第36卷 第18期 计 算 机 工 程 2010年9月
Vol.36 No.18 Computer Engineering September 2010
·安全技术· 文章编号：1000—3428(2010)18—0138—03 文献标识码：A 中图分类号：N945

基于直接操作内核对象的进程隐藏技术研究

潘茂如，曹天杰
(中国矿业大学计算机科学与技术学院，江苏 徐州 221116)

摘 要：分析直接操作内核对象和调用门的实现机制，提出通过使用调用门，在无驱动情况下提升用户程序的特权级，进而修改内核中的进程双向链表实现进程隐藏。设计并实现一个基于该思路的木马程序，在实验条件下验证该木马的隐蔽性和存活能力，分析应对该类型木马的检测策略。实验证明，该木马可以有效实现进程隐藏，躲过常见安全防护软件的检测与查杀。

关键词：木马；直接操作内核对象；调用门；进程隐藏

Research on Process Hiding Technology Based on Direct Kernel Object Manipulation

PAN Mao-ru, CAO Tian-jie
(School of Computer Science and Technology, China University of Mining and Technology, Xuzhou 221116, China)

【Abstract】 The realization mechanism of the Direct Kernel Object Manipulation(DKOM) and call gate are analyzed and proposed. By using call gate, it can promote the program's privilege to modify the kernel's process list to hide the process without the driver. A Trojan program is designed and implemented, and the hidden and survival functions are verified in experimental conditions based on the proposal. The experiments have proved that the Trojan can hide the process effectively and escape the detection and killing of the common security software. It also analyzes the Trojan program's detection method.

【Key words】 Trojan; Direct Kernel Object Manipulation(DKOM); call gate; process hiding

1 概述

随着网络普及程度的提高，日益泛滥的木马已经成为计算机系统面临的最通常的安全威胁。如今的木马为了躲避杀毒软件和安全工具的检测与查杀，广泛采用了隐藏进程方式来提升生存能力[1]。大多数木马具有一个共同的目的：对计算机进行持续访问，为入侵者提供潜行能力，隐藏入侵者在系统中各项行为的存在，隐藏进程是木马的一项重要工作。

Windows 进程隐藏分用户态和内核态，它们都通过修改系统进程的相关信息或挂靠到系统可信任的进程的方式来实现隐藏。用户态的实现包括注册为服务、修改动态链接库、应用程序的接口挂钩、远程线程注入等方式。内核态的实现包括内核挂钩，直接操作内核对象(Direct Kernel Object Manipulation, DKOM)等方式[2]。DKOM 是内核态进程隐藏常用的技术之一，其在内核层面的进程双向链表中摘除需要隐藏的进程信息，这种进程隐藏行为难以检测。不过，该进程隐藏过程通常需要用驱动程序来获取系统级权限，实现条件较为复杂，且由于驱动程序调用了一些敏感的系统函数[3]，容易被安全软件察觉。

本文在对直接操作内核对象的隐藏进程方式进行分析研究的基础上，提出通过安装调用门，无驱动情况下实现用户程序特权级由 ring3 到 ring0 的提升，在用户态操作内核数据，修改进程双向链表实现进程隐藏。

2 直接操作内核对象隐藏进程

研究基于直接操作内核对象隐藏进程的机制，先要了解 Windows 内核中的进程结构和进程双向链表列表结构。

每个进程和线程都有描述它们的结构，EPROCESS 结构描述进程，ETHREAD 结构描述线程。每个 EPROCESS 结构都有一个 LIST_ENTRY 结构，这个结构包含 2 个 DWORD 指针 Flink 和 Blink，分别指向下一个进程结构和前一个进程结构。除了 LIST_ENTRY 结构之外，EPROCESS 结构中还包含了进程的唯一标识符 PID 和进程名成员等。

2.1 LIST_ENTRY 结构

LIST_ENTRY 结构定义[4]如下：

```
Typedef struct_LIST_ENTRY {
 struct_LIST_ENTRY*Flink;
  struct_LIST_ENTRY*Blink;
} LIST_ENTRY, *PLIST_ENTRY;
```

