内核脱钩技术在检测 rootkit 木马信息隐藏中的应用

吴坤鸿， 舒 辉， 董卫宇
(信息工程大学 信息工程学院，河南 郑州 450002)

摘 要：简要讨论了 Windows 内核系统服务调用机制，分析了基于 rootkit 技术的木马通过内核态挂钩 SystemServiceDispatchTable 隐藏各种敏感信息的一般原理。在检测 SystemServiceDispatchTable 挂钩隐藏注册表键值的基础上，提出两种内核检测脱钩方法，实现了对 rootkit 挂钩的有效检测与脱钩，确保了系统获取注册表等敏感信息的完整性。

关键词：系统调用； 内核挂钩； 信息隐藏； 内核检测； 内核脱钩
中图法分类号：TP31 **文献标识码**：A **文章编号**：1000-7024 (2008) 14-3635-03

Application of kernel unhooking in detecting information hiding of rootkit trojan

WU Kun-hong, SHU Hui, DONG Wei-yu
(College of Information Engineering, Information Engineering University, Zhengzhou 450002, China)

Abstract: The mechanism of Windows kernel system services call is briefly discussed, and then made an analysis on universal principles of hiding various sensitive information by hooking SystemServiceDispatchTable of trojan based on rootkit technology on the kernel level. On the foundation of detecting registry hiding by kernel hooking, two kernel detecting-unhooking methods which are able to detecte and unhook rootkit hooking is presented, and the query outcome integrity of sensitive information such as system registry is ensured.

Key words: kernel call; kernel hooking; information hiding; kernel detecting; kernel unhooking

0 引 言

目前各种杀毒软件检测技术不断提高，采用传统用户态技术的木马生存空间越来越小，因此，木马技术的研究已经开始转向了操作系统的内核。Rootkit 是指具有系统内核权限工具集的总称，越来越多的恶意软件如典型的 Nt_rootkit[1]、AFXRootkit[2]、Fu_rootkit[3]等工具就采用这种基于系统内核的 rootkit 技术隐藏其敏感信息，包括注册表启动项，进程，文件实体，通讯端口等信息，从而实现其在目标机的深度隐藏，隐蔽地进行信息窃取和各种破坏。本文在 Windows 内核调用机制的基础上，分析了在内核态挂钩 SystemServiceDispatchTable (简称 SSDT)表实现木马各种信息隐藏的一般原理，并以挂钩隐藏注册表为例，提出两种内核检测脱钩方法——基准代码定位比较法和内核文件导出比较法，有效地检测隐藏 rootkit 挂钩的存在并脱钩和获取注册表等完整信息。

1 内核调用与挂钩技术

1.1 内核调用机制剖析

Windows NT 体系结构：出于安全和易维护性的考虑，Windows NT 采用分层的设计思想。系统服务调用将内核模式代码和用户模式代码分开，子系统使用系统服务调用为用户提供应用程序编程接口 API，而系统服务调用向下调用内核执行体实现各种功能。

从图 1 可以看出，从用户模式调用本机系统服务是通过 ntdll.dll 来实现的。表面上，Win32 函数为编程人员提供了大量的 API 接口来实现功能，但是这些 Win32 函数只不过是一个 API 接口的容器而已，它将本机 API 包装起来，通过系统服

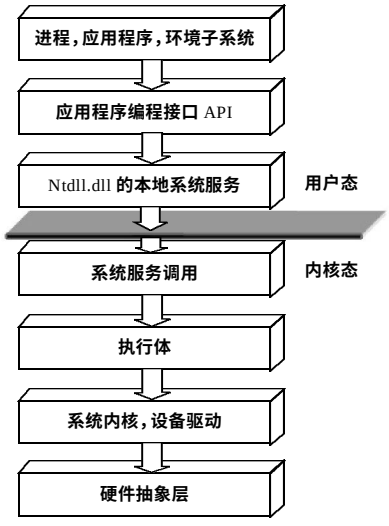

图 1 Windows NT 体系结构

收稿日期：2007-07-16 E-mail：rikky1981@163.com
作者简介：吴坤鸿 (1981－)，男，福建漳州人，硕士研究生，研究方向为网络信息安全； 舒辉 (1974－)，男，江苏盐城人，副教授，硕士生导师，研究方向为并行编译和网络信息安全； 董卫宇 (1977－)，男，辽宁沈阳人，讲师，研究方向为网络信息安全和驱动开发。

