文章编号:1009-3087(2014)01-0008-06

一种采用硬件虚拟化的内核数据主动保护方法

傅建明[1,2,3],沙乐天[1,2*],李鹏伟[1,2],彭国军[1,2]

(1. 武汉大学 空天信息安全与可信计算教育部重点实验室,湖北 武汉 430072;2. 武汉大学 计算机学院,湖北 武汉 430072;
3. 武汉大学 软件工程国家重点实验室,湖北 武汉 430072)

摘 要:为保护操作系统内核的完整性,提出了一种基于硬件虚拟化技术的保护方案。该方法对关键寄存器、代码指针表、函数代码等恶意代码攻击的关键点进行识别和放入保护区,利用硬件虚拟化的自动陷入机制检测对保护区的非法篡改。同时,利用单步执行技术和事件转发技术保障OS其它操作的兼容性。另外,通过保护页的合并减少保护区的长度以提高异常处理的效率。最后,实现了一个采用该技术的原型工具——HV_KDAP,该工具检测了主流的9款Rootkit样本,实验结果证实其增加的负载为12.7%。该工具还可以抑制内核本地权限提升的攻击,以及用于内核攻击的取证。

关键词:硬件虚拟化;内核数据完整性;Rootkit检测

中图分类号:TP31 **文献标志码**:A

An Active Protection of Kernel Data Using Hardware-assisted Virtualization

FU Jian-ming[1,2,3], *SHA Le-tian*[1,2*], *LI Peng-wei*[1,2], *PENG Guo-jun*[1,2]

(1. Key Lab. of Aerospace Info. Security and Trusted Computing of Ministry of Education, Wuhan Univ., Wuhan 430072, China;
2. School of Computer, Wuhan Univ., Wuhan 430072, China; 3. State Key Lab. of Software Eng., Wuhan Univ., Wuhan 430072, China)

Abstract: In order to protect the integrity of operating system kernel files, a method of active protection of kernel data was proposed based on hardware-assisted virtualization. The method recognizes the key points of some registers, code pointers, and function codes, which are often attacked by malicious codes, and maps these points into a protection table, and then it can avoid kernel modification through R/W bit of PTE. At the same time, single step execution is used to legally write data in protected points, and events injection keeps the compatibility of operation system. In addition, continuous pages in the protection table are merged to reduce the size of the protection table and improve the efficiency. Finally, based on this method, a prototype system, called HV_KDAP, was designed and implemented. HV_KDAP can detect 9 kinds of Rootkits, which contain popular techniques in Rootkit, and its overhead is about 12.7%. Moreover, HV_KDAP can also detect the attacking of local privilege escalation exploiting, and be applied to the kernel forensics.

Key words: hardware virtualization; kernel data integrity; Rootkit detection

操作系统内核是操作系统的核心,实现了应用程序访问系统资源的接口及其自身的管理功能。多数操作系统利用了Ring0和Ring3的协调机制来运行程序。具体来说,应用程序处于用户层,而内核则处于内存中的内核地址空间,内核层的代码可以访问全部内存空间并执行特权指令,用户层的代码则受到限制。然而内核是可以扩展的,应用程序通过加载驱动可以将新的代码加入内核层,从而为恶意代码攻击内核提供了方便。针对这一情况,目前的内核防护技术都建立在虚拟机上,其开销较大,而一般的终端主机没有安装其对应的虚拟机,从而影响其保护范围。针对这一问题,作者提出了一种采用硬件虚拟化的内核数据主动保护方法,把内核代码和数据汇聚到保护区,这样只需监控保护区,不需要逐个检查攻击点;该方法通过对Hook data和Hook code的保护,可以检测深度内联hook[1];该方法独立于已有的虚拟机平台,如Xen[2]或者BitVisor[3],直接利用硬件虚拟化技术,降低检测器的开销和对终端主机的要求,同时兼容OS对保护区部分数据

