第 32 卷 第 20 期 计 算 机 工 程 2006 年 10 月
Vol.32 № 20 Computer Engineering October 2006
·软件技术与数据库· 文章编号：1000—3428(2006)20—0095—02 文献标识码：A 中图分类号：TP316

Windows(2000/XP)下隐藏进程的检测机制

王麟峰，董亮卫
(电子科技大学计算机学院，成都 610054)

摘 要：随着计算机技术的不断发展，近期出现了利用 Windows(2000/XP)内核设计上的漏洞隐藏自身进程的入侵技术。针对这种隐藏技术提出了利用内核进程环境控制块(KPEB)、内核线程环境控制块(KTEB)以及 Windows 操作系统的调度机制来检测这些隐藏进程的新方法，并给出了代码示例。

关键词：进程；隐藏；内核进程环境控制块；内核线程环境控制块；检测

Detection on Windows(2000/XP) Hidden Process

WANG Linfeng, DONG Liangwei
(Department of Computer, University of Electronic Science and Technology, Chengdu 610054)

【Abstract】 With the development of computer technology, lately some malicious code use the leak of Windows(2000/XP) kernel design to hide their processes. In order to detect the hidden processes created by malicious code, a new technology has been described with the example of program. The technology involved includes: the kernel process environment block(KPEB), the kernel thread environment block(KTEB), the mechanism of traditional processes detection and dispatcher.

【Key words】 Process; Hidden; Kernel process environment block(KPEB); Kernel thread environment block(KTEB); Detection

由于 Windows 操作系统自身的不完善和不断暴露出来的漏洞，因此它被各种病毒、木马、后门等恶意入侵程序利用，这些入侵者利用各种方法掩盖它们存在于目标系统中的任何迹象来欺骗合法用户。隐藏它们的运行实例是最重要的手段之一，如比较著名的 fu rootkit，它是一种最新流行的后门工具，入侵者利用它来隐藏自身的运行实例。由于这种入侵技术出现较晚，针对其的反隐藏技术仍处于探索阶段。目前的防护工具很难发现这些入侵程序，严重地影响了系统的安全性和可靠性。因此针对检测这些进程的技术进行研究十分必要。根据项目实践，我们发现可以利用 Windows 内核的 KPEB 和 KTEB 以及 Windows 的进程调度原理，实现一种可靠的进程检测机制。

1 进程隐藏的原理

在 Windows 操作系统中每个进程都有自身的进程环境块——KPEB，进程是程序运行的实例，每个进程拥有各自的线程，进程的执行和调度是以线程为基础，而每个线程也拥有各自的线程环境块——KTEB。KPEB 和 KTEB 分别由结构 EPROCESS 与 ETHREAD 来表示，这 2 个结构中的成员包含了内核的方方面面，是 2 个比较大的结构。它们在 Windows 内核中的地位是不言而喻的。但 Microsft 并没有直接公开这些结构的内容，因此必须通过逆向工程的方法来查看它们的内容。结构的内容来源于 Microsoft Windbg 的 extension kdex2x86.dll。关于 Windbg 的使用请翻阅 Microsft 相关文档。下面先介绍 EPROCESS，因为其非常庞大，所以只列出了该结构中与本文相关的成员：

```
typedef struct _EPROCESS{
KPROCESS Pcb;                /*在结构中的偏移量为 0x000*/
DWORD UniqueProcessId;       /*代表了进程的 id*/
LIST_ENTRY ActiveProcessLinks;
BYTE ImageFileName[16]; /*代表进程名，进程名最多只能有 16 个字节*/
} EPROCESS, *PEPROCESS, **PPEPROCESS;
```