务来实现真正的功能，也就是 ntdll.dll 是系统调用接口在用户模式下一个外壳。

从内核模式调用系统服务是由 ntoskrnl.exe 提供的一系列类似 ZwXXX 和 NtXXX 函数实现的，图中的执行体和系统的内核就存在 ntoskrnl.exe 中，并且是分层的。

调用从用户态切换入内核态是通过 cpu 执行陷阱指令来完成。陷阱指令有 Int 0x2e (Windows 2000 intel), sysenter (Windows XP intel),syscall(Windows XP AMD)，下面是以 sysenter 为例给出的一个系统调用模型：

```
mov eax,ServiceID        /*调用服务号*/
lea edx,ParameterTable   /*参数表*/
sysenter (call ntdll! KiFastSystemCall)
ret
```

以下是系统内核导出的和系统调用相关的 3 个全局变量及 ServiceDescriptorEntry 结构[4]：

```
SystemServiceDispatchTable /*系统服务调用函数地址表*/
SystemSercviceParameterTable /*系统服务参数字节数表*/
KeServiceDescriptorTable /*指向上述两个表的描述符表*/
Typedef struct ServiceDescriptorEntry{
    Ulong *ServiceTableBase;          /*SSDT 基址*/
    Ulong *ServiceCounterTableBase; /*SSDT 函数调用计数表基址*/
    Ulong  NumberOfServices;  /*SSDT 服务函数个数*/
    Char  *ParamTableBase;    /*系统服务参数字节数表基址*/
}SSDT_ENTRY;
```

由用户态进入内核态调用 SSDT 的具体实现过程如图 2 所示：①当系统调用函数 ZwXXX 发生内核切换时，利用 ServiceID 服务号作为偏移索引，加上 ServiceTableBase 地址获得函数在 SystemServiceDispatchTable 表中的位置，取得对应函数真正地址；②经过对应 SystemSercviceParameterTable 表中的函数参数字节数检查；③执行内核 ntoskrnl.exe 中的服务代码，并将执行的结果返回给调用参数，用户态程序获得需要的信息，至此完成一次系统服务调用。

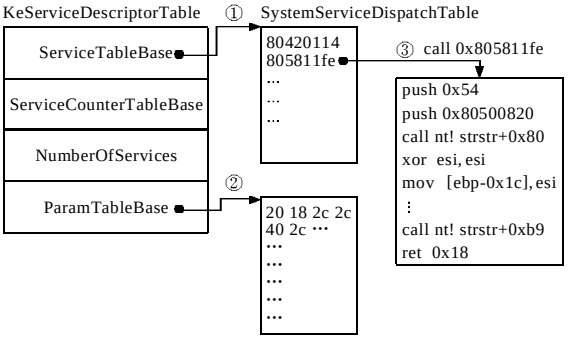

图 2　系统内核 SSDT 调用

1.2　SSDT 内核调用挂钩和注册表隐藏

以 Windows XP，补丁 sp2 某系统为例，当使用 Windows 提供的注册表管理器 regedit.exe 查看某个注册表子键时，系统将调用 advapi32.dll 的 RegEnumValue 函数，进而调用 ntdll.dll 的 ZwEnumerateValueKey 函数，从而切换入内核调用 ntoskrnl.exe 中服务代码(实验中地址为 0x805811fe)，然后将枚举的信息逐步返回给注册表管理器，这样就可以获得注册表这个子键的所有信息。而一般 rootkit 木马使用 SSDT 内核挂钩技术实现隐藏注册表的原理就是通过利用 ZwEnumerateValueKey 的服务号 ServiceID (Windows XP 此值为 0x49)，定位 SystemSercviceParameterTable 表中服务代码的地址，替换为 rootkit 代码地址(如图 3 所示)从而达到篡改系统执行结果，实现注册表敏感信息的隐藏。

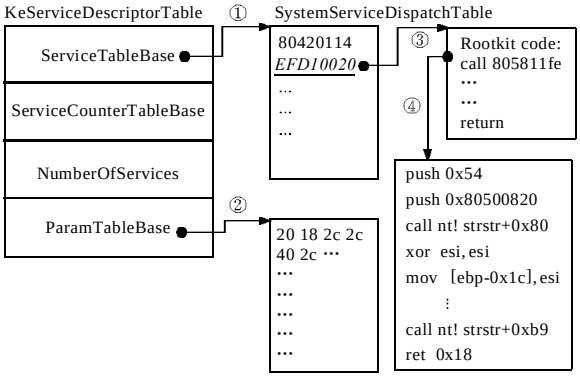

图 3　挂钩 SSDT 调用

挂钩后 SSDT 调用步骤如下(如图 3 所示)：①当系统调用函数 ZwEnumerateValueKey 发生内核切换时，利用 ServiceID 作为偏移索引，获得在 SystemServiceDispatchTable 表中的位置，得到的是被修改过的函数地址 0xefd10020；②经过对应 SystemSercviceParameterTable 表中的函数参数检查；③执行 0xefd10020 地址开始的 rootkit 代码，然后 call 0x805811fe，执行图 3 中第④步，调用内核 ntoskrnl.exe 中的服务代码，并将执行的结果进行修改，去掉所要隐藏的敏感项，最后将修改后的结果返回。