内核使用该结构将所有进程对象维护在一个双向链表中，通常情况下这个链成环形，也就是说最后一个 Flink 指针指向链表中的第 1 个 LIST_ENTRY 结构，而第 1 个 Blink 指针指向最后一个，这样就很容易双向遍历该链表。遍历整个链表过程中，需要保存当前第 1 个 LIST_ENTRY 结构的地址，以判断是否已遍历了整个链表。

2.2 当前活跃进程列表的枚举

所有的操作系统都在内存中记录各种系统信息，基于直接操作内核对象进程隐藏的原理便是通过修改这些内核级数据信息达到隐藏进程的目的。

Windows 具有唯一的内核处理器控制块结构(KPRCB)，

基金项目：江苏省自然科学基金资助项目(BK2007035)
作者简介：潘茂如(1985－)，男，硕士，主研方向：信息安全，计算机取证；曹天杰，教授、博士生导师
收稿日期：2010-01-20 **E-mail：**falcon0912@163.com

这个结构用来描述处理器当前处理所处理的所有信息，它位于内存地址 0XFFDFF120 处，这个地址偏移 0X124 便是指向当前 ETHREAD 的指针，ETHREAD 的指针向下偏移一个值(其中，PID 和 Flink 的偏移量在不同的操作系统下的值不同)便是当前的进程的 Flink，由此便找到了当前 EPROCESS 结构，从当前 EPROCESS 结构开始，遍历这个双向链表即可获取当前活跃的进程列表信息[5]，如图 1 所示。

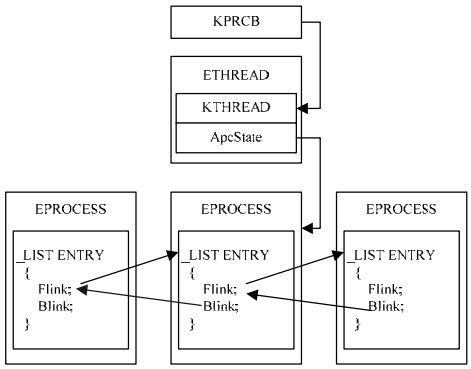

图 1　当前活跃进程双向链表的遍历流程

2.3　进程隐藏的实现

实现进程隐藏时首先需要遍历进程双向链表，通过进程的 PID 定位到需要隐藏的进程，返回它的 EPROCESS 地址。找到需要隐藏的进程结构之后，修改该进程 LIST_ENTRY 结构的值，将该进程的 EPROCESS 结构从链表中移除，该操作过程为双向链表的拖链操作。操作流程如图 2 所示。

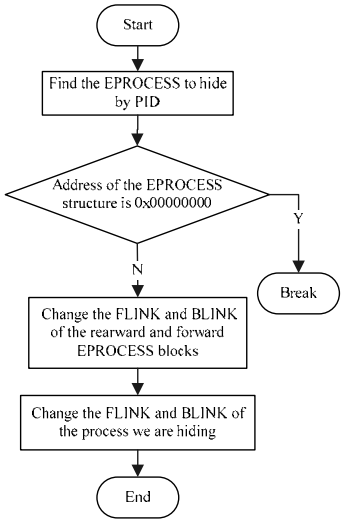

图 2　进程脱链的操作流程

程序在将自身进程从进程链表移除的过程时，将上一个 LIST_ENTRY 结构的 Flink 指针指向下一个 LIST_ENTRY 结构，将下一 LIST_ENTRY 结构的 Blink 指向上一个 LIST_ENTRY 结构。同时程序的 Flink 指针和 Blink 指针指向自身，这样避免因指针指向无效内存区域导致系统崩溃。经过如此操作，无论怎样对进程信息进行枚举，都无法获取已经隐藏的进程信息。基于直接操作内核对象的木马程序，便是通过此方法实现进程的隐藏。如图 3 所示，可以看出，通过直接操作内核对象隐藏进程的需要有对内核对象的读写权限，该过程通常由驱动程序来完成。由于检测软件对不明驱动程序的检测较为敏感，驱动程序获取 ring0 特权过程容易被安全程序发现。可见，如果可以在无驱动的情况下实现用户程序的特权提升，以便对内核数据操作，实现相应功能，将提高基于直接操作内核对象以及其他方式隐藏进程的用户程序的存活能力。

