

基于文件过滤驱动的文件型病毒防范方法的实现

陈建熊¹, 揭 摄¹, 张 鑫²

(1. 合肥电子工程学院 网络系, 安徽 合肥 230037;
2. 武警合肥指挥学院 院务部, 安徽 合肥 230031)

摘 要:计算机病毒给当今社会造成了较大危害, 为了有效防止病毒蔓延, 针对当前防病毒软件存在的不足, 给出了一种基于文件过滤驱动的文件型病毒防护方法。分析了文件过滤驱动原理, 通过在文件系统驱动上层附加一层过滤驱动, 来截获上层应用对文件的各种操作, 阻止可疑应用程序对可执行文件的写操作, 达到主动防御文件型病毒的目的。文章从病毒对文件的打开、写入、关闭以及用户层和驱动通信等方面给出了该方法的实现原理。实验证明该方法能够有效对文件型病毒进行防护。

关键词:文件过滤驱动; 计算机病毒; 主动防御

中图分类号: TP309

文献标识码: A

文章编号: 1673-629X(2013)03-0143-04

doi: 10.3969/j.issn.1673-629X.2013.03.036

Implementation of Virus Prevention Method Based on File System Filter Driver

CHEN Jian-xiong¹, JIE She¹, ZHANG Xin²

(1. Dep. of Network, Electronic Engineering Institute of Hefei, Hefei 230037, China;
2. Dep. of Logistics, Hefei Command College of Armed Police, Hefei 230031, China)

Abstract: Computer virus is harmful to society. In view of the defect of the virus detection method existing, a new method of virus prevention based on file system filter driver is brought forward to prevent computer virus propagation. Analyze the theory of file system filter driver. With attaching a filter driver to the file system driver to obtain upper application operating on the file of the disk, can prevent suspicious application program to write on executable file. Thus implement the aim of active defense of the computer virus. Present the detail implementation of this method with opening, writing, closing the executable file by virus and commutation between application and driver. The result indicates that this method can prevent the computer virus spreading efficiently.

Key words: file system filter driver; computer virus; active defense

0 引 言

文件型病毒是指感染可执行文件的一类病毒, 在 Windows 系统下, 文件型病毒主要是指对 PE 可执行文件^[1]进行感染的病毒类型, 这类病毒种类比较多, 危害比较大, 技术性较强, 比较典型的文件型病毒有 CIH、Funlove、莫国防、熊猫烧香等。

当前针对 PE 文件型病毒的检测^[2]方法主要有以下几种: 特征码查毒、行为监控、启发式查毒、虚拟机查毒等^[3], 但是这些方法都是在病毒出现之后的一种事后补救措施。近年来, 对未知恶意代码的检测和分析

变得极为重要^[4]。于是就出现了病毒主动防御技术^[5], 主动防御技术是能够在病毒将要出现恶意行为的时刻所采取的防御技术, 当前主动防御主要是通过截获特定系统调用函数, 分析调用序列, 从而发现恶意行为, 但是对系统调用的下层操作无法做到有效截获。文中提出了一种基于文件过滤驱动的文件型病毒主动防御方法。该方法在系统调用函数层以下工作, 实现更加底层, 能够更加有效地截获并发现病毒程序对可执行文件的恶意行为。

1 文件过滤驱动基本框架

1.1 文件过滤驱动简介

在 Windows 操作系统中, 文件系统驱动程序 (FSD, File System Driver) 是用来管理磁盘上的文件的, 它负责存储本地的数据, 接收本地磁盘文件的打

收稿日期: 2012-07-03; 修回日期: 2012-10-08

基金项目: 国家自然科学基金资助项目 (10990011); 军内科研计划项目 (KY08004)

作者简介: 陈建熊 (1980-), 男, 博士, 讲师, 研究方向为网络与信息安全。

开、读、写、关闭等请求,通常这些请求来自用户进程,通过 IO 管理器发送到文件系统。

文件过滤型驱动程序 (FSFD, File System Filter Driver)^[6,7] 是叠加在文件系统驱动程序上的过滤型驱动程序,FSFD 是针对 FSD 而言的。它运行于操作系统的内核模式,在系统内核驱动中加入新的层,在不影响上层和下层接口的情况下,能够截获到所有的文件系统请求,从而不需要修改上层的软件或下层的驱动程序,就可加入新的功能。文件系统过滤驱动的目标是捕获系统对文件的种种操作行为,比如文件的创建、打开、读写、改名,目录的创建、打开、枚举、改名、删除等。并且可以选择完成或者修改这些请求。正是由于这种能力,致使其应用在文件保护^[8,9]、文件透明加密^[10]、移动存储控制^[11]、病毒防治、远程文件复制等多方面。

