黑客防线

2007

精华奉献本

上册

→高危漏洞详尽分析

→木马免杀系列专题

→黑客编程高级指南

→系统防护终极策略

《黑客防线》编辑部 编

50个高危系统漏洞底层分析！

150例流行黑客技术实战指南！

200篇黑客入侵事件剖析！

1200MB黑客技术经典操作视频盘点！

人民邮电出版社
POSTS & TELECOM PRESS

黑客防线

2007

精华奉献本

上册

《黑客防线》编辑部 编

人民邮电出版社
北京

图书在版编目(CIP)数据

《黑客防线》2007精华奉献本/《黑客防线》编辑部编.
北京: 人民邮电出版社, 2007.1
ISBN 978-7-115-15560-3

Ⅰ.黑... Ⅱ.黑... Ⅲ.计算机网络-安全技术
Ⅳ.TP393.08

中国版本图书馆CIP数据核字(2006)第144844号

内容提要

《黑客防线2007精华奉献本》是国内最早创刊的网络安全类媒体之一《黑客防线》总第61期到第72期的精华文章摘要, 杂志理念"在攻与防的对立统一中寻求突破"完美地体现在本书中。全书按攻防对立的关系, 分为上、下两册(本册为上册), 并附带2张包含1200MB安全技术录像的光盘。收录的文章通俗易懂、图文并茂, 在每个关键知识点和细节上还有详细的"小知识"、"小提示", 方便大家理解和阅读。在栏目划分上, 选取了网络安全技术上最热门、读者最喜欢的各大栏目, 典型的有脚本攻防、漏洞攻防、黑器攻防等, 同时选取国内首创的"溢出研究"栏目中的精品文章, 非常适合各层次读者学习!

本书适合各高校在校生、网络管理员、安全公司从业者、黑客技术爱好者阅读。

黑客防线2007精华奉献本(上册)

编　　:《黑客防线》编辑部
责任编辑: 魏雪萍

出版发行: 人民邮电出版社发行　北京市崇文区夕照寺街14号A座(100061)
读者热线: 010-62145877
印　　刷: 中煤涿州制图印刷厂
经　　销: 全国各地新华书店

开　　本: 782×1092　1/16
印　　张: 36
字　　数: 3800千字
2007年1月第1版　2007年1月北京第1次印刷

ISBN 978-7-115-15560-3/TP·5870

全套(含上下册)定价: 39.00元(附两张光盘)

本书如有印刷质量问题(错页、掉页、残页等), 请您与我们联系, 我们负责调换。
联系电话: 010-62141446　E-mail: yougoubu@hacker.com.cn

致读者

黑客一词，源于英文Hacker，原指热心于计算机技术、水平高超的电脑专家，尤其是程序设计人员，他们发现系统漏洞和各种程序错误，并设法修补这样的漏洞。从原来的定义来说，黑客无疑是网络中最让人觉得神秘且崇高的人群。但世界上没有绝对好的东西，黑客也是这样。信息高科技时代的到来，给人类各方面发展都带来了前所未知的巨大进步，但同时也带来了前所未知的巨大灾难。黑客这一神秘群体，由于掌握了核心的技术、最有破坏力的漏洞，所以渐渐分离出来两个部分，一部分是为网络安全而奉献精力的真正黑客，而另一部分是在网络上为所欲为，拥有强大破坏力，行踪神秘的Cracker！

我国互联网业界风起云涌，网络从最初的新奇，到今弥散到人们的日常生活，它改变了传统的信息消费渠道、抢占了新闻受众、影响着大众娱情……面对汹涌而来的信息浪潮，网络的脆弱性也在经受着无数黑客、Cracker的一次次考验和打击。面对这样的转变，为了向广大网友普及网络安全常识、提高网络管理员的技术水平，《黑客防线》应运而生——一本崭新的集网络安全专业技术、专业资讯为一体的专业杂志面世了。

作为中国最早的网络安全类杂志之一，已经历了6年的风风雨雨。一直以"在攻与防的对立统一中寻求突破"为理念的我们，在普及网络安全知识的同时，针对网络上黑客攻击事件、网络漏洞等方面策划出最快、最新、最权威的技术文章奉献给大家。同时也以为"黑客"正名为己任，告诉我们的每个读者什么是真正的黑客，什么是黑客精神，什么是Cracker。

在黑客防线成长的过程中，通过我们不断推出的攻防实验室，非常多的朋友学习到了很多实用的网络安全攻击、防御知识，虽然不可避免地存在一些缺陷，但我们一直在积极地听取大家的意见，希望能更适合大家的要求 同时，很高兴地看到大家通过实验室而让技术水平得到提高，而且每个朋友的成长几乎都能带动周围一圈人意识的改变，这样才真正地体现出了"攻防对立统一"的精髓。

眼下互联网上的攻击事件越来越多，各大网络管理员、系统维护者都在为入侵防护而头疼，生怕自己一不小心就让虎视眈眈的"黑客"进入了服务器。其实管理员们根本没必要这样担惊受怕。要知道任何攻击技术都不是万能的，每种入侵方法都有它自己针对的漏洞，如果网管们能掌握这样的攻击技术，本身就是一名出色的善意"入侵者"，那在知己知彼的情况下，防护不就变得非常简单了吗？这也是《黑客防线》一直倡导的"在攻与防的对立统一中寻求突破"理念的最终体现。我们的攻防实验室就是为了深入这一理念而开展活动的，希望每个喜欢入侵技术、每个喜欢安全技术的朋友都能在我们的实验室中学到自己需要的知识。在此前提下，根据各读者的强烈建议，我们特别精选了200篇经典攻防技术文章，收集了200个全程演示的攻防录像，融合在《<黑客防线>2007精华奉献本》里，以便喜欢黑客技术的朋友们系统地学习最新的网络安全技能，做好最强的防护系统。

最后，我们再次感谢广大读者对我们的支持，感谢人民邮电出版社同我们的合作，使更多读者可以看到我们的图书，让国内更多的人了解到黑客技术，并借此来提高国内网络安全的整体水平。

读者有任何疑问可以来我们论坛发帖讨论，地址是http://www.hacker.com.cn/newbbs，我们的编辑和黑客爱好者们将在第一时间为您答疑，帮助您解决安全故障和系统防护问题，共同创造一片纯净的中国互联网空间！

