基于实时监测 WIN32 函数调用防范网络蠕虫攻击的新方法

文伟平 卿斯汉

(1.中国科学院，信息安全技术工程研究中心，软件研究所，北京 100080;2.中国科学院，软件研究所，北京 100080;
3.中国科学院，研究生院，北京 100039)

摘 要：网络蠕虫利用系统安全漏洞，控制目标系统和网络，构成对互联网络安全的最大威胁。本文从系统入侵防范的角度，提出一种基于实时监测 Win32 函数调用检测和防御网络蠕虫攻击的新方法，该方法通过对网络蠕虫攻击行为模式的分析，对网络蠕虫的防范机制作了新的探索，建立了网络蠕虫防御的原型系统，最后以“冲击波杀手”蠕虫为例对该系统进行了实例评估，测试数据表明该系统能够及时地发现网络蠕虫并阻止网络蠕虫的进一步扩散，是一种有效地防御网络蠕虫攻击的新方法。

关键字：网络蠕虫；实时监测；Win32 函数；攻击行为；“冲击波杀手”蠕虫

1 引言

随着互联网应用的深入，网络蠕虫对计算机系统安全和网络安全的威胁日益增加。特别是在网络环境下，多样化的传播途径和复杂的应用环境使网络蠕虫的发生频率增高、潜伏性变强、覆盖面更广，造成的损失也更大。与传统的主机病毒相比，网络蠕虫具有更强的繁殖能力和破坏能力。如表1所示。

表1 近年来网络蠕虫对社会造成的经济损失

蠕虫名称	持续时间	经济损失
莫里斯蠕虫	1988年	6000多台计算机停机，直接经济损失9600万美元。
美丽杀手	1999年3月	政府部门和一些大公司紧急关闭网络服务器，经济损失超过12亿美元
爱虫	2000年5月	众多用户电脑被感染，损失超过100亿美元以上
红色代码	2001年7月	网络瘫痪，直接经济损失超过26亿美元
求职信	2001年12月	大量邮件阻塞服务器，损失达数百亿美元
SQL蠕虫王	2003年1月	网络大面积瘫痪，银行自动提款机中断，直接经济损失超过26亿美元
冲击波蠕虫	2003年8月	8天内导致全球电脑用户损失高达20亿美元之多

从网络安全的角度检测网络蠕虫攻击目前主要有基于网络流量异常分析的检测方法。邹长春等在文献[1]中，通过对一定地址空间的流量监控来预测网络蠕虫的传播，从而采取更有效的措施来对抗网络蠕虫的大规模攻击。在文献[2]中，Cheung 等提出了基于行为图的检测方法，基于图的入侵检测系统（GrIDS）建立节点间的行为图（Activity Graph），通过它与预定义的行为模式图进行匹配，检测网络蠕虫是否存在，是当前防御分布式网络蠕虫入侵最有效的工具。Spitzner 引入了 HoneyPot 的思想[3][4]，HoneyPot 虚拟大量的 IP 地址，然后检测发往这些地址的数据包来判断是否有网络蠕虫攻击，HoneyPot 同时能够阻断网络蠕虫的攻击。由 Liston 所设计的 LaBrea 工具[5]，能够通过长时间阻断与被感染机器的 TCP 连接来降低蠕虫的传播速度。

从系统安全的角度预防网络蠕虫的攻击人们也做了大量的研究工作。Somayaji和Forrest的防御思想是首先建立一个正常系统调用序列库，然后将其他的调用序列模式与它们进行比较，如果不匹配就认为可能受到恶意代码的攻击[6]。Willianmson提出的“Virus Throttle”方法[7]，它通过检测一台主机是否大量地向其他地址发送SYN包，从而判断该主机是否已受到恶意代码的攻击。目前这两种方法检测网络蠕虫攻击的误警率较高。此外，传统的单机防病毒技术为对抗恶意代码攻击做了大量的贡献，但它依赖于一定的检测规则，不太适应网络蠕虫的检测和防御。

本文提出一种基于系统调用检测和防御网络蠕虫攻击的新技术，克服了以上单机防范的缺陷，通过截获网络蠕虫对 Win32 函数的异常调用来检测系统是否受到攻击，及时中止网络蠕虫的攻击行为并做进一步日志记录。这种方法在 Win32 函数被调用的时候就能及时地发现网络蠕虫攻击并中断恶意代码的执行，从而更好

