软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, 2011,22(Suppl.(2)):1−8

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

一种利用物理内存搜索硬件虚拟化 Rootkit 的检测方法*

周天阳+, 朱俊虎, 李鹤帅, 王清贤

(解放军信息工程大学 信息工程学院,河南 郑州 450002)

Approach for Hardware Virtualization-Based Rootkit Detection via Physical Memory Searching

ZHOU Tian-Yang+, ZHU Jun-Hu, LI He-Shuai, WANG Qing-Xian

(Institute of Information Engineering, PLA Information Engineering University, Zhengzhou 450002, China)

+ Corresponding author: E-mail: zhoutianyang2010@gmail.com

Zhou TY, Zhu JH, Li HS, Wang QX. Approach for hardware virtualization-based Rootkit detection via physical memory searching. *Journal of Software*, 2011,22(Suppl.(2)):1−8. http://www.jos.org.cn/1000-9825/11021.htm

Abstract: Hardware Virtualization-Based Rootkit (HVBR) is one of many new malwares appearing over the years. Compared to the traditional Rootkit, HVBR is stealthier and more difficult to detect. This paper analyzes the concealment and working mechanism of HVBR. By aiming at the stealth of HVBR on bypassing virtual memory scan to counter detection, this paper proposes a detection approach, based on physical memory search. The approach modifies Page Table Entry (PTE) to traverse the physical memory, and matches the fixed characteristic of HVBR with the raw memory data to detect and locate HVBR in memory. The experimental results show it is reliable and efficient.

Key words: hardware virtualization; HVBR (hardware virtualization-based Rootkit); physical pages search; detection

摘 要: 硬件虚拟化 Rootkit(hardware virtualization-based Rootkit,简称 HVBR)是近几年出现的新型恶意程序,相比传统 Rootkit 具有更强的隐藏性,难以被有效地检测出来.分析了 HVBR 的隐藏原理和运行机制,针对 HVBR 能够绕过直接内存扫描的隐藏特性,提出了一种基于物理内存搜索的检测方法.该方法通过修改页表项 PTE 遍历物理内存,通过比较 HVBR 的固有特征进行检测和定位.实验结果表明,该方法具有较好的可靠性和检测效率.

关键词: 硬件虚拟化;HVBR(hardware virtualization-based Rootkit);物理内存搜索;检测

借助 Rootkit 技术,恶意程序可以隐藏自身属性和攻击行为,长期潜伏在被侵入的系统中.为了消除这种潜在的系统安全隐患,安全软件根据被篡改的系统对象或行为来检测 Rootkit 的存在.Rootkit 及其检测技术是业界关注的焦点.

传统 Rootkit 一般通过 Hook[1]改变系统程序执行路径,过滤系统调用返回结果,或者采用直接操纵内核对象(direct kernel object manipulation,简称 DKOM)[2]技术篡改系统记录信息.由于传统 Rootkit 依赖于操作系统运

* 基金项目: 国家自然科学基金(60903220)
收稿时间: 2011-02-15; 定稿时间: 2011-07-28

行,无法彻底消除隐藏痕迹,许多基于操作系统的检测技术仍然可以将其检测出来,如完整性校验[3]、交叉试图比对[4]和启发式行为识别等.

为获取更高权限,Rootkit 不断向系统底层渗透,甚至独立于操作系统之外运行来躲避安全软件.近年来,随着虚拟化技术的复兴和飞速发展,出现了一类基于硬件虚拟化技术的 Rootkit(hardware virtualization-based Rootkit,简称 HVBR).不同于传统 Rootkit,HVBR 隐藏于操作系统与物理硬件之间,不依赖操作系统的内核服务、文件系统和网络协议栈就能实现其恶意功能,传统的 Rootkit 检测技术均对其无效[5,6].在虚拟化技术日益普及的今天,HVBR 给计算机安全带来极大的挑战,亟需研究相应的检测方法.现有的 HVBR 检测方法主要是通过探测虚拟化环境与真实环境之间的硬件特性差异来间接地判断 HVBR 存在的可能.但是,HVBR 利用其底层优势可以截获探测指令,从而对抗和绕过这些检测方法.同时,在桌面虚拟化技术日益普及的今天,对系统虚拟化环境的恶意性进行检测显得更为迫切.HVBR 与病毒和蠕虫不同,并不以广泛传播为目的,仅针对特定目标展开攻击,很难获取有效的代码特征和分析程序行为.