收稿日期:2013-06-19

基金项目:国家自然科学基金资助项目(61202387;90718005);高等学校博士学科点专项科研基金资助项目(20120141110002)

作者简介:傅建明(1969—),男,教授,博士. 研究方向:软件安全. E-mail:jmfu@whu.edu.cn

*通信联系人 E-mail:ltsha@whu.edu.cn

的合理修改,并转发 OS 的其他页面异常,从而保障了与 OS 的兼容性。

内核防御主要分成 2 大类,被动防御及主动防御,实现方式和防御功效各有不同。内核的被动防御[4-7]主要是利用已有的内核模式库,检测当前内核的内存映像是否被修改过,并给出修改的详细记录。内核的主动防御是指假设内核是安全的前提下,检测可能存在攻击的方法。该方法大都借用虚拟机实现攻击检测,Livewire[8]采用虚拟机检测异常,给出了修补在 VMM 和客户机 OS 之间语义鸿沟的方法。SecVisor[9]设计了一个微小的 hypervisor,利用 W ⊕X 和 SPT/NPT 机制保障代码的完整性,提供了对模块的加载和卸载的处理。

而硬件虚拟化方法具体是在硬件层面及 CPU 内部对虚拟技术提供直接支持,并通过这种设计提高虚拟效率、降低其开发难度。Pcanel/V2[10]利用 Intel VT 技术设计并实现了 VMM 架构,简化了传统的 VMM 设计复杂度并大大提高总体运行功效。文献[11]及文献[12]均基于虚拟化技术设计隔离执行模型,前者基于 AMD 的 SVM 硬件虚拟化设计了一种单向隔离执行环境,实现了轻量级的可信计算基,后者则基于本地虚拟化技术在 PC 平台下提供操作系统隔离及计算环境的重现,兼顾系统安全性及代码可用性。

本研究有别于已有的研究。主要利用已有针对内核的攻击知识,形成内核保护区,该保护区能覆盖常见的内核不变量,而不需要内核恶意代码的语义识别(特征码),从而可以避免被动防御中的误[7]。其次,其实现方式独立于已有的开源虚拟机,如 XEN 和 BitVisor,减小了检测器的开销,直接利用硬件虚拟化技术实现内核的完整性保护,可以扩大该技术的应用范围。最后,该技术不仅提供检测服务,而且支持响应服务,直接拒绝对内核的非法修改。

1　保护对象和保护原理

1.1　保护对象

全文内核的代码和数据都很多,保护这些对象是非常复杂的任务。通过对现有的 Rootkit[13]技术进行分析,获得内核的保护对象列表。这些对象包括关键的寄存器、函数表和函数代码等。另外也可根据未知恶意代码或病毒文件的攻击知识获取待保护的目标对象进而扩展对象列表,从而实现内核数据攻击的扩展防御。

CR0,控制寄存器,其中的 WP 位是操作系统的写保护开关。对于进入内核操作的 Rootkit 来说,具有写内核空间的特权是实现其它操作的前提。因此将 CR0. WP 加入保护列表。

SYSENTER_EIP,Windows 使用该寄存器中的地址定位内核空间入口[14]。Rootkit 可以窜改 SYSENTER_EIP 中的地址,将其指向自己的服务调度函数,因此必须监控 SYSE NTER_EIP 的内容不被修改,将其加入列表。

IDTR,中断描述符表寄存器,存放中断表的基地址。Rootkit 可以伪造 IDT 表,并将 IDTR 指向伪造表,就可以接管操作系统的中断处理流程[15]。因此将 IDTR 也加入保护列表。

除了以上关键寄存器,Rootkit 还可以通过修改内存数据和函数代码达到潜行和破坏的目的,如 SSDT Hook、IDT Hook、TDI Hook 等[16]。同样对相关内存数据进行监控防护,并可以根据具体实际情况添加其他保护对象。另外,为了防止深度内联 Hook,以及文件过滤驱动和网络过滤驱动,NTFS 和 TCP/IP 的 IRP 和相应的函数体也加入保护对象列表中,这在系统防护中通常被忽视[17]。具体保护对象如表 1 所示。

