

# 汇编语言与逆向技术实验报告

## Lab6 - RE Challenge #2

学号：2112492 姓名：刘修铭 专业：信息安全

### 一、实验目的

- 1.熟悉静态反汇编工具 IDA Freeware;
- 2.熟悉反汇编代码、伪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;
- 4.运用熟悉的编程语言，实现简单的脚本编写。

### 二、实验原理

- 1.通过 IDA Freeware 得到 ELF 文件的反汇编代码，如图 1 和图 2 所示。

```
.text:0000000000007CA ; ===== S U B R O U T I N E =====
.text:0000000000007CA ; Attributes: bp-based frame
.text:0000000000007CA ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:0000000000007CA public main
.text:0000000000007CA main proc near ; DATA XREF: _start+1D10
.text:0000000000007CA
.text:0000000000007CA var_60 = qword ptr -60h
.text:0000000000007CA var_54 = dword ptr -54h
.text:0000000000007CA var_4C = dword ptr -4Ch
.text:0000000000007CA endptr = qword ptr -48h
.text:0000000000007CA var_40 = qword ptr -40h
.text:0000000000007CA var_38 = qword ptr -38h
.text:0000000000007CA var_30 = qword ptr -30h
.text:0000000000007CA var_28 = qword ptr -28h
.text:0000000000007CA var_20 = qword ptr -20h
.text:0000000000007CA var_18 = qword ptr -18h
.text:0000000000007CA var_10 = qword ptr -10h
.text:0000000000007CA var_8 = qword ptr -8
.text:0000000000007CA ; __unwind {
.text:0000000000007CA push rbp
.text:0000000000007CB mov rbp, rsp
.text:0000000000007CE sub rsp, 60h
.text:0000000000007D0 mov [rbp+var_54], edi
.text:0000000000007D5 mov [rbp+var_60], rsi
.text:0000000000007D9 mov rax, fs:28h
.text:0000000000007E2 mov [rbp+var_8], rax
.text:0000000000007E6 xor eax, eax
.text:0000000000007E8 cmp [rbp+var_54], 5
.text:0000000000007EC jz short loc_804
.text:0000000000007EE lea rdi, s ; "argc nonono"
.text:0000000000007F5 call _puts
.text:0000000000007FA mov edi, 1 ; status
.text:0000000000007FF call _exit
.text:000000000000804 ;
.text:000000000000804 loc_804:
.text:000000000000804 mov rax, [rbp+var_60] ; CODE XREF: main+221j
.text:000000000000808 add rax, 20h ; ' '
.text:00000000000080C mov rax, [rax]
.text:00000000000080F lea rcx, [rbp+endptr]
.text:000000000000813 mov edx, 10h ; base
.text:000000000000818 mov rsi, rcx ; endptr
.text:00000000000081B mov rdi, rax ; nptr
.text:00000000000081E call _strtol
.text:000000000000823 mov edx, eax
.text:000000000000825 mov eax, 6543h
.text:00000000000082A sub edx, eax
.text:00000000000082C mov eax, edx
.text:00000000000082E mov [rbp+var_4C], eax
.text:000000000000831 mov eax, [rbp+var_4C]
.text:000000000000834 mov edi, eax
.text:000000000000836 call f
.text:00000000000083B mov [rbp+var_28], rax
.text:00000000000083F mov rax, [rbp+var_60]
.text:000000000000843 add rax, 8
.text:000000000000847 mov rax, [rax]
.text:00000000000084A lea rcx, [rbp+var_40]
.text:00000000000084E mov edx, 10h ; base
.text:000000000000853 mov rsi, rcx ; endptr
.text:000000000000856 mov rdi, rax ; nptr
```