针对上述问题,本文提出修改页表项 PTE,遍历物理内存空间,搜索 HVBR 固有的数据特征来枚举虚拟机控制结构的方法,通过比较实际控制结构数目与预期数目的差异来实现对 HVBR 的有效检测.由于该方法不依赖于特定的探测指令和寄存器信息,因此 HVBR 难以抵抗这种检测.

1 HVBR 的隐藏原理

HVBR 借助虚拟机监控器(virtual machine monitor,简称 VMM)进行构建.VMM(也称 Hypervisor)是在现有系统平台上引入的一薄层管理软件,主要实现对计算机底层硬件资源的抽象、封装和隔离,为上层的 1 个或多个客户机操作系统(guest OS)提供虚拟硬件资源.VMM 运行于主机硬件之上,操作系统之下,拥有最高特权级,操作系统无法感知出自身是运行在真实的硬件之上,还是运行在系统虚拟机环境中.HVBR 充分利用 VMM 的底层优势,实现深度隐藏.

1.1 基于硬件虚拟化技术的VMM

硬件虚拟化技术是生产厂商在硬件产品中提供的虚拟化技术支持.硬件虚拟化技术引入新的 CPU 模式和虚拟化指令集,帮助 VMM 提升性能,如 Intel VT-x[7]和 AMD SVM[8]等.下面以 Intel VT-x 为例介绍基于硬件虚拟化技术的 VMM 的工作原理.

Intel VT-x 为 CPU 提供了虚拟机扩展 VMX(virtual machine extension)功能[7].在 VMX 中,新的 CPU 模式称为根(root)模式,根模式仅供 VMM 使用;而 Guest OS 运行在非根模式下.VMM 可以截获 Guest OS 对硬件的访问,实现对系统资源的有效控制.VMM 通过虚拟机控制结构(virtual-machine control structure,简称 VMCS)设置需要截获的硬件访问,如指令、中断和异常等.当 Guest OS 在执行时触发了这些条件设置,将引起 VM 退出(VM exit),此时 CPU 从非根模式切换到根模式,VMM 获得系统控制权并执行相应处理程序.当 VMM 执行结束后,通过将控制权返回给 Guest OS,这称为 VM 进入(VM entry).CPU 在虚拟化指令 VMXON,VMXOFF 的控制下开启或关闭 VMX 功能,具体过程如图 1 所示.

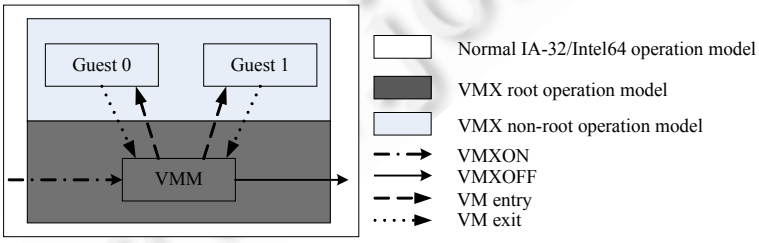

Fig.1 VMX working mechanism[7]

图 1 VMX 工作机制[7]

1.2 HVBR工作原理

HVBR 在目标操作系统运行时进行加载.首先,HVBR 获得内核权限,暂停操作系统运行,在确认系统支持虚拟化功能后,设置相关寄存器来开启 VMX 功能并进入根操作模式;然后,为 VMCS 分配内存空间并初始化,将程序计数器(PC)设为操作系统暂停时的下一条指令,为 VM 退出事件准备处理程序;最后,返回非根操作模式,操作系统从 PC 中的指令处继续执行.此时,目标操作系统已被迁移至虚拟机中,完全受 HVBR 的监控[5].

