**电子科技大学信息与软件工程学院**

**标 准 实 验 报 告**

**（实验）课程名称：软件安全**

**电子科技大学教务处制表**

**学生姓名：周玉川 学 号：2017221302006 指导教师： 钱伟中**

**实验地点：信软学院**  **实验时间：2019.10.25/2019.10.13**

**一、实验室名称：信软学院实验室**

**二、实验项目名称： 逆向分析**

**三、实验学时：4学时**

**四、实验原理：**

1. 软件逆向的概念

将可执行程序反汇编，通过分析反汇编代码来理解代码功能，然后用高级语言重新描述这段代码，逆向分析原始软件的思路，这个过程就称作逆向工程（Reverse Engineering），有时也简单地称作逆向（Reversing）。

1. 软件逆向原理

利用逆向工程技术，从可运行的程序系统出发，运用解密、反汇编、系统分析、程序理解等多种计算机技术,对软件的结构、流程、算法、代码等进行逆向拆解和分析，推导出软件产品的源代码、设计原理、结构、算法、处理过程、运行方法及相关文档等。随着用户需求的复杂度越来越高软件开发的难度也在不断地上升快速高效的软件开发已成为项目成败的关键之一。为了提高程序员的产品率开发工具的选择尤为重要因为开发工具的自动化程度可以大大减少程序员繁琐重复的工作使其集中关注他所面临的特定领域的问题。为此当前的IDE不可避地要向用户隐藏着大量的操作细节而这些细节包含了大量的有价值的技术。

1. 逆向工具使用方法简介

在对软件进行逆向工程时，不可避免地需要用到多种工具，工具的合理使用,可以加快调试速度，提高逆向工程的效率。对于逆向工程的调试环节来说，没有动态调试器将使用的调试工作很难进行。可以看出，各种有效的工具在逆向工程中占据着相当重要的地位，有必要对它们的用法做一探讨。

IDA Pro简介：IDA Pro（简称IDA）是DataRescue公司（www.datarescue.com）出品的一款交互式反汇编工具，它功能强大、操作复杂，要完全掌握它，需要很多知识。IDA最主要的特性是交互和多处理器。操作者可以通过对IDA的交互来指导IDA更好地反汇编，IDA并不自动解决程序中的问题，但它会按用户的指令找到可疑之处，用户的工作是通知IDA怎样去做。比如人工指定编译器类型，对变量名、结构定义、数组等定义等。这样的交互能力在反汇编大型软件时显得尤为重要。多处理器特点是指IDA支持常见处理器平台上的软件产品。IDA支持的文件类型非常丰富，除了常见的PE格式，还支持Windows,DOS,UNIX,Mac,Java,.NET等平台的文件格式。

当我们打开之后，IDA会提供3种不同的打开方式；New（新建），Go（运行），Previous（上一个）。当我们初次打开的时候选择GO就可以了。进入之后，选择左上角的file中的open打开文件，或者也可以直接将文件拖入窗口中。

Ollydbg简介：OLLYDBG是一个新的动态追踪工具，将IDA与SoftICE结合起来的思想，Ring 3级调试器，非常容易上手，己代替SoftICE成为当今最为流行的调试解密工具了。同时还支持插件扩展功能，是目前最强大的调试工具之一。

**五、实验目的：**

1）理解软件逆向的概念；

2）掌握逆向工具：Ollydbg和IDA pro；

3）掌握逆向分析软件的方法；

4）掌握对可执行文件逆向到高级语言的方法

**六、实验内容：**

现在得到一个可执行EXE执行程序，反汇编该程序得到的信息展示如下:

