文章编号:1001 - 9081(2010)05 - 1327 - 04

基于虚拟机技术的进程分析方法

高　勇[1,2]，范明钰[1,2]

(1. 电子科技大学 计算机科学与工程学院，成都 610054；　2. 电子科技大学 信息安全研究中心，成都 610054)

(gao6081295@163.com)

摘　要:针对现有进程分析方法存在的缺陷,提出了一种在 Windows 平台虚拟环境下分析进程的方法。该方法首先在宿主机下分析虚拟机的内存,捕捉当前线程,并通过内核数据结构得到当前线程所在进程，然后通过页目录表物理地址计算进程页面,对内存进行清零来结束进程。实例分析表明本方法在保护宿主机安全的同时,能快速监测到程序,并且可以有效地结束进程。

关键词:虚拟机;内核;进程;CR3

中图分类号: TP316.7　　**文献标志码**:A

Process analyzing method based on virtual machine

GAO Yong[1,2], FAN Ming-yu[1,2]

(1. *School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu Sichuan* 610054, *China*;
2. *Research Center of Information Security, University of Electronic Science and Technology of China, Chengdu Sichuan* 610054, *China*)

Abstract: In view of the shortcomings of the existing process analyzing methods, a new method was proposed based on virtual environment of Windows platform. This method captured the current thread by analyzing virtual machine's memory under host, got the current process by the kernel data structures, and set zero among the memory to kill the process. The physical address of memory could be worked out by using the base address of page table. The experimental result shows that the proposed method can quickly detect process, effectively kill the process, and maintain the host security at the same time.

Key words: Virtual Machine (VM); kernel; process; CR3

Journal of《计算机应用》Computer Applications

0　引言

随着计算机的普及,计算机给人们带来了越来越多的便利,但计算机安全问题也越来越严重。为了应对安全问题,计算机用户需要有安全意识,而安全厂商们也积极防御恶意程序。

现有恶意程序一般都对常用函数进行了挂钩或对某些内核数据结构进行了修改,并拥有着过高的权限,使得查杀越来越难。并且在对其进行分析的过程中很容易造成对系统不可估计的破坏。为了分析了解系统的安全状况,有必要对系统内的进程进行监控分析。但操作系统及系统中存在的各种软件很有可能存在安全隐患而易受攻击,且监控程序本身也易受到攻击。同时,系统层的监控程序常涉及系统内核的修改,对 Windows 等非开源的操作系统而言往往不可实现或实现后或多或少会造成系统的不稳定。因此,本文提出一种利用虚拟机(Virtual Machine, VM)对进程进行监视分析的方法。虚拟化是指分离操作系统和硬件抽象层来对计算机资源进行分配和应用,目前比较流行的虚拟机有:VMware、Xen、Virtualzzo、Parallels 等。本文选取 Windows 平台下最常用的 VMware 作为分析进程时使用的虚拟机。

1　现有进程分析方法

1.1　进程检测方法分析

现有的进程检测方法主要有以下几种。

1)系统函数枚举法。这类函数比较多,用户层的函数如 CreateToolhelp32Snapshot()、Process32First()、Process32Next() 等,内核层的函数如 ZwQuerySystemInformation() 等。

2)挂钩 SwapContext()。在 Windows 系统中每个线程切换时, SwapContext() 函数将当前运行线程的上下文与重新执行线程的上下文进行交换,在每次线程切换时定位线程,根据线程找到相应进程[1]。

3)遍历 EPROCESS 结构体双向链表。EPROCESS 结构描述了进程的信息,所有进程的 EPROCESS 结构由其中的 FLINK 和 BLINK 链接成一个环形的双向链表。EPROCESS 结构随着进程的创建而创建,并在进程的生命期始终存在。所以遍历 EPROCESS 结构链表即可找到所有进程[2]。

4)扫描调度线程表。在 Windows 操作系统中维持了系统调用的线程列表,在 Windows 2000 中有 KiWaitInListHead、KiWaitOutListHead 和 KiDispatcherReadyListHead 三个双向线程链表。前两个链表包含的是等待线程,第三个链表包含的是就绪线程。遍历链表通过 ETHRE-AD 线程结构得到该线程所属进程的 EPRO-CESS 结构指针,从而遍历所有进程[3]。