EPROCESS 结构含有 2 个重要的成员变量 Pcb 和 ActiveProcessLinks。Pcb 是 KPROCESS 类型的结构，这个结构含有进程运行时的一些状态信息，如进程的内核态工作时间、用户态工作时间、该进程是否已获得 CPU、进程页表的基址等，其中有名为 ThreadListHead 的成员变量，由它可以得到该进程所拥有的线程。ActiveProcessLinks 是一个 LIST_ENTRY 类型的双向链表节点，所有该进程的成员变量组成了一个双向链表。KPROCESS 和 LIST_ENTRY 分别定义如下：

```
typedef struct _KPROCESS{
  LIST_ENTRY ThreadListHead; /*在结构中的偏移量 0x50*/
}KPROCESS,*PKPROCESS,**PPKPROCESS;
typedef struct _LIST_ENTRY{
   struct _LIST_ENTRY *Flink;
    struct _LIST_ENTRY *Blink;
}LIST_ENTRY,*PLIST_ENTRY;
```

由这 3 个数据结构可以得出进程与其所拥有的线程在内核中的组织方式见图 1。

传统的进程检测方法主要是通过遍历由每个进程的 ActiveProcessLinks 所组成的链表(表头指针 PsActiveProcessList)得到所有进程的信息，如 Windows 提供的内核函数 ZwQuerySystemlnformation(在用户态下用于枚举进程的 API

作者简介：王麟峰(1978－)，男，硕士生，主研方向：安全理论与技术，网络计算技术；董亮卫，硕士

收稿日期：2005-11-02 **E-mail：**wanglf@cdnet.edu.cn

最终要调用这个函数)。而 Windows 对进程的调度和切换是基于线程，Windows 的线程分派器使用另外的数据结构，所以修改 ActiveProcessLinks，或者挂钩 ZwQuerySystemlnformation 都不会影响这些进程的执行(通过线程分派器仍然能获得 CPU 使用时间)，这就给入侵者提供了隐藏进程的机会。进程隐藏的实现主要有两种方法：

(1)更改系统内核中的数据结构，直接隐藏某些对象。该方法不修改任何代码，仅通过释放内核中 PsActiveProcessList 链上的节点 ActiveProcessLinks 来隐藏进程；

(2)通过挂钩或修改各种系统函数可以改变系统函数的行为从而实现进程的隐藏。如挂钩 ZwQuerySystemlnformation 等函数，使之列举不出某些进程。

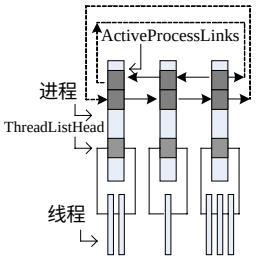

图 1 进程和线程的关系示意图

2 Windows 进程的调度机制

Windows 下线程是基本的执行单位，所有的调度都是基于线程的，Windows2000 和 WindowsXP 在调度时所使用的数据结构却不一样。但都要用到 ETHREAD 类型的 KTEB，ETHREAD 结构的第 1 个成员名为 Tcb，其结构如下：

```
typedef _ KTHREAD{
KAPC_STATE ApcState;          /*偏移量 0x034*/
 LIST_ENTRY WaitListEntry;    /*0x05C*/
LIST_ENTRY QueueListEntry;    /*0x110*/
LIST_ENTRY ThreadListEntry;   /*0x1A4*/
} KTHREAD, *PKTHREAD, **PPKTHREAD;
```

因为它是一个非常大的结构，所以只列出了与本文相关的部分，其中的成员 ApcState 也是一个含有 KPROCESS 类型结构的指针 Process，由它指向了进程的 KPEB 的第 1 个成员变量 Pcb，它所指向的地址值也是 KPEB 的地址值，所以通过 Process 可以直接访问 KPEB 的其它成员变量。

