

汇编语言与逆向技术实验报告

Lab6- RE Challenge #2

学号：2112514 姓名：辛浩然 专业：信息安全、法学

一、实验步骤

1. 通过 IDA Freeware 得到 ELF 文件的反汇编代码。
2. 使用 IDA 的反编译功能（F5）得到伪代码。
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！

二、反汇编代码

```
.text:00000000000007CA ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00000000000007CA public main
.text:00000000000007CA proc near ; DATA XREF: _start+1Df0
.text:00000000000007CA
.text:00000000000007CA var_60 = qword ptr -60h
.text:00000000000007CA var_54 = dword ptr -54h
.text:00000000000007CA var_4C = dword ptr -4Ch
.text:00000000000007CA endptr = qword ptr -48h
.text:00000000000007CA var_40 = qword ptr -40h
.text:00000000000007CA var_38 = qword ptr -38h
.text:00000000000007CA var_30 = qword ptr -30h
.text:00000000000007CA var_28 = qword ptr -28h
.text:00000000000007CA var_20 = qword ptr -20h
.text:00000000000007CA var_18 = qword ptr -18h
.text:00000000000007CA var_10 = qword ptr -10h
.text:00000000000007CA var_8 = qword ptr -8
.text:00000000000007CA ; __unwind {
.text:00000000000007CA push rbp
.text:00000000000007CB mov rbp, rsp
.text:00000000000007CE sub rsp, 60h
.text:00000000000007D2 mov [rbp+var_54], edi
.text:00000000000007D5 mov [rbp+var_60], rsi
.text:00000000000007D9 mov rax, fs:28h
.text:00000000000007E2 mov [rbp+var_8], rax
.text:00000000000007E6 xor eax, eax
.text:00000000000007E8 cmp [rbp+var_54], 5
.text:00000000000007EC jz short loc_804
.text:00000000000007EE lea rdi, s ; "argc nonono"
.text:00000000000007F5 call _puts
.text:00000000000007FA mov edi, 1 ; status
.text:00000000000007FF call _exit
.text:0000000000000804 ; -----
.text:0000000000000804 loc_804: mov rax, [rbp+var_60] ; CODE XREF: main+22fj
.text:0000000000000808 add rax, 20h ; '\t'
.text:000000000000080C mov rax, [rax]
.text:000000000000080F lea rcx, [rbp+endptr]
.text:0000000000000813 mov edx, 10h ; base
.text:0000000000000818 mov rsi, rcx ; endptr
.text:000000000000081B mov rdi, rax ; nptr
.text:000000000000081E call _strtol
.text:0000000000000823 mov edx, eax
.text:0000000000000825 mov eax, 6543h
.text:000000000000082A sub edx, eax
.text:000000000000082C mov eax, edx
.text:000000000000082E mov [rbp+var_4C], eax
000007CA 00000000000007CA: main (Synchronized with Hex View-1)
```

```

.text:0000000000000831      mov     eax, [rbp+var_4C]
.text:0000000000000834      mov     edi, eax
.text:0000000000000836      call    f
.text:0000000000000838      mov     [rbp+var_28], rax
.text:000000000000083F      mov     rax, [rbp+var_60]
.text:0000000000000843      add     rax, 8
.text:0000000000000847      mov     rax, [rax]
.text:000000000000084A      lea     rcx, [rbp+var_40]
.text:000000000000084E      mov     edx, 10h           ; base
.text:0000000000000853      mov     rsi, rcx           ; endptr
.text:0000000000000856      mov     rdi, rax           ; nptr
.text:0000000000000859      call    _strtol
.text:000000000000085E      mov     [rbp+var_20], rax
.text:0000000000000862      mov     rax, [rbp+var_60]
.text:0000000000000866      add     rax, 10h
.text:000000000000086A      mov     rax, [rax]
.text:000000000000086D      lea     rcx, [rbp+var_38]
.text:0000000000000871      mov     edx, 10h           ; base
.text:0000000000000876      mov     rsi, rcx           ; endptr
.text:0000000000000879      mov     rdi, rax           ; nptr
.text:000000000000087C      call    _strtol
.text:0000000000000881      mov     [rbp+var_18], rax
.text:0000000000000885      mov     rax, [rbp+var_60]
.text:0000000000000889      add     rax, 18h
.text:000000000000088D      mov     rax, [rax]
.text:0000000000000890      lea     rcx, [rbp+var_30]
.text:0000000000000894      mov     edx, 10h           ; base
.text:0000000000000899      mov     rsi, rcx           ; endptr
.text:000000000000089C      mov     rdi, rax           ; nptr
.text:000000000000089F      call    _strtol
.text:00000000000008A4      mov     [rbp+var_10], rax
.text:00000000000008A8      mov     rax, [rbp+var_28]
.text:00000000000008AC      sub     rax, [rbp+var_20]
