基于硬件辅助虚拟化技术的反键盘记录器模型

马建坤 黄 皓

(南京大学软件新技术国家重点实验室 南京 210093) (南京大学计算机科学与技术系 南京 210093)

摘 要 结合已有的键盘记录器,分析了Windows中从用户按键到应用程序处理消息的过程,并针对该过程详细分析了可能出现的安全威胁。在此基础上提出了基于硬件辅助虚拟化的反键盘记录器模型。利用CPU提供的硬件辅助虚拟化技术实现了虚拟机监控器,当获取用户输入时通过在虚拟机监控器中自主处理键盘中断并将读取到的键盘扫描码信息交由受保护的用户线程来保护用户键盘输入的安全。

关键词 反键盘记录器,硬件辅助虚拟化,中断,虚拟机监控器

中图法分类号 TP393 **文献标识码** A

Anti-key Logger Based on Hardware-assisted Virtualization

MA Jian-kun HUANG Hao

(State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210093 China)

(Department of Computer Science and Technology, Nanjing University, Nanjing 210093, China)

Abstract The message flow of Windows operating system was introduced and the potential threats was analyzed. A solution based on hardware-assisted virtualization was developed to defend against the software key logger. A virtual machine monitor was implemented using Intel virtual technology. When the protected thread is reading keyboard input, the keyboard interrupt is handled in the virtual machine monitor. By reading scan code of the keyboard in the virtual machine monitor, the keyboard input can be safely sent to the protected thread.

Keywords Anti-key logger, Hardware-assisted virtualization, Interrupt, Virtual machine monitor

1 引言

键盘记录器是用于截取和记录用户击键信息的工具。通过截取和记录用户的输入行为,攻击者可以很容易地监控用户的行为,甚至可以很轻松地获取系统的管理员权限。

目前的键盘记录器可分为软件和硬件两种[1]。软件键盘记录器通过在操作系统中安装软件的方式来截取用户的键盘输入信息;而硬件键盘记录器则是在硬件层次上截获用户的输入。由于硬件键盘记录器不在软件控制下,因此针对这种类型的键盘记录器的防护基本上只能依靠用户在输入时人为制造混淆信息,而无法使用软件的方法来防止该种攻击。所以本文主要研究针对软件键盘记录器的防护。

2 键盘记录器分析

2.1 Windows键盘工作原理

在Windows系统中,按键信息被组织成消息,这些键盘消息随后被发送到指定的GUI(Graphical User Interface)线程。GUI线程接收到键盘消息之后再对这些消息进行处理。整个过程可以总结为两步,第一步是键盘扫描码的获取,即系统读取硬件上的按键扫描码的过程;第二步是消息的流动,即消息从构造到到达目标线程的过程。

2.1.1 键盘扫描码的获取

Windows中用于处理设备的I/O请求的是设备对象,设备对象以栈的方式组织,通过向设备栈的栈顶设备对象发送一个IRP(I/O Request Package)来实现对设备的I/O操作[2]。图1显示了Windows中PS/2键盘的键盘设备栈结构。Windows中键盘等外部设备的输入由一个线程读取,该线程为RIT(Raw Input Thread)[3]。以PS/2键盘为例,当用户按下或松开键盘上的某个按键时,与键盘对应的中断处理程序会被调用,默认情况下该中断处理程序由i8042prt.sys驱动程序提供。键盘中断处理程序读取键盘端口的扫描码,然后交由上层键盘类驱动程序处理。上层键盘类驱动程序将键盘扫描码最终返回给I/O调用的发起方,即RIT。