2.1 Windows2000 的调度机制

Windows2000 的线程分派器使用以下 3 个数据结构：*pKiDispatchReadListHead，*pKiWaitInListHead，*pKiWaitOutListHead。第 1 个是已经处于“准备”状态可以通过分派器获得 CPU 的线程链表头指针。后 2 个是处于不同“等待”状态的线程链表头指针。这 3 个链表均是 LIST_ENTRY 类型的链表，后 2 个链表中的节点是处于不同类型等待状态线程的 ETHREAD 中的 WaitListEntry 成员。而 pKiDispatchReadListHead 是具有 32 个节点的链表，每个节点代表了一个优先级，每个节点又指向另一个链表，链表的每个节点是该优先级下某个线程中的 WaitListEntry。*pKiDispatchReadListHead 的组织方式如图 2。

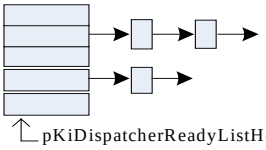

图 2 Windows2000 线程调度链表

2.2 WindowsXP 的调度机制

WindowsXP 下仅有 *pKiWaitListHead 和 *pKiDispatcherReadyListHead 结构，它的线程分派主要是利用 ETHREAD 中的 WaitListEntry、QueueListEntry 和 ThreadListEntry 来完成。这 3 个成员变量分别代表了 3 种状态的线程，其中由已经处于“准备”状态等待调度的线程成员 ThreadListEntry 组成了一个双向链表。其余 2 个成员组成的双向链表分别代表了各自处于“等待”状态的线程，如图 3。WindowsXP 就是利用了这 3 个链表实现了对线程的调度。

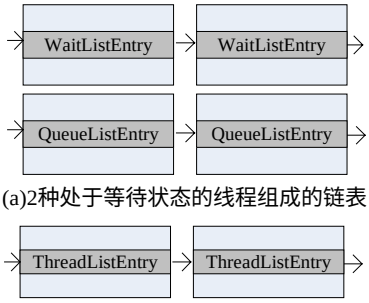

图 3 WindowsXP 下的线程链表

3 基于线程调度的隐藏进程检测机制

3.1 Windows 2000 的检测机制

Windows 2000 下主要是检测*pKiDispatchReadListHead、*pKiWaitInListHead、*pKiWaitOutListHead4 3 个链表。由这 3 个链表可以得到每个线程的 ETHREAD 和 ETHREAD 中的成员 Tcb，并由 Tcb.ApcState.Process 进入所属进程的 KPEB 得到所需要的进程信息。通过这样的方式可以得到所有的进程，隐藏的木马进程虽然可以从进程链表上释放自己，但无法从线程链表上释放自己，除非它不想获得 CPU，一个无法获得 CPU 时间的入侵者对用户来说是可以忽略不计的。其主要的代码实现如下：

```
/*KTEB 中的 Tcb 含有成员 WaitListEntry，它在 KTEB 中的偏移量为 0x5c*/
#define WAITLIST_OFFSET 0x5c
void createProcList () {
PLIST_ENTRY obj;
PETHREAD pethread;
for (obj=pKiWaitInListHead->Flink; obj && obj!= pKiWaitInListHead; obj = obj->Flink) {
   pethread = (PETHREAD) ((char*)obj - WAITLIST_OFFSET);
   insertProc (pethread->Tcb.ApcState.Process); }
for (obj=pKiWaitOutListHead->Flink; obj && obj!= pKiWaitOutListHead; obj = obj->Flink) {
   pethread = (PETHREAD) ((char*)obj - WAITLIST_OFFSET);
   insertProc (pethread->Tcb.ApcState.Process); }
for (int i=0; i<32; i++)
   for (obj=pKiDispatcherReadyListHead[i].Flink; obj!= &pKiDispatcherReadyListHead[i]; obj=obj->Flink) {
     pethread=(PETHREAD) ((char*)obj - WAITLIST_OFFSET);
      insertProc (pethread->Tcb.ApcState.Process); }  }
void insertProc (PEPROCESS obAddr) {
  for (int i=0; i<nprocs; i++)
   if (procs[i].obAddr == (int)obAddr) return;
   procs[nprocs].obAddr = (int)obAddr;
   procs[nprocs].pid = obAddr->UniqueProcessId;
```

(下转第 99 页)