HVBR 不挂钩操作系统的内核服务,也没有修改内核数据,很少干涉目标操作系统的运行,具有很高的隐藏性.HVBR 一般基于精简监控器(thin hypervisor)模型进行设计编写,系统性能开销极小.精简监控器(thin hypervisor)并不虚拟所有底层硬件,目标系统甚至可以直接访问内存和 I/O 设备.HVBR 拦截硬件访问指令或系统底层事件,窃取 CPU 时间以执行自己的恶意操作,如构造特殊的指令序列篡改当前进程,插入中断或异常改变程序执行流程,直接操纵网卡建立隐蔽通信等.HVBR 工作原理如图 2 所示.

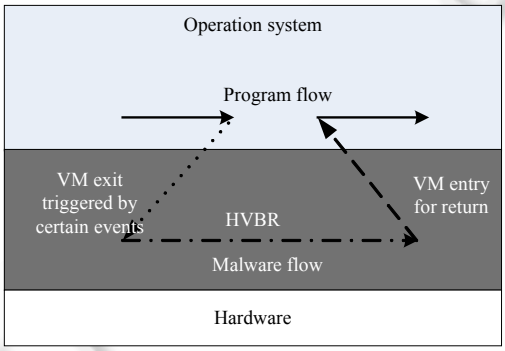

Fig.2 HVBR running principium

图 2 HVBR 工作原理

1.3 HVBR的反检测

HVBR 的底层特性使之具有天然的反检测优势.研究人员针对虚拟环境的特性提出一些 HVBR 的检测手段,例如寄存器标志位检测、TLB 检测、计时检测等,它们都是通过执行敏感指令测试虚拟硬件与物理硬件之间的特性差异,判断计算机中是否存在 VMM,从而达到检测 HVBR 的目的[9].然而,这种检测方法具有很大的局限性,一方面 HVBR 利用底层优势可以拦截到这些敏感指令,如 CPUID,RDMSR 等,并伪造指令执行结果,通过弥补差异使检测失败[10];另一方面,硬件虚拟化技术正得到广泛应用,许多商业服务器和个人计算机已部署了虚拟环境,仅通过判断 VMM 的存在性并不能有效检测出 HVBR,因为 HVBR 可以通过虚拟化嵌套的方法隐藏在一个合法使用的 VMM 中.

由于 HVBR 在主机中运行时要为自己的控制结构分配内存,如 Intel VT-x 的 VMCS,如果利用控制结构的固有特征在内存中进行搜索,不仅可以检测 VMM,而且可以具体确定出 VMM 的个数,从而判断是否存在 HVBR.但是,HVBR 在设计时都会使用内存隔离技术隐藏自己占用的内存空间.所谓内存隔离是指 HVBR 修改目标操作系统的页表,将敏感 PTE 重定向到垃圾物理页面,或者直接删除对特定物理页的映射,同时 HVBR 运行时使用自己的页表维持对内存的正确访问.在内存被隔离的情况下,操作系统内的检测程序无法通过直接遍历虚拟内存地址获取完整有效的内存信息,导致检测失效.

2 基于物理内存搜索的 HVBR 检测方法

采用内存隔离的 HVBR 与目标操作系统只是虚拟内存空间不同,两者仍使用同一个物理内存空间,同时硬件虚拟化技术要求 VMM 运行时虚拟机控制结构必须常驻内存,不可换出.如果直接扫描物理内存空间,就可以获取到完整的内存信息,通过特征匹配确定 VMM 的存在及个数,进而检测 HVBR.目前物理内存扫描主要有硬

件和软件两种方法.其中,硬件方法为利用专用的 PCI 卡 Tribble[11]或者通过 Firewire[12]端口访问物理内存.但是,Rootkit 通过操纵主板北桥上的内存控制芯片可以进行对抗[13].另外,专门的硬件要求也限制了这些方法的应用范围;软件方法主要有调用 Device\Physical Memory 内核对象[14]访问物理内存或者使用 Window Memory Dump Utilities 工具集转储 RAM[15].软件方法获取的内存映像缺乏实时性,并不能反映系统的真实状态,HVBR 可以通过"冬眠"策略躲过这种检测,另外 HVBR 还可以通过 DKOM 方法篡内核对象使得这些软件方法只能得到虚假的物理内存,使检测失效.由此可见,上述方法都无法保证能够提供实时、真实的物理内存映像.

