Windows内核恶意代码分析与检测技术研究

左黎明
(华东交通大学 基础科学学院,江西 南昌 330013)

摘 要:Windows内核恶意代码是指能够通过改变Windows执行流程或者改变内核审计和簿记系统所依赖的数据结构等手段以达到隐藏自身,实现恶意功能的程序或程序集,对操作系统安全造成很大的危害。对近年来基于NT内核的微软Windows操作系统下恶意代码主要的隐藏实现技术(包括对进程函数、注册表函数、SSDT等的HOOK行为)进行了深入分析研究,提出了一些具有实用价值的恶意代码检测技术方案。实践表明文中提出的恶意代码分析检测技术在实际中具有积极的指导意义。

关键词:内核;恶意代码;Native API;HOOK
中图分类号:TP393.08 **文献标识码**:A **文章编号**:1673-629X(2008)09-0145-03

Research of Analysis and Detection of Malicious Code in Windows Kernel

ZUO Li-ming
(School of Natural Science, East China Jiaotong Univ., Nanchang 330013, China)

Abstract: Malicious code in Windows kernel is a program or set of programs that an intruder uses to hide her presence and allow malicious actions on a computer system by altering the execution flow of the operating system or manipulating the data set that the operating system relies upon for auditing and bookkeeping, it does great harm to the safe of Windows operating system. Carries on the thorough analysis of main concealing techniques of malicious code (include every hooking action on process functions, registry functions, SSDT and etc.) in the Windows operation system based on NT kernel, which has been popular in recent years. Then it proposes some schemes on how to detect the malicious code in Windows kernel, practice has showed that the schemes have very high practicality.

Key words: kernel; malicious code; native API; HOOK

0 引 言

从20世纪90年代末,微软公司的基于NT内核的操作系统得到了长足的发展,系统变得更加稳定与高效,相关应用不断得到深入,越来越多的公司和个人都希望能够搞清楚NT内核操作系统的核心的工作原理,以便更好地在该平台下进行开发。2000年后的一次NT源代码泄露事件更引发了一次研究NT内核结构高潮。内核技术的普及也带来了病毒技术和恶意代码技术的又一次飞跃,很多恶意代码已经能够和杀毒软件工作在同一层次了,导致了此类恶意代码无法彻底清除。与内核恶意代码分析相关的学术文献非常少,文献[1~3]介绍和NT内核和驱动的工作原理和开发方法,文献[4~6]提到了一些特殊的内核恶意代码技术和内核恶意代码分析的入门知识。文中揭示了近年来恶意代码使用的新技术原理,提出了几种实用的检测方法。

收稿日期:2007-12-21
基金项目:江西省自然科学基金资助项目(0611009);江西省教育厅支持项目(赣教技2006123);华东交通大学校立科研基金资助项目(07JC03)
作者简介:左黎明(1981-),男,江西鹰潭人,硕士,讲师,研究方向为信息安全、非线性系统。

1 基于NT内核的Windows操作系统结构

如图1所示,NT内核操作系统一般分为两部分[2]:用户模式(ring3)和内核模式(ring0)。

一般的应用程序运行在用户模式,驱动程序和操作系统内核运行于内核模式(ring0),恶意代码开发者通过使用一些特殊的和微软公司未公开的属性(比如一些Native API)可以直接访问内核资源和系统服务(包括进程管理、内存管理、文件管理等等),并加载自己的驱动和修改已有的系统驱动程序,达到隐藏自己和实现病毒传播、远程监控、网络监听、系统后门等功能的目的[6]。

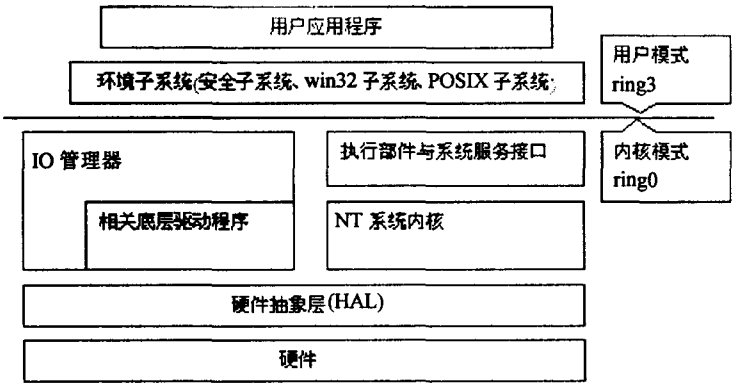

图 1 NT 内核操作系统分层结构

2 内核级恶意代码常用技术分析

2.1 内核级恶意代码使用技术概述

恶意代码开发者想尽了各种办法,对进程、文件、注册表、系统服务、网络服务等各方面信息进行了控制,内核级的恶意代码做得更加巧妙和隐蔽。从技术上进行分类,恶意代码使用的技术手段可以分为:(1)用户模式系统调用劫持;(2)核心模式系统调用劫持;(3)核心模式数据篡改;(4)核心模式中断处理程序劫持。