图 1 reverse 的反汇编代码

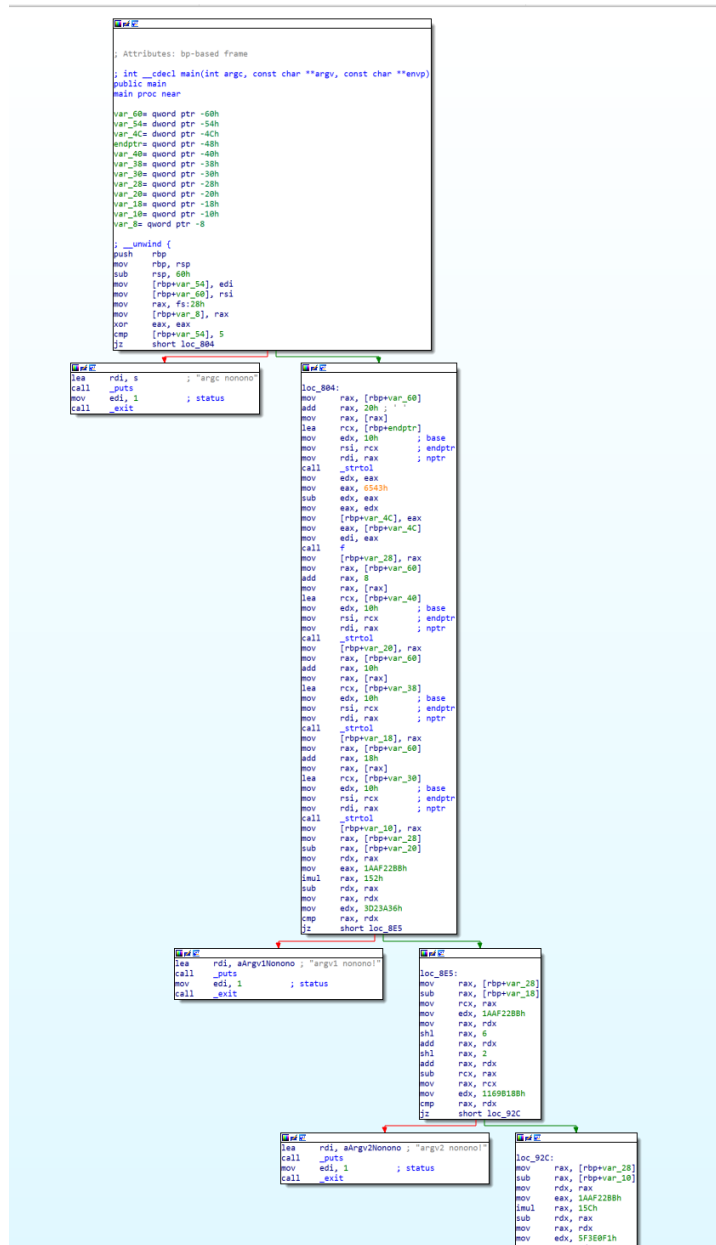


图 2 reverse 的反汇编代码的图形化显示

2.使用 IDA 的反编译功能（F5）得到伪代码，如图 3 所示。

```

1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3     unsigned int v4; // [rsp+14h] [rbp-4Ch]
4     char *endptr; // [rsp+18h] [rbp-48h] BYREF
5     char *v6; // [rsp+20h] [rbp-40h] BYREF
6     char *v7; // [rsp+28h] [rbp-38h] BYREF
7     char *v8; // [rsp+30h] [rbp-30h] BYREF
8     __int64 v9; // [rsp+38h] [rbp-28h]
9     __int64 v10; // [rsp+40h] [rbp-20h]
10    __int64 v11; // [rsp+48h] [rbp-18h]
11    __int64 v12; // [rsp+50h] [rbp-10h]
12    unsigned __int64 v13; // [rsp+58h] [rbp-8h]
13
14    v13 = __readfsqword(0x28u);
15    if ( argc != 5 )
16    {
17        puts("argc nonono");
18        exit(1);
19    }
20    v4 = strtol(argv[4], &endptr, 16) - 25923;
21    v9 = f(v4);
22    v10 = strtol(argv[1], &v6, 16);
23    v11 = strtol(argv[2], &v7, 16);
24    v12 = strtol(argv[3], &v8, 16);
25    if ( v9 - v10 != 0x233F0E151CCLL )
26    {
27        puts("argv1 nonono!");
28        exit(1);
29    }
30    if ( v9 - v11 != 0x1B45F81A32LL )
31    {
32        puts("argv2 nonono!");
33        exit(1);
34    }
35    if ( v9 - v12 != 0x244C071725LL )
36    {
37        puts("argv3 nonono!");
38        exit(1);
39    }
40    if ( (int)v4 + v12 + v11 + v10 != 0x13A31412F8CLL )
41    {
42        puts("argv sum nonono!");
43        exit(1);
44    }
45    puts("well done!decode your argv!");
46    return 0;
47 }