的保护操作系统和应用软件的安全。

本文的组织如下：第 2 节对网络蠕虫的检测防御机制进行原理性探索；第 3 节阐述 Win32 函数截获和防范网络蠕虫攻击(CheckDLLCall)的基本策略实现；第 4 节以"冲击波杀手"蠕虫为例，测试和评估 CheckDLLCall 原型系统的效果；最后对本文的工作进行总结，并讨论需要进一步研究的内容。

2 网络蠕虫检测和防御机制的原理性探索

网络蠕虫一般分为两类：①主动利用系统漏洞进行攻击，可以对整个互联网可造成瘫痪性后果的网络蠕虫。以"红色代码"，"尼姆达"，以及最新的"冲击波蠕虫"为代表。②针对个人用户，通过网络(主要是电子邮件，恶意网页形式)迅速传播的网络蠕虫，以爱虫，求职信蠕虫为例。在这两类网络蠕虫中，第一类具有很大的主动攻击性和突发性。本论文中，主要研究对第一类网络蠕虫的防范机理，后面文中提到的网络蠕虫均指第一类网络蠕虫。

缓冲区溢出是目前导致第一类网络蠕虫迅速扩散和传播的主要原因。从红色代码到 Slammer 蠕虫，再到 2003 年 8 月爆发的"冲击波蠕虫"，都是利用系统缓冲区溢出漏洞进行攻击和传播的典型。网络蠕虫在 Internet 上传播的过程一般为：①扫描。由蠕虫的扫描功能模块负责探测存在漏洞的主机。当程序向某个主机发送探测漏洞的信息并收到成功的反馈信息后，就得到一个可传播的对象。②攻击。攻击模块根据①扫描到的对象，通过缓冲区溢出获得一个 shell，取得该主机的权限(一般为管理员权限)。③复制。通过原主机和新主机的交互将蠕虫程序复制到新主机并启动。表 2 列举了 CERT 网站发布的目前利用缓冲区溢出攻击迅速扩散和传播的网络蠕虫[8]。

表 2 利用缓冲区溢出攻击扩散和传播自身的网络蠕虫

蠕虫	探测（端口）	攻击渗透	自我复制（端口）	发布文档
Nimda	80，139，600	IIS，Code Red II 和 Sadmind 后门	有（80，139，600），Email 和网络共享	CA-2001-06
Code Red I	80	IIS 4.0/5.0 Index Service	有（80）	CA-2001-13
Code Red II	80	IIS 4.0/5.0 Index Service	有（80）	CA-2001-13, IN-2001-09
Adore	23，53，111，515	Bind,LPRng,Rpc.statd ,wu-ftpd	有（23，53，111，515）	CA-2001-02,IN-2001-01
Sadmind/IIS	80，111	IIS, Solstice, Sadmind	有（80，111） 80：Windows 平台 111：Unix 平台	CA-2001-11,MS 00-078
Lion	53	BIND	有（53）	CA-2001-02
Ramen	21,111，515	wu-ftp,rpc.statd,LPRng	有（21,111，515） tar 压缩包：ramen.tgz	IN-2001-01
Digispid.B	1433	Microsoft SQL Server	有（1433）	IN-2002-04
Slapper	80，443	OpenSSL 和 Apache	有（80）	CA-2002-27
MSSQL Worm	1433	Microsoft SQL Server	有（1433）	CA-2003-04
W32/Blaster worm	135	Microsoft Rpc Dcom	有（4444，6666）	CA-2003-20

注：在表 1 中，CA（CERT Advisory）和 IN (CERT Incident Note)是由 CERT 发布的警告信息。

蠕虫扫描发送的探测包是根据不同的系统漏洞进行设计的，针对 WEB 服务器的漏洞就需要发送经过特殊构造的 HTTP 请求，针对一般远程缓冲区溢出漏洞可以发送溢出代码来探测。蠕虫攻击发送的数据包一般包含可以导致远程主机发生缓冲区溢出的溢出字符串。溢出字符串被远程主机接受后，通常由宿主程序分配在堆栈的缓冲区内。为了实现攻击的目的，溢出字符串中的可执行代码（ShellCode）必须调用一部分系统核心动态连接库（Dynamic Linking Library,简称 DLL）提供的输出函数(API 函数)，调用 API 函数必需知道待调