黑客防线2007精华奉献本目录（上册）

编程解析

黑器攻防

网管之家

密界寻踪

Socket: 有几个读者发来消息说想看C#类的文章，这次让大家得偿所愿。用C#做东西感觉还真容易

适合读者: 入侵爱好者，C # 爱好者

前置知识: C # 基础

用C#写经典小工具

文/图 ngaut

《黑客防线》杂志上有人提到C # 在网络安全方面的问题，于是我赶紧把自己的一些心得写下来与大家分享，很明显，用C # 来写木马之类的东东是不现实的。因为存在以下缺点: 生成的可执行程序太大，速度慢。当然了，最大的缺点是需要.net运行库。给别人装木马之前还要帮忙装一个.NET Framework，100多兆的运行库肯定让我们先崩溃，呵呵。不过不要伤心，C #用来开发简单的扫描器和系统自身的安全管理以及蜜罐等还是有用武之地的。下面我们先从最经典的进程管理开始:

C#提供给我们强大而便捷的Process类，使我们的开发更容易、更方便。这对于编写高效、快速的应用程序是非常有帮助的。考虑到很多穷人无力购买昂贵的Microsoft Visual Studio.NET 2003(我们学校机房的Microsoft Visual Studio .NET 2003，当然了是D版的)，我费了点时间读了SharpDevelop的文档，并且安装测试了SharpDevelop，这是个免费精致的C#的IDE。

按照标准文档安装不成功或程序启动出错，请运行SharpDevelop\bin\setup目录下的PostInstallTasks.bat，因为SharpDevelop对中文的支持太差，我用的版本都是乱码，所以有些注释是用英语写的，我英语太差，希望不会引起大家的误解。考虑到一部分人的需要，我写稿的时候加入了中文注释。另外限于篇幅关系，我不可能详细讲解每个函数的用途，如果大家遇到不清楚的地方请查阅MSDN。

列出所有进程

写出属于自己的Pslist，非常的简单，呵呵，比C语言方便多了。

```
    //winxp + SharpDevelop_1.0.3.1768
    //winxp + Microsoft Visual Studio .NET 2003测试通过 (偶跑到学校的机房又测试了一次)
    //GetProcesses() Creates a new Process component for each process resource on the local computer.
    using System;
    using System.Diagnostics;  //不要忘了这个名字空间
    class ListProcess
    {
        public static void Main()
        {
    Console.WriteLine("Process infomation:");
    //取得所有进程信息保存到allProcess数组
    Process[] allProcess = Process.GetProcesses();
    //allProcess.Length是进程总数
    Console.WriteLine("Process count:{0}:", allProcess.Length);
    //输出所有进程信息
    foreach (Process thisProc in allProcess)
    {
    string procName = thisProc.ProcessName;//进程名
    int procID = thisProc.Id;                    //进程ID
    Console.WriteLine("Process:{0}, ID:{1}", procName, procID);
        }
    Console.ReadLine(); //暂停程序，相当于c语言中的gets()
        }
    }
```

运行结果如图1 所示。

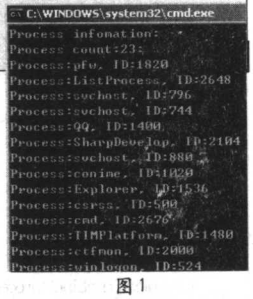


图1

杀死指定进程

杀死进程之前你总要列出系统进程吧，代码很简单。程序运行结果如图2 所示。

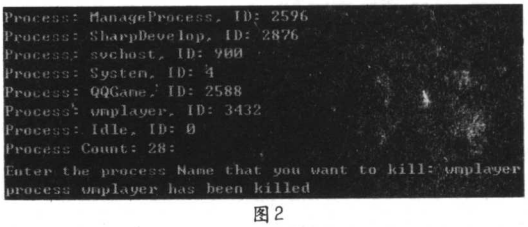


图2

```
    //winxp + SharpDevelop_1.0.3.1768
    //winxp + Microsoft Visual Studio.NET 2003测试通过
    using System;
    using System.Diagnostics;
    class ManageProcess
    {
        //List All Processes
        private void ListAllProcess()
        {
    Console.WriteLine("Process information:");
    Process[] allProcess = Process.GetProcesses();
    foreach (Process thisProc in allProcess)
        {
    string processName = thisProc.ProcessName;
    int processID = thisProc.Id;
    Console.WriteLine("Process: {0}, ID: {1}", processName, processID);
        }
        //输出进程总数
```

```
    Console.WriteLine("Process Count: {0}:", allProcess.
Length);
        Console.WriteLine();
        }
    //Kill specific Process
    private void KillProcess(string processName)
    {
        try
        {
    Process [] thisproc = Process.GetProcessesByName
(processName);
        //thisproc.Length：名字为的进程总数
        //可能是多进程哦，如svchost就有4个
        //Process found!
        if (thisproc.Length > 0)
        {
        for (int i=0; i<thisproc.Length; i++)
                {
        //尝试关闭进程，并释放资源
        if(!thisproc[i].CloseMainWindow())
                {       //强制关闭
                        thisproc[i].Kill();
                }
    Console.WriteLine("process {0} has been killed",
processName);
                }
        }
        else
        {
    Console.WriteLine("process {0} not found!", processName);
        }
        }
    catch   //出现异常，表明kill进程失败，提示用户
        {
    Console.WriteLine("kill process {0} faild", processName);
        }
        Console.ReadLine();
    }
    public static void Main()
    {
    ManageProcess process = new ManageProcess();
    process.ListAllProcess();          //列出所有进程
    Console.Write("Enter the process Name that you want
to kill: ");
    string procName = Console.ReadLine();//取得要杀
死的进程名
        process.KillProcess(procName);
        }
    }
```

取得指定URL的源代码

我知道的方法有两种，下面是第一种，程序运行结果如图3所示。

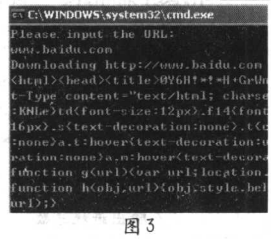


图3

```
    //winxp + SharpDevelop_1.0.3.1768
    //winxp + Microsoft Visual Studio .NET 2003测试通
过
    //DownloadData()函数原型：public byte[] DownloadData
(string address);
    //作用：Downloads data from a resource with the
specified URI.
    using System;
    using System.IO;
    using System.Net;
    using System.Text;
    class GetWebSourceCode1
    {
        public static void Main()
        {
    string remoteUrl;     //远程URL
    Console.WriteLine("Please input the URL:");
    //取得URL，注意这里要加上"http://"，如果不加
上，则必须有用户输入
    remoteUrl = "http://" + Console.ReadLine();
    //  建立一个WebClient实例
    WebClient myWebClient = new WebClient();
    //  下载主页数据
    Console.WriteLine("Downloading " + remoteUrl);
    // Download the Web resource and save it into a
data buffer.
        try
        {
    byte[] myDataBuffer = myWebClient.DownloadData
(remoteUrl);
    //输出网页的源代码
    string download = Encoding.ASCII.GetString
(myDataBuffer);
    Console.WriteLine(download);
    Console.ReadLine();
        }
        catch               //URL解析失败
        {
    Console.WriteLine("sorry, The URL address is invalid!!!
");
                Console.ReadLine();
        return;
        }
        }
    }
```

