第5章 PE病毒

5.1 实验概述

本章实验旨在让读者了解PE病毒的基本原理，熟悉PE病毒中的部分关键技术以及PE病毒的典型清除方法。实验过程分为以下四个阶段。

1. 熟悉masm32

先编写HelloWorld程序，总体步骤如下。

(1) 下载masm32v11。

(2) 熟悉masm32的基本环境。

(3) 写一个最简单的HelloWorld程序(如调用MessageBoxA弹出一个对话框)，并编译成功。

(4) 对得到的可执行文件进行反汇编，比较其反汇编代码与汇编代码异同。

(5) 查看并理解masm32\bin下各个批处理程序，了解他们的大致功能，以及与qeditor程序Project菜单的具体对应关系(Edit-Setting-Edit Menus)。

(6) 探索其他菜单功能，并列出你认为对本课程后续学习有用的功能。

2. 熟悉病毒重定位的基本思路和方法

在HelloWorld.exe中添加一段代码，具体要求如下。

(1) 该段代码弹出一个对话框(标题：武大网安病毒重定位，内容：姓名+学号)。

(2) 该段代码同时包括代码和字符串数据。

(3) 该段代码可以插入.text节的任意指令之间，而不修改该段代码中的任何字节，对其可移动性进行验证(移动产生的空闲区域可用nop代替)。

3. kernel32基地址定位及API函数地址搜索

搜索kernel32.dll的导出API函数地址。

(1) 用OllyDbg打开HelloWorld.exe，获取kernel32.dll模块基地址，定位到kernel32.dll模块。

(2) 从内存中的kernel32.dll模块获取函数LoadLibraryA和GetProcessAddress的函数地址，并实际检验获得的地址是否正确。

4. 分析病毒感染过程

分析并清除病毒，总体步骤如下。

(1) 编译本书中的感染示例程序 bookexample-old.rar，使用该感染示例程序对 HelloWorld.exe 进行感染。分析该病毒在感染文件时具体做了哪些操作，该病毒如何返回 HOST。利用 HelloWorld 继续感染其他目标程序，定位目标程序运行时存在的问题并予以解决。

(2) 编译本书中的感染示例程序 bookexample-new.rar，使用该感染示例程序对计算器程序(calc.exe)进行感染。

(3) 病毒感染示例程序在 64 位系统中无法正常感染，请定位其原因，并以最小范围的源码修改方案解决该问题。

(4) 清除病毒：在被感染的程序中定位 HOST 程序原程序入口地址，并手工恢复被感染的计算器程序(calc.exe)。

5.2 实验预备知识与基础

5.2.1 反汇编和反编译

(1) 汇编：将汇编源代码转变为目标程序(当然还不是最终的可执行的，因为还没有链接程序)。

(2) 编译：将高级语言编写的源程序通过编译器转变为目标程序。

(3) 反汇编：将可执行的文件中的二进制经过分析转变为汇编程序。

(4) 反编译：将可执行的程序经过分析转变为高级语言的源代码格式，由于编译器优化原因，一般难以进行完全一致的转换。

5.2.2 反汇编的原理

反汇编的核心工作就是要能够解析(parse)机器码，根据 CPU 的指令集规范理解它的含义，并且将其展现为对应的汇编文本形式。然而反汇编器中同样重要的一个组成部分，是对其所支持的平台的可执行文件格式的解析。

例如 Windows 上的 PE、PE+，*-nix 上的 ELF，macOS 上的 Mach-O 格式等。这些可执行文件的封装中包含可执行文件的结构元数据、导出/导入表、字符串常量池、调试符号信息等诸多辅助信息，以及很重要的，该文件的首选的加载地址以及代码的入口地址等。

通过这些封装在可执行文件里的辅助信息，反汇编器才可以正确识别文件中哪些部分是机器码，并对其进行反汇编操作。对用户友好的反汇编器还会进一步结合辅助信息来在反汇编结果中给出更丰富的内容，例如一个地址如果指向字符串常量则在注释中显示字符串内容，如果一个函数调用的参数列表已知，则根据调用约定(calling convention)分析调用点前传递参数的代码。

5.2.3 病毒重定位的原理

病毒重定位主要是利用 call 指令的 push+jmp 机制，push 到堆栈中的是 call 指令结束

之后的地址，jump 跳转的机制是从 call 指令结束之后的地址算起，偏移大小为 E8 之后紧跟的数值。利用 push 机制将数据压入堆栈，jump 跳转到下一条需要执行指令的地址。对于 MessageBoxA 和 ExitProcess 函数的调用，利用 call 间接调用的方式即 FF 15，跳转到 IAT 表中指向的函数入口点。