文件过滤驱动在 Windows 系统中所处的位置如图 1 所示,从图中可以看出文件过滤驱动负责接收上层的 IRP^[12] (I/O Request Packet) 请求,处理后传递给下层设备对象。

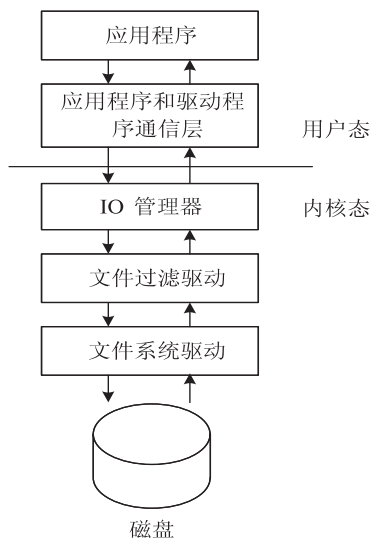


图 1 文件过滤驱动的位置

1.2 文件过滤驱动加载的基本原理

文件过滤驱动通过在各个系统卷设备对象上附加一层过滤设备对象,来截获对卷设备对象的操作请求,从而截获对文件各种操作,完成用户期望的目的。

为了在已有的系统卷上附加设备对象,并且能够在有新的卷发现时(比如插入移动存储介质),仍然能够附加过滤驱动,驱动程序加载的基本原理如下:

(1) 使用 IoRegisterFsRegistrationChange 系统注册一个回调函数,当系统中有文件系统被激活或者撤销时,该回调函数就被调用。在回调函数中去绑定文件系统的控制设备对象。

(2) 当有新的卷加入时,文件系统的控制设备对象会收到主功能码为 IRP_MJ_FILE_SYSTEM_CON-

TROL 和副功能码为 IRP_MN_MOUNT 的卷挂载 IRP 请求,因为在第一步已经在文件系统的控制设备对象上已完成监控了,所以这两个请求是能够被截获的。当截获到该 IRP 请求后就虚拟一个设备对象,将该设备对象挂载在新的卷设备对象上,完成卷的动态挂载。

(3) 通过系统调用 IoEnumerateDeviceObjectList 枚举当前系统已有的卷设备对象,同时新建一个过滤驱动并加载枚举到的卷设备对象上层。也可以通过卷设备名称来获取卷设备对象,从而来绑定特定的卷设备。

通过以上方法就完成了过滤驱动的加载过程,加载完成后,所有对文件的访问都会被截获(包括新的移动介质加入的情况),当截获后就可以根据需要对文件的访问做控制操作。在本方法中正是在这一层次上完成病毒防范目的的。

2 病毒检测基本原理及实现

目前较为流行的各种杀毒软件,主要是检测病毒的特征行为,而对于病毒的传播却重视不足。如果能够在病毒的传播时期就能将其拒之于系统之外,那么系统的安全性和稳定性将大为提高。

文件型病毒感染一个可执行文件,其本质就是对可执行文件进行写操作,然而合法程序一般情况下是不会对可执行文件进行写操作的,因此如果能够监控到对可执行文件的写操作就很值得怀疑是否是病毒的行为。文中所提出的方法是通过文件过滤驱动在操作系统内核级检测该行为,并加以阻止。但是在个别情况下,合法程序也会对可执行文件进行写操作,为了进行区分,在本方法中有白名单策略,其中保存着合法进程的名称。为了能够将白名单下发给底层驱动程序,还设计有用户程序和驱动程序的通信功能。