第二种方法的主要函数是OpenRead()，也很简单，限于篇幅关系请大家看光盘上代码。

取得远程Web服务器类型

通过http ResponseHeaders取得远程Web服务器类型，程序运行结果如图4所示。

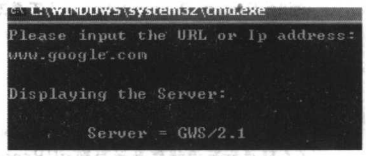


图4

探测结果显示Google用的GWS/2.1，而我们用的IIS/5.0，基本原理是通过分析http Response Headers，取得http ResponseHeaders中的server信息。

```
//取得远程Web服务器类型
//winxp + SharpDevelop_1.0.3.1768
//winxp + Microsoft Visual Studio.NET 2003测试通过
using System;
using System.Net;
using System.Text;
namespace GetRemoteServerType
{
class GetRemoteServerType
{
public static void Main()
{
Console.WriteLine("Please input the URL or Ip address:
");
string remoteUrl = "http://" + Console.ReadLine();
// Create a new WebClient instance.
WebClient myWebClient = new WebClient();
// Download the Web resource and save it into a
data buffer.
try
{
byte[] myDataBuffer = myWebClient.DownloadData(remoteUrl);
    }
    catch
    {
Console.WriteLine("The URI address is invalid or An
error occurred while downloading data!");
    Console.ReadLine();                    //暂停
            return;
    }
// 建立一个WebHeaderCollection对象
WebHeaderCollection myWebHeaderCollection =
myWebClient.ResponseHeaders;
Console.WriteLine("\nDisplaying the Server:\n");
    // 循环查找
for (int i=0; i < myWebHeaderCollection.Count; i++)
    {
// 只显示Server类型，若去掉"Server"判断条件则显
示更多信息
if (myWebHeaderCollection.GetKey(i) == "Server")
                {
Console.WriteLine ("\t" + myWebHeaderCollection.GetKey
(i) + " = "
    + myWebHeaderCollection.Get(i));
                    break;
            }
    }
    Console.ReadLine();
    }
}
}
```

简单的端口扫描

一般的端口扫描的原理其实非常简单，只是简单地利用操作系统提供的Connect()系统调用，与每一个感兴趣的目标计算机的端口进行连接。如果端口处于Listen状态，那么Connect()就能成功 否则，这个端口不能用，也没有提供服务。这个技术的一个最大的优点是不需要任何权限，系统中的任何用户都有权利使用这个调用 。

另一个好处就是速度快，如果对每个目标端口以线性的方式使用单独的Connect()调用，那么将会花费相当长的时间。可以同时打开多个套接字，从而加速扫描。使用非阻塞I/O，允许设置一个低的时间用尽周期，同时观察多个套接字。但这种方法的缺点是很容易被发觉，从而被过滤掉。目标计算机的Logs文件会显示一连串的连接和连接时出错的服务消息，并且能很快地使它关闭。

有了原理做事就比较容易了，呵呵，至少方向是对的，程序是使用C#提供的TcpClient类来实现的。

```
//sample port scanner
using System;
using System.Net;
using System.Net.Sockets;
class portScanner
{
public static void Main()
{
Console.WriteLine("Start scanning.....");
int port = 21;
Console.WriteLine("Please input the URL or Ip address:
");
string server = Console.ReadLine();
try
{
// 如果建立tcp连接成功，表明端口开放
TcpClient client = new TcpClient(server, port);
Console.WriteLine("{0} port {1} is Openned!!!", server,
port);
client.Close();
}
catch (SocketException e)
{
Console.WriteLine("{0} port {1} is not Openned!", server,
port);
// 建立连接不成功，提示不成功的原因
Console.WriteLine("SocketException: {0}", e);
    }
    Console.Read();
    }
}
```

简单的蜜罐

这个也比较简单，因为是简单的蜜罐，所以只显示Listen的数据就可以了。这里就不在继续介绍了，留给大家做吧，呵呵。

PS：为了这篇文章，花了一个星期的时间，写完的时候觉得自己好累了，文笔不好还请多见谅。

由于我没有自己的电脑，编程和写文章的时间都是蹭来的。其实仔细想想编程真的不是很难，我们这样的菜鸟也可以编程。希望大家在创造自己工具时一路走好。能有更多的朋友提供更好的意见和建议。

适合读者：C＃程序员，木马爱好者
前置知识：无

C＃实现从自身资源提取EXE

文　程晨

很多木马都有生成服务端的功能，即从自身的EXE文件再生成一个新的EXE文件。我在网上翻了翻代码，VB和VC++有很多方法实现。可是一直没有找到C＃的，可能是C＃编写木马不实用，因为大多数系统必须先安装微软的NET Framework才能运行C＃程序。然而笔者认为再过几年以后，随着微软新系统的推出，C＃木马必然会辉煌起来，所以懂得如何自身生成EXE文件还是有必要的。

在实现这个功能的同时我再把C＃是如何从资源中提取图片和字符串的代码一并写出来。

第一步：建立一个资源文件

1.新建一个控制台应用程序项目，在Class1.cs开头加上

```
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Drawing.Imaging;
// 以上部分是调用图片文件所必须添加的。
using System.Resources;
using System.IO;
```

2.在主函数中加上

```
// 声明对象，将资源存入MyResources.resources文件中
System.Resources.ResourceWriter rw = new System.Resources.ResourceWriter("MyResources.resources");
// 声明读取一个二进制文件。
```

小提示 这里的27136为要读取EXE文件的字节数！可以用鼠标右键调出文件属性查看！当然可以写得更智能一些，让程序自动读出字节数。

```
//filename.exe.是你读取的EXE文件名。
FileStream fs=File.Open("filename.exe",FileMode.Open);
// 用二进制方式读取文件
BinaryReader br= new BinaryReader(fs);
byte [] bytes=new byte[27136];
br.Read(bytes,0,27136);
fs.Close();
// 添加到资源
rw.AddResource("Myexe",bytes);
// 载入图片
System.Drawing.Image Img = System.Drawing.Image.FromFile("temp.jpg");
rw.AddResource("MyPic",Img);
// 声明字符串
string Mystr = "I love China.";
rw.AddResource("MyString",Mystr);
rw.Close();
// 将编译后的程序和filename.exe及temp.jpg放入同一子目录运行就可以得到资源文件：MyResources.resources
```