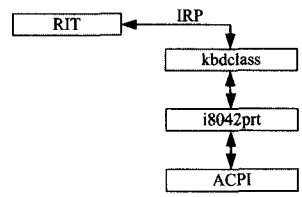

图1 Windows PS/2键盘设备栈

2.1.2 消息流

对于每一个GUI线程,Windows都为其维护了两个消息队列。其中一个消息队列用于存放系统输入消息,另一个消

到稿日期:2010-12-25 返修日期:2011-03-05 本文受国家自然科学创新群体项目(60721002)和江苏省高技术项目(BE2008124)资助。

马建坤(1986—),男,硕士生,主要研究方向为操作系统和主机信息安全,E-mail:tao-mjk-tx@hotmail.com;黄 皓(1957—),男,教授,博士生导师,主要研究方向为信息安全。

息队列用于存放寄入消息[3]。GUI 线程从消息队列中获取消息，并交由相应的窗口进行处理。图 2 显示了 Windows 的消息流。

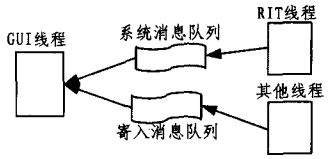

图 2　Windows 消息流

在每一个会话中，Windows 都会创建一个 RIT 用于读取外部设备的输入信息，并将其组织成消息后放入处于前台的 GUI 线程的系统消息队列中。对于键盘设备，RIT 读取到键盘扫描码后，根据扫描码生成按键消息，然后放入系统消息队列中。同时，其他线程以寄入方式发送的消息被存放在 GUI 线程的寄入消息队列中。当 GUI 线程被触发从而调用 NtUserGetMessage 或 NtUserPeekMessage 来获取消息时，GUI 线程从消息队列中获取消息并对消息进行处理。系统消息队列和寄入消息队列存放在 GUI 线程的 TRHEADINFO 结构中。

2.2　键盘记录器实例

基于上述分析，本节结合 Windows 的消息流给出一些键盘记录器实例。

(1)基于消息钩子的键盘记录器

消息钩子是 Windows 平台提供的一种机制，可以允许程序监控系统中消息的处理[4]。消息钩子分为两种：局部钩子和全局钩子。局部消息钩子允许线程监控当前线程的消息处理过程；全局消息钩子允许线程监控当前桌面上所有 GUI 线程的消息处理过程。针对局部消息钩子，系统在每个 GUI 线程的 THREADINFO 域中为其定义了一个 PHOOK 数组用于登记当前线程的局部消息钩子。对于全局消息钩子，系统在该 GUI 线程所属桌面的 DESKTOPINFO 域中也维护了一个钩子列表数组。应用程序通过调用函数 SetWindowsHookEx 来安装一个消息钩子，最终系统通过参数中指定的线程 ID 在该 GUI 线程或者 GUI 线程所属的桌面钩子列表中加入一个消息钩子。消息钩子处理函数根据钩子类型在消息流的处理过程中会被调用，例如当一个键盘输入消息被加入到系统消息队列时会调用系统中安装的类型为 WH_KEYBORD_LL 的钩子处理函数。键盘记录器通过安装消息钩子，从而可以截获到击键信息。

(2)基于调用 GetAsyncKeyState 的键盘记录器

RIT 读取到设备输入信息之后，除了构造消息并将其加入到系统消息队列之外，还会更新相应的按键信息。在 GUI 线程的系统消息队列数据结构中，还记录了键盘上各个键的状态。同时除了每个 GUI 线程保存的该线程接收的按键状态信息外，系统还维护了一个全局的按键状态数组。键盘记录器可以调用 GetAsyncKeyState 函数来从这两个数组中获取按键状态信息，从而可以截获到用户的按键信息。

(3)基于 IAT HOOK 技术实现的键盘记录器

IAT(Import Address Table)是 PE 文件的导入地址表，存放当前文件所引用的其他文件的函数[5]。通过 HOOK IAT 中导出的消息操作的相关函数即可实现消息的拦截。GUI 线程可以通过调用 GetMessage 函数从消息队列中获取消息，然后调用 DispatchMessage 函数对消息进行分发处理，这两个函数由 User32. dll 导出。所以只要 HOOK IAT 中 User32. dll 的这两个函数就可以实现消息的拦截。