表 1　保护对象列表

Tab. 1　List of protected objects

名称	说明	攻击
CR0. WP	写保护开关	禁止写保护
SYSENTER_EIP	系统服务调度入口	截持调用入口处
IDTR	存有 IDT 表基址	IDT Hook
SDT	系统描述表	SSDT Hook
SSDT	系统服务派发表	Table Hook
IDT	中断服务派发表	Table Hook
TCPIP IRP	网络驱动派发表	Table Hook
NTFS IRP	文件系统驱动派发表	Table Hook
SSDT CODE	系统服务函数	深度内联 Hook
TCPIP CODE	网络驱动函数	深度内联 Hook
NTFS CODE	文件驱动函数	深度内联 Hook

1.2　保护原理

图 1 展示了 Intel VT 技术的基本原理。虚拟机管理器 VMM 运行在 VMX root 模式,客户机系统 Guest OS 运行在 VMX non-root 模式。由 VMX root 转入 VMX non-root 称为 VMX Entry,反向称为 VMX Exit。2 种模式均具有独立的处理器硬件资源,VMX 引入了虚拟机控制域 VMCS 来维护这些独立资源,并控制 VMX non-root 模式下的处理器行为。

对硬件虚拟化的应用主要是指将正在运行的操作系统切换为 VMX non-root 模式,以监控操作系统的运行。监控框架的关键在于设置 VMCS。VMCS

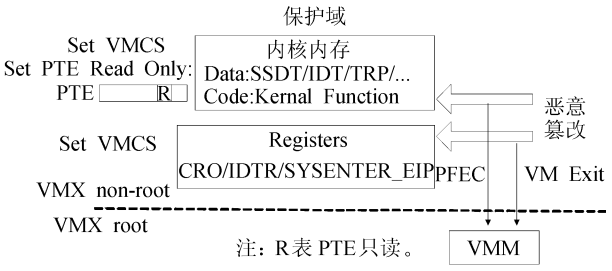

图1 内核保护的基本原理

Fig. 1 Fundamental of kernel protection

包含若干重要的数据域,主要用来定义在何种情况下完成控制权的切换,具体过程见2.1节。

2 系统实现

利用上面提出的内核数据的保护原理,设计和实现了一个采用硬件虚拟化的内核数据主动保护原型系统:HV_KDAP。HV_KDAP旨在保护内核数据、代码和相关寄存器,防止恶意代码对其进行窜改,并兼容操作系统其他功能的正常运行。HV_KDAP包括4个模块:AddrFounder用于初始化地址表AddrTable;Configurer用于初始化内存对象的PTE属性及VMCS设置;VMCheck用于决定如何处理,包括转发正常操作及调用VMProtect执行保护。

如图2所示,在系统初始化阶段,由AddrFounder获得内存对象的地址段,并保存入地址表AddrTable;而后Configurer组件设置VMCS以配置Intel VT监控框架,同时设置内存对象的PTE属性。

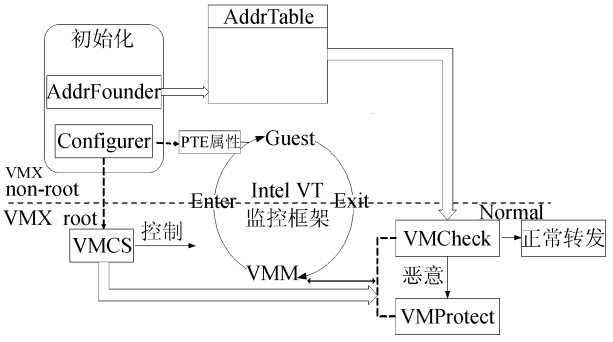

图2 HV_KDAP架构

Fig. 2 Framework of HV_KDAP

系统运行过程中,VMCheck和VMProtect对保护对象协同提供保护。VMCheck处理VM Exit事件,转发正常操作,对恶意操作调用VMProtect模块。VMProtect检查操作所属模块以及所改写的内核对象,向用户报告窜改操作并将恶意模块卸载恢复系统运行。