.text:00000000000008B0      mov     rdx, rax
.text:00000000000008B3      mov     eax, 1AAF22BBh
.text:00000000000008B8      imul    rax, 152h
.text:00000000000008BF      sub     rdx, rax
.text:00000000000008C2      mov     rax, rdx
.text:00000000000008C5      mov     edx, 3D23A36h
.text:00000000000008CA      cmp     rax, rdx
.text:00000000000008CD      jz      short loc_8E5
.text:00000000000008CF      lea     rdi, aArgv1Nonono ; "argv1 nonono!"
.text:00000000000008D6      call    _puts
.text:00000000000008D8      mov     edi, 1             ; status
.text:00000000000008DE      call    _exit
.text:00000000000008E5      ; -----
.text:00000000000008E5      ;

.text:00000000000008E5      loc_8E5:      ; CODE XREF: main+1031j
.text:00000000000008E5      mov     rax, [rbp+var_28]
.text:00000000000008E9      sub     rax, [rbp+var_18]
.text:00000000000008ED      mov     rcx, rax
.text:00000000000008F0      mov     edx, 1AAF22BBh
.text:00000000000008F5      mov     rax, rdx
.text:00000000000008F8      shl     rax, 6
.text:00000000000008FC      add     rax, rdx
.text:00000000000008FF      shl     rax, 2
.text:0000000000000903      add     rax, rdx
.text:0000000000000906      sub     rcx, rax
.text:0000000000000909      mov     rax, rcx
.text:000000000000090C      mov     edx, 1169B18Bh
.text:0000000000000911      cmp     rax, rdx
.text:0000000000000914      jz      short loc_92C
.text:0000000000000916      lea     rdi, aArgv2Nonono ; "argv2 nonono!"
.text:000000000000091D      call    _puts
.text:0000000000000922      mov     edi, 1             ; status
.text:0000000000000927      call    _exit
.text:000000000000092C      ; -----
.text:000000000000092C      loc_92C:      ; CODE XREF: main+14A1j
.text:000000000000092C      mov     rax, [rbp+var_28]
.text:0000000000000930      sub     rax, [rbp+var_10]
.text:0000000000000934      mov     rdx, rax
.text:0000000000000937      mov     eax, 1AAF22BBh
.text:000000000000093C      imul    rax, 15Ch
.text:0000000000000943      sub     rdx, rax
.text:0000000000000946      mov     rax, rdx
.text:0000000000000949      mov     edx, 5F3E0F1h
.text:000000000000094E      cmp     rax, rdx
.text:0000000000000951      jz      short loc_969
.text:0000000000000953      lea     rdi, aArgv3Nonono ; "argv3 nonono!"
.text:000000000000095A      call    _puts
.text:000000000000095F      mov     edi, 1             ; status
.text:0000000000000964      call    _exit
.text:0000000000000969      ; -----
.text:0000000000000969      loc_969:      ; CODE XREF: main+1871j
.text:0000000000000969      mov     rdx, [rbp+var_20]
.text:000000000000096D      mov     rax, [rbp+var_18]
.text:0000000000000971      add     rdx, rax
.text:0000000000000974      mov     rax, [rbp+var_10]
.text:0000000000000978      add     rdx, rax
.text:000000000000097B      mov     eax, [rbp+var_4C]
.text:000000000000097E      cdq     rdx
.text:0000000000000980      add     rdx, rax
.text:0000000000000983      mov     rax, 13A31412F8Ch
.text:000000000000098D      cmp     rdx, rax
.text:0000000000000990      jz      short loc_9A8
.text:0000000000000992      lea     rdi, aArgvSumNonono ; "argv sum nonono!"
.text:0000000000000999      call    _puts
.text:000000000000099E      mov     edi, 1             ; status
.text:00000000000009A3      call    _exit
.text:00000000000009A8      ; -----
.text:00000000000009A8      loc_9A8:      ; CODE XREF: main+1C61j
.text:00000000000009A8      lea     rdi, aWellDoneDecode ; "well done!decode your argv!"
.text:00000000000009AF      call    _puts
.text:00000000000009B4      mov     eax, 0
.text:00000000000009B9      mov     rsi, [rbp+var_8]
.text:00000000000009BD      xor     rsi, fs:28h
.text:00000000000009C6      jz      short locret_9CD
.text:00000000000009C8      call    __stack_chk_fail
.text:00000000000009CD      locret_9CD:      ; CODE XREF: main+1FC1j
.text:00000000000009CD      leave
.text:00000000000009CE      ret     ; // starts at 7CA
.text:00000000000009CE      main          endp
.text:00000000000009CF