(4)基于 Shadow SSDT HOOK 技术[5]实现的键盘记录器

当用户态程序调用 GetMessage 和 DispatchMessage 等函数时，User32. dll 通过发起一个软件中断而陷入到内核中的系统服务分发函数中来完成相应的处理。系统服务分发函数依据系统服务号来调用系统分发表中的处理函数。这里 GetMessage 和 DispatchMessage 函数对应的分发函数为 NtUserGetMessage 和 NtUserDispatchMessage。这两个函数的指针都存放在 Shadow SSDT(System Service Dispatch Table)中[2]。通过 HOOK Shadow SSDT 表中的这两个函数，就可以实现消息的拦截。

(5)基于键盘过滤驱动的键盘记录器

RIT 以 ZwReadFile 的方式读取设备的输入信息，而 ZwReadFile 会调用 NtReadFile 来构造一个 IRP 结构的 I/O 请求包，并发送到键盘设备对象栈上。键盘记录器通过将自己创建的设备对象附加在键盘类驱动创建的设备对象之上，就可以拦截到 RIT 对键盘驱动的读请求，从而可以截获到用户的击键信息[6]。

(6)基于 IDT HOOK 的键盘记录器

IDT(Interrupt Descriptor Table)存放了系统中的中断描述符，每一个中断描述符指定了一个中断处理函数，用于中断和异常的处理[8]。当键盘上有一个键被按下或者弹起的时候，键盘会产生一个中断通知系统，系统就可以读取相关信息然后再做后续的处理。对于 PS/2 的键盘，默认情况下 Windows xp 为键盘预留的中断号为 0x93。当一个 PS/2 的键盘上有击键信息时，中断号为 0x93 的中断服务例程将被调用，该服务例程会读取 0x60 端口的键盘扫描码。然后将读取到的键盘扫描码处理后交给上层的设备驱动做后续的处理。所以键盘记录器只要通过 IDT HOOK 方式 HOOK 0x93 号中断，就可以在键盘产生事件时获取通知，通过读取 0x60 号端口就可以截获到击键信息[7]。

(7)基于修改 I/OAPIC 的键盘记录器

APIC(Advanced Programmable Interrupt Controller)从奔腾处理器开始被引入进 IA-32 的处理器系统中，其可以完成更高级的中断管理。其中用于接收外部中断的 I/OAPIC 允许程序开发人员重新定义 IRQ(Interrupt ReQuest)和中断处理程序之间的映射关系[8,9,10]。PS2 键盘的硬件中断请求为 IRQ1，默认情况下在 Windows xp 中定义了 IDT 中 0x93 的中断处理例程用于处理 IRQ1 的中断。而键盘记录器可以修改这种映射关系，将中断请求为 IRQ1 的中断映射到 IDT 表中未使用的一项中，从而当键盘中断产生时，将会调用键盘记录器的处理函数[7]。这种方式不修改 IDT 表中对应的键盘中断，隐蔽性高，主流的 Anti-Rootkits 工具无法检测到。

(8)直接操作消息队列的键盘记录器

以这种方式实现的键盘记录器并不常见，但是依据 Windows 消息流的原理可以很容易实现一个这样的记录器。当一个 GUI 线程成为接收外设输入的线程时，所有的键盘输入信息都会被组织成消息放入到该 GUI 线程的系统消息队列中。所以只要直接从系统消息队列中读取消息，记录其中的

键盘消息，就可以实现截获用户的击键记录。

(9)基于截屏方式实现的键盘记录器

该种方式是通过截取用户输入时的屏幕，从所得的图像中获取到用户的输入信息。这种方法要求键盘的输入信息必须显示在屏幕上，或者输入信息可以在屏幕中看到。严格意义上讲，该种方法获取到的并非是用户的击键信息，而是用户的屏幕信息。