用函数的入口地址，这个地址也会随着 DLL 的不同和 DLL 文件加载顺序的不同而改变。为提高 ShellCode 的通用性，通常使用输入表来调用 LoadLibraryA()、GetProAddress（）和 ExitProcess()函数，而其它所需的函数地址则通过调用 LoadLibraryA（）和 GetProAddress()来得到[9]。由于 ShellCode 存放在堆栈中，对这些常用函数的调用都是从堆栈分配的缓冲区发出的。如果能够及时截获并中断网络蠕虫对这些函数的异常调用，成功地阻断了网络蠕虫对目标系统的缓冲区溢出攻击，那对于被攻击系统来说，尽管存在可以被网络蠕虫利用的系统漏洞，但它对于网络蠕虫仍然是免疫的。这就是本文实现对网络蠕虫攻击检测和防御的机理。

3 Win32 函数截获和防范网络蠕虫攻击的基本策略实现

CheckDLLCall 的实现主要通过两步来实现对网络蠕虫攻击的防范：①截获网络蠕虫对 Win32 函数的功能调用。CheckDLLCall 中集成了很多常被网络蠕虫调用的 Win32 函数，每当这些函数被调用时，初始化函数就会立即被执行。②安全检查。初始化函数检查函数调用者是否来自正常代码，如果调用来自正常代码，就跳回到原函数的入口处，如果调用来自网络蠕虫或其他的恶意代码，初始化函数就会终止程序的执行并进行日志记录。

3.1 监视函数功能调用

网络蠕虫需要获得远程 Shell（Windows 系列操作系统主要是运行一个 System 的 CMD）、打开一个端口（开启一定的服务）、修改系统重要的配置文件（留下后门）或者通过另一个攻击来复制或者传播自身（例如：冲击波、SQL slammer、Nimda 和 CodeRed 等蠕虫），它们需要调用 Win32 函数来实现它们的攻击目的。表 3 中列出了一部分常被网络蠕虫调用的 Win32 函数。

表 3. 网络蠕虫常调用的 Win32 函数

函数名	动态连接库	函数说明
LoadLibraryA	Kernel32.dll	获得DLL的句柄
GetProcAddress	Kernel32.dll	获得DLL中函数的入口地址
CreateProcessA	Kernel32.dll	创建另外的一个进程
CloseSocket	WS2_32.dll	关闭一个Socket连接
WSAStartup	WS2_32.dll	初始化Winsock DLL
ExitProcess	Kernel32.dll	中止一个进程
ExitThread	Kernel32.dll	中止一个线程
CloseHandle	Kernel32.dll	关闭一个内核对象
CreatePipe	Kernel32.dll	创建一个匿名管道
RegCreateKeyExA	Advapi32.dll	在指定项下创建新项（或键）
RegCloseKey	Advapi32.dll	关闭系统注册表中的一个项（或键）
RegSetValueExA	Advapi32.dll	设置指定项的值

CheckDLLCall 集成包括表 3 列举的网络蠕虫可能调用的 Win32 函数，这些函数集合能够轻易的进行扩展，而扩展后的集合不用重新加载 CheckDLLCall 就能够应用， CheckDLLCall 是动态监视系统函数调用的。

从表 3 可以看出，被监视的函数大部分来自 Kernel32.dll、Ws2-32.dll、Advapi32.dll 和其它一些动态连接库。一种思路是把这些函数调用地址和相应动态链接库的接口参数封装到另外的库中，从封装库中调用原来的动态连接库，完成相应函数的调用。BOWALL 就是使用这种方法来监视库函数的[10]。但这种思路实现起来有两个问题：①Windows 2000 不允许对象 Kernel32.dll 等这样的系统动态连接库的任何修改；②封装动态连接库在系统更新或者应用软件重装后都要重新装载。

Microsoft Detours[11]是常用的监视 Win32 函数的二进制库。它是一个合适的监视 Win32 函数调用的工具，只监视指定的函数并且它并不修改磁盘上的映像而是修改内存中加载的数据，所以它能够在运行的时候动态监视。它用一个指向用户提供的初始化函数的跳转指令替代目标函数的开始几条指令，目标函数开始的这几个指令保存在初始化函数中，这个初始化函数中不仅包含着从目标函数中转移的指令，而且包含一个指