2.1 VMCS的设置

Configurer组件中VMCS的设置是关键,它决定各种监控对象的陷入。利用Intel VT框架监控内核对象的修改主要是设置VMCS的控制域VM execution control,其中内存对象的监控还涉及到PTE属性设置、处理器单步执行及单步调试异常捕获。

1)寄存器对象

VM-execution control数据域提供2个相关控制位对CR0寄存器的访问:CR0 Guest/Host Mask与CR0 Read Shadow。为监控对CR0.WP的修改,将CR0 Guest/Host Mask和Read Shadow都置为0X00010000。SYSENTER_EIP属于模式指定寄存器。VM-execution control的控制位Use MSR Bitmaps决定是否使用位图(MSR Bitmaps)监控模式指定寄存器的访问,为0则所有的模式指定寄存器访问都引发VM Exit。本研究决定监控所有模式指定寄存器的访问,在VM Exit处理例程中筛选出修改SYSENTER_EIP操作。

另外,IDTR、GDTR、LDTR和TR都是描述符寄存器,VM-execution control的控制位"descriptor table exit"决定了访问这些寄存器是否引发VM Exit。若置为1,则Guest OS中的相关处理例程将引发VM Exit。寄存器对象的监控设置见表2,包括CR0、SYSENTER_EIP和IDTR、GDTR、LDTR等相关寄存器的设置以及对应的退出原因标记:Basic Exit Reason。退出原因标记主要用于记录退出操作语义。

表2 VMCS设置中的寄存器对象

Tab. 2 VMCS configuration of register objects

防护对象	VMCS设置	Basic Exit Reason
CR0.WP	guest/host mask:0x00010000 read shadow:0x00010000	28
SYSENTER_EIP	VM-Execution Controls[28]:1	32
IDTR	VM-Execution Controls[2]:1 VM-Execution Controls[31]:1	46

2)内存对象

设置监控内存对象包括内存对象PTE属性的设置和VMCS设置。改写PTE的R/W位即可将对应页面设为只读属性。而VM-execution的子域"Exception"Bitmaps设置对Guest OS中异常事件的监控,对应位置1的异常将引发VM Exit。VMCS还提供了PFEC_MASK和PFEC_MATCH用于限定页面异常监控的范围,是否引发VM Exit将由PFEC、PFEC_MASK和PFEC_MATCH共同决定,具体相关VMCS设置见表3。

表 3 内存对象相关 VMCS 设置

Tab. 3　VMCS configuration of memory objects

捕获应用	VMCS 设置	Exit Reason
Page Fault	Exception[14]:1	Fault Number:14
	PFEC_MASK:0x3	
	PFEC_MATCH:0x3	
Step Fault	Exception Bitmaps[3]:1	Fault Number:3

2.2　VM Exit 处理

监测到内核对象修改后,控制流程转入 VMM 提供的 VM Exit 处理例程,Exit 处理例程负责阻止修改操作,以及对正常操作情况进行转发。保护对象中定义了保护类型(MainType)和子类型(SubType)。MianType 有 2 种类型:PROTECTED_OBJ 和 PTECONFLICT_OBJ。PTECONFLIC T_OBJ 是指与保护对象处于同一页面,但并非禁止修改的区域。而子类型标识用于说明该保护对象的具体类型,如 IDT、SSDT 或 SSDT_CODE 等。具体的处理过程见 PVE(Procedure of VM Exit)。

首先考虑对寄存器对象修改的情况以及没有正常转发的情况都需要处理,通过虚拟机自省获得操作所属模块,将其卸载,并恢复 Guest OS 执行,参见 PVE 中 3.1 ~ 3.3 行。