上述方法中，只有第9种不是基于消息流截获实现的，由其特点得知其不安全的原因是由于应用程序对接收到的键盘输入的处理不安全所造成的。应用程序将键盘输入直接输出到不安全的屏幕中才导致了信息泄露，而只要对接收到的键盘输入的处理得当就可避免这种攻击，例如不显示键盘输入或显示其他信息。对键盘输入的安全处理的职责在于应用程序本身而不在于反键盘记录器，反键盘记录器的主要职责在于保护键盘输入安全地传输到应用程序，所以反键盘记录器可以只关注消息流的安全。基于消息流的前8种键盘记录器中，利用键盘过滤驱动和IDT HOOK实现的键盘记录器是通过在从RIT读取按键信息到将消息放入系统队列这一段路径上截取的键盘输入信息。由消息钩子造成的信息泄露是围绕消息队列所产生的，消息进入消息队列和从消息队列中移出都会导致钩子函数被调用，从而都会导致信息泄露。而通过调用函数GetAsyncKeyState能够获取按键信息的原因是因为RIT在将按键信息放入消息队列的同时更新了相应的按键状态信息。而以直接操作消息队列的方式来获取按键信息的方式可以获取键盘输入的根本原因是消息队列本身没有受到保护。上述列出了基于消息流的8种键盘记录器，并没有包含所有的键盘记录器，因为可以通过对应关系将其他的基于消息流的键盘记录器对应到消息流中的某个位置，从而可以用上述的8种键盘记录器来分析。例如陈俊杰等人提出的基于SSDT及回调函数的键盘记录器[11]，虽然其实现方法不同，但该记录器结合了基于Shadow SSDT和基于键盘类过滤驱动技术，因此依然可以将其对应到消息流中的相应位置来进行分析。

3 相关工作分析

3.1 消息钩子检测和清除

消息钩子由于实现简单，因此是反键盘记录器必须考虑的内容。郭津之等提到的消息钩子的检测和清除技术[12]可以很容易地检测和清除系统中的安装消息钩子，但是该方法并没有对消息钩子进行安全性检验，所以应用程序安装的消息钩子也会被认为是恶意钩子而一并清除。

3.2 反汇编进程的指令流

Muhammad Aslam等人提出了扫描系统中的所有进程，通过发现其中函数SetWindowsHookEx的调用语句来查找系统中所有可能安装消息钩子的模块的方法[13]。该方法只能检测出系统中安装消息钩子的模块，无法检测基于其他方式实现的键盘记录器。

3.3 Anti-Rootkits技术

Rootkit是一组对系统具有高权限的代码，并通过一些隐藏手段使得自身对管理员隐蔽[6]。键盘记录器隶属于Rootkits，针对Rootkits的防御研究也可以用于防御键盘记录器。目前主流的Anti-Rootkits技术侧重于对Rootkits的检测上，这种方法很难检测出一些键盘记录器，例如通过调用GetAsyncKeyState实现的键盘记录器就很难被检测到。虽然Anti-Rootkits工具可以清除一些检测到的键盘记录器，但是不是所有的键盘记录器都可以安全清除。由于Windows的驱动设备栈结构在卸载时必须从设备栈的顶部开始删除设备[2]，因此当系统中安装了基于过滤驱动实现的键盘记录器，且在键盘记录器之上还附载了其他设备对象时，将无法安全清除键盘记录器，而在这些系统上防御无疑比检测更重要。

3.4 屏幕键盘

利用软件自己绘制的键盘来代替真实的键盘，是一种有效地防止多数基于消息流截获的键盘记录器的措施。该种方法将鼠标点击消息通过软件转换成键盘按键消息，从而有效地防御了多数键盘记录器的攻击，但是该种方法存在两个缺陷：其一，无法防御通过截屏方式实现的键盘记录器；其二，由于增加了针对鼠标消息的处理，然而该方法并未对鼠标消息进行保护，因此通过截获鼠标消息并调用与防护软件相同的处理流程，键盘记录器依然可以截获到用户的输入信息。