对一个文件的访问被操作系统抽象为文件的打开、读、写、关闭以及文件控制等,当过滤驱动加载到卷设备对象上层的时候,应用程序包括病毒对于文件的操作都将被过滤驱动截获。在底层过滤驱动中相应的也有文件的建立、文件读写以及文件关闭的各种分发例程,这些分发例程分别对应的 IRP 为 IRP_MJ_CREATE、IRP_MJ_READ、IRP_MJ_WRITE、IRP_MJ_CLOSE,为了截获对文件的这些操作,在文件过滤驱动中需要分别对这些 IRP 所对应的分发例程加以实现。

其方法是在过滤驱动中分别实现如下例程:

```
DriverObject->MajorFunction[IRP_MJ_CREATE] = SfCreate;
```

```
DriverObject->MajorFunction[IRP_MJ_CLOSE] = SfCleanupClose;
```

```
DriverObject->MajorFunction[IRP_MJ_WRITE] = SfWrite;
```

```
DriverObject->MajorFunction[IRP_MJ_READ] = SfRead;
```

```
DriverObject->MajorFunction[IRP_MJ_DEVICE_CONTROL]
=SfDeviceIoControl
```

当上层应用程序包括病毒对文件进行读写操作的时候就会向下层驱动发出读写请求,相应的就会调用如上代码中相应的读写例程。

其中在 SfCreate 分发例程中需要监控对各种可执行文件的打开过程,对于打开文件的 IRP 的处理是系统设计的关键,其它 IRP 处理过程也依赖于这种 IRP 的处理结果。在该过程中需要监控的信息有文件名称以及文件对象。这些信息可在当前 IRP 堆栈中的 FileObject 域中得到,通过监测文件的扩展名是不是可执行文件,如果是则将该文件对象插入到全局链表中,以便在文件其它操作中进行对比识别。加入链表记录后需要将该 IRP 直接下发给下层文件系统驱动程序进行处理。对于其它类型的文件建立操作则直接下发给下层驱动处理。其实现代码如下:

```
RtlUnicodeStringToAnsiString ( &fileName, &pFileObj ->
FileName,TRUE); //获得文件名称
.....
//判断该文件名称是不是可执行文件,如果是则执行下列操作
pBuffer = ExAllocatePoolWithTag ( NonPagedPool, newSize,
SFLT_POOL_TAG );
RtlInitEmptyUnicodeString ( & ( fsContext. fileName ), ( PW-
CHAR) ( pBuffer ), ( USHORT) ( newSize ) );
fsContext. fsCtx = pFileObj->FsContext;
RtlCopyUnicodeString( &( fsContext. fileName ), &( pFileObj->
FileName ) );
ExAcquireFastMutex( &pDevExt->fsCtxMutex );
pFsContext = RtlLookupElementGenericTable ( &pDevExt ->
fsCtxTable, &fsContext );
if( ! pFsContext ) {
//判断是否第一次建立,如果是就要加入全局链表
pFsContext = RtlInsertElementGenericTable ( &pDevExt ->
fsCtxTable, &fsContext, sizeof( FILE_CONTEXT ), &newElement );
//加入新的节点,以便在写例程中判断哪些允许写入
}
ExReleaseFastMutex( &pDevExt->fsCtxMutex )
```

在 SfWrite 分发例程中,驱动的主要工作为:驱动将检查当前写操作的文件对象是否在全局链表中,若不在,则直接发送该 IRP 到下层设备驱动中;若在链表中,则表明该文件是可执行文件,需要进行禁止写操作。为了保证合法程序完成写操作,还需要查找进程白名单,如果当前进程在白名单链表的时候,就允许进行写操作,如果不在则拒绝写入操作。

其实现代码如下:

```
fsContext. fsCtx = pFileObj->FsContext;
ExAcquireFastMutex( &pDevExt->fsCtxMutex );
```

```
pFsContext = RtlLookupElementGenericTable ( &pDevExt ->
fsCtxTable, &fsContext );
if( ! pFsContext ) { //不在链表中,直接下发 IRP
ExReleaseFastMutex( &pDevExt->fsCtxMutex );
IoSkipCurrentIrpStackLocation( Irp );
return IoCallDriver( pDevExt->AttachedToDeviceObject, Irp );
}
ExReleaseFastMutex( &pDevExt->fsCtxMutex );

pTmpInfoNode = pInfoListHead->pNext;
cxjGetProcess( runProcName ); //获得当前进程名称
while( pTmpInfoNode ) {
//检测当前操作的进程是不是在进程白名单链表中
if( memcmp( pTmpInfoNode->procName, runProcName,
strlen( runProcName ) ) == 0 ) {
break;
}
pTmpInfoNode = pTmpInfoNode->pNext;
}
if( pTmpInfoNode ) {
//当前进程在白名单链表中,则直接下发该 IRP
IoSkipCurrentIrpStackLocation( Irp );
return
IoCallDriver( pDevExt->AttachedToDeviceObject, Irp );
} else { //若不在,则拒绝该操作
Irp->IoStatus. Status = STATUS_ACCESS_DENIED;
Irp->IoStatus. Information = 0;
status = Irp->IoStatus. Status;
IoCompleteRequest( Irp, IO_NO_INCREMENT );
return status;
}
```