第二步：从资源文件中提取资源

1.新建一个Windows应用程序项目，添加一个PictureBox控件来放图片（设Name为pictureBox再加几个按钮控件，具体代码程序请见附件）

在Form1.cs开头加上

```
using System.Resources;
using System.Reflection;
using System.IO;
```

2.在主函数中加上

```
// 通过反射得到资源程序集
System.Reflection.Assembly asm = Assembly.GetExecutingAssembly();
// 初始化资源管理对象
// 这一步很多人会出错！一定要注意这里的ResourceConsumer，其实这是你命名空间的名字。在这个例子中项目名我取的是ResourceConsumer，系统自动将命名空间设为：namespace ResourceConsumer
rm = new ResourceManager("ResourceConsumer.MyResources",asm);
```

3.在相应的按钮控件处理函数中添加

```
// 从资源中读取EXE文件并还原
private void button1_Click(object sender, System.EventArgs e)
{
// 创建EXE文件
FileStream fs=File.Open("filename.exe",FileMode.Create);
// 定义文件大小
byte[] bytes=new byte[27136];
// 读取资源文件
bytes=(byte[])rm.GetObject("Myexe");
// 二进制方式写入文件！
BinaryWriter bw=new BinaryWriter(fs);
bw.Write(bytes);
fs.Close();
}
// 从资源中提取图片并显示
private void LoadImg_Click(object sender, System.
```

```
EventArgs e)
    {
    pictureBox.Image = (Image)rm.GetObject("MyPic");
        }
    //从资源中提取字符串并显示
    private void LoadString_Click(object sender, System.
EventArgs e)
    {
    string str = (string)rm.GetString("MyString");
    MessageBox.Show(str);
        }
```

4．别忘了还要把原来编译好的的资源文件MyResources.resources调入。调入方法：文件->添加现有项->。此时将文件类型设为*.*，然后找到前面编译好的资源文件MyResources.resources打开即可！以上代码在Windwos 2003+Visual Studio.net 2003 下调试通过！

在这个例子中，仅仅是实现了如何从自身资源提取EXE文件，并没有加入一些像木马那样的配置信息，因此本文也仅仅是抛砖引玉的作用！文中不妥之处还望高手指教！为了让大家更好地研究这段代码，我提供了一个简单的源代码文件（参见），它是实现ASP文件加密的一个程序。此程序会自动在Windows临时文件夹生成temp.exe（其实就是screnc.exe），然后调用temp.exe对ASP文件实现加密操作！呵呵，其实就可以给在CMD下运行的程序开发Windows的GUI SHELL程序了。

适合读者：程序员，游戏爱好者
前置知识：C/C++基础

自己写汉诺塔游戏

文 袁嘉尉

本来我只想做一个动画演示的程序，然后发给黑防大编，后来想了想，不如做成游戏吧，我把想法告诉了Socket，他说很好，然后我就做了这个程序，其实还有很多功能可以完善，但是我觉得挺浪费时间的，也就没加了。

我先说说我编动画的思路吧，汉诺塔，顾名思义，引用谭浩强老爷爷书中的句子：也就是老和尚挪盘子，把*n*个盘子按照从小到大的顺序从A挪到C上，途中要经过B，所以我就先定义3个数组int c[99],d[99],e[99],数组中的99也就是可以挪动的盘子数的上限，其实设置成76是最好的，因为一般的电脑的分辨率都是1024×768，所以如果一个盘子的高度是10的话，那么最多能放76个盘子，有了数组后，就可以用它来记录盘子的大小和顺序了，然后就是定义一些头文件和一些函数还有定义一些宏：

```
#include <graphics.h>。
#include <dos.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <process.h>
#define a xy[0]
#define b xy[1]
#define lt 0x4b00
#define rt 0x4d00
#define et 0x1c0d
char xy[1],rr;
int m,i=1,g=0,h=0,k=1,l=1,o=0,hh,gg,xx=1,yy,zz,n=1,p=1,
w=1,ww=1,pp=1;
```

接下来就可以看程序部分了，首先我们从main函数开始看：

```
main()
{
char dd;
int driver=DETECT,mode;
initgraph(&driver,&mode,"");
bar(0,0,1024,768);
closegraph();/*清屏*/
clrscr();/*光标回到起始点*/
printf("      1.play hanoi\n\n\n\n\n\n\n\n\n
2.see computer play\n\n\n\n\n\n\n\n\n
3.exit");/*打印出选项*/
while(1)
{
dd=getch();
if (dd==49)play();
else if(dd==50)print();
else if(dd==51)abort();/*判断按键，如果为1，则运行play
()函数，如果为2，则运行print(),函数，如果为3则退出*/
}
}
```

接下来我们可以先看演示的部分，也就是选择“2。看电脑玩”，具体代码如下：

```
void print()
{
int driver=DETECT,mode;
initgraph(&driver,&mode,"");
bar(0,0,1024,768);
closegraph();
clrscr();/*同样是清屏，把光标移到初始位置*/
printf("input the number of diskes:");
scanf("%d",&m);/*输入碟子的数目*/
hanoi(m,m,'A','B','C');/*调用hanoi函数来计算路径*/
```

```
}
extern hanoi(int n,int i,char one,char two,char three)
{
if(n==1)move(i,one,three);
else
{
hanoi(n-1,i,one,three,two);
move(i,one,three);
hanoi(n-1,i,two,one,three);
}
}
/*通过递归调用hanoi函数来实现自动确定步骤*/
/*在调用hanoi的过程中调用了move函数，用来写和计算出c,d,e数组中的数值*/
void move(int f,char x,char y)
{
int oo=0;/*这个是为了区分游戏和演示部分用的变量*/
if(k==1)
{
for(i=1;i<=f;i++)
{d[i]=i;
k=2;
}
/*给最左边的汉诺塔A，也就是c数组赋初值，因为在没运行任何步骤之前，A座是由上到下顺序排列的，B,C座都是空的，所以只要给c数组赋值就可以，变量k的作用就是让这个赋值只运行一边，因为c数组的内容在以后的递归调用中会不断的变，不能每次都赋值*/
for(i=1;i<=(f+1)/2;i++)
{
hh=f+1-i;
gg=d[i];d[i]=d[hh];d[hh]=gg;
}/*颠倒数组里面的内容，例如原来数组是1,2,3,4运行这段代码后就会变成4,3,2,1，调用它的原因在下面再解释*/
draw(oo,f,l,g,h);/*调用一次draw函数，就一次*/
}
a=x,b=y;/*纯属为了书写方便*/
for(i=1;i<=(f+1)/2;i++)
{
hh=f+1-i;
gg=d[i];d[i]=d[hh];d[hh]=gg;
}
```