3.5 基于键盘记录器的方法

Muzammil M. Baig和W. Mahmood提出了一种采用和键盘记录器同样的方法，即通过截取键盘中断，然后由程序自己处理中断，并将处理结果直接发送到应用程序的消息队列中[14]。该种方法可以有效地防御那些针对消息流中从外部设备到消息队列的路径上实现的键盘记录器，但是无法防御基于消息钩子等实现的键盘记录器。而且由于防护程序和操作系统处于同一级别，Rootkit仍然可以修改键盘中断，从而使得防护程序无效。

由上述分析可以看出，要防止键盘记录器，必须对消息流的整个过程进行防御。从消息产生到消息最终被应用程序接收，要确保恶意程序无法从中截取到真实的按键信息。要实现这一要求的根本是要将防护程序和其他程序区别开来，即将防护程序运行于较高的权限级别上，而将其他软件运行于较低的权限级别。目前主流的操作系统的设计中，驱动程序和操作系统都运行在最高的特权级，基于驱动程序的Rootkit对系统有完全的权限，所以基于传统模式的架构无法满足这一要求。

4 基于硬件辅助虚拟化技术的反键盘记录器设计与实现

4.1 系统工作模型

基于上述分析，本文提出了基于本地虚拟化的反键盘记录器模型，具体实现结构如图3所示。

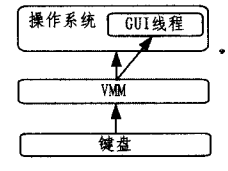

图3 基于虚拟化技术的反键盘记录器模型

虚拟机监控器(Virtual Machine Monitor，VMM)运行在更高的特权模式下，操作系统的运行受到虚拟机监控器的监控。当键盘中断到达时，CPU陷入到虚拟机监控器中。如果此时是受保护的GUI线程正在接收外部输入，则虚拟机监控器读取键盘的扫描码并保存在缓冲区中以待GUI线程读取；否则，虚拟机监控器将此中断交由客户机操作系统处理。当VMM获取到用户的击键信息时，VMM通知GUI线程，GUI

线程再利用 VMM 提供的服务调用机制(Hypercall)来获取键盘扫描码。

Goldberg 在文献[15]中定义了两种 VMM,Type I VMM 和 Type II VMM,结构如图 4 所示。Type I VMM 运行于硬件上,操作系统运行于虚拟机中。Type II VMM 运行于宿主机操作系统中,客户机操作系统运行于虚拟机上。Type II VMM 由于需要宿主机操作系统支持,因此不适用于那些已经安装了操作系统的主机。而 Type I VMM 可以利用 CPU 提供的硬件辅助虚拟化技术实现,不需要修改操作系统或对操作系统重新部署。目前主流的 CPU 都提供了硬件辅助虚拟化支持,例如 Intel-VT[9] 和 AMD-V[16] 技术。本文的 VMM 是 Type I VMM。Goldberg 模型中的虚拟机由 Type I VMM 创建并调度运行,而本文的 VMM 以内核驱动程序的方式提供,由操作系统加载运行。当驱动程序开启 CPU 的虚拟化功能后,驱动程序运行在 Root 模式[9],而操作系统运行在 Non-Root 模式[9],这样运行在 Root 模式的驱动程序构成了 VMM,而运行在其上的操作系统则作为 VM 由 VMM 管理。