本文利用内存地址映射机制,提出通过修改 PTE 来遍历物理页的真实内存获取方法,确保检测材料的有效性.下面以 X64 架构下的 Intel VT-x 为例阐述通过搜索物理内存中的 VMCS 来检测 HVBR 的方法.

2.1 X64架构下的物理内存扫描

X64 架构的寻址模式采用 4 级页表结构映射.虚拟内存地址为 64 位,高 16 位为保留位,实际有效位为低 48 位.分别为 9 位的第四级页映射表、9 位的页目录指针表、9 位的页目录、9 位的页表和 12 位页内偏移.四级页表的表项均为 64 位,页表基地址为 0xFFFFF68000000000,对于某个虚拟地址 *VA*,可以通过如下公式计算其对应的页表项 PTE 的地址:

$$PTEAddress = ((\mathrm{ULONG64})VA \gg 9)\ \&\ 0x7FFFFFFF8 + 0xFFFFF68000000000 \quad (1)$$

PTE 中包含了 4K 内存页的物理页号,代表实际的物理地址.通过修改 PTE 中的物理页号,就可以改变某一虚拟地址所指向的物理地址.因此,若申请一个 4K 大小的页,反复修改该页虚拟地址的 PTE,使其从指定页面直到物理地址高端,每次修改后再通过该虚拟地址进行访问,则可以实现全物理内存的扫描.

HVBR 可以采用内存虚拟化技术(如影子页表技术)来隔离物理内存,但是这需要将 HVBR 所占用的页面标记为缺页,当检测程序访问"缺页"的页面时会引发 VM Exit,HVBR 再通过处理该退出事件来过滤内存访问,才能实现内存隔离.本文方法中申请页面属于正常的系统行为,HVBR 不会干涉,在遍历物理内存页时并不会引发缺页异常,修改 PTE 也没有采用特权指令实现,即使 HVBR 拦截到了刷新 TLB 的动作,也无法据此判断上层的检测意图,因为 TLB 的刷新在系统正常运行时非常频繁.另外,HVBR 一旦引入影子页表技术会给系统带来严重的性能损耗,这反而会暴露 HVBR 的存在,因此很少采用这种方法进行内存隔离.

2.2 虚拟机控制结构的特征

虚拟机控制结构(virtual-machine control data structure,简称 VMCS)[7]是 Intel VT-x 专门为 VMX 操作定义的数据结构.它管理 VMX 操作模式之间的转换(进入退出 VMX 非根操作模式)和处理器在非根操作模式中的行为.VMM 可以为每个虚拟处理器定义一个 VMCS.

VMCS 结构由版本标识、VMX 中止标识和 VMCS 数据这 3 个基本信息域组成[7],见表 1.

Table 1 VMCS basic format

表 1 VMCS 结构的基本格式

字节偏移	内容
0	VMCS 版本标识
4	VMX 中止标识
8	VMCS 数据

VMCS 的前 4 个字节为版本标识,不同的版本 VMCS 具有不同的数据存储格式.数据区包含 6 个部分[5]:

- 客户机状态域:VM 进入时,处理器状态从该域中加载,而 VM 退出时,处理器状态保存在该域中;
- 宿主机状态域:VM 退出时,处理器状态从该域中加载;
- 虚拟机执行控制域:设置退出条件,控制处理器在 VMX 非根操作模式中的行为;
- 虚拟机退出控制域:控制虚拟机退出时的行为;
- 虚拟机进入控制域:控制虚拟机进入时的行为;
- 虚拟机退出信息域:接收虚拟机退出信息,描述虚拟机退出原因.

对于使用 Intel VT-x 技术的 HVBR,VMCS 是必备的,其数据区格式由 VMCS 版本号指定且不可更改.各数据域中的参数位置相对固定,除少数参数外,大部分参数必须填写,其中有很多参数的值是常量.例如,参数 GUEST_CR0 保存在客户机状态域中,记录了 CPU 在非根模式下 CR0 寄存器的值,CR0 寄存器用于控制处理器的模式和状态,HVBR 在初始化 VMCS 时必须填写这个参数.因此,可以选择 GUEST_CR0 作为特征值来标识 VMCS.

