ecx, [edx+edx*2]
0040152e shl eax, 0x5
00401531 add ecx, ecx
00401533 add eax, esi
00401535 sub ecx, eax
00401537 cmp ecx, 0xfffff613
0040153d jne 0x401596 {"h`K@"}

```

该部分的功能是将第五个字符存入 esi 中，第八个字符存入 edx 中，首先将 $esi * 7Dh$ （即 125）放入 ecx 中，再进行 $18 * edx + ecx$ 的操作，化简后即判断“ $125 * \text{第五个数} + 18 * \text{第八个数}$ ”是否与 10969 相等；再判断“ $6 * \text{第八个数} - 33 * \text{第五个数}$ ”是否与 -2541 相等；

```

0040153f movsx edx, byte [esp+0xa {var_4e}]
00401544 movsx esi, byte [esp+0x6 {var_52}]
00401549 lea ecx, [edx+edx*4]
0040154c add ecx, ecx
0040154e lea eax, [esi+esi*2]
00401551 sub ecx, eax
00401553 cmp ecx, 0x351
00401559 jne 0x401596 {"h`K@"}

```

```

0040155b mov ecx, esi
0040155d lea eax, [edx*8]
00401564 shl ecx, 0x5
00401567 sub eax, edx
00401569 sub ecx, esi
0040156b add ecx, ecx
0040156d sub ecx, eax
0040156f cmp ecx, 0x1460
00401575 jne 0x401596 {"h`K@"}

```

```

00401577 push data_404b68 {"Correct\n"}
0040157c call j_sub_401610

```

将第七个数存入 edx 中，第三个数存入 esi 中，判断“ $10 * \text{第七个数} - 3 * \text{第三个数}$ ”是否与 849 相等；”判断“ $62 * \text{第三个数} - 7 * \text{第七个数}$ ”是否与 5216 相等，如果以上判断均成立则跳转到成功的输出。

3.1.3 成功截图

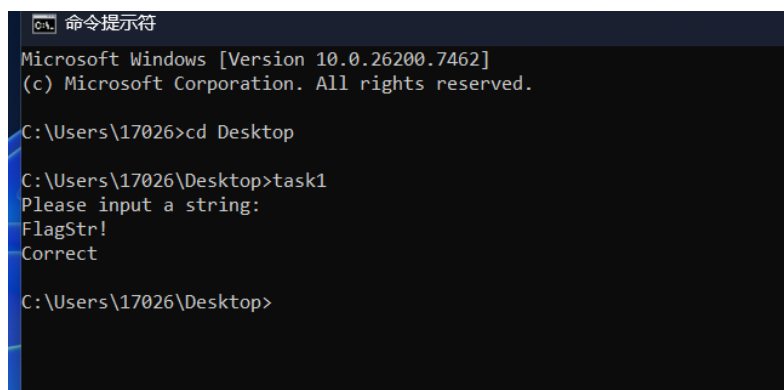
编写 c++ 代码将解析出逆向结果：

```

1  #include<iostream>
2  using namespace std;
3
4  int main() {
5      int a0, a1, a2, a3, a4, a5, a6, a7;
6
7      a0 = (178 - 38) / 2;
8      a1 = 178 - a0;
9      a2 = (7 * 849 + 10 * 5216) / (62 * 10 - 3 * 7);
10     a3 = (541 - 2 * (-708)) / (3 + 2 * 8);
11     a4 = (10969 - 3 * (-2541)) / (125 + 3 * 33);
12     a5 = 8 * a3 - 708;
13     a6 = (849 + 3 * a2) / 10;
14     a7 = (33 * a4 - 2541) / 6;
15
16     cout << char(a0) << char(a1) << char(a2)
17           << char(a3) << char(a4) << char(a5)
18           << char(a6) << char(a7) << endl;
19
20     return 0;
21 }

```

根据 c++ 程序，我们可以得到结果为 flagStr!, 运行 task1 并输入后得到截图：



```

C:\> 命令提示符
Microsoft Windows [Version 10.0.26200.7462]
(c) Microsoft Corporation. All rights reserved.