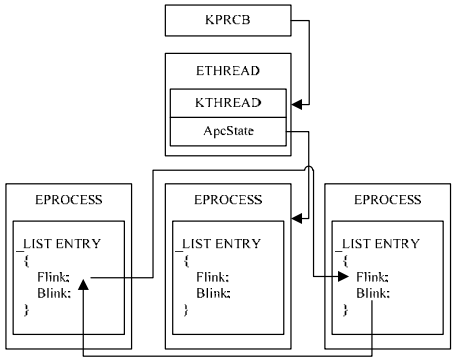

图 3　在链表中移除 EPROCESS 信息的流程

3　ring3 进 ring0 之门

3.1　x86&x64 门机制

x86 下段保护的核心就是门机制，为了在不同特权级的代码段之间进行控制访问，处理器特别提供了一组称为门(调用门、中断门、任务门、陷阱门)描述符的描述符，可以通过这组门描述符中的任何一个进行权限切换[6]。门提供了受保护的间接调用，为任务内的特权转移提供了安全可靠的方法。其中调用门的描述符及其工作原理如图 4 所示。

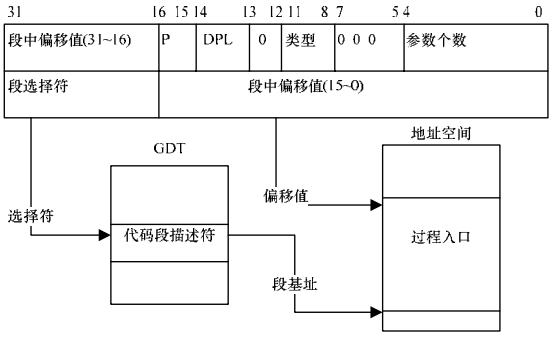

图 4　调用门描述符结构及其工作原理

图 4 中的 P(present)标志是存在标志，类型指门类型(此处为调用门)，DPL(Descriptor Privilege Level)表明访问的目标将需要什么级别的权限，假设调用门的 DPL 值为 3，则任何用户程序都可以访问它。段中偏移值指出将要执行的代码相对于段开始位置的偏移量，段选择符指向 GDT(Global Descriptor Table)。

调用门工作时，首先执行特权检查，对访问权限的检测包括：CPL(Current Privilege Level)，RPL(Requestor Privilege Level)，调用门 DPL 和目标段描述符 DPL。权限检查的规则为：RPL<=调用门 DPL&& CPL<=调用门 DPL。

执行特权检查之后，调用门实现特权原理过程如下：首先，调用门描述符的选择符给出目标代码的代码段描述符，由目标代码的代码段描述符的基地址得出代码的段基址，这个段基址加上由调用门描述符的 2 个段中偏移值求得的偏移值，最终得到代码在地址空间中的入口点，最后，跳转到地址中执行。

3.2　无驱动操作内核对象

在通过直接修改内核对象隐藏进程的过程中使用调用门时，通过 ring3 特权级的 sgdt 指令获得 GDT 地址，由于 sgdt

指令获取的是 GDT 的线性地址，首先需要将 GDT 的线性地址转化到物理地址。然后遍历 GDT，在其中找到一个 present 为 0 的空位，然后在该位置创建一个 DPL 为 3，目标代码段是 ring0 特权级的调用门。虽然 GDT 表位于内核区域，一般用户态程序无权访问，可以通过 GetSecurityInfo、SetEntriesInAcl、SetSecurityInfo 这些 API 来修改叫 PhysicalMemory 的 Section 内核对象的 ACE，该对象可以对物理内存操作。通过该调用门，可在无驱动情况下实现程序特权级由 ring3 到 ring0 的提升，修改内核对象，完成进程隐藏任务。

4 进程隐藏测试与检测分析

4.1 进程隐藏测试

通过对基于门机制的直接操作内核对象过程的分析，可以模拟用户程序隐藏进程的过程。在 Windows 环境下使用 VC6.0 进行代码实现一个木马程序。过程如图 5 所示。图 6 为隐藏木马程序。