```

三、反编译代码

```

int __cdecl main(int argc, const char **argv, const char **envp)
{
    unsigned int v4; // [rsp+14h] [rbp-4Ch]
    char *endptr; // [rsp+18h] [rbp-48h] BYREF
    char *v6; // [rsp+20h] [rbp-40h] BYREF
    char *v7; // [rsp+28h] [rbp-38h] BYREF
    char *v8; // [rsp+30h] [rbp-30h] BYREF
    __int64 v9; // [rsp+38h] [rbp-28h]
    __int64 v10; // [rsp+40h] [rbp-20h]
    __int64 v11; // [rsp+48h] [rbp-18h]
    __int64 v12; // [rsp+50h] [rbp-10h]
    unsigned __int64 v13; // [rsp+58h] [rbp-8h]

    v13 = __readfsqword(0x28u);
    if ( argc != 5 )
    {
        puts("argc nonono");
        exit(1);
    }
    v4 = strtol(argv[4], &endptr, 16) - 25923;
    v9 = f(v4);
    v10 = strtol(argv[1], &v6, 16);
    v11 = strtol(argv[2], &v7, 16);
    v12 = strtol(argv[3], &v8, 16);
    if ( v9 - v10 != 0x233F0E151C11 )
    {
        puts("argv1 nonono!");
        exit(1);
    }
    if ( v9 - v11 != 0x1B45F81A3211 )
    {
        puts("argv2 nonono!");
        exit(1);
    }
    if ( v9 - v12 != 0x244C07172511 )
    {
        puts("argv3 nonono!");
        exit(1);
    }
    if ( (int)v4 + v12 + v11 + v10 != 0x13A31412F811 )
    {
        puts("argv sum nonono!");
        exit(1);
    }
    puts("well done! decode your argv!");
    return 0;
}

```

00000951:main:35 (951)

```

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 }

```

四、 逆向分析

```

; int __cdecl main(int argc, const char **argv, const char **envp)
public main
main      proc near          ; DATA XREF: _start+1D1fo

var_60    = qword ptr -60h
var_54    = dword ptr -54h
var_4C    = dword ptr -4Ch
endptr    = qword ptr -48h
var_40    = qword ptr -40h
var_38    = qword ptr -38h
var_30    = qword ptr -30h
var_28    = qword ptr -28h
var_20    = qword ptr -20h
var_18    = qword ptr -18h
var_10    = qword ptr -10h
var_8     = qword ptr -8

; __unwind {
push      rbp
mov       rbp, rsp
sub       rsp, 60h
mov       [rbp+var_54], edi
mov       [rbp+var_60], rsi
mov       rax, fs:28h
mov       [rbp+var_8], rax
xor       eax, eax
cmp       [rbp+var_54], 5
jz        short loc_804
lea       rdi, s              ; "argc nonono"
call      _puts
mov       edi, 1              ; status
call      _exit