5.2.4 获取 kernel32 基地址的方法

(1) CreateProcess 函数在完成装载应用程序后，会先将一个返回地址压入堆栈顶端，而这个返回地址恰好在 kernel32.dll 中，利用这个原理可以顺着这个返回地址按 64KB 大小往地址搜索，那么一定可以找到 kernel32 模块的基地址。

(2) 通过 PEB 枚举当前进程空间中用户模块列表也可以获取 kernel32 模块的基地址，fs:[0]指向 TEB，fs:[30H]指向 PEB，PEB 偏移 0ch 是 LDR 指针，以下可以分别通过加载顺序、内存顺序、初始化顺序获取 kernel32 模块的基地址。此方法对于 32 位程序有效，在 64 位系统下，PEB 指向位置位 gs:[60]。

(3) 通过遍历 SEH 链的方法，在 SEH 链中查找成员 prev 的值为 0xFFFFFFFFh 的 EXCEPTION_REGISTER 结构。该结构中的 handler 值是系统异常处理例程，总是位于 kernerl32.dll 中。当前线程的 TIB 保存在 fs 段选择器指定的数据段的 0 偏移处，所以 fs:[0] 的地方就是 TIB 结构中的 ExceptionList 字段。而 ExceptionList 指向一个 EXCEPTION_REGISTRATION 结构，SEH 异常处理回调函数的入口地址就由 EXCEPTION_REGISTRATION 结构指定。此方法在 Windows XP 系统有效。

5.2.5 PE 病毒感染文件恢复

结合前面分析出的病毒的感染过程，要修复感染后的文件，至少要知道以下几个信息，同时也是需要修复的数据。

(1) 感染前的文件大小。前面已经知道，病毒直接从文件末开始写入 shellcode，要去掉病毒写入的内容，就必须知道感染前原文件大小。

(2) 感染前最后一个区段的 RawSize。因为 RawSize 增加的大小等于文件增加的大小，所以知道了感染前文件大小也就是 RawSize 的大小。

(3) 感染前最后一个区段的 VirtualSize。在获取正确的 RawSize 之后就可以根据对齐来计算 VirtualSize，也就是说第二个问题解决了，这个问题也就解决了。

(4) 感染前的映像和。在获取最后一个区段正确的 VirtualSize 之后就可以计算映像和，第三个问题解决了，这个问题也就解决了。

(5) 感染前的入口点。前四个信息一环扣一环，所以总体来说，要获取的关键数据只有两个，感染前的文件大小和感染之前的入口点。

5.2.6 修复过程总结

(1) 通过在感染后的文件中搜索 shellcode 得到原始入口点，原文件大小。

(2) 根据当前文件大小，得出增加部分的大小 ExtraSize。

(3) 根据 ExtraSize 修正最后一个区段的 RawSize，当前值减去多的部分即可得正确 RawSize。

(4) 由正确的 RawSize 和内存对齐大小,计算正确的 VirtualSize。

(5) 根据 PE 头的大小和所有区段映像大小,计算总的 SizeofImage。

(6) 原始入口点减去映像基址即得 AddressofEntryPoint。

(7) 将文件大小减小 ExtraSize,设置结束标记。

5.3 熟悉 masm32

5.3.1 实验目的

掌握 masm32 工具的基本使用方法和环境配置,学习编写、编译和调试简单的汇编语言程序,理解 masm32 工具在汇编语言程序开发中的重要性,并通过本实验为后续更复杂的汇编语言实验打下坚实的基础。

5.3.2 实验内容及实验环境

1. 实验内容

(1) 下载 masm32v11。

(2) 熟悉 masm32 的基本环境。

(3) 写一个最简单的 HelloWorld 程序,并编译成功。

(4) 对得到的可执行文件进行反汇编,比较其反汇编代码和最初的汇编代码有哪些异同。

(5) 查看并理解 masm32\bin 下各个批处理程序,了解它们的大致功能,以及与 QEditor 程序 Project 菜单的具体对应关系(Edit-Setting-Edit Menus)。