SfCleanupClose 例程主要是监控文件对象的关闭操作以及进行缓存清理工作^[13],通过操作中的文件对象,检查该对象是否在全局链表中,若在链表中,则需要将全局链表中该节点清除掉,以免每次在打开例程中都加入节点,导致节点过多的现象,影响性能。实现代码如下:

```
ExAcquireFastMutex( &pDevExt->fsCtxMutex );
fsContext. fsCtx = pFileObj->FsContext;
RtlLookupElementGenericTable ( &pDevExt -> fsCtxTable,
&fsContext );
ExReleaseFastMutex( &pDevExt->fsCtxMutex )
```

为了下发白名单策略,需要实现应用层和驱动层之间的通信,该功能可以通过 WDM 中的 DeviceIoControl 机制实现,在驱动程序中对应的例程是 SfDeviceIoControl。这需要在应用层和内核层定义共用的 IOCTL 代码,其中传输方式均定义为 Buffered。Buffered 相对于其它传输方式,效率比较低,但使用起来比较方便。然后上层程序通过调用 DeviceIoControl 函数来访问底

层驱动,这时 I/O 管理器会发送对应的 IOCTL 代码的 IRP 到底层驱动,底层驱动对 IRP 堆栈信息域指针的读写就可以实现用户模式和内核模式的信息交换。实现代码如下:

```
pInfoNode = ( PINFO_NODE ) Irp->UserBuffer;  
if( pInfoNode->flag == 1 ) { //添加进程白名单  
    pInfoNodeNew = ( PINFO_NODE ) ExAllocatePoolWithTag (   
NonPagedPool, sizeof( INFO_NODE ), SFLT_POOL_TAG );  
    RtlCopyMemory( ( char * ) pInfoNodeNew, ( char * ) pInfo-  
Node, sizeof( INFO_NODE ) );  
    KeEnterCriticalRegion();  
    pInfoNodeNew->pNext = pInfoListHead->pNext;  
    //在进程白名单链表中插入节点  
    pInfoListHead->pNext = pInfoNodeNew;  
    KeLeaveCriticalRegion();  
}
```

以上从对文件的打开、写入以及关闭分别对整个方法的实现作了一个介绍,该方法的实现更加底层,能够从细粒度上实时监控病毒的恶意行为。

根据以上实现原理,功能结构如图 2 所示。

通过实验验证,该方法能够检测到病毒的传播行为并能加以阻止,达到主动防御病毒的目的。

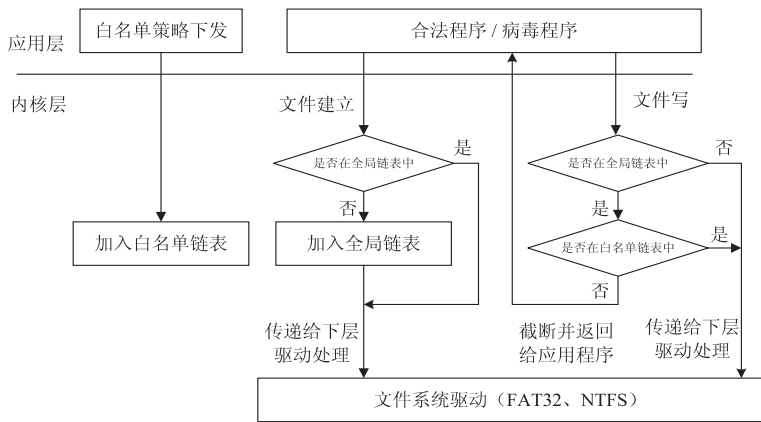


图 2 功能结构图

3 结束语

基于文件过滤驱动,通过在文件驱动上层附加一层过滤驱动,从而在操作系统内核截获文件的读写操作,通过判断是不是对 PE 可执行文件进行写操作,来判断是否是病毒行为。实验结果表明,该方法能够有