```

函数调用约定为 `_cdecl`，从右向左传参，由 caller 清理堆栈。

```

push  rbp
mov   rbp, rsp
sub   rsp, 60h

```

程序执行时，首先执行启动函数。当所有的初始化操作完成后，启动函数调用 `main` 函数。执行 `main` 函数时，`main` 函数需要有自己的栈帧，进行下述操作：（1）保存旧的帧指针；（2）创建新的帧指针，即直接把当前的栈顶地址作为当前函数的帧指针；（3）`rsp` 减小 96 个字节，在栈中分配局部变量的空间。

对于 `main` 函数的参数：`argc` 表示传入 `main` 函数的参数个数；第二个参数 `char** argv`，是字符串数组，用来存放指向的字符串参数的指针数组，每一个元素指向一个参数。各成员含义如下：`argv[0]`：指向程序运行的全路径名；`argv[1]`：指向执行程序名后的第一个字符串，表示真正传入的第一个参数；`argv[2]`：指向执行程序名后的第二个字符串，表示传入的第二个参数……所以需要输入的 `main` 函数的参数个数应该是 `argc-1` 个；`envp` 是系统的环境变量。

```

mov  [rbp+var_54], edi
mov  [rbp+var_60], rsi

```

随后，把参数保存到变量中。`main` 函数第一个参数是 `argc`，使用 `edi` 传递，存到 `rbp-54` 的位置；第二个参数是 `argv`，即命令行输入的参数序列，为 `char**` 类型，存到 `rbp-60` 的位置。

```
mov rax, fs:28h
mov [rbp+var_8], rax
xor eax, eax
```

程序使用了 **canary 保护机制**：从 `fs` 寄存器偏移为 `0x28` 的位置中取出 8 字节放入 `rax` 寄存器中，`rax` 会将其放在 `rbp-8` 的位置，最后将 `rax` 的值清零。程序接收输入后会进行检查，如果 **canary** 被覆盖就会执行 `stack_chk_fail` 函数，从而阻止程序继续运行，以防止程序被栈溢出攻击。

判断 `argc` 值是否为 5，即命令行输入的参数数量是否为 4，如果不相等，输出“`argc nonono`”，程序结束。如果相等，跳转 `804` 地址处。这里说明：**命令行输入的参数数量为 4！**

```
loc_804:                                ; CODE XREF: main+221j
mov rax, [rbp+var_60]
add rax, 20h ; ' '
mov rax, [rax]
lea rcx, [rbp+endptr]
mov edx, 10h ; base
mov rsi, rcx ; endptr
mov rdi, rax ; nptr
call _strtol
```

以 `rbp` 的值为地址并向上偏移 `60h` 字节，该位置的值，即 `argv[0]` 的地址，赋值给 `rax`。该地址加 32，即 `argv[0]` 的地址后偏移 32 字节，即 `argv[3]` 的地址。将其指向的值赋值给 `rax`，即 `argv[3]` 首字符的地址，类型为 `char*` 类型。

随后调用 **`strtol` 函数**。函数有三个参数：`base`、`endptr` 和 `nptr`。其功能是将输入的字符串根据给定的 `base` 转换为 `base` 进制的长整数，并将该长整数转换为十进制数返回。输入的字符串为 `nptr`，即为 `argv[3]`；`base` 值为 16。所以该步将输入的第四个字符串转为 16 进制长整数，返回的十进制数存至 `eax` 中。

```
mov edx, eax
mov eax, 6543h
sub edx, eax
mov eax, edx
mov [rbp+var_4C], eax
mov eax, [rbp+var_4C]
mov edi, eax
call f
```

将转换后的数减去 `6543h`，存到 `rbp-4Ch` 地址处。将转换后的数赋值给 `edi`，作为接下来调用函数的参数。调用 **`f` 函数**。


```

__int64 __fastcall f(int a1)
{
    int i; // [rsp+1Ch] [rbp-14h]
    __int64 v3; // [rsp+20h] [rbp-10h]
    __QWORD *ptr; // [rsp+28h] [rbp-8h]

    if ( a1 <= 1 || a1 > 200 )
        return 0LL;
    ptr = malloc(8LL * a1);
    *ptr = 1LL;
    ptr[1] = 1LL;
    v3 = 0LL;
    for ( i = 2; i < a1; ++i )
    {
        ptr[i] = ptr[i - 1] + ptr[i - 2];
        v3 = ptr[i];
    }
    free(ptr);
    return v3;
}