再次颠倒，在下面还会有一个颠倒，要反复颠倒的原因就是，画图和计算对于数组顺序的需求是不一样的，计算需要正确的顺序，也就是c[i]=i的顺序，而画图需要颠倒过来，否则就会画出倒着摆放的汉诺塔，这样配合下面的颠倒就会形成循环，使得在调用draw函数的时候用的是逆序，在调用计算代码的时候用的是顺序。

```
printf("The step to moving %3d diskkes:",f);
printf("\n%c-->%c",x,y);/*写出挪动汉诺塔的步骤*/
if(a=='A'&&b=='C')
{h=h+1;e[h]=d[l];d[l]=0;l=l+1;}
else if(a=='A'&&b=='B')
{g=g+1;d[g]=d[l];d[l]=0;l=l+1;}
else if(a=='B'&&b=='C')
{h=h+1;e[h]=d[g];d[g]=0;g=g-1;}
else if(a=='B'&&b=='A')
{l=l-1;d[l]=d[g];d[g]=0;g=g-1;}
else if(a=='C'&&b=='A')
{l=l-1;d[l]=e[h];e[h]=0;h=h-1;}
else if(a=='C'&&b=='B')
{g=g+1;d[g]=e[h];e[h]=0;h=h-1;}/*这里面的A,B,C分别代表的是3个放碟子的底座，因为在hanoi函数中计算了步骤，所以在这里判断步骤，来确定3个座上汉诺塔的情况，也就是c,d,e函数的情况*/
for(i=1;i<=(f+1)/2;i++)
{
hh=f+1-i;
gg=d[i];d[i]=d[hh];d[hh]=gg;
}/*第3次颠倒，为画图做准备*/
draw(oo,f,l,g,h);
```

调用draw函数来画出汉诺塔的移动情况，因为是不断的计算，调用，每一步就计算一次，画一次，所以这个演示程序原理就是靠不断地刷新c,d,e数组的内容来确定3个底座上碟子的情况。每个碟子的大小等于本身代表它的数字，3个数组记录了所有碟子的位置，但是实际带入draw函数的只有最上面的那个碟子的大小，因为每次需要移动的碟子只有一个，只要不断判断3个底座最上面那个碟子就可以完成所有的步骤。

```
void draw(int o,int u,int aa,int bb,int cc)
{
int driver=DETECT,mode;
initgraph(&driver,&mode,"");/*图形初始化*/
if(d[aa]==0&&d[bb]==0&&e[cc]==1)win();/*因为汉诺塔移动完成以后A和B底座都是空的了，而C底座变成了初始的A底座，也就是最上面的碟子的大小是1，所以当达到这种情况时也就代表成功了*/
if(o!=0)
{
if(o==1)circle(100,290-(aa+u+1)*10,6);
if(o==2)circle(300,290-(aa+u+1)*10,6);
if(o==3)circle(500,290-(aa+u+1)*10,6);
if(pp!=1)
{
for(i=1;i<=(m+1)/2;i++)
{
hh=m+1-i;
gg=d[i];d[i]=d[hh];d[hh]=gg;
}
}
}
```

这段代码是跟游戏部分有关的，因为在选择和移动时无法知道光标到了哪里，这段函数首先判断是游戏还是演示，也就是通过"o"变量，如果是游戏就会根据o的值来确定光标究竟落到了哪座塔上。至于后面那段颠倒的代码则是在第一次调用draw函数的时候不调用它，等第2次，p!=1的时候才调用，也是因为计算和画图对数组顺序的需求不同。

```
for (i=1,aa=1,bb=1,cc=1;i<=u+1;i++,aa=aa+1,bb=bb+1,cc=cc+1)
{
```

```
    bar(100-c[aa]*10,350-(aa+u+1)*10,100+c[aa]*10,360-
(aa+u+1)*10);
    bar(300-d[bb]*10,350-(bb+u+1)*10,300+d[bb]*10,360-
(bb+u+1)*10);
     bar(500-e[cc]*10,350-(cc+u+1)*10,500+e[cc]*10,360-
(cc+u+1)*10);
          }
```

这段就是画汉诺塔和中间那根柱子的代码。循环碟子的次数加1 原因是为了让柱子多个头，方便看，而实现这个的原因就是bar 函数在左与右相等的时候就会画一条线。画碟子的方法就是通过循环u+1 次，把c，d，e 数组中的每个数值都画出来。pp=2 时；改变pp 的值，让上面运行的代码在第2次的时候可以运行sleep(1)；让程序停止1秒，用来观看，否则将会直接看到结果，没有过程。*/closegraph();/*关闭图形初始化，如果没有这句，就会不断地加载图形，依据我的电脑的配置，到了第1 2 步就进行不下去了，自动跳出，因为没有释放图形，导致内存不够。

紧接着就会回到move函数，move函数又回到hanoi，不断地调用，直到c[aa]=0,d[bb]=0,e[cc]=1的时候执行win 函数：

```
win()
{
int driver=DETECT,mode;
initgraph(&driver,&mode,"");
bar(0,0,1024,768);
closegraph();
clrscr();
printf("you win! ");
sleep(3);
main();
}
```

这个不需要注释什么，停顿3 秒后回到主函数。下面再说一下游戏部分，看主函数可以知道游戏部分是调用play函数

```
void play()
{
int key;
int driver=DETECT,mode;
initgraph(&driver,&mode,"");
bar(0,0,1024,768);
closegraph();
clrscr();
printf("input the number of diskes:");
scanf("%d",&m);/*初始化的一些东西，同样是询问你盘子
的个数*/
if (p==1)
{
for(i=1;i<=m;i++)
{c[i]=i;p=2;}
for(i=1;i<=(m+1)/2;i++)
{
hh=m+1-i;
gg=c[i];c[i]=c[hh];c[hh]=gg;
}
draw(n,m,xx,yy,zz);
}
/*这部分和move 函数里面的基本一样，这个也是3 次颠倒，只不过最后一次颠倒*/
while(1)
{
for(i=1;i<=(m+1)/2;i++)
{
hh=m+1-i;
gg=c[i];c[i]=c[hh];c[hh]=gg;
}
/*颠倒一次，为了计算*/
key=bioskey(0);
if (key==rt&&n==3)n=1;
else if (key==lt&&n==1)n=3;
else if (key==lt&&n>1)n=n-1;
else if (key==rt&&n<3)n=n+1;
/*判断方向按键与n 的情况，来决定光标的位置*/
else if (key==et&&ww==1)
{
ww=2;
if((n==1&&c[xx]==0)||(n==2&&d[yy]==0)||(n==3&&e[zz]
==0))
{clrscr();printf("can not put a empty line!");sound(440);
delay(500);nosound();ww=1;}
if(n==1&&ww==2){w=c[xx];c[xx]=0;xx=xx+1;}
if(n==2&&ww==2){w=d[yy];d[yy]=0;yy=yy-1;}
if(n==3&&ww==2){w=e[zz];e[zz]=0;zz=zz-1;}
}
else if (key==et&&ww==2)
{
ww=1;
if((n==1&&c[xx]<w&&c[xx]!=0)||(n==2&&d[yy]<w&&d[yy]!
=0)||(n==3&&e[zz]<w&&e[zz]!=0))
{clrscr();printf("can not put the big plate on the little
plate!");sound(440);delay(500);nosound();ww=2;}
if(n==1&&ww==1){xx=xx-1;c[xx]=w;}
if(n==2&&ww==1){yy=yy+1;d[yy]=w;}
if(n==3&&ww==1){zz=zz+1;e[zz]=w;}
}
draw(n,m,xx,yy,zz);
}
}
```