(6) 探索其他菜单功能,并列出你认为对本课程后续学习有用的功能。

2. 实验环境

(1) 硬件:一台装有 Windows 操作系统的普通 PC(使用虚拟机亦可)。

(2) 软件:masm32。

5.3.3 实验步骤

1. 软件和环境准备

进入官网下载地址:http://www.masm32.com,进入下载页面后,单击 Australia 1/2 开始下载,压缩后得到安装文件,成功安装后,右击“计算机”→属性→高级系统设置→环境变量,在用户变量新建并添加如下内容,配置环境变量。

然后将 bin 添加到 Path 中,编辑系统变量 Path,如图 5-1～图 5-3 所示。

2. masm 的使用

首先打开 HelloWorld.asm,如图 5-4 所示。

大致对程序功能进行分析,如图 5-5 所示,程序调用了 user32、kernel32。

在数据段存储了弹窗中的数据,如图 5-6 所示。

图 5-1　新建系统变量 lib

图 5-2　新建系统变量 include

图 5-3　编辑系统变量 Path

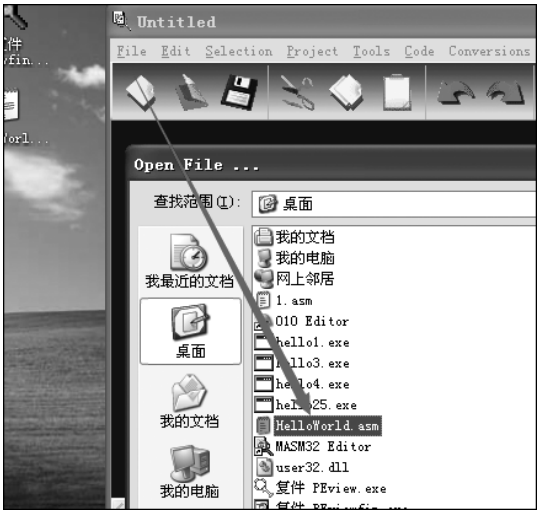

图 5-4　打开 HelloWorld.asm

```
;////////////////////////////////////////////////////////////////
; Include 文件定义
;////////////////////////////////////////////////////////////////
include          windows.inc
include          user32.inc
includelib  user32.lib
include          kernel32.inc
includelib  kernel32.lib
;////////////////////////////////////////////////////////////////
```

图 5-5　分析库文件

```
;////////////////////////////////////////////////////////////////
; 数据段
;////////////////////////////////////////////////////////////////
          .data
szCaption   db    'A MessageBox !',0
szText      db    'Hello, World !',0
;////////////////////////////////////////////////////////////////
```

图 5-6　分析数据段

图 5-7 显示了程序的代码段，它会调用一个 messageBoxA 弹窗，然后退出。

```
;////////////////////////////////////////////////////////////////
; 代码段
;////////////////////////////////////////////////////////////////
          .code
start:
          invoke      MessageBox,NULL,offset szText,offset szCapti
          invoke      ExitProcess,NULL
;////////////////////////////////////////////////////////////////
          end   start
```