2　内核检测脱钩法

根据前面对系统内核调用机制和挂钩系统调用实现隐藏原理的分析，一旦 SSDT 表被挂钩，通过 ServiceID 服务号和 ServiceTableBase 地址无法获得 SystemServiceDispatchTable 表中对应服务函数的原始地址，而且这些服务函数绝大部分没有导出，那么，如何有效地检测 SSDT 表挂钩的存在并正确获得这些函数的原始地址呢？针对这个问题，本文提出了两种内核检测脱钩方法——基准代码定位比较法和内核文件导出比较法，传统的 rootkit 检测工具如 RootkitRevealer[5]、RootKitHook Analyzer[6]一般只检测报告存在基于 SSDT 挂钩的 rootkit，而本文这两种方法不仅有效检测到基于 SSDT 挂钩的 rootkit，而且能够进行正确函数定位和脱钩，保证用户在使用系统工具如 regedit.exe 可以获取相关的完整信息。

2.1　基准代码定位比较法

基准代码定位比较法的主要思想是筛选干净即无 SSDT 表挂钩的系统中 ntoskrnl.exe 内核目标函数开头一段代码作为基准代码，并利用此基准代码在被挂钩的系统的 ntoskrnl.exe 内核中匹配定位真正函数的地址，之后比较检查 SSDT 表中的地址，对被挂钩的地址进行脱钩。

下面以 Windows XP 系统为例，假设通过 ZwEnumerateValueKey 函数定位到 SSDT 表中的地址命名为 NtEnumerateValueKey，基准代码定位比较法脱钩的实现过程如下：

(1) 获取系统中 ntoskrnl.exe 内核模块的加载首地址和长度。使用 NTDLL.DLL 中的 Native API ZwQuerySystemInformation 函数可以定位 ntoskrnl.exe，当参数 SystemInformationClass 为 SystemModuleInformation (此值为 11) 时，指针参数 SystemInformation 就会指向一块模块信息缓冲区，其结构为 SYSTEM_MODULE_INFORMATION，从而得到首地址 Base 和长度 Size。

ZwQuerySystemInformation 函数原型和_SYSTEM_MODULE_INFORMATION 结构声明如下[7]：

```
NTSYSAPI NTSTATUS NTAPI ZwQuerySystemInformation(
IN  SYSTEM_INFORMATION_CLASS SystemInformationClass,
OUT PVOID SystemInformation,
IN  ULONG SystemInformationLength,
OUT PULONG ReturnLength OPTIONAL);
typedef struct _SYSTEM_MODULE_INFORMATION {
ULONG  Reserved[2];
PVOID  Base;        /*模块首地址*/
ULONG  Size;        /*模块长度*/
ULONG  Flags;
USHORT Index;
USHORT Unknown;
USHORT LoadCount;
USHORT ModuleNameOffset;
CHAR   ImageName[256];
}  SYSTEM_MODULE_INFORMATION,  *PSYSTEM_MODULE_INFORMATION;
```

(2)准基准代码选取。准基准代码的选择必须在无 SSDT 表挂钩的一定数量 n(实验测试时取 n = 8)的不同补丁不同语言版本系统中实现，存在挂钩与否可以根据 NtEnumerateValueKey 是否落在第(1)步计算 ntoskrnl.exe 内核模块的范围之内来判断；若是，存储 NtEnumerateValueKey 开始的 NumOfBytes(初值为 32)个字节于 n 个数组 bytes[NumOfBytes]中，之后对 n 个 bytes [NumOfBytes] 对应序号的字节进行比较计算，提取一致的字节，作为准基准代码存放于 Referbytes [NumOfBytes]。