5)挂钩系统调用。每个进程在其生命期中都会调用 API, 大多数这类 API 服务请求都会通过系统调用转入内核。基于这个事实,在从用户层转向内核层的时候将其拦截,获取指向当前进程 EPROCESS 的指针,从而发现进程[4]。

6)遍历 PspCidTable。在 Windows 下所有的资源都是以对象的方式进行管理,PspCidTable 是一种句柄表,通过遍历

收稿日期:2009 - 11 - 26;**修回日期**:2010 - 01 - 25。　**基金项目**:国家 863 计划项目(2009AA01Z403;2009AA01Z435)。

作者简介:高勇(1985 -),男,山东德州人,硕士研究生,主要研究方向:信息安全;　范明钰(1962 -),女,四川成都人,教授,博士生导师,主要研究方向:信息安全、网络安全、密码学。

PspCidTable 中的所有句柄值[5]，记录句柄类型为进程值，这样系统内所有的进程对象都可以被检测出来。

7）遍历内存。因每个进程的 EPROCESS 结构存在于内存之中，则可以通过遍历保存 EPROCESS 结构的内存区域 0x80000000 ~ 0x90000000 来查询所有 EPROCESS，并通过 EPROCESS 中的 ExitTime 域来判断该进程是否结束，以记录所有活动进程[6]。

上面常用的方法中，对于 1）、2）两种方法，挂钩相应函数或更底层函数便无法检测到进程；对于方法 3），把要隐藏的进程的相关结构从双向链表中脱离，在遍历时便无法发现进程；对于方法 4），只能在 Windows 2000 中使用，因为在 Windows XP 中线程调度只有 KiWaitListHead 和 KiDispatcherReadyListHead 两个线程链表，其中后者很难查找，它受到操作系统版本的限制；对于方法 5），某些进程可能长时间处于等待状态或此段时间不进行系统调用，这样就无法检测到；另外也可以采用修改执行系统调用的中断或用全局描述符表（Global Descriptor Table，GDT）中的调用门实现系统调用避开这种方法的检测；对于方法 6），可利用 Rootkit 技术更改 PspCidTable 中的信息，那么检测程序将无法将进程对象区分出来；对于方法 7）遍历区域比较大，消耗时间长，不能实时监视活动进程。

1.2 进程结束方法分析

现有的结束进程方法主要有以下几种。

1）直接调用函数。最简单的结束进程的方法就是直接调用系统导出的函数，用户层的函数如 TerminateProcess()，内核层的函数如 Zw/NtTerminateProcess()，利用这些函数可直接结束进程。

2）消息机制。一种方法是用 FindWindow() 等得到的进程窗口句柄，然后用 SendMessage() 等向出口发送 WM_CLOSE 消息；另一种方法是找到进程的主线程，用 PostThreadMessage() 发送 WM_QUIT 消息结束主线程。当进程响应这些消息时自动退出。

3）创建远程线程。在目标线程的内存中开辟一块内存空间，创建一个远程线程，可以让远程线程执行 ExitProcess()、DebugActiveProcess() 等函数结束进程。

4）破坏进程资源。以 VM_OPERATION 方式打开进程，然后调用 NtUnmapViewOfSection 卸载掉进程的 dll 文件；或打开进程后利用 DuplicateHandle() 以 DUPLICATE_CLOSE_SOURCE 方式复制进程中的句柄，这样目标进程中的句柄在被复制的同时也被关闭了。这两种方法都破坏了进程中的资源，使得进程再次使用资源时因资源无效而退出。

上述四种方法同时存在一个致命的弱点：需要调用系统函数。这样可以针对不同的方法挂钩相应的函数或更底层的函数来过滤进程，就无法达到结束进程的目的。

2 基于 VM 技术的进程分析方法

目前利用 VM 的技术还局限于 Xen 等开源虚拟机上，Xen 是由英国剑桥大学研究的开源项目，只能在 Linux 上安装，不支持 Windows，因此在 Windows 下不能像 Linux 下利用直接修改源代码来完成对虚拟机的监视工作，这给不熟悉 Linux 的研究人员带来很大的不便。本文通过对虚拟机 VMware 内存文件 vmem 的研究来完成对虚拟机内进程的监视和结束。

