

南开大学

汇编语言与逆向技术课程实验报告

实验九：REExercise



学 院 网络空间安全学院
专 业 信息安全
学 号 2313546
姓 名 蒋枘言
班 级 信息安全班

一、实验目的

1. 熟悉反编译工具 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、打开文件

使用 Binary Ninja 打开待分析的 exe 文件： task1.exe 和 task2.exe。

反汇编代码如下图所示：

task1.exe

task2.exe

三、逆向分析过程

(一) task1

1. 打印“Please input a string:”，引导用户输入字符串。

```
00401470 sub_401470:
00401470 // 首先通过减少esp寄存器的值来为局部变量分配84（即16进制数54h）字节的栈空间。
00401470 83ec54          sub    esp, 0x54
00401473 // 以下3行代码从全局变量__security_cookie中读取一个值,
00401473 // 将其与esp（当前栈指针）进行异或操作,
00401473 // 然后将结果存储在栈上的局部变量位置（esp+50h）。
00401473 // 这是一种栈保护机制，用于检测栈溢出攻击。
00401473 a100604000      mov    eax, dword [__security_cookie]
00401478 33c4            xor    eax, esp {var_54}
0040147a 89442450       mov    dword [esp+0x50 {var_4}], eax
0040147e // 将字符串“Please input a string:\n”的地址压入栈中。
0040147e 68304b4000      push   data_404b30 {__saved_esi} {"Please input a string:\n"}
00401483 // 调用函数sub_401610（是一个打印函数）来显示这个字符串。
00401483 e8ebfbffff     call   j_sub_401610
00401488 8d442404       lea    eax, [esp+0x4 {var_54}]
0040148c 50              push   eax {var_54} {var_5c}
0040148d 684c4b4000      push   data_404b4c {var_60}
00401492 e887fbffff     call   j_sub_401650 // 输入函数。
00401497 8d4c240c       lea    ecx, [esp+0xc {var_54}]
0040149b 83c40c          add    esp, 0xc
0040149e 8d5101          lea    edx, [ecx+0x1 {var_53}]
```

2. 读取输入字符串的长度，并与8作比较。

```
004014a1 // 将ecx指向的内存地址中的字节值加载到al中。
004014a1 8a01          mov    al, byte [ecx]
004014a3 // 将ecx寄存器的值增加1。
004014a3 // 由于ecx之前被用作内存地址的指针,
004014a3 // 故为了将指针移动到下一个字节。
004014a3 41           inc    ecx
004014a4 // 使用test指令对al寄存器中的值进行位测试。
004014a4 // test指令实际上执行的是按位与（AND）操作,
004014a4 // 但它不保存结果,
004014a4 // 只更新标志寄存器（如零标志ZF、符号标志SF等）。
004014a4 // 这里test al, al相当于检查al是否为0。
004014a4 // 如果al为0，则零标志ZF被设置;
004014a4 // 如果al不为0，则零标志ZF被清除。
004014a4 84c0          test   al, al
004014a6 // 如果零标志ZF未被设置（即al不为0）,
004014a6 // 则跳转到地址004014a1。
004014a6 // 该循环会执行直到al寄存器中的值为0。
004014a6 75f9          jne   0x4014a1
```

```
004014a8 // 将ecx-edx赋给ecx。
004014a8 // 此时得到的ecx的值为输入字符串的长度。
004014a8 2bca          sub    ecx, edx {var_53}
004014aa 83f908         cmp    ecx, 0x8
004014ad 740d          je    0x4014bc
```

3. 若字符串的长度不等于8，则会输出错误提示“Wrong length”。

```
004014af // 将字符串“Wrong length\n”的地址压入栈中。
004014af 68504b4000      push   data_404b50 {__saved_esi} {"Wrong length\n"}
004014b4 // 调用函数sub_401610（是一个打印函数）来显示这个字符串。
004014b4 e8bafbffff     call   j_sub_401610
004014b9 83c404         add    esp, 0x4
```

4. 将第一个字符和第二个字符相加，结果与 0xb2（即十进制数 178）作比较，若不相等则跳转到最后的输出错误提示。

5. 将第一个字符和第二个字符相减，结果与 0xfffffd (即有符号十进制数-38) 作比较，若不相等则跳转到最后的输出错误提示。

```
004014d4 // 将ecx-edx的值赋给ecx寄存器。
004014d4 2bca           sub    ecx, edx
004014d6 // 比较ecx寄存器的值与立即数fffffdah。
004014d6 83f9da         cmp    ecx, 0xffffffffda
004014d9 0f85b7000000     jne    0x401596 {"h`K@\"}
```