向目标函数剩余部分的无条件跳转指令，如图1所示。

```
; 目标函数
            ...
TargetFunction:
            push ebp
            mov ebp,esp
            push ebx
            push esi
TargetFunction+5:
            push edi
            ...
            Ret
            ...
```

```
; 初始化函数
TargetFunction:
            jmp DetourFunction
TargetFunction+5:
            push edi
            ...
            ret
DetourFunction:
            ...
            push ebp
            mov ebp,esp
            push ebx
            push esi
            jmp TargetFunction+5
```

图1 目标函数与初始化函数的指令结构

我们将安全检查代码放到 DetourFunction()中来检测函数是否是被恶意代码所调用。当通过调用 TargetFunction ()和 DetourFunction()将我们想要监视的函数注册后，每当这些目标函数被调用时安全检查代码就会自动执行。CheckDLLCall 是作为一个动态链接库来实现的，每当可执行性文件将 DllMain()捆绑在自身之上时，监视就开始起作用了。这种监视在整个系统范围内都是有效的，这就意味着这个动态链接库应该捆绑到每个易受到溢出攻击的应用程序之上，一种实现方式是指定注册值 AppInit_DLLs，它能在注册键：[HKLM\Software\Microsoft\Windows NT\CurrentVersion\Windows]中找到。当那些调用 Kernel32.dll 中函数的应用程序启动时，这个键值指定的动态链接库同时被装载。由于很多系统程序和应用程序都调用 Kernel32.dll 的函数，所以对 Win32 函数实现系统范围内动态监视是非常容易的。

3.2 安全检测

在 DetourFunction()中的安全检验代码决定了一个调用是否来自网络蠕虫。其它监视系统关心的可能是函数的参数而 CheckDLLCall 并不如此，CheckDLLCall 检验一个 Win32 函数调用者是否来自正常的地址空间，而不是确认一个函数的调用是否合法。

堆栈注入技术将堆栈中的函数返回地址覆盖掉，劫持特权用户执行权限给已注入堆栈中的代码。基于堆的缓冲区溢出攻击采取相同的步骤把非法代码注入到堆空间中。所以如果一个函数调用来自堆栈或堆，那么就可以认为该调用是可疑的，然后中止它并做好日志记录。剩下的问题是如何鉴别调用者是否处在有效的代码段。在函数调用的过程中，只有返回地址被压入堆栈，代码段的值并不被压入堆栈。在 Win32 的操作系统中，缓冲区溢出攻击并不改变寄存器 CS 的值，即使这段代码存放在堆或者堆栈区域，寄存器 CS，DS，SS 的值总是相同的，所以在特权程序被网络蠕虫劫持后寄存器 CS 的值仍然保持不变。既然不能依赖寄存器 CS 来鉴别调用者，那么唯一可用的信息就是返回地址了。

溢出攻击将它们的代码存放在堆或者堆栈区域中，返回地址也被定位到堆或者堆栈区域。所以能够简单地把返回地址从内存中读取并回写过去。如果调用者来自代码页，这个写操作将导致异常，因为代码页是只读属性的。而如果调用者来自堆栈或堆，这个写操作就会成功完成，因为这些页可以被写入数据而无异常产生。这样就找到了一种非常直接的检测的方式，如果写操作导致异常，那么这个返回地址就位于代码页中，相应的调用者是一个正常的应用，否则，就是一个非法攻击，检测代码如图2所示。

```
        push    offset ExceptionHandler
        push    dword ptr fs:[0]
        mov     fs:[0], esp ;建立异常链
        mov     ax, RetAddr
        mov     RetAddr, ax ; 检测返回地址是否代码页
; 如果没有发生写异常，则说明返回地址处在堆栈或堆区域，可能是攻击
; 代码对API函数进行非法调用，做好日志记录，并中止程序运行
    ExceptionHandler:
        mov     eax, [esp+8] ;如果代码页发生异常则执行
        mov     esp, eax     ; 恢复堆栈和异常链
        pop     dword ptr fs:[0];
```

图 2 对返回地址的有效性检测及异常处理

由于该写异常是由 CheckDLLCall 自身引起的，它应该自行掩饰掉这个异常。为了捕捉这个异常，CheckDLLCall 在写异常产生之前建立了一个结构化的异常处理器(SEH)，并在异常产生后将其清除。Windows 中的 SEH 在文献[12]中有所描述。