要求: 写出其对应的高级语言

附件1为逆向EXE的反汇编代码。

**七、实验器材（设备、元器件）：**

PC机

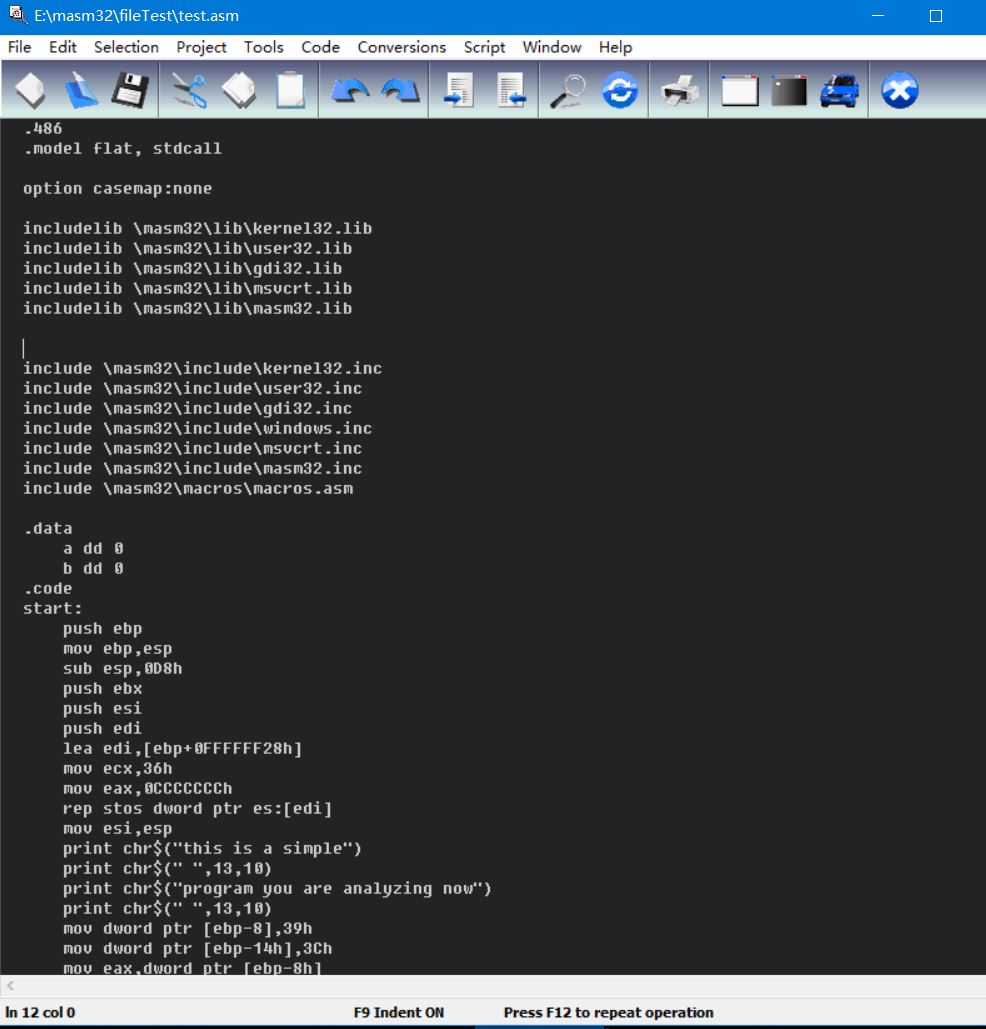
**八、实验步骤：**

1） 阅读反汇编程序，分析程序模块，整理成汇编程序

汇编程序如下，平台masm32

|  |
| --- |
| .486  ;告诉汇编器应该生成486处理器（或更高）的伪代码。  .model flat, stdcall  ;使用平坦内存模式并使用stdcall调用习惯,函数的参数从右往左压入  option casemap:none  ;控制字符的映射为大写。为了Windows.inc文件能正常工作，这个应该为”none”  includelib \masm32\lib\kernel32.lib  includelib \masm32\lib\user32.lib  includelib \masm32\lib\gdi32.lib  includelib \masm32\lib\msvcrt.lib  includelib \masm32\lib\masm32.lib  include \masm32\include\kernel32.inc  include \masm32\include\user32.inc  include \masm32\include\gdi32.inc  include \masm32\include\windows.inc  include \masm32\include\msvcrt.inc  include \masm32\include\masm32.inc  include \masm32\macros\macros.asm  ;为了使用来自WindowsAPI的函数，你需要导入dll和包含文件  .data ;数据段  a dd 0  b dd 0  .code ;代码段  start: ;程序开始的标志  push ebp ;保存上一个栈的栈底，所白了就是备份，方便后续回复  mov ebp,esp ;把上一个栈的栈顶，当做当前栈的栈底，相当于创造了一个新的栈  sub esp,0D8h ;为当前栈预留本地变量的空间  push ebx ;  push esi ;  push edi ;保护寄存器的值  lea edi,[ebp+0FFFFFF28h] ;  mov ecx,36h ;  mov eax,0CCCCCCCh ;  rep stos dword ptr es:[edi] ;初始化  mov esi,esp ;保存esp值  print chr$("this is a simple")  print chr$(" ",13,10)  print chr$("program you are analyzing now")  print chr$(" ",13,10)  mov dword ptr [ebp-8],39h ;  mov dword ptr [ebp-14h],3Ch ;初始化本地变量，所以是栈低减一个数值(进栈)  mov eax,dword ptr [ebp-8h] ;  mov ecx,dword ptr [ebp-14h] ;因为寄存器的锁机制，存入值后读取一次后，寄存器就会释放，所以要再存一次。  .IF eax <= ecx ;伪指令比较,相当于条件转址,注意点：其中至少一个值在寄存器中  ;例如：把值mov给data中定义的变量a,b，然后.IF a<b，这样会报错  mov eax,dword ptr [ebp-14h] ;  push eax ;  mov ecx,dword ptr [ebp-8h] ;  push ecx ;把参数按从右到左顺序压入栈中,同样需要重新读  call fun ;调用函数  add esp,8 ;这里加8的原因是函数参数占用8字节,因为函数已经执行完，所以回收栈  .ELSE  print chr$("hello")  .ENDIF  