6.计算第四个字符*3+第六个字符*2，结果与 0x21d（即十进制数 541）作比较，若不相等则跳转到最后的输出错误提示。

```
004014df // 将esp+7h指向的地址处的字节值符号扩展到edx中。
004014df 0fbe542407 movsx edx, byte [esp+0x7 {var_51}]
004014e4 // 将esp+9h指向的地址处的字节值符号扩展到ecx中。
004014e4 0fbe4c2409 movsx ecx, byte [esp+0x9 {var_4f}]
004014e9 // 计算edx+edx*2（即edx*3），并将结果加载到eax寄存器中。
004014e9 8d0452 lea eax, [edx+edx*2]
004014ec // 计算eax+ecx*2，并将结果加载到eax寄存器中。
004014ec 8d0448 lea eax, [eax+ecx*2]
004014ef // 比较eax寄存器的值与立即数21dh。
004014ef 3d1d020000 cmp eax, 0x21d
004014f4 0f859c000000 jne 0x401596 {"h`K@"}
```

7.计算第六个字符-第四个字符*8，结果与 0xfffffd3c（即有符号十进制数-708）作比较，若不相等则跳转到最后的输出错误提示。

```
004014fa // 计算edx*8，并将结果加载到eax寄存器中。
004014fa 8d04d500000000 lea     eax, [edx*8]
00401501 // 将ecx-eax的值赋给ecx寄存器。
00401501 2bc8             sub     ecx, eax
00401503 // 比较ecx寄存器的值与立即数fffffd3ch。
00401503 81f93cfdffff cmp    ecx, 0xfffffd3c
00401509 0f8587000000 jne    0x401596 {"h`K@`"}
```

8.计算第五个字符*125+第八个字符*18，结果与 0x2ad9（即十进制数 10969）作比较，若不相等则跳转到最后的输出错误提示。

```
004014fa // 计算edx*8，并将结果加载到eax寄存器中。
004014fa 8d04d500000000 lea    eax, [edx*8]
00401501 // 将ecx-eax的值赋给ecx寄存器。
00401501 2bc8        sub    ecx, eax
00401503 // 比较ecx寄存器的值与立即数fffffd3ch。
00401503 81f93cfdf7ff cmp    ecx, 0xfffffd3c
00401509 0f8587000000 jne    0x401596 {"h`K@"}
```

9.计算第八个字符*6-第五个字符*33，结果与 0xfffff613（即有符号十进制数-2541）作比较，若不相等则跳转到最后的输出错误提示。

```
00401529 8bc6        mov    eax, esi
0040152b // 计算edx+edx*2（即edx*3），并将结果加载到ecx中。
0040152b 8d0c52        lea    ecx, [edx+edx*2]
0040152e // 将eax寄存器的值左移5位（相当于乘以32）。
0040152e c1e005        shl    eax, 0x5
00401531 03c9        add    ecx, ecx
00401533 03c6        add    eax, esi
00401535 2bc8        sub    ecx, eax
00401537 // 比较ecx寄存器的值与立即数fffff613h。
00401537 81f913f6ffff cmp    ecx, 0xfffff613
0040153d 7557        jne    0x401596 {"h`K@"}
```

10.计算第七个字符*10-第三个字符*3，结果与 0x351（即十进制数 849）作比较，若不相等则跳转到最后的输出错误提示。

```
0040153f 0fbe54240a    movsx  edx, byte [esp+0xa {var_4e}]
00401544 0fbe742406    movsx  esi, byte [esp+0x6 {var_52}]
00401549 8d0c92        lea    ecx, [edx+edx*4]
0040154c 03c9        add    ecx, ecx
0040154e 8d0476        lea    eax, [esi+esi*2]
00401551 2bc8        sub    ecx, eax
00401553 81f951030000  cmp    ecx, 0x351
00401559 753b        jne    0x401596 {"h`K@"}
```

11.计算第三个字符*62-第七个字符*7，结果与 0x1460（即十进制数 5216）作比较，若不相等则跳转到最后的输出错误提示。

```
0040155b 8bce        mov    ecx, esi
0040155d 8d04d500000000 lea    eax, [edx*8]
00401564 c1e105        shl    ecx, 0x5
00401567 2bc2        sub    eax, edx
00401569 2bce        sub    ecx, esi
0040156b 03c9        add    ecx, ecx
0040156d 2bc8        sub    ecx, eax
0040156f 81f960140000  cmp    ecx, 0x1460
00401575 751f        jne    0x401596 {"h`K@"}
```