用回车和ww 变量判断提取，落下的代码，应该说是这个程序里比较好的一段代码。ww 是用来识别是提取还是落下，在ww=1 也就是提取情况下，首先令ww=2，也就是变为落下的标识，然后判断是否提取的底座上没有碟子。如果没有再变回提取状态，使ww=1，而且还有错误警报，这里我本来想加入个声音警报，但是不知道为什么警报只能响一次，如果以后再犯错就不响了，还希望哪位高手指点我一下。如果没有违规，则会将要提取的盘子在原来的底座上清除，当再次按回车，也就是执行落下指令的时候，再次判断，是否要落下地方最上面的盘子比提取出来的盘子小，如果是，则把状态还原为落下。如果不是，则将落下的汉诺塔添加到新的底座上，然后执行draw 函数，画出状态。这样整个程序就做完了，挺高兴的，这是我做的第一个游戏，希望大家指点我一下。

适合读者：程序员，远程控制软件开发者
前置知识：C/C++基础

网络执法官 核心技术初探（上）

文/图 四川大学信息安全研究所 古开元

大家都知道，现在学校内的IP资源越来越有限，相信很多在校的读者都有过痛苦的抢IP与被抢IP的经历，如果你是黑防的忠实读者（什么？你不是？开门，放老毒，WTF……），也许你此刻正在偷笑，因为黑防很多期都介绍过一个抢IP暴强的工具——网络执法官。

网络执法官是一款国产的局域网管理辅助软件，采用网络底层协议，能穿透各种客户端的防火墙对网络中的每一台主机进行监控。它采用网卡号（MAC地址）识别用户，可靠性高；软件本身占用网络资源少，对网络没有什么不良影响。软件不需运行于指定的服务器，在网内任何一台主机上运行即可有效监控所有本机连接到的网络（支持多网段监控），功能十分强大。网络执法官（v2.92试用版）的主界面如图1所示。

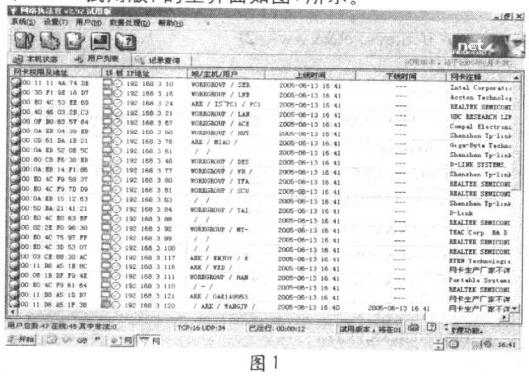


图1

不过，本文的目的不是介绍网络执法官的功能及用法，而是探索隐藏在该软件实现的核心技术，这才是黑客精神的本质嘛（呵呵，开玩笑，我可不敢自称黑客）。废话少说，让我们开始吧。

从图1的主界面，我们选择查看“本机状态”的视图，如图2所示。大家注意有框框住的那3个地方，分别是本机网卡参数，当前连接（TCP/UDP）状态和CPU使用率。今天我们就先暂时讨论并实现这3个技术吧。

以下程序均在Windows XP sp1 + VC7环境下编译通过！

核心技术一：获得本机网卡参数

细心的读者可能发现，网络执法官获得的网卡参数就跟我们在CMD命令下用“ipconfig /all”这条命令得到的结果是一样的，如图3所示。

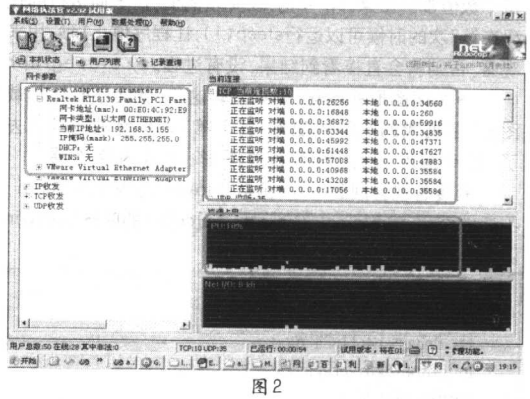


图2

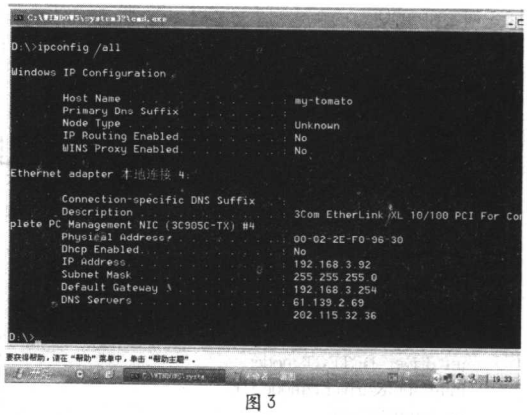


图3

哈哈，这下就好办了。搬出强大的IDA（The Interactive Disassembler）反汇编利器来，从系统目录system32里把ipconfig.exe找出来并加载到IDA去（有读者问，为什么不反汇编网络执法官呢？呵呵，网络执法官比ipconfig.exe大得多，包含的导入函数肯定暴多，本人后来也曾用IDA来反汇编网络执法官的主程序Robocop.exe，等我喝完茶，上完厕所，听了首歌，它才很辛苦地反编译完…），然后选择主菜单->Windows->Imports（或直接ALT+4），看到只有为数不多的几个函数，我们的目光一下子就可以定位在如图4所示的地方了。