```

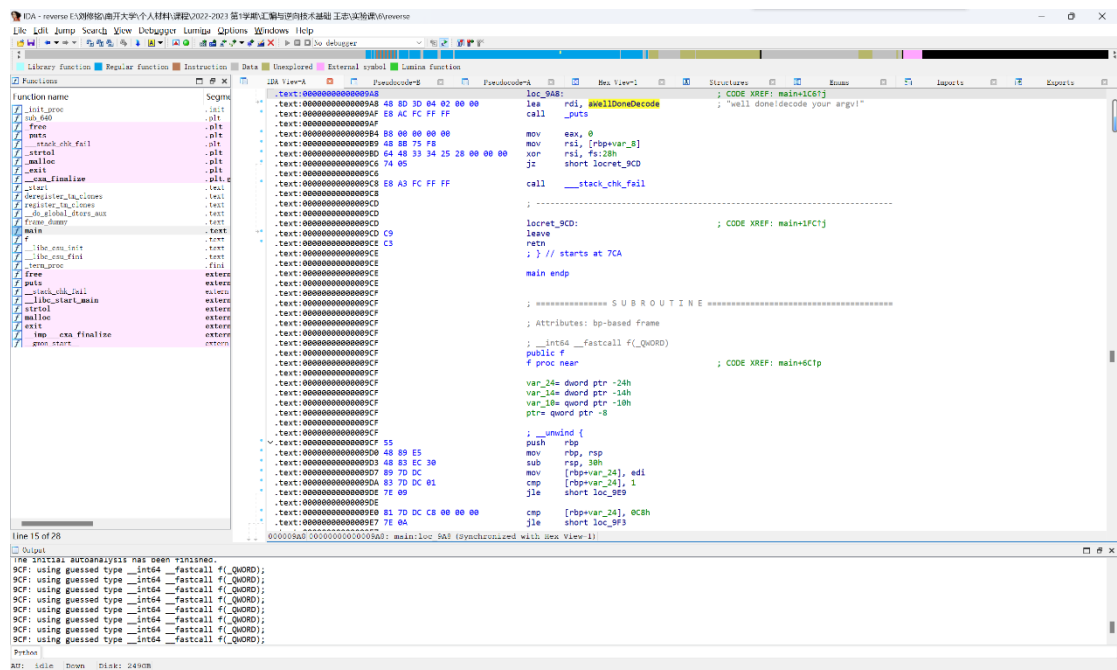
图 3 reverse 的反编译伪代码

3.对汇编代码和反编译伪代码的计算过程、条件判断、分支结构等信息进行分析，逆向推出待解方程组。

4.编写脚本实现暴力破解，解出方程组，得到参数“V9”、“V10”、“V11”、“V12”的正确取值，完成逆向分析挑战！

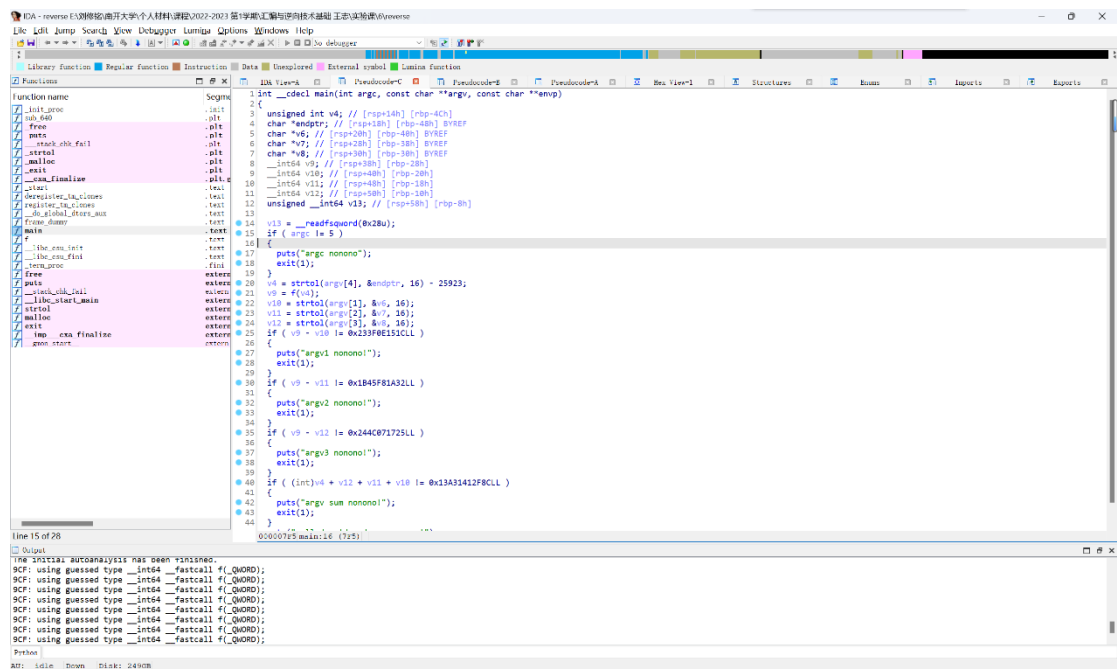
### 三、获取反汇编代码

通过 IDA Freeware 可以得到二进制代码的反汇编代码及反汇编伪代码：



The screenshot shows the IDA Freeware interface with the 'main' function selected. The assembly view displays instructions such as `lea rdi, wellDoneDecode`, `call puts`, `mov eax, 0`, `mov rax, [rbp+var_8]`, `xor rsi, fs:28h`, and `short locret_9CD`. The function name list on the left includes `__init_func`, `__free`, `__puts`, `__strchr_rbx_fail`, `__strtol`, `__malloc`, `__main_finalize`, `__atexit`, `__atexit_rbx_fail`, `__register_tls_callbacks`, `__do_global_ctors_aux`, `__frame_dummy`, `__main`, `__lib_tls_init`, `__lib_tls_fini`, `__term_proc`, `__free`, `__puts`, `__strchr_rbx_fail`, `__lib_tls_init`, `__strtol`, `__malloc`, `__exit`, `__imp_cxx_finalize`, and `__gnu_start`. The output window at the bottom shows the initial autoanalysis results.

### 反汇编代码



The screenshot shows the IDA Freeware interface with the 'main' function selected. The assembly view displays instructions such as `int __cdecl main(int argc, const char **argv, const char **envp)`, `unsigned int v4; // [rsp+140h] [rbp-4Ch]`, `char *endptr; // [rsp+180h] [rbp-48h] BYREF`, `char *v6; // [rsp+200h] [rbp-40h] BYREF`, `char *v7; // [rsp+240h] [rbp-38h] BYREF`, `char *v8; // [rsp+280h] [rbp-30h] BYREF`, `__int64 v10; // [rsp+400h] [rbp-20h]`, `__int64 v11; // [rsp+440h] [rbp-18h]`, `__int64 v12; // [rsp+480h] [rbp-10h]`, `unsigned __int64 v13; // [rsp+580h] [rbp-8h]`, `v13 = __readfsqword(0x28u);`, `if (argc != 5)`, `puts("arg nonono");`, `exit(1);`, `v4 = strtol(argv[4], &endptr, 16) - 25923;`, `v10 = v4;`, `v10 = strtol(argv[1], &v6, 16);`, `v11 = strtol(argv[2], &v7, 16);`, `v12 = strtol(argv[3], &v8, 16);`, `if (v9 - v10 != 0x239F0E151CCL)`, `puts("arg1 nonono!");`, `exit(1);`, `if (v9 - v11 != 0x1B45F81A32LL)`, `puts("arg2 nonono!");`, `exit(1);`, `if (v9 - v12 != 0x244C071725LL)`, `puts("arg3 nonono!");`, `exit(1);`, `if ((int)v4 + v12 + v11 != 0x13A31412F8CL)`, `puts("arg sum nonono!");`, `exit(1);`, and `return 0;`. The function name list on the left includes `__init_func`, `__free`, `__puts`, `__strchr_rbx_fail`, `__strtol`, `__malloc`, `__main_finalize`, `__atexit`, `__atexit_rbx_fail`, `__register_tls_callbacks`, `__do_global_ctors_aux`, `__frame_dummy`, `__main`, `__lib_tls_init`, `__lib_tls_fini`, `__term_proc`, `__free`, `__puts`, `__strchr_rbx_fail`, `__lib_tls_init`, `__strtol`, `__malloc`, `__exit`, `__imp_cxx_finalize`, and `__gnu_start`. The output window at the bottom shows the initial autoanalysis results.