效地检测病毒程序对可执行文件的写操作,由于直接截获文件读写 IRP,因此该方法比其它方法在实现上更加底层,截获恶意行为更加有效。

参考文献:

- [1] Pietrek M. Peer Inside the PE: A Tour of the Win32 Portable Executable File Format [EB/OL]. 2002-11-01. <http://www.microsoft.com>.
- [2] 樊震,杨秋翔. 基于 PE 文件结构异常的未知病毒检测[J]. 计算机技术与发展, 2009, 19(10): 160-163.
- [3] Skoudis E, Zeltser L. Malware: Fighting Malicious Code [EB/OL]. 2005. <http://dl.acm.org/citation.cfm?id=1212670&coll=DL&dl=GUIDE&CFID=146756038&CFTOKEN=30644509>.
- [4] Conklin W A, White G B, Cothren C, et al. 计算机安全原理[M]. 王昭,译. 北京: 高等教育出版社, 2006.
- [5] 杨阿辉, 陈鑫昕. 基于 SSDT 的病毒主动防御技术研究[J]. 计算机应用与软件, 2010, 27(10): 288-290.
- [6] Microsoft Corporation. Installable File System Development Kit [EB/OL]. 2004-12. www.microsoft.com/whdc/devtools/ifskit/default.mspx.
- [7] Nagar R. Windows NT file system internals: developer's guide [M]. Cambridge: O'Reilly & Associates, 1997: 615-667.
- [8] 刘亮, 周安民, 沈东. 基于文件过滤驱动的文件保护技术研究[J]. 四川大学学报(自然科学版), 2009, 46(3): 589-593.
- [9] 李凡, 刘学照, 卢安, 等. Windows NT 内核下文件系统过滤驱动程序开发[J]. 华中科技大学学报(自然科学版), 2003, 31(1): 19-21.
- [10] 王全民, 周清, 刘宗明, 等. 文件透明加密技术研究[J]. 计算机技术与发展, 2010, 20(3): 147-150.
- [11] 曹成龙, 傅德胜, 曹凤艳. 基于文件过滤驱动的移动存储控制方法[J]. 计算机应用, 2011, 31(6): 1498-1501.
- [12] Russionvich M E, Solomon D A. 深入解析 Windows 操作系统[M]. 潘爱民, 译. 第 4 版. 北京: 电子工业出版社, 2007: 539-541.
- [13] 谭文, 杨潇, 邵坚磊, 等. 寒江独钓: Windows 内核安全编程[M]. 北京: 电子工业出版社, 2009: 252-255.

(上接第 142 页)

- [8] 杨新英. 基于网络爬虫的 Web 应用程序漏洞扫描器的研究和实现[D]. 成都: 电子科技大学, 2007.
- [9] 陶亚平. Web 应用安全漏洞扫描工具的设计与实现[D]. 成都: 电子科技大学, 2007.
- [10] 陈秀真, 郑庆宏, 管晓宏, 等. 网络化系统安全态势评估的

研究[J]. 西安交通大学学报, 2004, 38(4): 404-408.

- [11] 王廷博, 徐世超. 基于层次分析法的网络安全态势评估方法研究[J]. 电脑知识与技术, 2008, 5(4): 56-58.
- [12] 朱振国, 鄢羽, 张闽, 等. 一种量化的网络安全态势评估方法[J]. 微计算机信息, 2007, 23(3): 62-65.

基于文件过滤驱动的病毒防范方法的实现

作者：

陈建熊，揭摄，张鑫

作者单位：

陈建熊,揭摄(合肥电子工程学院 网络系,安徽 合肥230037)，张鑫(武警合肥指挥学院 院
务部,安徽 合肥230031)

刊名：

计算机技术与发展

英文刊名：

Computer Technology and Development

年，卷(期)：

2013(3)

本文链接：http://d.g.wanfangdata.com.cn/Periodical_wjfz201303038.aspx