2.1 虚拟内存

在 32 位机上系统为每个进程都分配 4 GB 的地址空间，一般情况下前 2 GB 供应用程序使用，后 2 GB 供内核使用。这里所说的都是虚拟内存，那么要想从虚拟内存里读出数据，就需要知道虚拟内存和物理内存的转换机制。Windows NT 内存有两种分页方式：4 KB 分页和 4 MB 分页。下面详细介绍两种分页机制从线性地址到物理地址的转换[7]。

1）4 KB 分页机制。在把线性地址转换为物理地址时，CR3 寄存器保存了页目录的地址，页目录由 1 024 项（PDE）组成。根据线性地址的前 10 位来确定页目录中的偏移，找到对应的 PDE；再根据 PDE 的前 20 位定位页表，每个页表由 1 024 个 PTE 组成；根据线性地址的中间 10 位确定页表中的偏移，找到对应的 PTE；根据 PTE 的前 20 位来定位相应的数据页；最后根据线性地址的最后 12 位确定数据页中的偏移，以此得到线性地址所对应的物理地址。转换方式如图 1 所示。

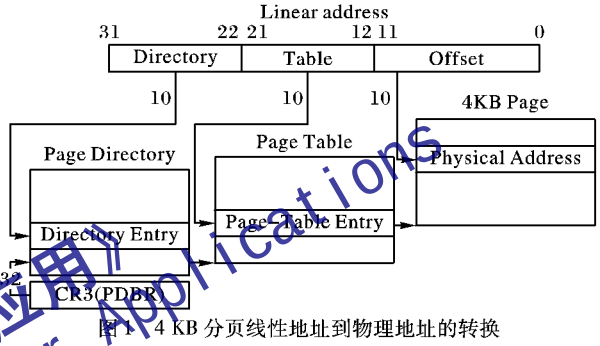

图 1 4 KB 分页线性地址到物理地址的转换

2）4 MB 分页机制。在把线性地址转换为物理地址时，CR3 寄存器保存了页目录的地址。根据线性地址的前 10 位来确定页目录中的偏移，找到对应的 PDE；再根据 PDE 的前 10 位定位数据页；根据线性地址的最后 22 位确定数据页中的偏移，以此得到线性地址所对应的物理地址。转换方式如图 2 所示。

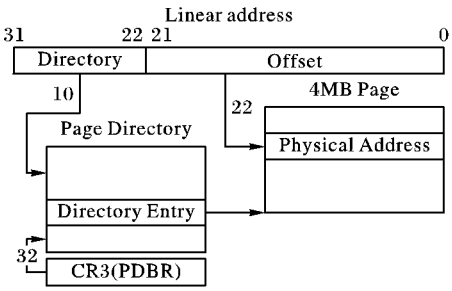

图 2 4 MB 分页线性地址到物理地址的转换

在转换过程中，通过 PDE 的第 8 位判断是哪种分页机制，PDE 的 0 ~ 11 位为标志位，其中第 8 位成为 PS 位，如果为 1，表示采用的是 4 MB 分页机制；如果为 0，则采用的是 4 KB 分页机制。通过页目项的第 0 位判断其是否在物理内存中达到安全访问的目的，当为 1 时，表明页表已经在物理内存中。

2.2 进程检测方法

在 Windows 系统中，一般情况内存 0x80000000 ~ 0xFFFFFFFF 是系统内核地址空间，其中存在处理器控制块结构（KPRCB）来描述 CPU 当前所处理的所有信息[8]，可在 Windbg 里面使用 dt _KPRCB 命令得到其结构，此结构的 0x4 偏移处是 Ptr32 _KTHREAD 类型的 CurrentThread，即一个指向当前线程 KTHREAD 结构的指针，接下来可以在 KTHREAD 偏移 0x44 处得到此线程所在进程的 KPROCESS 结构。以上