3.3 中止程序并对攻击做日志记录

CheckDLLCall 通过截获 Win32 函数调用检测到网络蠕虫攻击，就会立即中止该宿主程序的执行从而中断网络蠕虫的攻击行为，然后它会记录该事件及一些详细信息，包括拦截时间、应用程序名、堆栈中相关数据，返回 EIP、ESP 和错误码等，这样管理员和软件开发商能够根据这些相关信息进行安全配置或开发出相应的系统补丁。

4 相关测试数据及试验结果

4.1 相关的测试环境

我们以冲击波杀手蠕虫（Worm.KillMSBlast）为例[13]，设计了一个测试环境验证和评估 CheckDLLCall 系统检测和防御网络蠕虫的有效性。采用数据嗅探工具实时监测 Worm.KillMSBlast 蠕虫的网络行为；CheckDLLCall 系统实时监测 Worm.KillMSBlast 蠕虫对被攻击主机的缓冲区溢出攻击行为。测试环境拓扑如图 3 所示。

在测试环境中，启动数据嗅探工具和所有参与测试的节点主机（A，B，C，D）。这些测试节点的操作系统均为Windows 2000 Advance Server，经瑞星防病毒软件（版本：16.03.20）扫描，没有发现任何病毒，补丁情况见图3所示，节点A（192.168.0.1）为Worm.KillMSBlast的释放主机；节点D（192.168.1.3）装有CheckDLLCall系统，如图3所示。

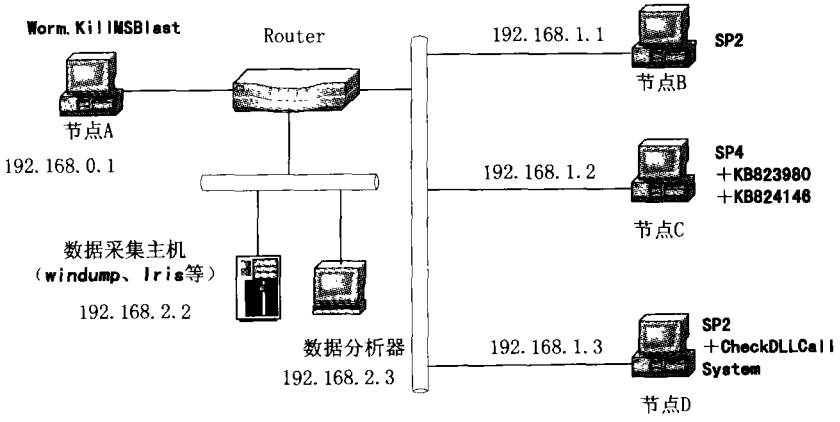

图3 测试环境拓扑图

4.2 冲击波蠕虫（Worm.KillMSBlast）调用的相关函数

Worm.KillMSBlast 对系统函数调用主要有两类：①蠕虫程序体对API函数的调用；②蠕虫ShellCode数据

发送到被攻击主机后，ShellCode被执行时产生的API函数调用。具体如表4所示：

表4. 冲击波蠕虫（Worm.KillMSBlast）调用的相关函数

函数名	动态连接库	函数说明	调用位置
LoadLibraryA	Kernel32.dll	获得DLL的句柄	程序体，ShellCode
GetProcAddress	Kernel32.dll	获得DLL中函数的入口地址	程序体，ShellCode
CreateProcessA	Kernel32.dll	创建另外的一个进程	ShellCode
WSAStartup	Ws2_32.dll	初始化Winsock DLL	ShellCode
CloseSocket	Ws2_32.dll	关闭一个Socket连接	ShellCode
ExitProcess	Kernel32.dll	中止一个进程	程序体，ShellCode
RegCloseKey	Advapi32.dll	关闭系统注册表中的一个项（或键）	程序体
IcmpSendEcho	Icmp.dll	发送数据包	程序体
Exit	Msvcrt.dll	结束程序运行	程序体
URLDownloadToFileA	Urlmon.dll	从其他web站点下载文件	程序体
ExitWindowsEx	User32.dll	退出Windows 操作系统	程序体