全部调用的都是iphlpapi库里的API，就在我利用MSDN查看上面用框框包含的每一个API时，无意中找到了GetAdaptersInfo()这个API函数，该函数内部已经封装了

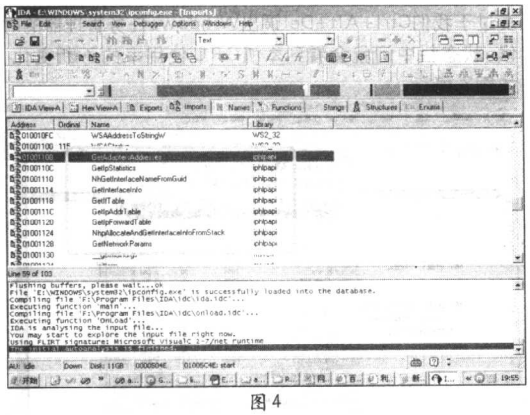


图4

ipconfig.exe可以实现的大部分功能函数。定义如下。

```
DWORD GetAdaptersInfo(
PIP_ADAPTER_INFO pAdapterInfo,
// 取得的网卡适配器信息结构
PULONG pOutBufLen  // 网卡适配器信息结构长度
);
```

这里注意几点:

1）使用该函数必须加上"#include <iphlpapi.h>"头文件和"#pragma comment(lib, "iphlpapi.lib")"语句。当然,你也可以用"LoadLibrary("iphlpapi.dll")"来显示加载库。

2）在真正使用GetAdaptersInfo()之前必须先作一次初始化,调用GetAdaptersInfo()取得指向该函数的第2个参数必要的大小值,如下:

```
    // 分配空间
    pAdapterInfo = (IP_ADAPTER_INFO *) malloc( sizeof
(IP_ADAPTER_INFO) );
    // 必须先初始化 IP_ADAPTER_INFO 结构大小
    ULONG ulOutBufLen = sizeof(IP_ADAPTER_INFO);
    // 取得指向 ulOutBufLen 的值
    if (GetAdaptersInfo( pAdapterInfo, &ulOutBufLen) ==
ERROR_BUFFER_OVERFLOW)
    {
        free(pAdapterInfo); // 释放掉原来分配的
        pAdapterInfo = (IP_ADAPTER_INFO *) malloc
(ulOutBufLen);  // 重新分配空间
    }
```

这样,我们感兴趣的东西都放在了pAdapterInfo所指向的结构里了,该结构的MSDN描述如下:

```
    typedef struct _IP_ADAPTER_INFO {
    struct _IP_ADAPTER_INFO* Next; // 下一块网卡信
息
    DWORD ComboIndex; // 保留
    Char AdapterName[MAX_ADAPTER_NAME_LENGTH
+ 4]; // 适配器名字
    char Description[MAX_ADAPTER_DESCR
IPTION_LENGTH + 4]; // 适配器描述
    UINT AddressLength; // MAC 地址长度
    BYTE Address[MAX_ADAPTER_ADDRESS_LENGTH];
// MAC 地址
    DWORD Index; // 网卡索引号
    UINT Type; // 网卡类型
    UINT DhcpEnabled; // 是否开启 DHCP
    PIP_ADDR_STRING CurrentIpAddress; // 当前 IP 地址
    IP_ADDR_STRING IpAddressList; // IP 地址列表
    IP_ADDR_STRING GatewayList; // 网关 IP 地址
    IP_ADDR_STRING DhcpServer; // DHCP 服务器 IP 地址
    BOOL HaveWins; // 是否使用 Windows Internet
Name Service (WINS). 服务
    IP_ADDR_STRING PrimaryWinsServer;
    // 主 WINS 服务 IP 地址
    IP_ADDR_STRING SecondaryWinsServer;
    // 次 WINS 服务器 IP 地址
    time_t LeaseObtained;
    time_t LeaseExpires;
    } IP_ADAPTER_INFO, *PIP_ADAPTER_INFO;
```

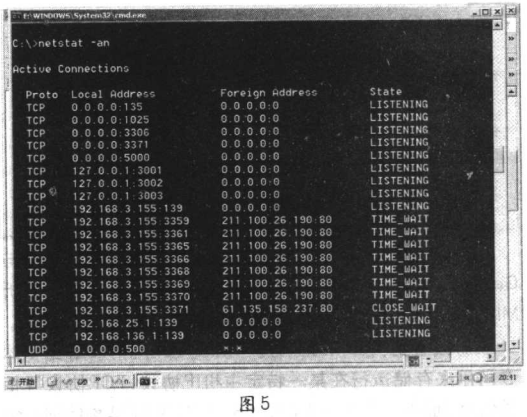
哇,看到可以获得这么多重要的信息,你是不是已经在狂流口水啦……

3）我们当然也可以用IDA里显示的那几个文档化了的API来实现,大家可以自己当做练习来实现。

具体的实现代码参看附件(MacInfo.cpp)。

核心技术二:显示TCP/UDP当前连接

看完上面技术一的实现,聪明的读者肯定想到了我们这一技术的实现可以如法炮制。要想成为高手不是靠一股冲动和蛮力就可以做到的,更多的要靠机智和变通,要学会举一反三。TCP/UDP的当前连接?让你想起了什么?对,netstat!我们在命令行下的输入"netstat_an"。得到截图。如图5所示。



```
C:\>netstat -an

Active Connections

  Proto  Local Address          Foreign Address        State
  TCP    0.0.0.0:135            0.0.0.0:0              LISTENING
  TCP    0.0.0.0:1025           0.0.0.0:0              LISTENING
  TCP    0.0.0.0:3306           0.0.0.0:0              LISTENING
  TCP    0.0.0.0:3371           0.0.0.0:0              LISTENING
  TCP    0.0.0.0:5000           0.0.0.0:0              LISTENING
  TCP    127.0.0.1:3001         0.0.0.0:0              LISTENING
  TCP    127.0.0.1:3002         0.0.0.0:0              LISTENING
  TCP    127.0.0.1:3003         0.0.0.0:0              LISTENING
  TCP    192.168.3.155:139      0.0.0.0:0              LISTENING
  TCP    192.168.3.155:3359     211.100.26.190:80      TIME_WAIT
  TCP    192.168.3.155:3361     211.100.26.190:80      TIME_WAIT
  TCP    192.168.3.155:3365     211.100.26.190:80      TIME_WAIT
  TCP    192.168.3.155:3366     211.100.26.190:80      TIME_WAIT
  TCP    192.168.3.155:3368     211.100.26.190:80      TIME_WAIT
  TCP    192.168.3.155:3369     211.100.26.190:80      TIME_WAIT
  TCP    192.168.3.155:3370     211.100.26.190:80      TIME_WAIT
  TCP    192.168.3.155:3371     61.135.158.237:80      CLOSE_WAIT
  TCP    192.168.25.1:139       0.0.0.0:0              LISTENING
  TCP    192.168.136.1:139      0.0.0.0:0              LISTENING
  UDP    0.0.0.0:500            *:*
```