2.3 VMCS的搜索

VMCS 总存在于物理内存页中,对物理内存进行扫描并将原始内存数据与 GUEST_CR0 的值进行匹配可以实现 VMCS 的搜索.Intel VT-x 规定 VMM 必须在物理内存的页边界(4KB 对齐)创建 VMCS,HVBR 按各参数在 VMCS 中的偏移位置填写参数值.为了提高内存搜索的效率,本文计算 GUEST_CR0 在页内的相对偏移,利用修改 PTE 的方式获取 4K 大小的物理页,直接将该页内相应偏移处的数据与 GUEST_CR0 的值进行比较,从而判断 VMCS 的存在.

首先,确定 GUEST_CR0 的偏移.Intel 并未公布 VMCS 各参数的具体偏移,而是均以编号代替,HVBR 的编写者使用这些编号填写 VMCS 的参数.这是因为不同版本的 VMCS 数据格式不同,参数位置无法固定.Intel 提供了 VMWRITE 指令用于写 VMCS,因此可以先向 VMCS 的 GUEST_CR0 中写入一个特征码,再扫描 VMCS 所在页,通过特征码匹配确定 GUEST_CR0 的页内偏移,具体过程如下:

- 向 GUEST_CR0 中写入特征码 0xabcdabcdabcdabcd,语句如下:

VmxWrite(0x00006800,0x0xabcdabcdabcdabcd);

其中,0x00006800 为 GUEST_CR0 在 VMCS 中的编号.

- 扫描 VMCS 所在页,根据特征码进行匹配,得到 GUEST_CR0 偏移 Offset,关键代码如下:

```
for (Scan=Page, i=0; i<512; i++, Scan++)
{
 if (*Scan==0x0xabcdabcdabcdabcd)
 {
  Offest=(((ULONG64)Scan) & 0xfff);
  KdPrint((“GUEST_CR0 Offest Is 0X%X\n”,Offest));
  break;
 }
 else
 continue;
}
```

其次,获取 GUEST_CR0 的值.目标操作系统运行时 GUEST_CR0 与当前 CR0 寄存器的值是一致的,可通过读取系统 CR0 寄存器得到 GUEST_CR0 的值,语句如下:

```
CR0=RegGetCr0();
```

在获得 GUEST_CR0 偏移和取值后,即可进行物理内存扫描,步骤如下:

- 申请一个 4K 大小的不可换出页;
- 计算出该页的 PTE;
- 修改 PTE,使其指向指定物理内存页;
- 访问其 PTE 所指向的页面,并读取该页指定偏移处的数据 Data,与 CR0 进行匹配,若匹配成功,则该页可能存在 VMCS.

需要说明的是,每次修改完 PTE 所指的物理地址后,都要刷新该页所对应的 TLB 项,否则计算机硬件将直接通过 TLB 寻址而使修改动作无效.

由于内存中存在大量随机数据,为防止检测误报,还可以通过匹配 VMCS 的其他特征,提高确检测的可靠

性.这可以通过增加比对 VMCS 的版本标识来实现.VMCS 的版本标识位于 VMCS 的开始位置,其值可以通过查询 MSR 寄存器 IA32_VMX_BASIC 获得.当一个页面的数据与 CR0 匹配成功后,再用版本标识匹配该页面 0 偏移处的数据,如果匹配成功,则确定该页存在 VMCS,记录该页的物理地址,并增加 VMCS 的计数.