CheckDLLCall系统主要监视对Kernel32.dll中LoadLibraryA()和GetProcAddress()的函数调用。

4.3 测试结果

在节点A释放Worm.KillMSBlast,蠕虫首先检测本地网络是否与Internet 连通，如果连通，则到网址(http://download.microsoft.com/download/2/8/1/281c0df6-772b-42b0-9125-6858b759e977/Windows2000-KB823980-x86-CHS.exe)下载对抗冲击波蠕虫的补丁并开始自身传播；否则，程序驻留内存，不开始新的扫描行为。从图4能够看出，节点A的Worm.KillMSBlast对192.168.0.0-192.168.254.254 开始扫描，从图5的数据能够看出，节点B与节点A有数据交换行为，节点B存在Rpc-dcom漏洞，节点B被感染，从图6进一步证明节点B被感染并开始自身传播，节点C由于已安装Rpc-dcom漏洞的补丁，没有被节点A或节点B感染，节点D装有CheckDLLCall系统，从网络行为来看，和节点B一样，与节点A有数据交换行为，但没有象节点B一样被感染并开始自身传播，图7显示的CheckDLLCall系统的日志记录，证明节点D存在 Rpc-dcom漏洞且已受到网络蠕虫攻击，但攻击被CheckDLLCall系统拦截而中止了蠕虫的攻击行为。

```
……
14:35:57.960611 IP 192.168.0.1 > 192.168.0.206: icmp 72: echo request seq 466
14:35:57.970276 IP 192.168.0.1 > 192.168.0.207: icmp 72: echo request seq 722
14:35:57.980176 IP 192.168.0.1 > 192.168.0.208: icmp 72: echo request seq 978
……
```

图 4 节点A冲击波杀手蠕虫的扫描行为

```
14:39:06.446007 IP 192.168.0.1 > 192.168.1.1: icmp 72: echo request seq 3573
14:39:06.447313 IP 192.168.0.1.4839 > 192.168.1.1.135: S 1246699886:1246699886(0) win 16384
                                    <mss 1460,nop,nop,sackOK> (DF)
14:39:06.448072 IP 192.168.0.1.4839 > 192.168.1.1.135: . ack 1908695868 win 17520 (DF)
14:39:06.448373 IP 192.168.0.1.4839 > 192.168.1.1.135: P 0:72(72) ack 1 win 17520 (DF)
14:39:06.452787 IP 192.168.0.1.4839 > 192.168.1.1.135: . 72:1532(1460) ack 61 win 17460 (DF)
14:39:06.452896 IP 192.168.0.1.4839 > 192.168.1.1.135: P 1532:1776(244) ack 61 win 17460 (DF)
14:39:06.469713 IP 192.168.0.1.707 > 192.168.1.1.1122: S 1246762444:1246762444(0) ack
                                    1908755484 win 17520 <mss 1460,nop,nop,sackOK> (DF)
14:39:06.474601 IP 192.168.0.1.4839 > 192.168.1.1.135: . ack 62 win 17460 (DF)
14:39:06.474749 IP 192.168.0.1.4839 > 192.168.1.1.135: F 1776:1776(0) ack 62 win 17460 (DF)
14:39:06.646060 IP 192.168.0.1.707 > 192.168.1.1.1122: . ack 43 win 17478 (DF)
14:39:06.647159 IP 192.168.0.1.707 > 192.168.1.1.1122: P 1:23(22) ack 105 win 17416 (DF)
14:39:06.846259 IP 192.168.0.1.707 > 192.168.1.1.1122: . ack 126 win 17395 (DF)
14:39:06.847740 IP 192.168.0.1.707 > 192.168.1.1.1122: F 23:23(0) ack 381 win 17140 (DF)
14:39:06.848983 IP 192.168.0.1.707 > 192.168.1.1.1122: R 1246762468:1246762468(0) win 0 (DF)
```

图 5 节点A和节点B冲击波杀手蠕虫的数据交换行为

```
14:40:00.543978 IP 192.168.1.1 > 192.168.0.1: icmp 72: echo request seq 211
14:40:00.553860 IP 192.168.1.1 > 192.168.0.2: icmp 72: echo request seq 467
14:40:00.563971 IP 192.168.1.1 > 192.168.0.3: icmp 72: echo request seq 723
14:40:00.574027 IP 192.168.1.1 > 192.168.0.4: icmp 72: echo request seq 979
                    ……
```