12.若以上 8 个条件都满足，则输出正确提示“Correct”，程序结束。

```
00401577 // 将字符串“Correct\n”的地址压入栈中。
00401577 68684b4000      push    data_404b68 {"Correct\n"}
0040157c // 调用函数sub_401610 (是一个打印函数) 来显示这个字符串。
0040157c e8f2faffff     call    j_sub_401610
00401581 83c404          add     esp, 0x4
00401584 33c0            xor    eax, eax {0x0}
00401586 5e              pop    esi {_saved_esi}
00401587 8b4c2450        mov    ecx, dword [esp+0x50 {var_4}]
0040158b 33cc            xor    ecx, esp {var_54}
0040158d e827fbffff     call    CookieCheckFunction
00401592 83c454          add    esp, 0x54
00401595 // 从当前函数返回。
00401595 c3              retn   {_return_addr}
```

13.若有任意一个条件不满足，则输出错误提示“Wrong”，程序结束。

```
00401596 // 将字符串“Wrong\n”的地址压入栈中。
00401596 68604b4000      push    data_404b60 {"Wrong\n"}
0040159b // 调用函数sub_401610 (是一个打印函数) 来显示这个字符串。
0040159b e8d3faffff     call    j_sub_401610
004015a0 8b4c2458        mov    ecx, dword [esp+0x58 {var_4}]
004015a4 83c404          add    esp, 0x4
004015a7 b801000000      mov    eax, 0x1
004015ac 5e              pop    esi {_saved_esi}
004015ad 33cc            xor    ecx, esp {var_54}
004015af e805fbffff     call    CookieCheckFunction
004015b4 83c454          add    esp, 0x54
004015b7 // 从当前函数返回。
004015b7 c3              retn   {_return_addr}
```

(二) task2

1.打印“Please input a number:” , 引导用户输入一个数。将其赋值到寄存器 eax 中。然后开始计算：

- ①将 eax 的值加上 0x13ac6d22;
- ②与 0x9bf39868 进行异或;
- ③减去 0x61bacb1a;
- ④与 0x4a8bd66c 进行异或;
- ⑤加上 0x74ebdec3;
- ⑥与 0x1325a73d 进行异或;
- ⑦加上 0x217008e;
- ⑧与 0x217008e 进行异或。

```
00401470 sub_401470:
00401470 51              push    ecx {var_4}
00401471 68304b4000      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 684c4b4000      push    data_404b4c {var_10}
00401485 e894fbffff     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, 0x61bacb1a
```

004014a0	356cd68b4a	xor	eax, 0x4a8bd66c
004014a5	05c3deeb74	add	eax, 0x74ebdec3
004014aa	353da72513	xor	eax, 0x1325a73d
004014af	058e001702	add	eax, 0x217008e
004014b4	358e001702	xor	eax, 0x217008e
004014b9	3defbeadde	cmp	eax, 0xdeadbeef
004014be	7511	jne	0x4014d1 {"h\K@"}

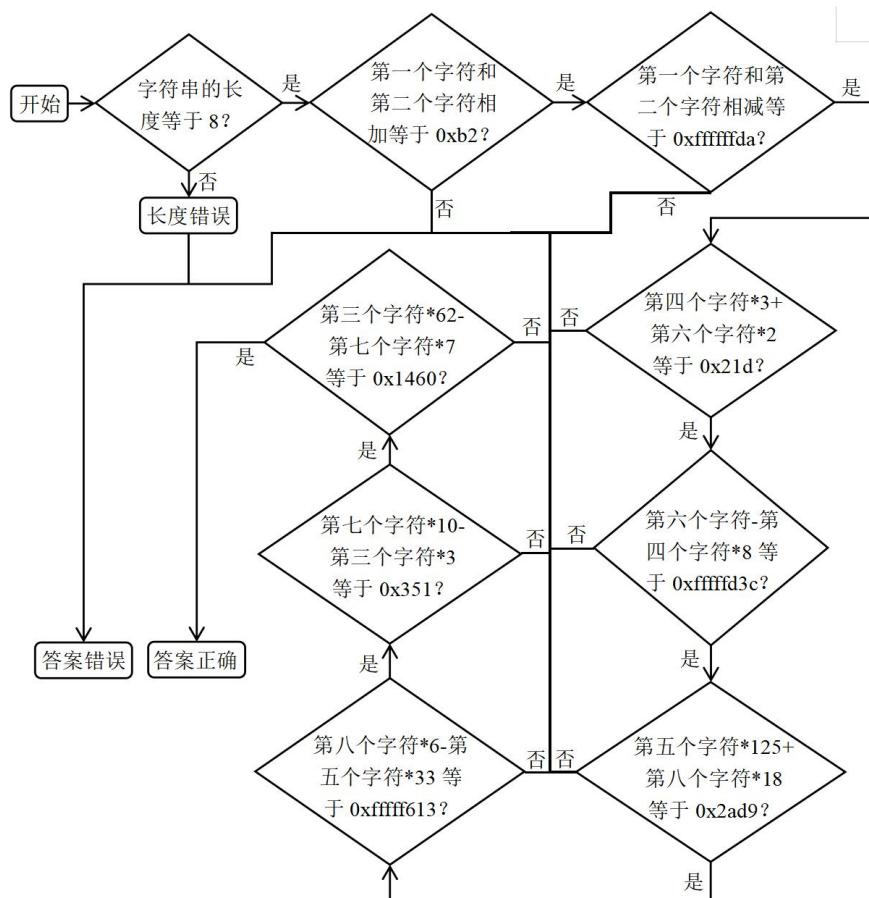
2. 将结果与 0xdeadbeef 作比较。若相等，则输出正确提示“Correct!”，程序结束；否则输出正确提示“Wrong!”，程序结束。