《计算机应用》Journal of Computer Applications

所使用的数据结构信息可在 Windbg 下通过 dt 命令加相应的数据结构名称得到。数据结构关系如图 3 所示。

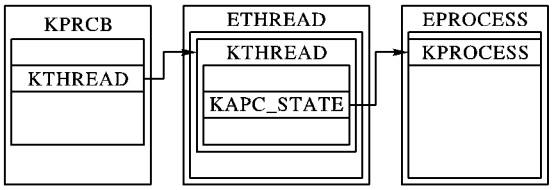

图 3　从 KPRCB 得到 EPROCESS

本文正是根据这些数据结构的联系,通过监视 KPRCB 来实时监视进程的运行,本方法的具体流程如图 4 所示。

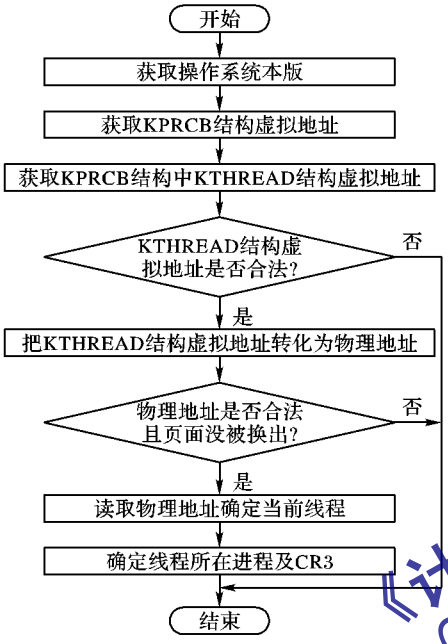

图 4　进程检测方法流程

部分代码如下:

```
ReadVirMem((PVOID)(0xffdff124), &KThread, sizeof(ULONG));
ReadVirMem ((PVOID)(KThread + 0x34), &KApcstate,
  sizeof(ULONG));
ReadVirMem ((PVOID)(KApcstate + 0x10), &KProcess,
  sizeof(ULONG));
ReadVirMem ((PVOID)(KProcess + PROCNAME_OFF),
  ProcName, sizeof(CHAR) * 20);
ReadVirMem ((PVOID)(KProcess + 0x18), &cr3,
  sizeof(ULONG));
```

其中函数 ReadVirMem(PVOID virBaseAddr, PVOID outBuffer, ULONG uSize)需要根据 2.1 节的方法把线性地址转换成物理地址。

2.3　内存清零方法

在 Windows 下每个进程的 4 GB 空间中系统内核空间被所有进程共享,而用户空间是进程独占的,这里面保存着进程的资源。所以当破坏掉进程用户空间数据时,进程无法继续执行,且由于用户空间的独占性,此时并不影响其他进程的运行,也不影响操作系统的稳定性。

对内存清零是根据 2.2 节中得到的 CR3 的值即页目地址,对其页面中的数据进行清除。其具体步骤如下:

1)为确保在对进程用户空间数据进行清零的过程中不会发生进程切换的现象,需要把虚拟机暂停,此处采取在宿主机挂起虚拟机进程的方法;

2)根据 2.1 节的介绍利用 CR3 的值计算进程页面,把虚拟地址转化为物理地址,并判断页面地址是否有效和页面是否没有被换出内存,若是,则用 0xCC 填充进程的用户空间以达到清空内存的目的;

3)恢复虚拟机的运行,即在宿主机激活虚拟机进程。

部分代码如下:

```
VOID KillProcess()
{
  hlib = GetModuleHandle("ntdll.dll");
  myNtSuspendProcess = (ULONG) GetProcAddress(hlib,
    "NtSuspendProcess");
  __asm
  {
    push hproc
    call myNtSuspendProcess
    mov uret, eax
  }
  WriteDataToPhyMem(cr3, 0xcc);
  myNtResumeProcess = (ULONG) GetProcAddress(hlib,
    "NtResumeProcess");
  __asm
  {
    push hproc
    call myNtResumeProcess
    mov uret, eax
  }
}
```

3　实验分析

为了验证本文方法的有效性和高效性,本章对本文方法和其他方法进行了比较。实验选取了部分正常软件和利用不同技术来得到的恶意代码,包括灰鸽子[9]、Ntrootkit[10]、Fu[11]、AV 杀手[12]、魔兽贼[13]。实验平台如下:宿主机操作系统:Windows XP SP3,虚拟机:VMware 6.0,虚拟机操作系统:Windows XP SP3。