(3)测试准基准代码 Referbytes[NumOfBytes]。在第(2)步中选取的 n 个系统进行测试，在 ntoskrnl.exe 模块中匹配查找准基准代码 Referbytes [NumOfBytes] 测试其定位到的地址 NtEnumerateValueKey '是否是 NtEnumerateValueKey，若不是，则增加基准代码长度 NumOfBytes，重复测试，直到能够精确定位，最后存放基准代码于数组 Referbytes[NumOfBytes]。

(4)检测并脱钩。在可能存在 SSDT 表挂钩的 XP 系统中，比较 SSDT 表中 NtEnumerateValueKey，若其不在第 (1) 步计算出的 ntoskrnl.exe 模块的范围内说明存在 rootkit 挂钩，用基准代码匹配定位到的 NtEnumerateValueKey '替换 NtEnumerateValueKey 进行脱钩。

基准代码定位比较法的关键在于 Referbytes[NumOfBytes] 的正确选择和匹配速度。改进基准代码定位比较法可以通过增加 n 的数量和测试强度进行更精确选择基准代码，并通过结合函数中其它的特征代码来提高匹配速度和精度，如 XP 系统中 NtEnumerateValueKey 内部都调用函数 ObReference-ObjectByHandle，此函数由系统导出，其地址可以作为一个重要的特征码。

2.2 内核文件导出比较法

这种方法的基本思想是从 ntoskrnl.exe 文件中导出服务函数的原始地址，并以此为基准地址和 SSDT 表中的地址进行比较，发现不同则进行脱钩。因为 KeServiceDescriptorTable 结构初始化工作是由文件 ntoskrnl.exe 加载进内存时调用 KiInitSystem 函数完成的，因此这种方法定位原始地址的基本原理是通过搜索 ntoskrnl.exe 文件中 KiInitSystem 函数初始化 KeServiceDescriptorTable->Base 特殊指令，在实验测试的 Windows XP 系统中，初始化代码反汇编的伪指令为 mov ds: KeServiceDescriptorTable.Base, offset KiServiceTable，搜索到这条指令之后，就可以从 KiServiceTable 指向的位置读出服务函数的原始地址。内核文件导出比较法基本实现过程如下：

(1)从内核内存模块 ntoskrnl.exe 中读取 SSDT 表中服务函数地址，存放于数组 DwordSSDT[1024]。

(2)从内核文件 ntoskrnl.exe 中读取服务函数原始地址，存放于数组 DwordFile[1024]。

在 ntoskrnl.exe 文件中，指令是以二进制编码形式存放的，由于 KeServiceDescriptorTable 是导出的变量，通过导出表计算出其地址，实验中此地址为 0x0046dfa0，以此地址和操作码 0xc705 在 ntoskrnl.exe 函数 KiInitSystem 中搜索 KeServiceDescriptorTable->Base 初始化指令。

一旦找到这条指令，未导出的变量 KiServiceTable 的偏移地址(实验中为 0x4742b8)就找到了，由此偏移地址开始即可读出服务函数原始地址。

(3)检测并脱钩。比较 DwordSSDT[1024]和 DwordFile[1024] 相同序号的函数地址，若不同则说明存在 rootkit 挂钩 SSDT 表，使用 DwordFile[1024]中的函数地址进行替换脱钩。

3 结束语

本文提出的基准代码定位比较法和内核文件导出比较法两种脱钩方法，经过实验环境测试，在安装 Nt_rootkit、AFXRootkit、Fu_rootkit 的系统中，能够有效地检测基于 rootkit 的 SSDT 表挂钩存在，并成功地进行脱钩恢复，能够保证系统中查询注册表等相关敏感信息的完整性。另外，由于基准代码定位比较法不依赖于系统内核文件 ntoskrnl.exe，而内核文件也有被修改的可能，这种检测方法的可靠性要略优于内核文件导出比较法，但是基准代码定位比较法与系统版本和补丁等关联度较大，所以其在稳定性方面不如内核文件导出比较法，在实际检测中二者可以取长补短，结合使用。

参考文献:

[1] GregHoglund. Nt_rootkit [EB/OL]. http://www.megasecurity.org/trojans/n/nt_rootkit/Nt_rootkit.html. (下转第 3641 页)