004014c0	68504b4000	push	data_404b50 {var_8} {"Correct!\n"}
004014c5	e8a9fbffff	call	j_sub_401510
004014ca	83c404	add	esp, 0x4
004014cd	33c0	xor	eax, eax {0x0}
004014cf	59	pop	ecx {var_4}
004014d0	c3	ret	{__return_addr}

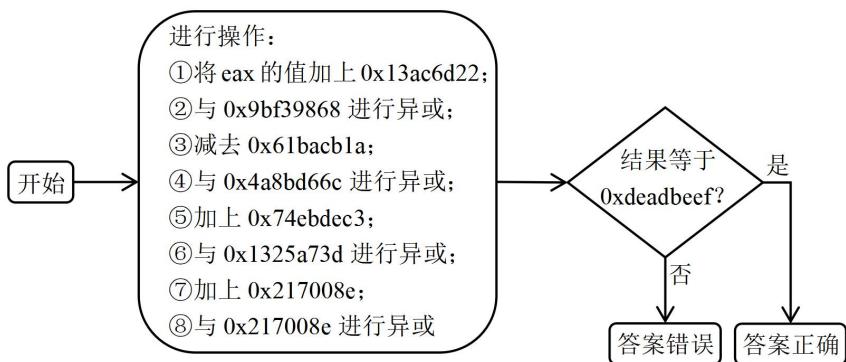
004014d1	685c4b4000	push	data_404b5c {var_8} {"Wrong!"}
004014d6	e898fbffff	call	j_sub_401510
004014db	83c404	add	esp, 0x4
004014de	b801000000	mov	eax, 0x1
004014e3	59	pop	ecx {var_4}
004014e4	c3	ret	{__return_addr}

四、逆向分析流程图

(一) task1



(二) task2



五、结果验证

(一) task1

正确的字符串有 8 个字符。设第 1,2,3,4,5,6,7,8 个字符的 ASCII 码分别为 $a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8$, 则有

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 6 & 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -8 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 125 & 0 & 0 & 18 \\ 0 & 0 & 0 & 0 & -33 & 0 & 0 & 6 \\ 0 & 0 & -3 & 0 & 0 & 0 & 10 & 0 \\ 0 & 0 & 62 & 0 & 0 & 0 & -7 & 0 \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \\ a_8 \end{pmatrix} = \begin{pmatrix} 178 \\ -38 \\ 541 \\ -708 \\ 10969 \\ -2541 \\ 849 \\ 5216 \end{pmatrix}$$

根据 Cramer 法则知 $a_i = \frac{D_i}{D}$ ($i = 1, 2, \dots, 8$) , 故可解得

$$\begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \\ a_8 \end{pmatrix} = \begin{pmatrix} 70 \\ 108 \\ 97 \\ 103 \\ 83 \\ 116 \\ 114 \\ 33 \end{pmatrix}$$