3.1　检测进程方法实验

将上述的 5 种恶意代码分别安装在虚拟机中使其正常运行,然后分别使用本文提出的方法(在表 1 中用 0 表示)和1.1 节中阐述的现有检查方法:系统函数枚举(在表 1 中用 1 表示),挂钩 SwapContext()(在表 1 中用 2 表示),遍历 EPROCESS(在表 1 中用 3 表示),挂钩系统调用(在表 1 中用 4 表示),遍历 PspCidTable(在表 1 中用 5 表示),遍历内存(在表 1 中用 6 表示),进行分析检测。

使用本文方法检测进程过程如图 5 所示,上述 7 种方法检测结果和检测耗时对比如表 1 所示,其中“Y”代表可检测到进程,“—”表示无法检测到进程。

表 1　检测结果对比

检查方法	恶意代码					耗时/s
	灰鸽子	Ntrootkit	Fu	AV 杀手	魔兽贼	
0	Y	Y	Y	Y	Y	实时(<0.5)
1	Y	—	—	—	—	1
2	Y	Y	Y	Y	Y	130~180
3	Y	—	—	—	—	1
4	Y	—	—	—	Y	50~90
5	Y	Y	Y	—	—	2~4
6	Y	Y	Y	Y	Y	6~10

通过实验结果可以看出,方法 1、3、4、5 在检测某些技术

《计算机应用》Journal of Computer Applications

的时候有效,但是没有普遍性,当遇到相应的反检测技术时失效。方法 2、6 虽然也能检测出所有的进程,但是和本文中方法相比速度慢了很多。本文中方法可检测出所有类型的线程进程,并且检测用时也极少。

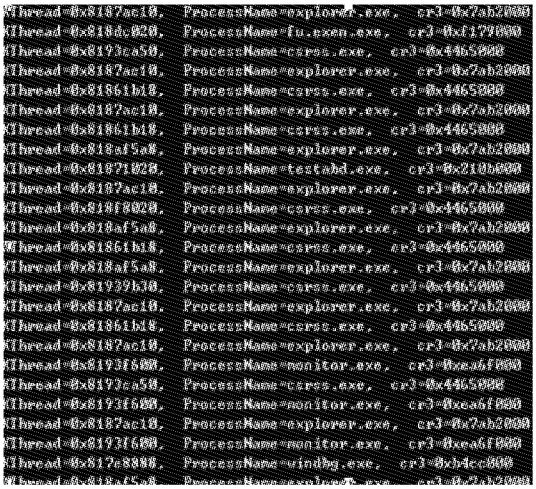

图 5 检测进程截图

3.2 结束进程方法实验

实验时首先使上述 5 种恶意代码正常运行于虚拟机中,然后分别使用本文中结束进程的方法(在表 2 中用 0 表示)和 1.2 节中阐述方法:直接调用函数(在表 2 中用 1 表示),消息机制(在表 2 中用 2 表示),创建远程线程(在表 2 中用 3 表示),破坏进程资源(在表 2 中用 4 表示),对上面所列五个恶意代码进行清除,清除结果如表 2 所示,"Y"代表可结束,"—"代表不可结束。

表 2 结束进程结果对比

检查方法	恶意代码				
	灰鸽子	Ntrootkit	Fu	AV 杀手	魔兽贼
0	Y	Y	Y	Y	Y
1	Y	—	—	—	—
2	Y	—	—	—	—
3	Y	—	Y	—	—
4	Y	—	—	Y	Y

从实验结果可知,现有方法在遇到相应的防杀技术时失去作用,而本文方法可以结束所列举的恶意进程,避免了调用 Windows 自身函数,防止了函数被挂钩时造成的不可执行,并且避免了其运行环境被结束等破坏,保证了本文方法的有效性。

4 结语

随着 Rootkit 技术的发展,恶意进程越来越难以发现,给计算机安全防护工作带来了更大的挑战。本文提出了一种基于监视 VM 内存来实时监视进程,通过对进程内存清零来清除进程的方法。最后通过实验和其他方法进行了比较,实验结果表明本文的方法主要有以下几个优点:1)目标进程运行在虚拟机中,不会对宿主机中的系统和检测程序造成破坏;2)能在第一时间检测到目标程序,具有实时性;3)没有调用系统 API,避免了系统 API 被挂钩造成结果错误。

参考文献:

[1] KIMMO K. Detecting hidden process by hooking the swapcontext function [EB/OL]. [2009 - 08 - 26]. https://www.Rootkit.com/newsread.php?newsid=170.