图 5-7　分析代码段

启动汇编，如图 5-8 所示。也可在命令行中输入“ml /c /coff　HelloWorld.asm”。

单击链接，如图 5-9 所示。也可在命令行输入 link /subsystem:Windows HelloWorld.obj。

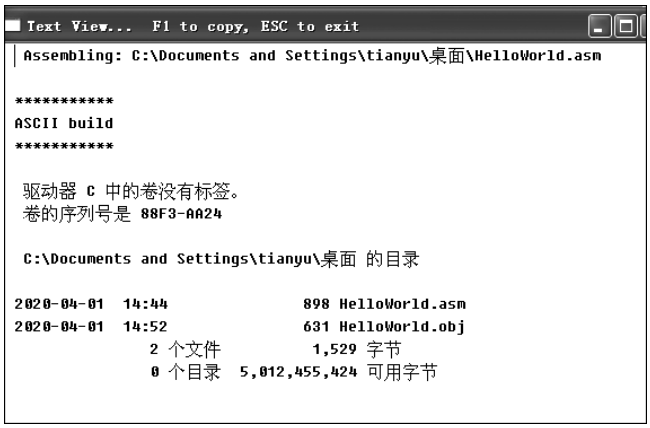

图 5-8　启动汇编

现在就可以运行程序了，结果如图 5-10 所示。

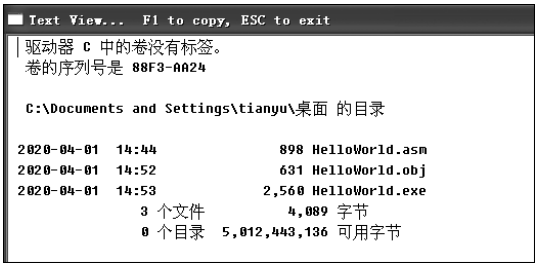

图 5-9　单击链接

图 5-10　程序运行结果

3. 反汇编

使用 masm32 Editor 打开 HelloWorld.exe，在 Tools 工具栏下打开如图 5-11 所示的反汇编 exe 文件。

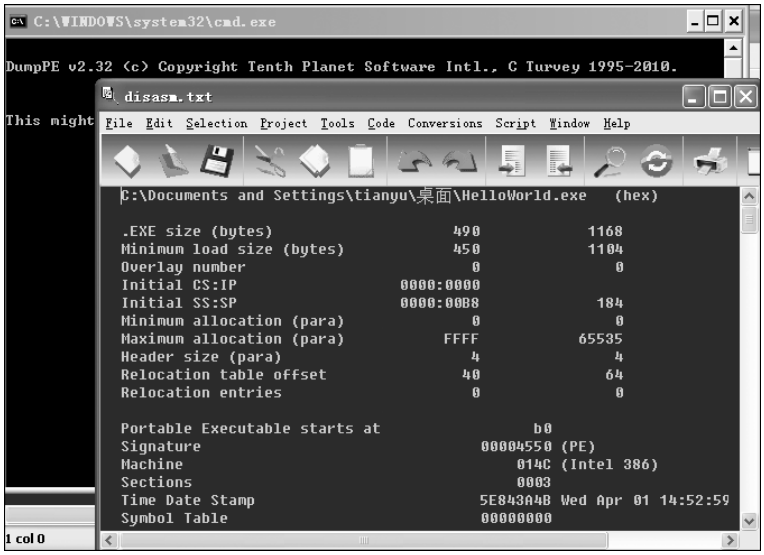

图 5-11　打开反汇编 exe 文件

仔细观察，可以发现相关内容是 PE 文件格式的内容。

然后在后面是反汇编的代码，如图 5-12 所示。

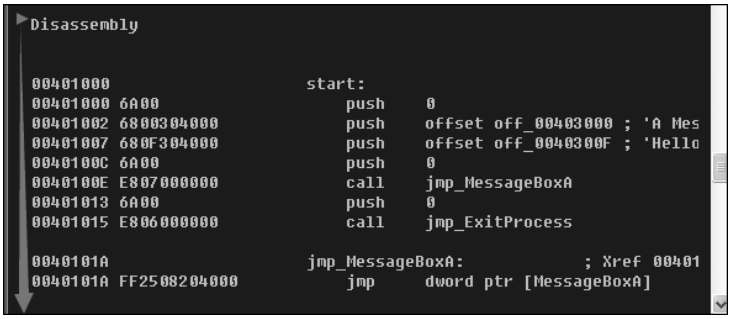

图 5-12　反汇编代码

通过图 5-13 的对比可以发现很明显的区别。

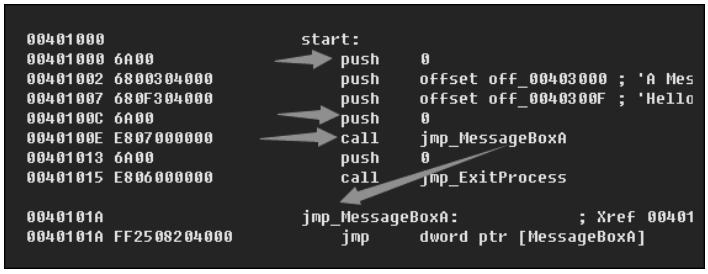

图 5-13　对比

反汇编代码的逻辑大致是将数据压栈，首先执行 call jmp_messageBoxA，然后开始执行弹窗操作，最后再跳转到 jmp_ExitProcess。

5.4　病毒重定位

5.4.1　实验目的

深入理解病毒重定位的基本思路和方法，掌握病毒在内存中的重定位技术，为后续深入研究和防范恶意软件奠定坚实的基础。

5.4.2　实验内容及实验环境

1. 实验内容

在 HelloWorld.exe 中添加一段代码，具体要求如下。

(1) 该段代码弹出一个对话框(标题：武大网安病毒重定位，内容：姓名＋学号)。

(2) 该段代码同时包括代码和字符串数据。

(3) 该段代码可以插入.text 节的任意指令之间，而不修改该段代码中的任何字节，对其可移动性进行验证(移动产生的空闲区域可用 nop 代替)。