物理内存搜索的关键代码如下所示:

```
Page=MmAllocateContiguousMemorySpecifyCache(PAGE_SIZE,l1,l2,l3,MmNonCached);
PTE=(PULONG64) ((((ULONG64)Page>>9) & 0x7ffffffff8)+PT_BASE);
*PTE &=0xfff0000000000fff;
//读取 VMCS 版本标识
RevisionID=MsrRead(MSR_IA32_VMX_BASIC) & 0xffffffff
for (i=0; i<N; i++)
{
//读取页面指定偏移处的数据
 Data=*(PULONG64)((ULONG64)Page+GUEST_CR0_Offset);
//特征匹配
if (Data==CR0 && *Page==RevisionID)
   {
    //获取 Page 的物理地址
    PA=MmGetPhysicalAddress (Page);
    KdPrint((“Blue Pill Vmcs’s PA Is 0x%X\n”,PA));
    VMCSCount++;
   }
    //刷新 TLB 项
    CmInvalidatePage(Page);
    *PTE+=PAGE_SIZE;
}
```

为了减少内存遍历的时间,提高检测速度,本文仅扫描特定地址范围的物理内存,而不是扫描全部物理内存.在实际环境中,HVBR 初始化时将自身加载到系统的内核区域(如 Blue Pill 以驱动的方式加载进内核),使用非换页内存存储数据.在 X86 系统中,系统代码、系统加载的驱动程序位于 0x80000000 至 0x9FFFFFFF 中.这个区域的虚拟地址直接减去 0x80000000 就可以得到对应的物理地址,进而缩小需要扫描的物理内存范围;在 X64 系统中,虚拟地址和物理地址并不存在特定的转换关系,内核地址从 0xFFFFF800’00000000 开始,非换页内存池为 0xFFFFFAC0’0000000 至 0xFFFFFADF’FFFFFFFF 结束.由于这个区域的内存是连续分配的,可以通过查找系统页表来定位这些特定区域的起始物理页,然后在按页帧号增长的顺序进行扫描.通过上述方法,可以减少需要搜索的物理内存区域,从而提高检测效率.

3 实验结果

本文实验用的硬件环境是搭载 Intel Core 2 Duo E7500 处理器的平台,操作系统为 64 位的 Windows XP Professional x64 Edition (Build 3790 ServicePack1),测试样本选为 Blue Pill[5].Blue Pill 是 2006 年 Black Hat 会议上由著名计算机安全专家 Rutkowska 发布的 HVBR 原型程序,可以运行在 Intel VT-x 平台上,是一种有代表性的 HVBR.

安装 Blue Pill 时,选择与目标操作系统分用不同页表.测试时,使用传统的虚拟内存搜索方法与本文提出的物理内存搜索方法分别进行检测,结果如图 3、图 4 所示.

图 3 和图 4 显示的是检测驱动程序运行后,DbgView 捕获的系统打印信息.图 3 显示虚拟内存搜索没有检

测出 Blue Pill,这是因为 Blue Pill 在操作系统页表中清除了 VMCS 所在页的 PTE 项,从而绕开了虚拟内存扫描.图 4 中显示,不仅检测出系统中有 VMCS,而且检测到两个 VMCS,这是由于实验用计算机配备的是双核 CPU,Blue Pill 攻击了两个 CPU 内核,因而内存中存在两个 VMCS.同时,图 4 中的 Time 区域显示检测所耗的时间仅为毫秒级.

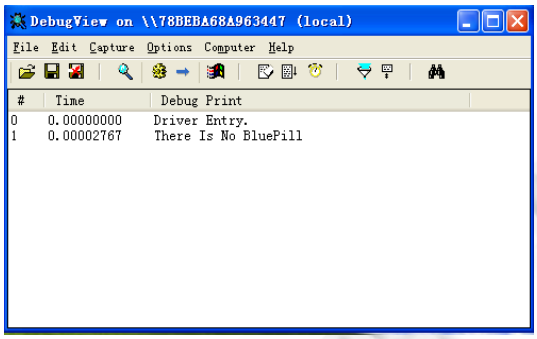

Fig.3 Detection result via virtual memory searching

图 3 基于虚拟内存搜索的检测结果

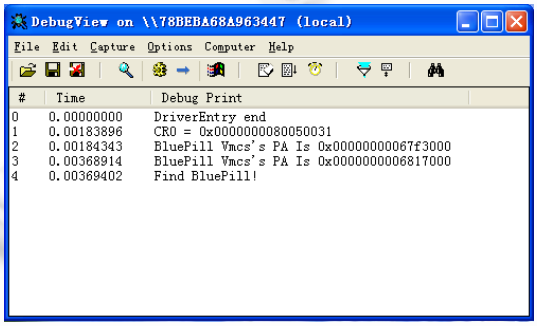

Fig.4 Detection result via physical memory searching

图 4 基于物理内存搜索的检测结果

重复实验多次,结果相同且没有出现误警.由此证明了基于物理内存搜索的检测方法对 HVBR 的可靠性和有效性,并且具有较好的检测效率.