$a_1 \sim a_8$ 的 ASCII 码对应的字符分别为 “F” “l” “a” “g” “S” “t” “r” “!” , 故最终的字符串为 “FlagStr!” , 验证发现结果正确。

```
命令提示符
Microsoft Windows [版本 10.0.26100.2454]
(c) Microsoft Corporation. 保留所有权利。

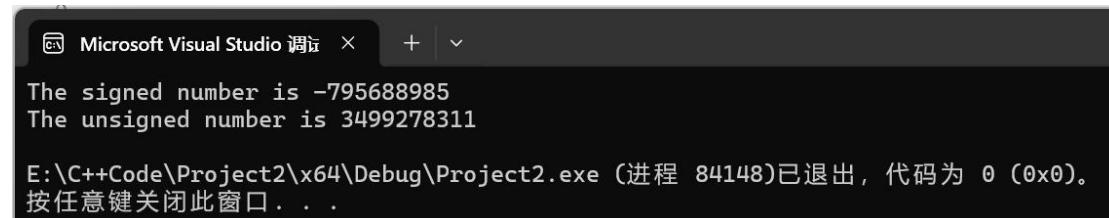
C:\Users\Lenovo>D:\南开批事情\第3学期-汇编语言与逆向技术课件和实验\实验\task1.exe
Please input a string:
FlagStr!
Correct
```

(二) task2

写一个 C++ 程序, 将过程倒推, 来计算结果:

```
#include <iostream>
using namespace std;
int main() {
    long long res = 0xdeadbeef;
    res ^= 0x217008e;
    res -= 0x217008e;
    res ^= 0x1325a73d;
    res -= 0x74ebdec3;
    res ^= 0x4a8bd66c;
    res += 0x61bacb1a;
    res ^= 0x9bf39868;
    res -= 0x13ac6d22;
    cout << "The signed number is " << (int)res << endl;
    cout << "The unsigned number is " << (unsigned int)res << endl;
    return 0;
}
```

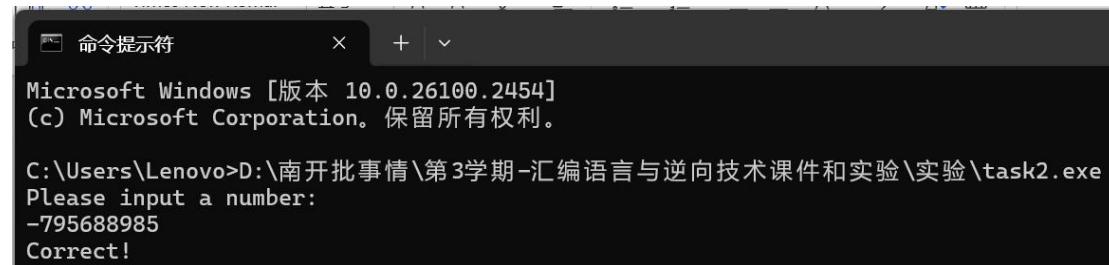
运行结果为：



The signed number is -795688985
The unsigned number is 3499278311

E:\C++Code\Project2\x64\Debug\Project2.exe (进程 84148)已退出，代码为 0 (0x0)。
按任意键关闭此窗口 . . .

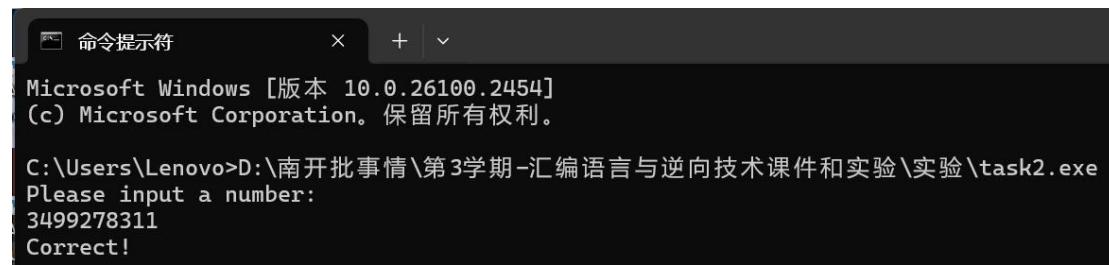
故结果为-795688985 或 3499278311，验证发现二者结果正确。



命令提示符

Microsoft Windows [版本 10.0.26100.2454]
(c) Microsoft Corporation。保留所有权利。

C:\Users\Lenovo>D:\南开批事情\第3学期-汇编语言与逆向技术课件和实验\实验\task2.exe
Please input a number:
-795688985
Correct!



命令提示符

Microsoft Windows [版本 10.0.26100.2454]
(c) Microsoft Corporation。保留所有权利。

C:\Users\Lenovo>D:\南开批事情\第3学期-汇编语言与逆向技术课件和实验\实验\task2.exe
Please input a number:
3499278311
Correct!