C:\Users\17026>cd Desktop
C:\Users\17026\Desktop>task1
Please input a string:
FlagStr!
Correct
C:\Users\17026\Desktop>

```

图 10: task1.exe 的成功截图

得到 correct，结果正确。

3.2 task2

3.2.1 反汇编代码

```

00401470      int32_t __fastcall sub_401470(int32_t arg1)
00401470  51          push     ecx (var_4)
00401471  68304b4008 push     data_404b30 (var_8) ("Please input a number:\n")
00401476  e8f8fbffff call     j_sub_401510
0040147b  8d442404    lea      eax, [esp+0x4 (var_4)]
0040147f  50          push     eax (var_4) (var_c)
00401480  684c4b4008 push     0x404b4c (var_10)
00401485  e804fbffff call     j_sub_401550
0040148a  8b44240c    mov      eax, dword [esp+0xc (var_4)]
0040148e  83c40c     add      esp, 0xc
00401491  05226dac13 add      eax, 0x13ac6d22
00401496  356898f39b xor      eax, 0x9bf39868
0040149b  2d1acbba61 sub      eax, 0x61acb1a
004014a0  35c40b66c6 xor      eax, 0x4a8b66c
004014a5  05c3e0eb74 add      eax, 0x74e0ebc3
004014aa  333da72513 xor      eax, 0x1325a73d
004014af  058e001702 add      eax, 0x217008e
004014b4  358e001702 xor      eax, 0x217008e
004014b9  3defbeadde cmp      eax, 0xdeadbeef
004014be  7511       jne      0x4014d1 ("h!Ke")
  
```

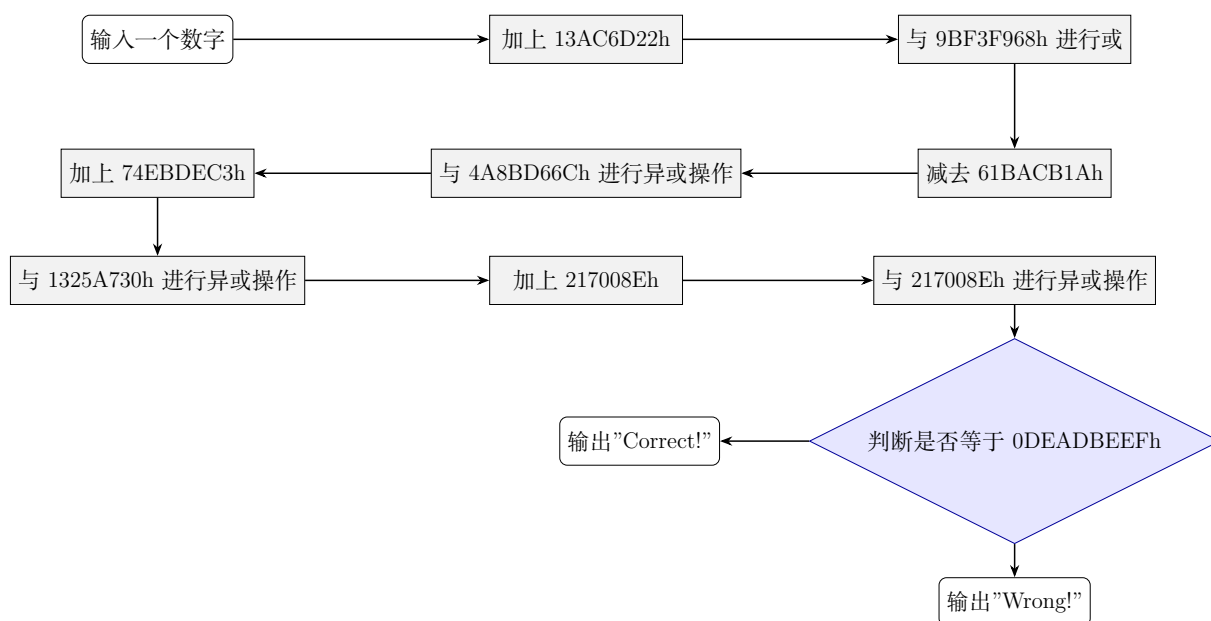
图 11: task1 反汇编代码

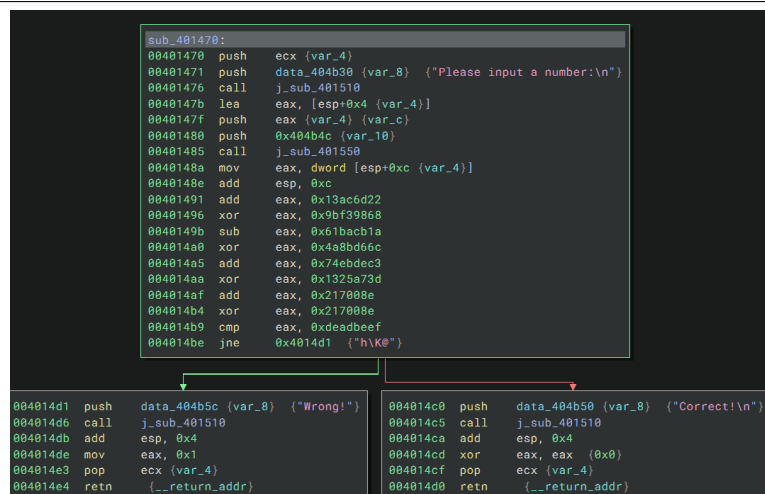
```