4 结 语

基于物理内存搜索的检测是一种针对现有 HVBR 隐藏技术行之有效的检测方法.与基于差异化的检测和基于虚拟内存搜索的检测相比,该方法实现了对物理内存页的完全扫描,能够获取真实的系统内存,同时利用 HVBR 固有的数据特征进行匹配搜索,可以得到可信的检测结果,而这是传统 HVBR 检测方式所做不到的.

未来,HVBR 在隔离物理内存时可能借助于 Intel 的扩展页表技术(EPT),但是该技术需要专门的硬件支持,仅针对特定类型的 CPU,同时开启 EPT 需要设置 MSR 寄存器,而且会在内存指定位置留下使用痕迹,探测这些信息可以获知 EPT 是否被滥用,这是下一步要研究的重点.

References:

[1] 韩小明.Hook 启思录. 2006. http://blog.csdn.net/xiam-my/category/259006.aspx

[2] Nguyen AQ, Yoshiyasu T. Towards a tamper-resistant kernel Rootkit detector. In: Proc. of the 2007 ACM Symp. on Applied Computing. New York: ACM, 2007. 276−283. [doi:10.1145/1244002.1244070]

[3] 吴坤鸿,乐宏彦.反 Rootkit 的内核完整性检测与恢复技术.计算机工程,2008,34(21):129−131.

[4] 白光冬,郭耀,陈向群.一种基于交叉视图的 Windows Rootkit 检测方法.计算机科学,2009,36(8):133−137.

[5] Rutkowska J, Tereshkin A. Subverting Vista kernel for fun and profit. In: Proc. of the Black Hat Conf. Vegas, 2006.

[6] Zovi D. Hardware virtualization Rootkits. In: Proc. of the Black Hat Conf. Vegas, 2006.

[7] Intel. Intel®64 and IA-32 Architecture Software Developer's Manual, Volume3B: System Programming Guide, Part2. 2007. http://download.intel.com/design/processor/manuals/253669.pdf

[8] AMD. AMD64 Architecture Programmer's Manual Volume 2: System Programming. 2007. http://support.amd.com/us/Processor_TechDocs/24593.pdf

[9] Ptacek T, Lawson N, Ferrie P. Don't tell Joanna, the virtualized Rootkit is dead. In: Proc. of the Black Hat Conf. Vegas, 2007.

[10] Rutkowska J. IsGameOver(), anyone? In: Proc. of the Black Hat Conf. Vegas, 2007.

[11] Grand J, Carrier B. Method and apparatus for preserving computer memory using expansion card. United States Patent, 7181560, 2007.

[12] Becher M, Dornsif M, Klein CN. FireWire: All your memory are belong to us. In: Proc. of the CanSecWest Conf. Vancouver, 2005.

[13] Rutkowska J. Beyond the CPU: Defeating hardware based RAM acquisition. In: Proc. of the Black Hat Conf. Arlington, 2007.

[14] Waits C, Akinyele JA, Nolan R. Computer forensics: Results of live response inquiry vs. memory image analysis. 2008. http://handle.dtic.mil/100.2/ADA488423

[15] Schuster A. Searching for process and threads in Microsoft Windows memory dumps. Digital Investigation, 2006,3(1):10−16. [doi:10.1016/j.diin.2006.06.010]

周天阳(1979－),男,河南郑州人,硕士生,主要研究领域为网络与信息研究,虚拟化,安全应用.

李鹤帅(1987－),男,硕士生,主要研究领域为网络与信息研究.

朱俊虎(1974－),男,副教授,主要研究领域为网络与信息安全.

王清贤(1960－),男,教授,博士生导师,主要研究领域为信息安全,算法分析.