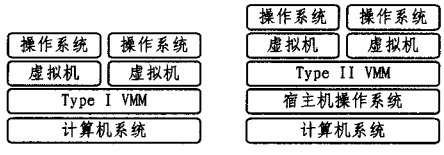

图 4 两种 VMM 体系架构

4.2 系统安全性分析

当受保护的 GUI 线程在接收外部键盘输入时,所有的键盘中断都将交由 VMM 处理,操作系统本身不会得到调用,所以图 2 所示的消息流将不再工作。基于消息流上的键盘记录器将无法截获到用户的击键信息。同时在接收到键盘按键信息后,不是回显真实的按键信息,而是回显星号,从而可以有效地抵御基于截屏方式的键盘记录器的攻击。由于 VMM 运行在更高的特权模式下,因此键盘记录器企图通过破坏 VMM 的结构或者直接读取 VMM 中缓冲区的方式来获取键盘输入信息将无法成功。而对于那些修改 APIC 的重定位信息从而企图修改键盘中断号的键盘记录器,可以通过动态获取键盘中断对应的中断号来防御。当外部中断产生时,CPU 陷入到 VMM 中,VMM 读取此时 I/OAPIC 中键盘中断对应的中断号,从而确定此时键盘的中断号。这样无论键盘记录器如何修改 I/OAPIC 中键盘中断对应的中断号,VMM 都可以准确获取在键盘中断发生时刻外部键盘的中断号,从而可以有效地防御这种攻击。在 VMM 将键盘的扫描码信息交给 GUI 线程的这段路径上采用的是基于 Hypercall 的方式,恶意程序无法在中间拦截,因此键盘扫描码信息在该路径上是安全的。同时为防止键盘记录器在受保护的 GUI 线程接收键盘输入期间通过 Hypercall 调用来读取键盘中断扫描码,在读取 VMM 的键盘扫描码的过程中增加认证,即在 GUI 线程通知 VMM 监控键盘中断的时候记录下该线程的 ID,当线程读取扫描码时验证发起 Hypercall 的线程是否为记录的线程即可防御这种攻击。综上所述,当受保护的 GUI 线程在获取键盘输入时,键盘扫描码信息在从外部键盘设备到 GUI 线程的过程中是安全的。

4.3 实现

本文利用 Intel-VT 技术实现了基于本地虚拟化技术支持的反键盘记录器模型。其中 VMM 以驱动程序的形式被加载到 Windows 操作系统中。当驱动程序完成相应的初始化后,开启虚拟化功能,从而将自身运行于根模式,操作系统和其上运行的应用程序均运行于非根模式。相关实现具体包含以下 6 个方面:

(1)VMM 的构建

VMM 的构建是在驱动程序的初始化函数中完成的。初始化完成的主要工作是准备好虚拟化支持所需要的一些信息,并将 CPU 的虚拟化功能开启。这分以下几步:①初始化 VMXON 结构。VMXON 结构用于当前处理器对虚拟化的支持。②初始化 VMCS 结构。VMCS 结构是用于控制非根模式和模式切换的关键结构,其中包含了宿主机和客户机的状态信息以及模式之间切换的控制信息。初始化该结构使用 VMCLEAR 指令,读取该结构可以通过指令 VMREAD 来实现,而写该结构可以通过指令 VMWRITE 来实现。③开启虚拟化功能。利用前面准备的 VMXON 结构调用 VMXON 指令,可以将当前 CPU 置于根模式之下。而通过 VMPTRLD 指令可以将 VMCS 结构的指针加载到当前 CPU 中,之后通过调用 VMLAUCH 指令就可以将 CPU 置于非根模式,并运行 VMCS 中指定的客户机代码。

(2)外部中断的接管

要接管外部中断,只要使得当外部中断产生时会导致 VM-Exit,从而可以陷入到 VMM 中即可。VMCS 中 VM-Execution 控制域中位 0 控制外部中断是否产生 VM-Exit,当设置该位为 1 时,外部中断会产生 VM-Exit。这样当外部中断到达时,会首先陷入到 VMM 中。