图5

是不是和图2中网络执法官实现一样呢(读者甲:不一样啊,本地和远程的位置放反了!狂倒!)。重复上面的工作,用IDA加载系统目录system32里的netstat.exe,发现还是加载了iphlpapi库的API函数,如图6所示。

图6框中显示的分别是GetUdpStatsFromStackEx()、GetTcpStatsFromStackEx()、GetIpStatsFromStackEx()、GetIcmpStatsFromStackEx()、AllocateAndGetTcpExTableFromStack()和AllocateAndGetUdpExTableFromStack()这6个API函数。查了查MSDN,晕死,这些API全都没查到,换言之它们应

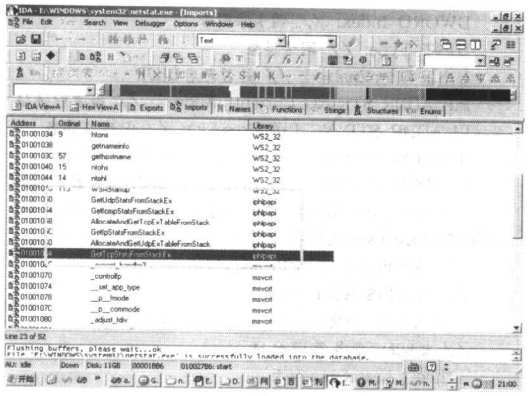


图6

该都是微软未文档化的函数(我猜的),那怎么办?不能就这样的给凉拌了。注意到MSDN中iphlpapi库给出的API函数有几个与上面列出的极为相似,比如:

```
DWORD GetIpStatistics(
//  获得当前计算机的IP统计数据
  PMIB_IPSTATS pStats
//  指向运行在计算机上的IP协议信息的存储结构
);
```

GetUdpStatistics(),GetTcpStatistics()和GetIcmptatistics()原型与GetIpStatistics()类同,其中重点就在于分析MIB_IPSTATS, MIB_TCPSTATS等结构体的成员上,从这些函数的名字就可以知道,它们的作用在于获取当前计算机的IP, TCP,UDP, ICMP等协议的统计数据。看下面这个函数:

```
DWORD GetTcpTable(          // 获得TCP连接表
  PMIB_TCPTABLE pTcpTable, // 指向MIB_TCPTABLE
结构的缓冲区
  PDWORD pdwSize,          // 指向pTcpTable所
值结构大小的指针
  BOOL bOrder     // 是否按 本地IP->本第端
口->远程IP->远程端口 排序
);
```

它获取本机TCP当前连接状态信息,GetUcpTable()与GetTcpTable()原型类同。同理,这里的重点则在于分析MIB_TCPTABLE和MIB_UDPTABLE这两个结构体的成员。

可以看到,MIB_IPSTATS, MIB_TCPSTATS等结构体专门用来存储运行在某一特定主机IP协议,TCP协议等相关信息,而MIB_TCPTABLE和MIB_UDPTABLE结构体则包含了TCP/UDP连接表的详细信息,由此可以看到通过获取这些关键结构体的相关数据成员,就可以获取TCP/UDP当前的连接信息。由于篇幅原因,加上这些结构体的成员暴多,这里就不一一列举,大家应该参阅MSDN获取更详尽的信息。

附件中提供了著名安全站点nologin提供的netstat.c源代码。相信读者们经过仔细分析阅读,会有很大的收获。

核心技术三:CPU使用率的实现

显示CPU使用率好像很多时候都要用到,最典型的莫过于我们Ctrl+Alt+Del调出Windows任务管理器的时候,状态栏中间就会看到本机CPU使用率的不断变化,如图7所示。

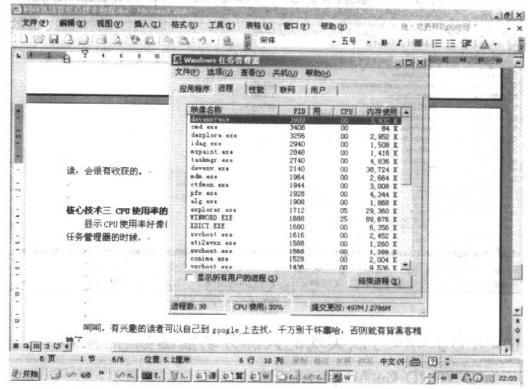


图7

网络执法官显示的CPU占用率是整个系统的占用率(与Windows任务管理器显示的是一样的),而非单指本软件的占用。该软件在刚启动时,占用CPU资源相当多,当扫描范围较大时更为明显。

笔者经过查询,从网上找到了实现该技术的一些关键资料,即获得CPU使用率的实现可以用到NtQuerySystemInformation()这个未文档化的函数。原型如下:

```
NTSTATUS NtQuerySystemInformation(
  SYSTEM_INFORMATION_CLASS
SystemInformationClass, // 系统信息类
  PVOID SystemInformation,  // 保存系统信息
  ULONG SystemInformationLength,
// 取得系统信息长度
  PULONG ReturnLength   // 返回的长度
);
```

我们首先要利用此函数通过SYSTEM_BASIC_INFORMATION结构获得CPU处理器的个数,NtQuerySystemInformation(SystemBasicInformation, &SysBaseInfo,sizeof(SysBaseInfo),NULL);然后,通过SYSTEM_TIME_INFORMATION结构获得新的系统时间,NtQuerySystemInformation(SystemTimeInformation, &SysTimeInfo,sizeof(SysTimeInfo),0);再通过SYSTEM_PERFORMANCE_INFORMATION结构获得CPU空闲时间,NtQuerySystemInformation(SystemPerformanceInformation,&SysPerfInfo,sizeof(SysPerfInfo),NULL)。根据下面的算法:

```
if (liOldIdleTime.QuadPart != 0) // 如果是第一次调
用,跳过
  {
  // 当前空闲时间值=新的空闲时间值-旧的空闲时间值
  dbIdleTime = Li2Double(SysPerfInfo.liIdleTime) -
Li2Double(liOldIdleTime);
  // 当前系统时间值=新的系统时间值-旧的系统时间值
  dbSystemTime = Li2Double(SysTimeInfo.liKeSystemTime)-
Li2Double(liOldSystemTime);
  // 当前CPU空闲时间=空闲时间÷系统时间
  dbIdleTime = dbIdleTime / dbSystemTime;
```