```

对函数 **f** 的反编译代码分析，其功能为通过对 **f** 函数分析，实则就是斐波那契数列求解。如果参数 **a1** 小于等于 **1** 或者大于 **200**，返回 **0**；其他情况下，返回值是斐波那契数列（从 **1,1,2,3,5,8...** 开始）的第 **a1** 项。

```

mov     [rbp+var_28], rax
mov     rax, [rbp+var_60]
add     rax, 8
mov     rax, [rax]
lea     rcx, [rbp+var_40]
mov     edx, 10h           ; base
mov     rsi, rcx           ; endptr
mov     rdi, rax           ; nptr
call    _strtol

```

函数 **f** 调用结束后，将返回值存到 **rbp-28h** 地址处。随后，类似 **argv[3]** 的处理方法，得到 **argv[1]**，并作为参数调用 **strtol** 函数。将输入的字符串转为 **16** 进制长整数。

```

mov     [rbp+var_20], rax
mov     rax, [rbp+var_60]
add     rax, 10h
mov     rax, [rax]
lea     rcx, [rbp+var_38]
mov     edx, 10h           ; base
mov     rsi, rcx           ; endptr
mov     rdi, rax           ; nptr
call    _strtol

```

将返回的十进制数存到 **rbp-20h** 地址处。随后，得到 **argv[2]**，并作为参数调用 **strtol** 函数。将输入的第二个字符串转为十六进制长整数。

```

mov     [rbp+var_18], rax
mov     rax, [rbp+var_60]
add     rax, 18h
mov     rax, [rax]
lea     rcx, [rbp+var_30]
mov     edx, 10h           ; base
mov     rsi, rcx           ; endptr
mov     rdi, rax           ; nptr
call    _strtol

```

将刚返回的数存到 `rbp-18h` 地址处。随后，得到 `argv[3]`，并作为参数调用 `strtol` 函数。将输入的第三个字符串转为十六进制长整数。

```

mov     [rbp+var_10], rax
mov     rax, [rbp+var_28]
sub     rax, [rbp+var_20]
mov     rdx, rax
mov     eax, 1AAF22BBh
imul    rax, 152h
sub     rdx, rax
mov     rax, rdx
mov     edx, 3D23A36h
cmp     rax, rdx
jz      short loc_8E5
lea     rdi, aArgv1Nonono ; "argv1 nonono!"
call    _puts
mov     edi, 1             ; status
call    _exit

```

将返回的十进制数存到 `rbp-10h` 地址处。

对应反编译代码，`rbp-28h` 地址至 `rbp-10h` 地址，分别存放 `v9`、`v10`、`v11`、`v12` 的值，其中 **`v9` 是 `f(v4)`**，即斐波那契数列第 `v4` 项。**`v4+25923`、`v10`、`v11`、`v12` 分别为输入的第四、一、二、三字符串转为的长整数。**

```

mov     rax, [rbp+var_28]
sub     rax, [rbp+var_20]
mov     rdx, rax
mov     eax, 1AAF22BBh
imul    rax, 152h
sub     rdx, rax
mov     rax, rdx
mov     edx, 3D23A36h
cmp     rax, rdx
jz      short loc_8E5
lea     rdi, aArgv1Nonono ; "argv1 nonono!"
call    _puts
mov     edi, 1             ; status
call    _exit

```

继续分析反汇编代码：

取出 `v9`，减去 `v10`，存在 `rdx` 中。计算 `1AAF22BBh × 152h`，结果为 `233B3BDAE6h`，存到 `rax` 中。`v9` 减去 `v10` 再减去 `rax`，与 `3D23A36` 比较，如果不相等，输出 `argv1 nonono`，程序结束。所以，`v9` 减去 `v10` 再减去 `rax == 3D23A36h`，即 **`v9-v10==233F0E151Ch`**。