[2] JAMES R, BUTELER I I. Detecting compromises of core subsystems and kernel function in Windows NT/ 2000 / XP [D]. Baltimore, USA: University of Maryland, 2002.

[3] 梁晓,李毅超.基于线程调度的进程隐藏检测技术研究[J].计算机科学,2006,33(10):114 - 118.

[4] 何志,范明钰.基于 HSC 的进程隐藏检测技术[J].计算机应用,2008,28(7):1772 - 1775.

[5] 段俊锋.一种基于 PspCidTable 的进程检测方法[J].黑客防线,2007(10):107 - 109.

[6] 王晨,武东英.基于内存扫描的隐藏进程检测技术[J].计算机应用,2009,29(Z1):89 - 91.

[7] KATH R. The virtual-memory manager in Windows NT [EB/OL]. (1992 - 12 - 21) [2009 - 09 - 16]. http://msdn.microsoft.com/en - us/library/ms810616.aspx.

[8] 毛德操.Windows 内核情景分析[M].北京:电子工业出版社,2009.

[9] 葛军,黄土平.灰鸽子远程控制系列[EB/OL].(2005 - 06 - 11)[2009 - 07 - 26].http://www.huigezi.net/index.asp.

[10] HOGLUND G. Ntrootkit [EB/OL]. (2005 - 12 - 19) [2009 - 06 - 06]. http://www.rootkit.com/project.php?id=11.

[11] Fuzen_op, FU Rootkit [EB/OL]. (2007 - 11 - 23) [2009 - 07 - 01]. http://www.rootkit.com/project.php? id=12.

[12] AV 杀手利用 Windows 映像劫持技术改注册表[EB/OL].(2008 - 04 - 10) [2009 - 07 - 01]. http://tieba.baidu.com/f?kz=353608650.

[13] 于捷.谨防"魔兽贼"木马病毒[EB/OL].[2009 - 06 - 29]. http://msn.chinabyte.com.sixxs.org/a/8933647.shtml.

《计算机应用》Journal of Computer Applications

(上接第 1326 页)

[4] FORGY C L. Rete: A fast algorithm for the many pattern/many object pattern match problem [J]. Artificial Intelligence, 1982, 19(1):17 - 37.

[5] ZHOU DONGDAI, FU YIFAN, ZHONG SHAOCHUN, *et al*. The rete algorithm improvement and implementation [C]// Proceedings of the 2008 International Conference on Information Management, Innovation Management and Industrial Engineering. Washington, DC: IEEE Press, 2008: 426 - 429.

[6] NELSON M L, RARIDEN R L, SEN P. A lifecycle approach towards business rules management [C]// Proceedings of the Proceedings of the 41st Annual Hawaii International Conference on System Science. Washington, DC: IEEE Computer Society, 2008: 113.

[7] CHENG I, SRINIVASAN S, BOYETTE N. Exploiting XML technologies for intelligent document routing [C]// Proceedings of the 2005 ACM symposium on Document Engineering. New York: ACM Press, 2005: 26 - 28.

[8] 徐黎,糜宏斌,冯元勇,等.基于业务对象模型的业务规则语言的设计及实现[J].计算机应用研究,2005,22(1):36 - 37.

[9] LIU TIE, TIAN CHUNHUA, ZHANG HAO, *et al*. Learning the priority for rule execution [C]// SOLI'09: IEEE/INFORMS International Conference on Service Operations, Logistics and Infomatics. Washington, DC: IEEE Press, 2009: 565 - 572.

[10] 王海滨.基于业务规则方法的 MIS 系统研究与实现[D].天津:天津大学,2007.

[11] 缴明洋,谭庆平.Java 规则引擎技术研究[J].计算机与信息技术,2006(3):41 - 43.

[12] 张渊,夏清国.基于 Rete 算法的 Java 规则引擎[J].科学技术与工程,2006(11):1548 - 1550.