(3)键盘中断的处理

在 VM-Exit 的处理中,当读取到的原因为外部中断时,通过分析该中断号是否为键盘中断来决定是否要特殊处理。键盘中断号可以通过读取 I/OAPIC 中 IRQ 的重定位信息来获取[8]。以 PS/2 键盘为例,I/OAPIC 中分配给 PS/2 键盘的 IRQ 为 IRQ1,只要读取 IRQ1 对应的中断号即可知道此时键盘的中断号。如果是键盘中断则进行后续处理,如果不是则通过事件注入方式将该事件注入到客户机操作系统中,交由客户机操作系统完成中断的处理。对于键盘中断,首先判断当前是否是受保护的 GUI 线程在接收键盘输入,不是则将该事件注入到客户机操作系统,是则读取键盘扫描码,并放入临时队列中,然后通知 GUI 线程有键盘事件到达。对于 PS/2 键盘,当键盘上有按键被按下或放起的时候,I/O 端口 0x60 中存放了键盘的扫描码信息,VMM 读取 I/O 端口 0x60 就可以获取到键盘的扫描码。

(4)键盘扫描码的传输

GUI 线程得到通知后,通过向 VMM 发起 Hypercall,从 VMM 读取键盘扫描码。这里在 GUI 线程端发起 Hypercall 通过调用 VMCALL 指令来实现;而在 VMM 端 Exit Reason 为 VMCALL 的事件即为 Hypercall。参数通过寄存器来传送,其中 EAX 存放服务号,EBX 存放第一个参数,ECX 存放第二个参数,EDX 存放第三个参数。定义服务号 0 为开启保护功能,开启保护功能后,所有的键盘输入都将被 VMM 直接处理。定义服务号 1 为关闭保护功能,关闭保护功能后,键盘输入都将以事件注入的方式注入到操作系统中。定义服务号 2 为设置事件对象,事件对象是用于 VMM 和 GUI 线程同步的对象,即当 VMM 需要通知 GUI 线程接收键盘输入时,VMM 设置该事件对象为有信号状态,等待该事件对象的

GUI线程可以被唤醒，从而发起读取键盘扫描码的 Hypercall。定义服务号 3 为读取键盘扫描码，其中第一个参数指向用于存放扫描码的缓冲区，第二个参数指定该缓冲区大小。GUI线程开启保护功能的时机是其获得了系统的输入焦点之后，关闭保护功能的时机是当其失去了系统的输入焦点。GUI线程获取到输入焦点时会接收到系统发送的 WM_SETFOCUS 消息，而当其失去输入焦点时会接收到系统发送的 WM_KILLFOCUS 消息。通过在这两个消息的处理中来开启和关闭保护功能。

(5)键盘扫描码的处理

在处理键盘按键信息时通常还需要使用键盘的虚拟按键信息，而从 VMM 读取到的是键盘的扫描码信息，所以还必须将键盘的扫描码转换为虚拟键信息。GUI线程读取到键盘扫描码后，通过系统调用 MapVirtualKeyEx 可以将键盘扫描码信息转换成虚拟键信息。除了虚拟键信息，GUI线程在处理键盘输入时还需要知道键盘的 Windows 消息类型，这可以通过键盘扫描码来确定。当键盘扫描码的 0x80 位设置为 1 时表示键盘上有一个按键被松开，对应的 Windows 消息是 WM_KEYUP，否则表示键盘上有一个按键被按下，对应的 Windows 消息是 WM_KEYDOWN。这些信息获取到之后，GUI线程就可以进入到对键盘事件的消息处理中。

(6)额外的保护

这里主要是针对 PS/2 键盘的额外保护。由于键盘扫描码信息存放在 0x60 端口，键盘记录器可以直接读取 0x60 端口来获取键盘扫描码，因此必须对 I/O 端口进行保护。只要设置 VM-Execution 控制域中的 Use I/O bitmap 为 1，并且将需要保护的 I/O 端口在 I/O 位图中的位设置为 1 即可。这样任何针对该端口的 I/O 访问都将陷入到 VMM 中，VMM 可以在受保护的 GUI线程接收键盘输入的期间拒绝针对该端口的访问。这里只要将端口 0x60 在 I/O bitmap A 中的相应位设置为 1 即可实现对键盘数据端口访问的拦截，从而实现对端口的保护。