### 反汇编伪代码

## 四、逆向过程分析

### 1. “argc nonono”

```
if ( argc != 5 )
{
    puts("argc nonono");
    exit(1);
}
```

此代码块的功能是：

- 将 argc 和 5 比较
- 若 输入的参数个数为 4，进入下一步； 否则报错"argc nonono"

### 2.v4

```
v4 = strtol(argv[4], &endptr, 16) - 25923;
```

此代码块的功能是：

- 将输入的第四个参数（十六进制）减去 0x6543 后存入 v4

### 3.v9

```
v4 = strtol(a
v9 = f(v4);
v10 = strtol(
```

此代码块的功能是：

- 将 v4 的值作为参数调用函数 f，并将函数的返回值赋值给 v9

```
1 int64 __fastcall f(int a1)
2 {
3     int i; // [rsp+1Ch] [rbp-14h]
4     int64 v3; // [rsp+20h] [rbp-10h]
5     _QWORD *ptr; // [rsp+28h] [rbp-8h]
6
7     if ( a1 <= 1 || a1 > 200 )
8         return 0LL;
9     ptr = malloc(8LL * a1);
10    *ptr = 1LL;
11    ptr[1] = 1LL;
12    v3 = 0LL;
13    for ( i = 2; i < a1; ++i )
14    {
15        ptr[i] = ptr[i - 1] + ptr[i - 2];
16        v3 = ptr[i];
17    }
18    free(ptr);
19    return v3;
20 }
```

- 查阅函数 f 的代码后可知，函数 f 为斐波那契数列的求值函数，其中参数 v4 表示加和次数。

4.v10、v11、v12 分别表示传入的参数的第一、二、三位（十六进制）

```
v10 = strtol(argv[1], &v6, 16);
v11 = strtol(argv[2], &v7, 16);
v12 = strtol(argv[3], &v8, 16);
```

此代码块的功能是：

- 将 v10、v11、v12 分别传入参数的第一、二、三位（十六进制）

## 5.if 语句

```
if ( v9 - v10 != 151381742876LL )
{
    puts("argv1 nonono!");
    exit(1);
}
if ( v9 - v11 != 117138004530LL )
{
    puts("argv2 nonono!");
    exit(1);
}
if ( v9 - v12 != 155894355749LL )
{
    puts("argv3 nonono!");
    exit(1);
}
if ( v4 + v12 + v11 + v10 != 1349446086540LL )
{
    puts("argv sum nonono!");
    exit(1);
}
```

此代码块的功能是：

- 将 v10、v11、v12（十六进制）分别与 v9 进行运算，得到相应的值

综上，得到如下运算关系：

```
f(a4-0x6543) - a1      = 0x233F0E151C
f(a4-0x6543) - a2      = 0x1B45F81A32
f(a4-0x6543) - a3      = 0x244C071725
a4-0x6543 + a3 + a2 + a1 = 0x13A31412F8C
a4-0x6543 + 3* fib(a4-0x6543) = sum()
```

## 五、参数求解

由逆向分析可知，需结合斐波那契数列的求和公式与原始代码中的限定关系联合求解四个参数。

脚本代码如下：

```
def f(n):
    a=1
    b=1
    i=2
    while i<n:
        a,b = b,a+b
        i+=1
    return b
sum = 0x233F0E151C + 0x1B45F81A32 + 0x244C071725 + 0x13A31412F8C
for v4 in range(58,201):
    v9 = f(v4)
    if v4+3*v9 == sum:
        print(v4)
        print(bytes.fromhex(hex(v9-0x233F0E151C)[2:])+
bytes.fromhex(hex(v9-0x1B45F81A32)[2:]) +bytes.fromhex(hex(v9-
0x244C071725)[2:]) + bytes.fromhex(hex(v4+0x6543)[2:])))
        break
```

运行可得 $v4 = 58$ 。

故 $v9 = f(54) = 591286729879 = 0x89AB6F7C97$

由四个 if 语句可知：

$$v10 = v9 + 0x233F0E151C = 0x666C61677B$$

$$v11 = v9 + 0x1B45F81A32 = 0x6e65776265$$

$$v12 = v9 + 0x244C071725 = 0x655f686572$$

即，四个参数的值分别为：

$$v9 = 0x89AB6F7C97$$

$$v10 = 0x666C61677B$$

$$v11 = 0x6e65776265$$

$$v12 = 0x655f686572$$