页面修改异常发生时,需要考虑 3 种情况:阻止内存对象修改;处理 PTE 冲突区写入操作;转发内核层用户页面写入异常操作。

第 1 种情况处理方式与寄存器对象相同,参见 PVE 的第 6 行以及 6.1 ~ 6.3 行。

第 2 种需要借助处理器单步执行转发,首先重新设置该页的 PTE 为读写属性,而后设置处理器进入单步执行状态。由于页面属性重设为读写,写 PTE 冲突区的指令得以正常执行,参见 PVE 第 7 行以及 7.1 ~ 7.3 行。

第 3 种则利用 Intel VT 的 event injection 机制转发。当异常发生而引发 VM Exit 时,VMM 可以选择不对其进行处理,而是将异常信息转发给 Guest OS,VM-entry control 和 VM-exit control 的相关域用于进行设置。参见 PVE 的第 8 行以及 8.1 ~ 8.2 行。

若单步异常发生,首先判断是否为 HV_KDAP 开启的单步,则重新设置地址所属页面属性为只读,关闭单步执行,并恢复 Guest 运行,参见 PVE 的第 11 行;若为调试器等其它程序开启单步,则直接转发该操作,参见 PVE 的第 10 到 12 行以及 12.1 ~ 12.2 行。

```
PVE:Procedure of VM Exit
Switch(Exit Reason) {
1 Case (CR0.WP):
2 Case (SYSENTER_EIP):
3 Case (IDTR):
  /* 阻止修改操作 */
3.1 MoudleName = GetMoudle(GuestEIP);
    /* 获取引发 VM Exit 指令所属模块 */
3.2 Uninstall(MoudleName);
    /* 卸载对应的模块 */
3.3 ResumeGuestOS();
4 Case (Page Fault):
5 Switch (item.Maintype) {
6 Case PROTECTED_OBJ:
6.1     Warn(item.SubType);
        /* 根据子类型发出具体报警 */
6.2     MoudleName = GetMoudle(GuestEIP);
        /* 获取引发 VM Exit 指令所属模块 */
6.3     Uninstall(MoudleName);
        /* 卸载对应的模块 */
7 Case PTECONFLICT_OBJ:
7.1     SetPTE(g_addr,READ|WRITE);
        /* 重设 PTE 为读写属性 */
7.2     SetLable();
        /* 设置标志,HV_KDAP 开启单步 */
7.3     StartStep();
        /* 开启单步执行 */
8 Case USER_PAGE:
  /* 当 GetType 查询失败时设置该标记 */
8.1     CopyFrom(VM_Entry,VM_Exit);
        /* event injection */
8.2     GuestEIP = GetNext(GuestEIP,Length);
        /* 执行下一条指令 */
9     }
10 Case (Step Fault):
11      if (CheckLable())
        /* 判断由 HV_KDAP 开启单步 */
        {
11.1      SetPTE(g_addr,READONLY);
          /* 设置 PTE 只读 */
11.2      ShutDownStep();
          /* 关闭单步执行 */
        }
12      else
```

```
        }
        /* 调试器等其它程序 */
12.1    CopyFrom(VM_Entry,VM_Exit);
        /* event injection */
12.2    GuestEIP = GetNext(GuestEIP,Length);
        /* 执行下一条指令 */
        }
    }
```

3 实验与验证

为了验证该内核保护方法,共收集了9个Rootkit样本,见表4,其中包含各种恶意功能:IDT Hook、TCP/IP Hook、SSDT Hook、SYSENTER_EIP Hook、内核数据结构修改以及深度内联Hook。并作了全面的测试。测试环境为干净安装的Windows XP SP3系统,处理器Intel Core,内存为2 GB。首先开启原型系统,尝试在系统中操作常规应用软件,这些软件运行结果正常,表明HV_KDAP可以兼容应用软件的正常运行。查看HV_KDAP系统的输出记录,发现HV_KDAP对2类操作进行了转发,即正常访问PTE冲突区的操作以及DDK编译时产生的内核修改操作。