mov dword ptr [ebp-14h],32h  mov eax,dword ptr [ebp-8h]  mov ecx,dword ptr [ebp-14h]  .IF eax > ecx  mov eax,dword ptr [ebp-14h]  push eax  mov ecx,dword ptr [ebp-8h]  push ecx  call fun  add esp,8 ;这里同上理  .ENDIF  print chr$("this is true end! but you should not relax yourself!!! Be careful")  print chr$(" ",13,10)  print chr$("trap")  print chr$(" ",13,10)  mov eax,dword ptr [ebp-8]  pop edi ;  pop esi ;  pop ebx ;  add esp,0D8h ;  mov esp,ebp ;  pop ebp ;这里的作用就是恢复现场  ret    fun:  push ebp  mov ebp,esp ;把上一个状态保存下来，创造一个新栈  sub esp,0CCh  push ebx ;  push esi ;  push edi ;保存寄存器值  lea edi,[ebp+0FFFFFF34h]  mov ecx,33h  mov eax,0CCCCCCCh ;初始化  rep stos dword ptr es:[edi]  mov eax,dword ptr [ebp+8h] ;解释：当前栈低ebp是上一状态栈的栈顶esp,上一个栈的栈顶是call产生的下一条指令的ip地址,->  ;倒数第二个是参数a，倒数第三个是参数b，示意图: 高地址->[参数b,参数a,ip地址(call指令的下一条)]->低地址  ;所以说[ebp+8]是参数a，[ebp+0Ch]是参数b  mov a,eax  print chr$("The value of A is ")  print str$(a)  print chr$(" ",13,10)  mov ecx,dword ptr [ebp+0Ch] ;获取参数b，道理同上  mov b,ecx  print chr$("The value of B is ")  print str$(b)  print chr$(" ",13,10)  mov eax,dword ptr [ebp+8h]  mov ecx,dword ptr [ebp+0Ch]  .IF eax < ecx  mov eax, dword ptr [ebp+0Ch]  sub eax, dword ptr [ebp+8]  mov dword ptr [ebp-8],eax  .ELSE  mov eax,dword ptr [ebp+8]  sub eax,dword ptr [ebp+0Ch]  mov dword ptr [ebp-8],eax  .ENDIF ; 通俗讲：大的减小的，将结果保存在[ebp-8]  mov esi,esp  mov eax,dword ptr [ebp-8]  push eax  print chr$("their sum is ")  mov eax,dword ptr [ebp-8]  print str$(eax)  print chr$(" ",13,10)  print chr$("can you have ")  print chr$("the ability of ")  print chr$("reverse analysis")  print chr$(" ",13,10)  mov eax,dword ptr [ebp-8]  pop edi  pop esi  pop ebx  add esp,0CCh  mov esp,ebp  pop ebp ;恢复现场  ret ;返回上一个状态  end start |