sub_401470:
00401470 push     ecx (var_4)
00401471 push     data_404b30 (var_8) ("Please input a number:\n")
00401476 call     j_sub_401510
0040147b lea      eax, [esp+0x4 (var_4)]
0040147f push     eax (var_4) (var_c)
00401480 push     0x404b4c (var_10)
00401485 call     j_sub_401550
0040148a mov      eax, dword [esp+0xc (var_4)]
0040148e add      esp, 0xc
00401491 add      eax, 0x13ac6d22
00401496 xor      eax, 0x9bf39868
0040149b sub      eax, 0x61acb1a
004014a0 xor      eax, 0x4a8b66c
004014a5 add      eax, 0x74e0ebc3
004014aa add      eax, 0x1325a73d
004014af add      eax, 0x217008e
004014b4 xor      eax, 0x217008e
004014b9 cmp      eax, 0xdeadbeef
004014be jne      0x4014d1 ("h!Ke")
004014d1 push     data_404b5c (var_8) ("Wrong!")
004014d6 call     j_sub_401510
004014db add      esp, 0x4
004014de mov      eax, 0x1
004014e3 pop      ecx (var_4)
004014e4 retn     [...return_addr]
004014c0 push     data_404b50 (var_8) ("Correct!\n")
004014c5 call     j_sub_401510
004014ca add      esp, 0x4
004014cd xor      eax, eax (0x0)
004014cf pop      ecx (var_4)
004014d0 retn     [...return_addr]
  
```

图 12: task1 图形化显示

3.2.2 逆向分析





首先输出一个“Please input a number!”输入一个数字，将其赋值到寄存器 `eax` 中。然后开始计算，先将 `eax` 的值加上 `13AC6D22h`，然后再与 `9BF39868h` 进行异或操作，然后减去 `61BACB1Ah`，再与 `4A8BD66Ch` 进行异或操作，接着加上 `74EBDEC3h`，接着与 `1325A73Dh` 进行异或操作，然后加上 `217008Eh`，再与 `217008Eh` 进行异或操作，最后与 `0DEADBEEFh` 进行比较，如果相等的话就输出“Correct!”，反之输出“Wrong!”

3.3.3 成功截图

由上面的计算流程可知，最后结果是十六进制 `0xD092BFE7`，其十进制对应为：

- 有符号：-795688985
- 无符号：3499278311

所以输入 3499278311 和 -795688985 得到截图：

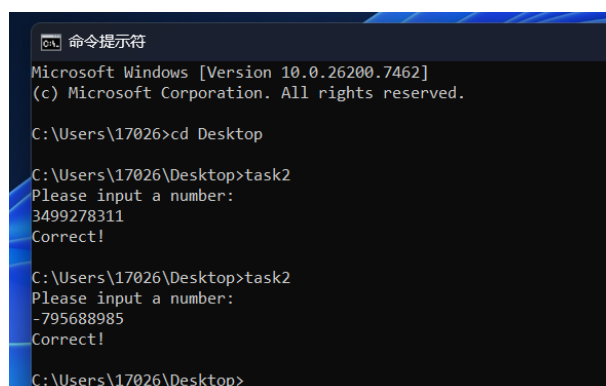


图 13: task2.exe 的成功截图

得到 correct，结果正确。