表4 测试样本列表

Tab.4 List of tested samples

样本名称	功能描述
Strace.sys	IDT Hook
Irphook.sys	TCPIP table Hook
TCPIRPhook.sys	TCPIP table Hook
Bytehook	SSDT table Hook
RegKeyHide	SSDT table Hook
ProcessHide	修改内核数据结构 隐藏进程
PortHide	修改内核数据结构 隐藏端口
Sysenter.sys	SYSENTER_EIP: 调用入口劫持
Migbot.sys	深度内联hook

而后测试HV_KDAP的保护功能。测试过程分为2个步骤:第一,关闭原型系统,测试样本对系统的影响。在样本运行前后对比2次系统运行快照发现内核多处位置被修改。第二,将系统还原至测试前状态,打开原型系统,再次进行测试,对比快照发现内核是完整的,同时HV_KDAP的日志记录也显示了拦截的内核修改操作。因此,测试的结果表明HV_KDAP确实可以检测并阻止这些样本对内核的攻击。

最后,测试HV_KDAP在性能上的损耗。通过编写程序测试其在不同环境下的运行时间(单位秒)。经过5次测试,统计对比发现原型系统的性能损耗为15.7%,表5显示了测试结果,限于篇幅原因省略其中测试2~4的具体数据,给出平均系统开销时间、对应的性能比、以及相对大页面的性能损耗。HV_KDAP的输出记录显示,只有极少操作(如DDK编译)会引起写用户区页面异常,其它转发操作均是对PTE冲突区的处理。

表5 HV_KDAP的性能测试

Tab.5 Performance test of HV_KDAP

属性	时间消耗/s				性能比/%	损耗/%
	测试1	…	测试5	平均		
大页面	118.6	…	120.1	119.2	100	0
小页面	122.8	…	122.7	122.6	102.9	-2.9
KDAP	138.5	…	137.3	137.7	115.7	-15.7

另外,由于HV_KDAP基于页面异常对内存对象进行监控,涉及到页面大小的选择。页面尺寸有2种:小和大,大页面的优势在于地址转译速度快,然而副作用是降低了页面保护的粒度。观察发现,对于Windows XP系统,采用大页面映射时,SSDT这样的重要结构都是可读写的,恶意代码可以肆意窜改。而且当采用大页面时,修改PTE属性所造成的PTE冲突区的面积大大增加,转发这些区域上的正常操作的损耗是巨大的。所以无论出于系统本身安全的考虑,还是原型系统性能的考虑,应该选择小页面。小页面设置带来的开销仅比大页面多为2.9%,而由HV_KDAP直接带来的性能损耗为12.7%。

4 结 论

Rootkit的出现给用户的安全及隐私带来了极大的威胁,从内核的Rootkit需要修改内核代码或内核数据这个事实出发,提出了基于硬件虚拟化的内核数据主动保护方法。该方法从已有的攻击威胁和OS内核自身的保护规律构建内核数据的保护列表,利用硬件虚拟化的自动陷入机制主动检测各种对内核的非法修改,可以积极有效的检测并阻止窜改内核的行为。另外,利用系统提供的单步执行机制和事件转发机制,保障了该方法与OS其他操作的兼容性。最后,在Windows XP环境下实现了一个原型保护工具HV_KDAP。通过对该工具的测试,其实验结果表明该方法具有很好的防护功能,且性能

损耗比较小。

HV_KDAP 的功能可以进一步扩展,支持安全软件和系统补丁对内核的修改,可以引入用户交互和白名单机制实现内核的合法修改。另外,内核的 Hook 点可能分散在 Heap 中,如何把这种堆中不变的 Hook 点加以保护也是下一步的研究工作。

参考文献:

[1]Gong guang, Li Zhoujun, Hu Zhaojian, et al. Research on stealth technology of Windows kernel-level rootkits[J]. Computer Science, 2010, 37(4): 59 – 62. [龚广, 李舟军, 忽朝俭, 等. Windows 内核级 Rootkits. 隐藏技术的研究[J]. 计算机科学, 2010, 37(4): 59 – 62.]