在Masm32 Editor中编写汇编程序，如图1-1所示



2） 编译程序，输出EXE文件

在Masm32 Editor中按ctrl+s保存文件，依次点击project->Console Assemble & Link。

然后执行文件，结果如图2-1所示

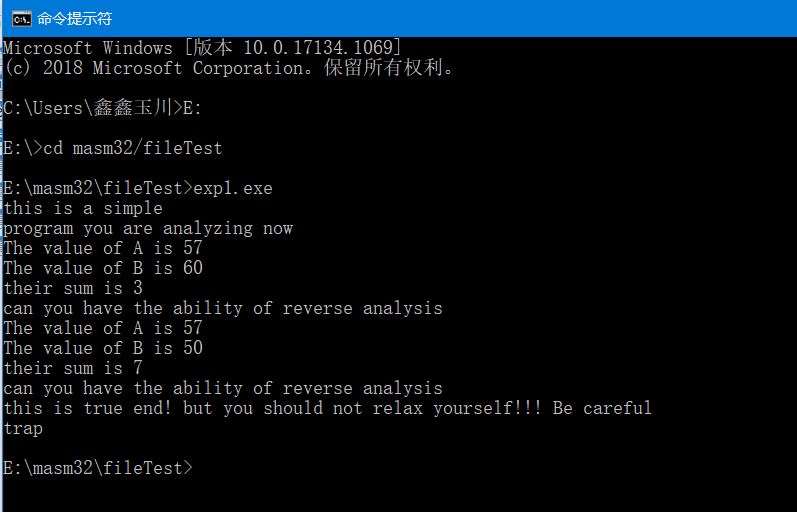
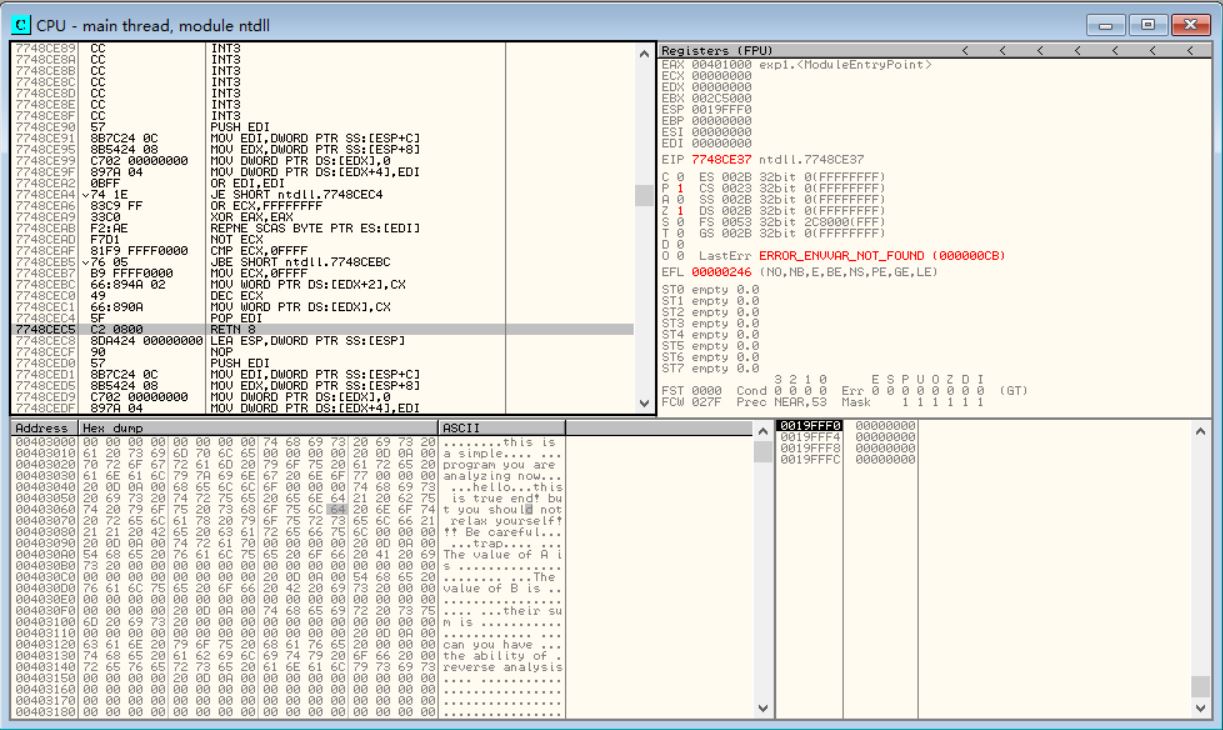