2. 实验环境

(1) 硬件：一台装有 Windows 操作系统的普通 PC(使用虚拟机亦可)。

(2) 软件：OllyDebug。

5.4.3　实验步骤

1. 观察 call 指令

call＝push＋jmp。

如果 call 调用一个内存地址，那么编译器编译的是相对偏移。

call 指令调用的地址＝call 指令所处偏移＋5＋相对偏移。

首先用 OD 打开 HelloWorld.exe 文件，在地址 40100E 开始使用 nop 填充字段，结果如图 5-14 所示。

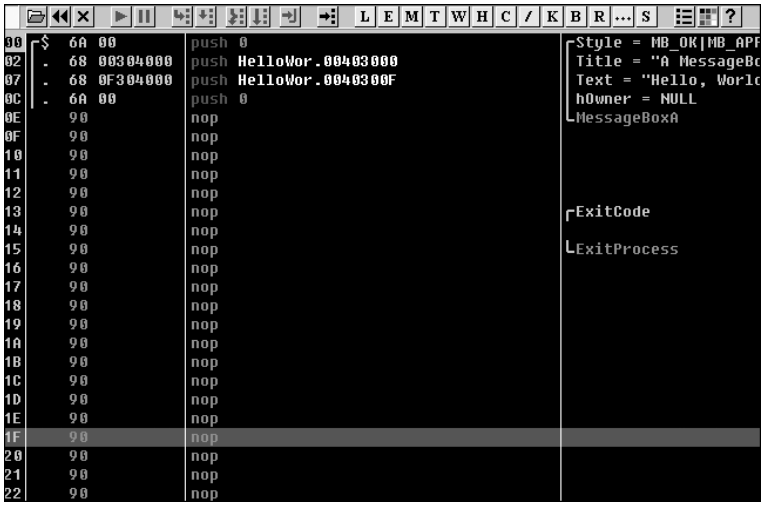

图 5-14　填充 nop

2. 在地址 100E 处添加汇编指令 call MessageBoxA

该步骤如图 5-15 所示。

图 5-15　添加汇编指令

3. 选择 1020 地址插入代码

首先添加 push 0，之后 push 的第二个参数是“武大网安病毒重定位”，再加上最后一字节为 0，一共 19 字节，也就是 13H。需要使用 call 指令的特性，如图 5-16 所示。

注意最后添加 0，这样一来 call 指令向后跳转 13H，如图 5-17 所示。

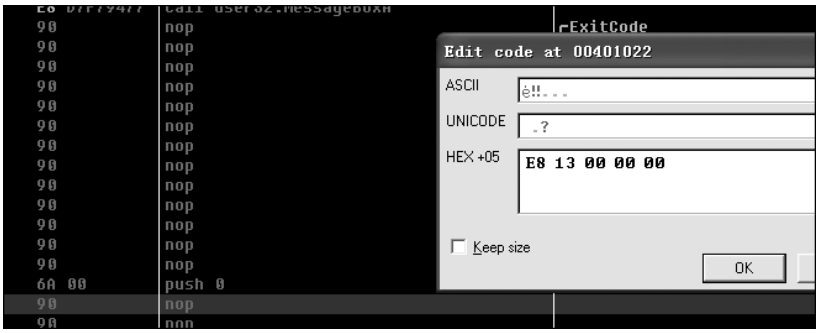

图 5-16　添加 push 0

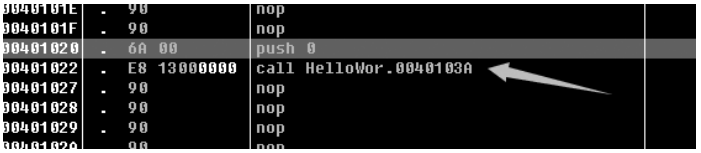

图 5-17　添加 call

接下来在下一条插入上述数据(在 ASCII 处输入“武大网安病毒重定位”),如图 5-18 所示。

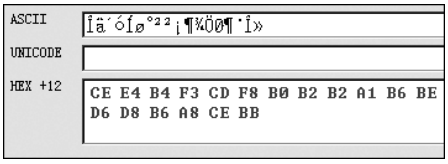

图 5-18　添加 ASCII 文字数据

运行结果如图 5-19 所示。

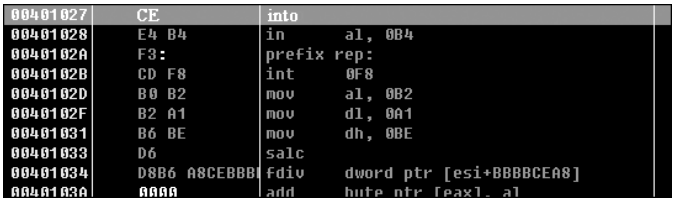

图 5-19　运行结果

然后输入第三个参数,“名字+学号”一共 16 字节,10H。此时插入这个数据后,数据偏移到了 40104F,所以先添加一个 call 指令到这个地址,如图 5-20 所示。