图 6 节点B冲击波杀手蠕虫的扫描行为

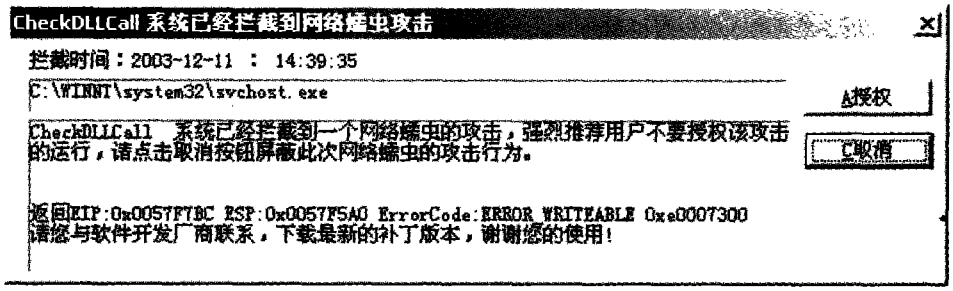

图7 CheckDLLCall 系统拦截网络蠕虫攻击的日志记录

测试证明，CheckDLLCall系统能很好地拦截Worm. KillMSBlast蠕虫攻击，由于CheckDLLCall系统在方法上主要与Win32函数调用相关，与攻击者采用何种攻击模式无关，所以该系统对其他网络蠕虫的攻击也是有效的。

5 结论及未来研究方向

本文提出一种基于系统函数调用检测和防御网络蠕虫攻击的新方法，克服了传统单机防范的缺陷，通过截获网络蠕虫对 Win32 函数的异常调用来检测网络蠕虫攻击，及时中止蠕虫的攻击行为并做进一步日志记录。这种方法在 Win32 函数被调用的时候就能及时地发现网络蠕虫攻击并阻止网络蠕虫的进一步扩散。该方法具有三大特点：①能检测未知网络蠕虫的攻击，该方法主要与 Win32 函数调用相关，与蠕虫采用何种攻击模式无关；②不需要作任何源代码的安全性分析；③原型系统是作为一个动态链接库动态生成的，封装的代码量小，监测函数调用时占用资源少，基本不影响系统的工作性能。

网络蠕虫的检测与防御是一个长期的过程，这主要因为：①网络蠕虫的种类繁多，形态千变万化；②网络蠕虫的入侵、感染、发作机制千差万别；③不能准确地预见新产生的网络蠕虫。所以，我们既要掌握当前网络蠕虫的实现机理，还要加强对未来网络蠕虫发展趋势的研究，真正作到防患于未然。

参考文献：

[1] C Zou, L Gao, W Gong , D Towsley. Monitoring and Early Warning for Internet Worms. Umass ECE Technical Report TR-CSE-03-01,2003.

[2] S Cheung, J Hoagland, K Levitt, J Rowe, C S Staniford, R Yip, D Zerkle. The Design of GrIDS: A Graph-Based Intrusion Detection System. Technical Report CSE-99-2. U.C. Davis Computer Science Department. http://citeseer.nj.nec.com/cheung99design.html. 1999.

[3] L Spitzner, Strategies and Issues:Honeypots-Sticking It to Hackers

[4] Honeypot技术讲解.安全焦点网站.http://www.xfocus.net/articles/200103/121.html.

[5] T Liston, Welcome to My Tarpit – The tactical and Strategic Use of LaBrea. http://www.hack.buster.net.

[6] A Somayaji, S Forrest. Automated Respose Using System – Call Delays, Proceedings of 9th Usenix Security Symposium, Denver, Colorado 2000.

[7] M M Williamson. Throttling Virus:Restricting propagation to defeat malicious mobile code. http://www.hpl.hp.com/techreports/2002/HPL-2002-172.pdf.

[8] CERT Security Website. http://www.cert.org.

[9] 通用ShellCode深入剖析.安全焦点网站.http://www.xfocus.net/articles/200401/655.html.

[10] A Kolishak. Buffer overflow protection for NT 4.0 binary – BOWall. http://www.security.nnov.ru/bo/eng/BOWall/.

[11] G Hunt, D Brubacher. Detours: Binary Interception of Win32 Functions. In Proceedings of the 3rd USENIX Windows NT Symposium.(1999).

[12] P Matt. A Crash Course on the Depths of Win32 Structured Exception Handling. Microsoft Systems Journal, 1997.1

[13] KillMSBlast病毒及其处理方法. 金山毒霸安全资讯网. http://www.duba.net/c/2003/08/21/90290.shtml