图 2-1

3） 用Ollydbg加载EXE文件，调试EXE

如图3-1所示



4) 用IDA获得C伪代码

将获得的可执行文件拖入IDA中，获得伪代码，如图表4-1，4-2

主函数4-1

|  |
| --- |
| unsigned int start()  {  char v1; // [esp+Ch] [ebp-D8h]  int v2; // [esp+D0h] [ebp-14h]  unsigned int v3; // [esp+DCh] [ebp-8h]  memset32(&v1, 214748364, 0x36u);  sub\_401278(aThisIsASimple);  sub\_401278(asc\_40301C);  sub\_401278(aProgramYouAreA);  sub\_401278(asc\_403040);  v3 = 57;  v2 = 60;  sub\_4010D5(57, 60);  v2 = 50;  if ( v3 > 0x32 )  sub\_4010D5(v3, v2);  sub\_401278(aThisIsTrueEndB);  sub\_401278(asc\_403090);  sub\_401278(aTrap);  sub\_401278(asc\_40309C);  return v3;  } |

子函数4-2

|  |
| --- |
| int \_\_cdecl sub\_4010D5(unsigned int a1, unsigned int a2)  {  char v3; // [esp+Ch] [ebp-CCh]  int v4; // [esp+D0h] [ebp-8h]  memset32(&v3, 214748364, 0x33u);  dword\_403000 = a1;  sub\_401278(aTheValueOfAIs);  sub\_401210(dword\_403000, &unk\_4030B4);  sub\_401278(&unk\_4030B4);  sub\_401278(asc\_4030C8);  dword\_403004 = a2;  sub\_401278(aTheValueOfBIs);  sub\_401210(dword\_403004, &unk\_4030E0);  sub\_401278(&unk\_4030E0);  sub\_401278(asc\_4030F4);  if ( a1 >= a2 )  v4 = a1 - a2;  else  v4 = a2 - a1;  sub\_401278(aTheirSumIs);  sub\_401210(v4, &unk\_403108);  sub\_401278(&unk\_403108);  sub\_401278(asc\_40311C);  sub\_401278(aCanYouHave);  sub\_401278(aTheAbilityOf);  sub\_401278(aReverseAnalysi);  sub\_401278(asc\_403154);  return v4;  } |