5 实验结果

本节给出相应的实验结果。针对以上分析，本文选取其中有代表性的 5 个键盘记录器，用于本文实验测试。KeyLogger 是基于消息钩子实现的键盘记录器；NetLogger 是基于调用 GetAyncKeyState API 的方式实现的键盘记录器；KLOG 是基于键盘过滤驱动的键盘记录器；Kb_sniff 是基于 IDT HOOK 技术实现的键盘记录器；ps2intcap 是基于修改 I/OAPIC 中重定位信息实现的键盘记录器。实验所使用的机器为 ThinkPad X61 7673，操作系统为 Windows XP sp2。表 1 给出测试结果。如测试结果显示，当受保护的线程在接收键盘输入时，选取的 5 个键盘记录器无法截获到按键信息，表明反键盘记录器的功能工作正常。

表 1 测试结果

键盘记录器	拦截结果
KeyLogger	无
NetLogger	无
KLOG	无
Kb_sniff	无
ps2intcap	无

结束语 本文分析了常用的键盘记录器的特点及其实现方法，同时详细分析了 Windows 的消息机制，并结合该机制提出了一种有效防止键盘记录器的软件方法。利用 Intel-VT 技术实现了 VMM，以监控键盘中断的处理过程。当受保护的 GUI线程接收键盘输入时，VMM 自行读取键盘扫描码并交由 GUI线程。该方法有效地防止了键盘记录器的攻击。

但是该方法需要硬件支持，CPU 必须支持硬件辅助虚拟化技术。随着支持虚拟化的 CPU 的普及，这一限制也将变得无关紧要。本文只是实现了 PS/2 键盘的安全防护，而对于 USB 键盘的支持则是下一步的工作。

参 考 文 献

[1] keystroke logging [EB/OL]. http://en.wikipedia.org/wiki/Keystroke_logging

[2] Russinovich M E, Solomon D A, Ionescu A. Windows Internals, Covering Windows Server 2008 and Windows Vista (5th Ed) [M]. Microsoft Press, 2009

[3] 潘爱民. Windows 内核原理与实现[M]. 北京：电子工业出版社，2010

[4] Richter J. Programming application for Microsoft Windows(4th Ed)[M]. Washington: Microsoft Press, 2000

[5] Peering P M. Inside the PE: A Tour of the Win32 Portable Executable File Format[J]. Microsoft SystemsJournal, 1994

[6] Hoglund G, Butler J. Rootkits: Subverting the Windows Kernel [M]. 北京：清华大学出版社，2007

[7] 谭文，杨潇，邵坚磊，等. 寒江独钓-Windows 内核安全编程[M]. 北京：电子工业出版社，2009：56-99

[8] Intel Corporation. Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 3A: System Programming Guide, Part 1[Z]. 2010

[9] Intel Corporation. Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 3B: System Programming Guide, Part 1[Z]. 2010

[10] Intel Corporation. Intel 82093AA I/O Advanced Programmable Interrupt Controller(I/O APIC) Datasheet[Z]

[11] 陈俊杰，施勇，薛质，等. 基于 SSDT 及回调函数的键盘记录方法[J]. 计算机工程，2010，36(11)

[12] 郭津之，龙海，黄皓. Windows 消息钩子的拦截和清除[J]. 计算机工程与设计，2009，30(18)4201-4203

[13] Aslam M, Idrees R N, Baig M M, et al. Anti-Hook Shield against the Software Key Loggers[C]// National Conference on Emerging Technologies. 2004

[14] Baig M M, Mahmood W. A Robust Technique of Anti Key-Logging using Key-Logging Mechanism[C]// 2007 Inaugural IEEE International Conference on Digital Ecosystems and Technologies(IEEE DEST 2007). 2007

[15] Goldberg R P. Architecture principles for virtual computer systems [D]. Harvard University, Cambridge, MA, 1972

[16] AMD. AMD64 Virtualization Codenamed "pacifica" Technology: Secure Virtual Machine Architecture Reference Manual[Z]. 2005