同理继续分析，可以得到另外三个方程：

$$v9 - v11 == 1B45F81A32h$$

$$v9 - v12 == 244C071725h$$

$$v9 + v12 + v11 + v10 == 13A31412F8Ch$$

上述条件均满足情况下，输出正确提示，程序结束。

五、脚本编写

综合上述分析，得到方程：

$$v9 == f(v4);$$

$$v9 == v10 + 0x233F0E151C;$$

$$v9 == v11 + 0x1B45F81A32;$$

$$v9 == v12 + 0x244C071725;$$

$$v4 + v12 + v11 + v10 == 0x13A31412F8C.$$

后四个方程相加得到：

$$v4 + 3v9 == 0x233F0E151C + 0x1B45F81A32 + 0x244C071725 + 0x13A31412F8C$$

$$\text{又 } v9 == f(v4)$$

二式联立则可求得 $v4$ 和 $v9$ ，继而求得 $v10$ 、 $v11$ 、 $v12$ 。

代码如下

```
def f(n):
    if n>200 or n<2:
        return 0
    a,b,i=1,1,2
    for i in range(2,n):
        a,b=b,a+b
    return b
sum = 0x233F0E151C + 0x1B45F81A32 + 0x244C071725 + 0x13A31412F8C
for v4 in range(0,200) :
    v9 = f(v4)
    if v4 + 3 * v9 == sum :
        break
v10 = v9 - 0x233F0E151C
v11 = v9 - 0x1B45F81A32
v12 = v9 - 0x244C071725
print (v9)
print (v10)
print (v11)
print (v12)
print (hex(v10))
```



```
print (hex(v11))
print (hex(v12))
print (hex(v4+25923))
```

输出截图：

The screenshot shows a Python IDE window titled 'main.py'. The code contains a loop that calculates values for v10, v11, and v12 based on v9, and then prints their hexadecimal representations. The output window shows the results of these calculations.

```
12     break
13     v10 = v9 - 0x233F0E151C
14     v11 = v9 - 0x1B45F81A32
15     v12 = v9 - 0x244C071725
16     print (v9)
17     print (v10)
18     print (v11)
19     print (v12)
20     print (hex(v10))
21     print (hex(v11))
22     print (hex(v12))
23     print (hex(v4+25923))
```

Output:

```
开始运行 ...
591286729879
439904987003
474148725349
435392374130
0x666c61677b
0x6e65776265
0x655f686572
0x657d
运行结束。
```

实验结果：

v9 为 591286729879;
v10 为 439904987003;
v11 为 474148725349;
v12 为 435392374130.

将 v10、v11、v12 转为 16 进制，并将 v4+25923 转为 16 进制，得到：

v10 为 0x666c61677b
v11 为 0x6e65776265
v12 为 0x655f686572
v4+25923 为 0x657d

将每位转为字符串：**666c61677b 6e65776265 655f686572 657d**

这是所输入字符串的一种代表性结果。

如果在每个字符串后接以非法字符(非 0-9、非 A-F、非 a-f)开头的任意字符串也是可以的。如：**666c61677bcccccc 6e65776265m12345 655f686572sssss 657dtptpt**

六、实验截图

The screenshot shows a terminal window titled 'machine@localhost:~/Desktop'. The user has entered a command to reverse a shell, and the terminal shows the response.

```
machine@localhost:~/Desktop
File Edit View Search Terminal Help
[machine@localhost Desktop]$ ./reverse 666c61677b 6e65776265 655f686572 657d
well done!decode your argv!
[machine@localhost Desktop]$
```

```
machine@localhost:~/Desktop
File Edit View Search Terminal Help
[machine@localhost Desktop]$ ./reverse 666c61677btccccc 6e65776265m12345 655f68
5572sssss 657dtptpt
well done!decode your argv!
[machine@localhost Desktop]$
```

七、实验心得

结合反汇编代码与反编译代码，提高逆向分析的能力和熟练度；

在逆向分析中，对汇编语言的知识加以复习、提高；

在本次实验中，对课上讲的 C 语言程序逆向分析中函数识别的知识点理解更为深刻。