2.2 对进程信息的控制

Windows 操作系统提供了两套公开的 API 函数来获得进程信息:一套是 PSAPI,通过 EnumProcesses() 函数来枚举进程;另一套是 ToolHelp32,通过 Process32First()和 Process32Next()函数来获得整个进程列表。现在运行在用户层的恶意代码一般都会让自己在系统进程列表中消失,原理是 HOOK 上述进程相关 API 函数,将自身进程从最后的进程列表中去除,但这样做一般很容易发现。

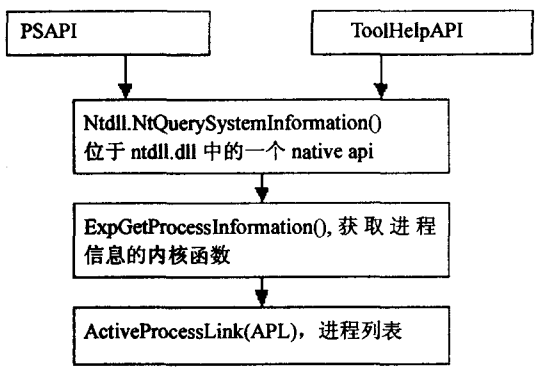

图 2 NT 内核操作系统获取进程过程

如图 2 所示,PSAPI 和 ToolHelp32 都是通过调用位于 ntdll.dll 中的一个 native API 函数 NtQuerySystemInformation()获取进程信息,而函数 NtQuerySystemInformation() 则是依靠内核函数 ExpGetProcessInformation()遍历 ActiveProcessLinks,通过 EPROCESS 结构(其中包含 ActiveProcessLinks 项,类型为 LIST_ENTRY)获得真实进程列表信息。整个环节中,内核态的恶意代码可以试图通过 HOOK 函数 NtQuerySystemInformation() 和函数 ExpGetProcessInformation(),将自己的相关进程从返回结果中去除或者直接从 ActiveProcessLinks 摘除自身相关进程信息,即要把要隐藏的进程的 EPROCESS 从 LIST_ENTRY 中摘除。另外几乎所有的反病毒查杀软件和系统进程软件使用 PsSetCreateProcessNotifyRoutine()来监视进程创建和销毁,但是令人遗憾的是该函数最多只能设置 8 个回调函数[4],因此有不少内核级的恶意代码被动隐藏自身相关进程的时候,还主动地销毁掉其它进程监控软件的使用的回调函数,让进程监控失效。

2.3 对文件和注册表信息的控制

内核恶意代码往往要对自身宿主文件和所在目录进行隐藏,达到保护自身的目地。在 NT 内核操作系统中,与文件和目录相关的 API 函数基本上都是调用 ntdll.dll 中的几个 Native API 函数 NtQueryVolumeInformationFile,NtQueryDirectoryFile,NtCreateFile,NtVdmControl(所对应的内核态下为 ZwQueryVolumeInformationFile, ZwQueryDirectoryFile, ZwCreateFile,ZwVdmControl),因此许多内核级的恶意代码基本上都会 HOOK 上面的这些函数。以 HOOK 函数 ZwQueryDirectoryFile 为例来阐明实现的基本原理:自己定义一个 HookZwQueryDirectoryFile 函数来处理一个名为 alice 的文件名,调用老的 ZwQueryDirectoryFile 获取真实文件和目录信息列表,然后在这个文件和目录信息列表中删除要隐藏的"alice"相关信息,最后将加工后的结果返回。对注册表的信息的控制也是采用同样的方法,对 ntdll.dll 中的 NtEnumerateKey 、NtEnumerateValueKey 或者它们的内核版本 ZwEnumerateKey 、ZwEnumerateValueKey 进行 HOOK 实现隐藏注册表。