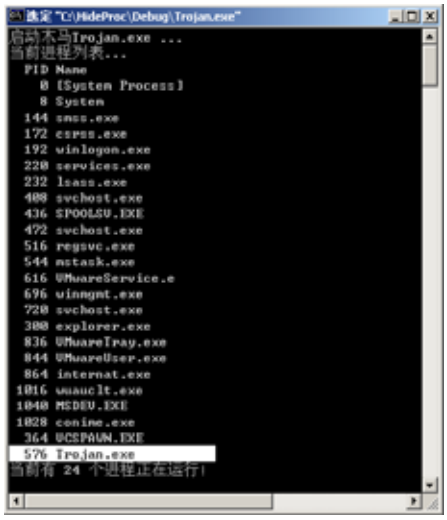

图 5 运行木马程序

图 6 隐藏木马程序

从图 6 可以看出，启动木马程序后，枚举当前系统活跃进程列表，获取当前活跃进程的 PID 和进程名。虽然 PID 是由系统伪随机分配的，获取进程列表之后，很容易获得木马进程的 PID。按提示输入木马程序的 PID，将木马进程从链表中摘除，再次枚举进程列表，可以看到木马进程已经被隐藏。此外，除了 PID 为 0 的系统进程，输入当前活跃进程的 PID，都可以实现对其进程的隐藏。

实验结果表明，木马载入时，可以躲过当前主流的杀毒防护软件。木马运行后，Windows 任务管理器、EF Process Manager、ProcessExplorer 等进程查看工具和 360 木马查杀、IceSword 等木马查杀工具都查看和检测不到被隐藏的木马信息。

4.2 进程隐藏的检测分析

对于直接操作内核对象隐藏进程的检测，其实现难度较大且复杂，由于程序已经将自身进程从链表中摘除，无法通过枚举进程方式检测其隐藏行为。不过 Windows 操作系统基于线程对象的资源调度规则，木马程序可以修改进程链表来隐藏进程，却不能修改操作系统的线程列表。一旦程序解开了想要隐藏的线程数据结构，对应线程也就无法获得 CPU 资源，程序也就无法运行了，可见只要程序在运行，就一定有与之对应的线程资源和进程信息，因此，可以利用基于线程调度的方式枚举线程对象，查找与之对应的进程对象来检测隐藏进程。

此外，对于一个进程来说，无论如何隐藏，EPROCESS 结构总是存在的，那么进程创建时随机分配的 PID 也就存在，而 PID 的值有上限，如 32 位计算机的 PID 范围应为 0~65 535，也可通过穷举 PID 然后调用 OpenProcess 函数返回指定进程的句柄，比对枚举到的进程列表，判断是否有隐藏的进程。

由此延伸思考，任何当前活跃的进程都会在内存留有信息，对内存区域进行逐页扫描，判断内存中是否有类型为进程对象的 Windows 内核对象存在，进而可以得到存在系统中的进程信息，然后比对枚举到的进程列表可得出结论。

5 结束语

本文提出在基于直接操作内核对象隐藏进程过程中安装调用门，在无驱动情况下实现用户程序特权级由 ring3 到 ring0 的提升，在用户态对系统内核数据进行操作隐藏进程。经过试验证明，与常见通过驱动程序获取 ring0 特权方式相比，该隐藏过程更为简单和容易，且无需调用一些敏感的系统函数，进程隐藏的成功率更高，检测难度更大。如何提高程序的稳定性和隐蔽性，如何分析和检测基于该机制木马程序对系统可能造成破坏，设计基于该技术的系统安全防护软件，都是下一步需要考虑和研究的。

参考文献

[1] 杨 彦, 黄 皓. Windows Rootkit 隐藏技术研究[J]. 计算机工程, 2008, 34(12): 152-154.

[2] 王建华, 张焕生, 侯丽坤. Windows 核心编程[M]. 北京: 机械工业出版社, 2001.

[3] Walter O. Programming the Microsoft Windows Driver Model[M]. [S. l.]: Microsoft Press, 2003.

[4] MSDN. LIST_ENTRY[EB/OL]. (2009-09-16). http://msdn.microsoft. com/en-us/library/aa491571.aspx.

[5] Greg H, James B. RootKits: Subverting the Windows Kernel[M]. [S. l.]: Addison Wesley Professional, 2005.

[6] 邓 志. x86 & x64 沉思录[EB/OL]. (2008-12-16). http://linux.chinaunix. net/bbs/thread-1052389-1-1.html.

编辑 陈 文