信息。部分字段的 XML 描述如下：

人力资源系统的部分字段描述：

```
<xs:sequence>
  <xs:element name = "EM_ID" msdata:DataType = "System.Guid, mscorlib, Version = 2.0.0.0, Culture = neutral, PublicKeyToken = b77a5c561934e089" type = "xs:int" minOccurs = "0" />
  <xs:element name = "Name" type = "xs:string" minOccurs = "0" />
  <xs:element name = "Code" type = "xs:string" minOccurs = "0" />
  <xs:element name = "Sex" type = "xs:unsignedByte" minOccurs = "0" />
  <xs:element name = "Nationality" type = "xs:string" minOccurs = "0" />
  <xs:element name = "UnitName" type = "xs:string" minOccurs = "0" />
  <xs:element name = "Mobile" type = "xs:string" minOccurs = "0" />
  <xs:element name = "telphone" type = "xs:string" minOccurs = "0" />
  <xs:element name = "Address" type = "xs:string" minOccurs = "0" />
  <xs:element name = "Status" type = "xs: string" minOccurs = "0" />
</xs:sequence>
```

办公自动化系统的部分字段描述：

```
<xs:sequence>
  <xs:element name = "user_id" type = "xs:int" minOccurs = "0" />
  <xs:element name = "user_name" type = "xs:string" minOccurs = "0" />
  <xs:element name = "Login_name" type = "xs:string" minOccurs = "0" />
  <xs:element name = "user_sex" type = "xs:string" minOccurs = "0" />
  <xs:element name = "user_people" type = "xs:string" minOccurs = "0" />
  <xs:element name = "department_id" type = "xs:int" minOccurs = "0" />
  <xs:element name = "hand_telephone" type = "xs:string" minOccurs = "0" />
  <xs:element name = "telephone" type = "xs:string" minOccurs = "0" />
  <xs:element name = "address" type = "xs:string" minOccurs = "0" />
  <xs:element name = "status" type = "xs:string" minOccurs = "0" />
</xs:sequence>
```

得到两边部分重要字段的对应描述，某些字段需要通过一些转换，比如人力资源系统中部门是名称，办公系统中部门是用id标识的，要通过查询办公系统中的部门基本信息表，将部门名称转化为 id 号写入人员表中，从而最终将人力资源系统的人员数据写入办公自动化系统中，达到数据同步目的。注 :部分字段还需要 XML 描述并对应起来，本文略。

4 结束语

本文分析了当前各企业信息化过程中遇到的问题，即相关的异构系统之间要进行互操作，所以需要集成异构系统。本文在分析了 SOA 的思想和 Web 服务技术后，提出了一种基于 SOA 思想的以 Web 服务为基础的异构系统集成框架。分析了 SOA 在异构系统集成中的优点，用 Web 服务技术构建 SOA，以实现系统集成，并提出了一个集成框架解决实际的应用问题。

参考文献:

[1] Ian Sommerville.软件工程[M].北京:机械工业出版社&中信出版社,2005.

[2] Eric Newcomer,Greg Lomow.Understanding SOA with web service[M].北京:电子工业出版社,2006.

[3] 高月,胡敏.基于.NET 的 Web Service 技术的分布式异构数据库集成[J].Modern Computer,2004(7):18-21.

[4] 孙锦程,殷兆麟. J2EE 与.NET 框架的互操作研究综述[J].计算机工程,2005,31(18):10-13.

[5] 宁葵,滕金芳.新一代的分布式计算技术——Web 服务[J].计算机工程,2003,28(3):196-198.

[6] Stewart Fraser,Steven Livingstone.C# XML 入门经典[M].北京:清华大学出版社,2003.

[7] Fabio Claudio Ferracchiati,Jay Glynn. .NET 数据服务 C#高级编程[M].北京:清华大学出版社,2002.

[8] 陈静,高川,申建军.ASP.NET 在异构数据源转换中的应用[J].重庆大学学报(自然科学版),2006(2):76-78.

(上接第 3637 页)

[2] Fuzen_op. FU_Rootkit [EB/OL]. https://www.rootkit.com/vault/fuzen_op/fu_rootkit.zip.

[3] Prasad Dabak,Milind Borate.Undocumented Windows NT[M]. America:John Wiley & Sons,2000.

[4] Mark Russinovich. RootkitRevealer[EB/OL]. http://www. microsoft.com/technet/sysinternals/Utilities/RootkitRevealer.mspx.

[5] Resplendence. RootKit Hook Analyzer [EB/OL]. http://www.resplendence.com/hookanalyzer.

[6] Gary Nebbett. Windows NT 2000 native API reference [M]. America:Sams,2001.

[7] Walter Oney.Windows driver model[M].America:Microsoft Press, 2001.

[8] Walter Oney.Programming the Microsoft Windows driver model [M].America:Microsoft Press,2003.

[9] Art Baker,Jerry Lozano.The Windows 2000 device driver book [M].America:Prentice HallPTR,2000.