对应的在恶意代码驱动程序入口 DriverEntry()也要做相应的处理,通过修改系统服务信息表将系统对 ZwQueryDirectoryFile 的调用指向对 HookZwQueryDirectoryFile 的调用,完成 HOOK 过程。

```
...
//利用 pOldZwQueryDirectoryFile 保留原有函数 ZwQueryDirectoryFile 的调用
pOldZwQueryDirectoryFile = (ZWQUERYDIRECTORYFILE)(
```

```
KeServiceDescriptorTable - > ntoskrnl. ServiceTable[ * (PULONG)((PUCHAR)ZwQueryDirectoryFile + 1)]);
_ asm //嵌入汇编模式,取消写保护
{
PUSH EAX
CLI //禁止中断
MOV EAX, CR0 //将 CR0 中写到 EAX 寄存器
AND EAX, NOT 10000H //禁止 WP 标志位
MOV CR0, EAX //写回 CRO
POP EAX
}
KeServiceDescriptorTable - > ntoskrnl. ServiceTable[ * (PULONG)((PUCHAR)ZwQueryDirectoryFile + 1)] = (ULONG)HookZwQueryDirectoryFile; //将对原来的 ZwQueryDirectoryFile 的调用指向自定义的函数
_ asm //恢复写保护
{
PUSH EAX
MOV EAX, CR0 //将 CR0 中写到 EAX 寄存器
OR EAX, 10000H //恢复 WP 标志位
MOV CR0, EAX //写回 CRO
STI //允许中断
POP EAX
}
...
```

2.4　对系统服务和网络信息的控制

对系统服务的控制主要是针对 advapi32. dll 的四个系统函数 EnumServiceGroupW, EnumServicesStatusExW, EnumServicesStatusExA, EnumServicesStatusA 进行 HOOK,过滤掉返回结果中的恶意代码启动的服务信息,原理和前面的文件信息控制几乎是一样的。对网络信息的控制方法更多,比如在 TDI 层次上进行拦截,TDI 导出了两个设备“\ Device \ Tcp”与“\ Device \ Udp”,内核恶意代码用设备过滤驱动的方法把这两个设备的所有 IRP 包接管过来进行处理后再传给下层驱动,这样可以达到隐藏任意端口的目的。在 NDIS 驱动层上进行拦截,恶意代码自己处理自身远程控制用的封包。

3　内核恶意代码检测

针对目前出现的各种隐藏的非常巧妙的各种内核恶意代码,目前为止没有一种通用的高效方法来进行检测[5~7]。

在与内核恶意代码的较量中,也积累了一些内核恶意代码的检测经验:

(1)比较法。通过直接利用一些底层方法在内核驱动中直接获得信息,再与通过常规使用系统函数方法获取信息比较,如果发现两种方法得到的信息不一样,有理由怀疑存在恶意代码。如图 3 所示,可以通过比较从底层直接遍历 ActiveProcessLinks 枚举进程,可以检测 Hook 了 NtQuerySystemInformation()函数的行为。检测注册表、系统服务、网络端口的方法与此方法一致。

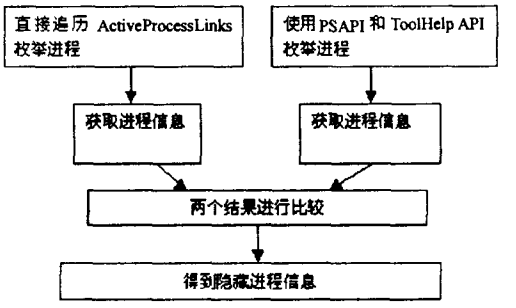

图 3　比较法获取隐藏进程信息

(2)进程状态分析法。例如通过内核线程函数 KiWaitInListHead、KiWaitOutListhead 等和一些寄存器标志来检测从那些 ActiveProcessLinks 上摘除自身隐藏的恶意代码进程。

(3)统计法。根据操作系统取某项系统信息所执行的 CPU 操作执行指令和指令时间的统计特征与真正系统的比较,也可以发现一些恶意代码的踪迹。

以上方法中(1)和(2)较为成熟,许多杀毒软件都采用了,方法比较可靠,但速度慢。方法(3)速度较快,但目前还不成熟,误报率较高,离真正实用高效还有很长的一段路要走,以后还将继续在这方面做一些工作。

参考文献:

[1] Nagar R. Windows NT file system internals[M]. New York: O. Reilly, 2007.

[2] Richter J. Windows 核心编程[M]. 北京:机械工业出版社, 2006.

[3] Microsoft Inc. Microsoft Windows2000 Driver Development Kit[M]. USA: Microsoft Press, 2000.

[4] PolyMeta. 突破 Windows NT 内核进程监视设置限制[EB/OL]. 2007 - 06 - 10. http://www.whitecell.org.

[5] 易　宇,金　然. 基于符号执行的内核级 Rootkit 静态检测[J]. 计算机工程与设计, 2006(16):3064 - 3068.

[6] 康治平,向　宏,胡海波. Windows 系统 Rootkit 隐藏技术研究与实践[J]. 计算机工程与设计, 2007(14):3337 - 3343.

[7] 阮文波,张长河,刘胜利. 基于指令跳转分析的 Windows RootKit 动态检测技术[J]. 信息工程大学学报, 2007(2):221 - 226.