图 5-20　添加 call 指令

接着编辑代码输入数据“名字+学号”,如图 5-21 所示。

然后添加 push 0,再调用 MessageBoxA,如图 5-22 所示。

注意:由于为了能使得弹窗在不同的位置都能被调用,所以这里需要使用 IAT 来进行 MessageBox 的寻址。通过查看 PE 文件结构,可知 IAT 的地址为 00402008H。

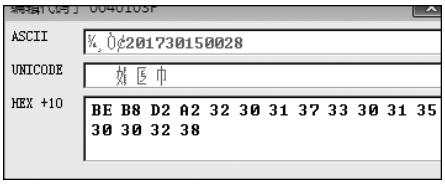

图 5-21　编辑数据

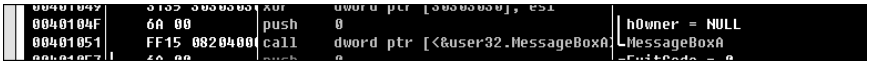

图 5-22　调用 MessageBoxA

最后退出进程，如图 5-23 所示。

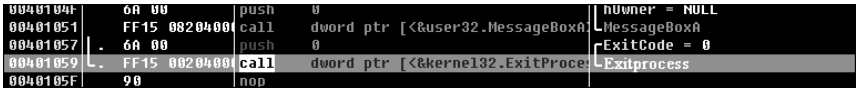

图 5-23　退出进程

这里也需要找到 IAT 的地址然后调用退出函数，如图 5-24 所示。

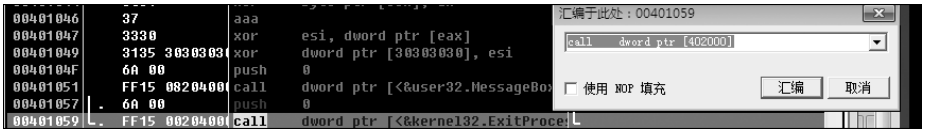

图 5-24　调用退出函数

4. 将修改复制到可执行文件

存储之后，对代码进行移动。

首先进行数据跟随，结果如图 5-25 所示。

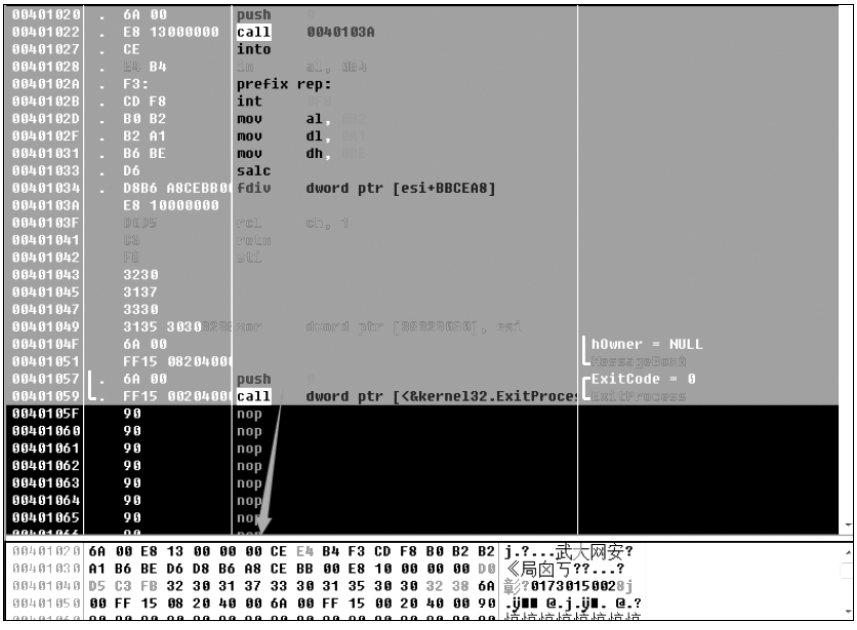

图 5-25　数据跟随结果