[2]石磊, 金海, 邹德清. Xen 虚拟化技术[M]. 武汉: 华中科技大学出版社, 2009: 300 – 323.

[3]Geeknet. BitVisor[EB/OL]. (2012 – 12 – 12)[2013 – 01 – 18]. http://sourceforge. net/projects/bitvisor/.

[4]Wu Kunhong, Le Hongyan. Anti-rootkit technology of kernel integrity detection and Restoration[J]. Computer Engineering, 2008, 34(21): 129 – 131. [吴坤鸿, 乐宏彦. 反 rootkit 的内核完整性检测与恢复技术[J]. 计算机工程. 2008, 34(21): 129 – 131.]

[5]Jamie B, Greg H. VICE—Catch the hookers! [EB/OL]. (2004)[2012 – 12 – 09]. http://blackhat. com/html/bh-usa-04/bh-usa-04-speakers. html.

[6]PJF. IceSword[EB/OL]. [2012 – 08 – 21]. http://www. majorgeeks. com/files/details/icesword. html.

[7]Rutkowska J. System virginity verifier[EB/OL]. [2012 – 09 – 04]. http://www. invisiblethings. org/papers/hitb05_virginity_verifier. ppt.

[8]Garfinkel T, Rosenblum M. A virtual machine introspection based architecture for intrusion detection[C]//Proceeding of the Network and Distributed Systems Security Symposium. New York: IEEE, 2003: 174 – 190.

[9]Seshadri A, Luk M, Qu Ning, et al. SecVisor: A tiny hypervisor to provide lifetime kernel code integrity for commodity OSes[C]//Proceeding of the ACM Symposium on Operating Systems Principles. New York: ACM, 2007: 335 – 350.

[10]Chen Wenzhi, Yao Yuan, Yang Jianhua, et al. Pcanel/V2: A VMM architecture based on Intel VT-x[J]. Chinese Journal of Computers, 2009, 32(7): 1312 – 1320. [陈文智, 姚远, 杨建华, 等. Pcanel/V2——基于 Intel VT-x 的 VMM 架构[J]. 计算机学报, 2009, 32(7): 1312 – 1320.]

[11]Wen Yan, Wang Huaimin. A isolated execution model based on local virtualization technology[J]. Chinese Journal of Computers, 2008, 32(10): 1768 – 1779. [温研, 王怀民. 基于本地虚拟化技术的隔离执行模型研究[J]. 计算机学报, 2008, 31(10): 1768 – 1779.]

[12]Li Xiaoqing, Zhao Xiaodong, Zeng Qingkai. One-way isolation execution model based on hardware virtualization[J]. Journal of Software, 2012, 23(8): 2207 – 2222. [李小庆, 赵晓东, 曾庆凯. 基于硬件虚拟化的单向隔离执行模型[J]. 软件学报, 2012, 23(8): 2207 – 2222.]

[13]Wikimedia Fundation. Rootkit[EB/OL]. [2012 – 10 – 27]. http://en. wikipedia. org/wiki/Rootkit.

[14]Russinovich M E, Solomon D A. Microsoft Windows internals[M]. 4th ed. 北京: 电子工业出版社, 2008: 375 – 448.

[15]Greg H, James B. Rootkits: Subverting the Windows kernel[M]. Indianapolis: Addison-Wesley Professional, 2005: 108 – 119.

[16]金海. 计算系统虚拟化——原理与应用[M]. 北京: 清华大学出版社, 2008: 1 – 16.

[17]Wang Zhi, Jiang Xuxian, Cui Weidong, et al. Countering kernel rootkits with lightweight hook protection[C]//Proceeding of the 16th ACM Conference on Computer and Communications Security. New York: ACM, 2009: 545 – 554.

(编辑　杨　蓓)