5） 分析调整第4步获得的C伪代码

|  |
| --- |
| unsigned int start()  {  int v1; // [esp+Ch] [ebp-D8h]  int v2; // [esp+D0h] [ebp-14h]  int v3; // [esp+DCh] [ebp-8h]  memset32(&v1, 214748364, 0x36u);  printf(this is a simple\n”);  printf("program you are analyzing now\n");// 4157ACh  v3 = 57;  v2 = 60;  sub\_fun (57, 60);  v2 = 50;  if ( v3 > 0x32 )  sub\_fun(v3, v2);  printf("this is true end! but you should not relax yourself!!! Be careful\n");//4162A0h  printf("trap");//415BC8h  return v3;  }  sub\_fun(int a1,int a2){  int v3;  printf(“The value of A is %d\n”,a1);  printf(“The value of B is %d\n”,a2);  if ( a1 >= a2 )  v3 = a1 - a2;  else  v3 = a2 - a1;  printf("their sum is ");// 41576Ch  printf("can you have ");//41575Ch  printf("the ability of ");//415748h  printf("reverse analysis\n");//415858h  return v3;  } |

6） 用Visual studio，将第5步获得的代码输入进行测试验证。

测试结果如图6-1

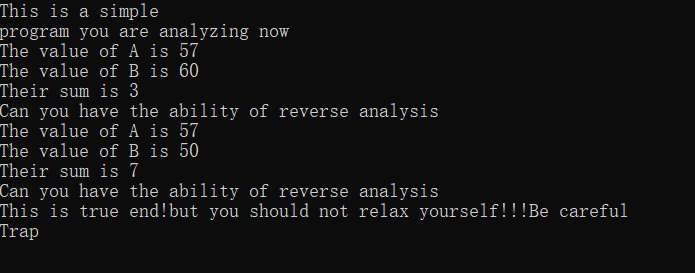


图6-1

**九、实验数据及结果分析：**

**（1） 附件1代码对应的高级语言程序**

|  |
| --- |
| **exp1.cpp**  #include<stdio.h>  int fun(int a,int b){  int c;  if (a<b){  c = b - a;  } else{  c = a - b;  }  printf("their sum is %d",c);// 41576Ch  printf("can you have ");//41575Ch  printf("the ability of ");//415748h  printf("reverse analysis\n");//415858h  return c;  }  int main(){  printf("this is a simple\n"); //415AB0  printf("program you are analyzing now\n");// 4157ACh  int a = 0x39; //[ebp-8h]  int b = 0x3C; //[ebp-14h]  if (a<=b){  //push b,push a  fun(a, b);  } else{  printf("hello\n");//4162E4h  }  b = 0x32;  if (a>b)  //push b push a  fun(a, b);  }  printf("this is true end! but you should not relax yourself!!! Be careful\n");//4162A0h  printf("trap");//415BC8h  return 0;  } |

**（2） 实验分析**

1.在本质上汇编代码和高级语言代码的逻辑是相同的。

2.高级语言在编译成汇编语言时会优化代码以及初始化。

3.汇编代码在进入一个段的时候会进行寄存器值得保护，以及通过修改bp产生新的栈，而且会给本地局部变量预留空间。

**十、实验结论：**

1. 程序在开始执行的时候都会把相应寄存器的值入栈保存，在程序结束后恢复现场。
2. 程序在执行call执行进入子程序后，子程序会构造新的栈，把上一个状态的栈顶当作自己的栈低，而且会给自己的局部变量在栈中留一些位置，子程序通过栈低减去一个正值获得传递给他的参数列表。

**十一、总结及心得体会：**

1. 掌握汇编语言非常重要，不仅可以进行逆行分析程序，而且提升对高级语言的理解。
2. 经过编写汇编代码，发现寄存器的锁机制，将内容写入寄存器后，一旦寄存器使用一次就会被释放，所以可能出现前后的eax值完全不同的现象，这也是汇编语言和高级语言不同的地方。

**十二、对本实验过程及方法、手段的改进建议：**

无。

**报告评分：**

**指导教师签字：**