



研究与开发

Rootkit 攻防机制与实现方法

李馥娟, 王群

(江苏警官学院计算机信息与网络安全系, 江苏 南京 210031)

摘要: Rootkit 是一类能够攻击系统内核且实现深度隐藏的恶意代码, 已对网络安全造成了严重威胁。首先, 介绍了 Rootkit/Bootkit 的基本特征, 对比分析了用户模式和内核模式下 Rootkit 攻击的特点; 接着, 重点剖析了 Rootkit 攻击涉及的挂钩、DKOM 和虚拟化技术的实现原理及工作机制; 最后, 结合具体的攻击行为讨论了针对 Rootkit 攻击的主要检测方法和防御技术。

关键词: 网络攻防; 恶意代码; Rootkit; 挂钩攻击; 网络安全

中图分类号: TP393

文献标识码: A

doi: 10.11959/j.issn.1000-0801.2018298

Mechanism and implementation of Rootkit attack and defense

LI Fujuan, WANG Qun

Department of Computer Information and Cyber Security, Jiangsu Police Institute, Nanjing 210031, China

Abstract: Rootkit is a set of malicious codes that can attack the system kernel and achieve deep hiding, which has posed serious threats to cyber security. Firstly, the basic features of Rootkit/Bootkit were introduced, and the characteristics of Rootkit attacks in user mode and kernel mode were compared and analyzed. Thereafter, the implementation principles and working mechanisms of Hook, DKOM and virtualization technologies involved in Rootkit attacks were emphatically analyzed. Combined with the specific attack behaviors, the main detection methods and defense techniques for Rootkit attacks were discussed at the end.

Key words: network attack and defense, malware, Rootkit, hooking attack, network security

1 引言

网络攻防表面上是一种攻守双方的技术对

抗, 其实质则是攻击者与防守者之间的力量较量, 是人与人之间的智力博弈。近年来, 随着人们对网络的依赖性越来越强, 功能各异的软硬件系统

收稿日期: 2018-06-13; 修回日期: 2018-12-10

基金项目: “十三五”江苏省重点建设学科建设工程资助项目 (No.2016-0838); 江苏高校品牌专业建设工程资助项目 (No.PZY2015C203); 江苏省第五期“333 工程”科研项目资助 (No.BRA2017443); 江苏高校哲学社会科学研究基金项目 (No.2018SJA0456)

Foundation Items: Key Construction Discipline Construction Project of Jiangsu Province During the 13th Five-Year Plan Period (No.2016-0838), Jiangsu University Brand Professional Construction Project Subsidy Project (No.PZY2015C203), Jiangsu Province Fifth Phase “333 Project” Scientific Research Project (No.BAR2017443), Jiangsu University Philosophy and Social Science Research Fund Project (No.2018SJA0456)



在丰富了应用的同时,其自身存在的漏洞也成为网络攻击者不断挖掘和利用的资源。网络攻防是一种“矛与盾”的关系,防守者总希望能够通过获取并分析攻击者的痕迹来溯源攻击行为,不断完善防御策略,以应对更加复杂多样的攻击。然而,随着攻击技术的不断提升,大量的攻击手段利用高级隐遁技术(advanced evasion technique, AET)^[1-2]来绕过正常的程序运行检测机制,通过组合、修改、乱序等方式直接攻击操作系统的内核,致使越来越多的网络攻击行为无法查明具体原因,更无法提出应对策略。Rootkit 便是在这种特殊而复杂的网络环境下逐渐演变形成的一项主要针对系统内核及固件的攻击技术。

Rootkit 并非一项全新的技术,但却是恶意代码家族中发展最快、安全威胁最大的技术之一。McAfee 实验室在《McAfee Labs Threats Predictions for 2017 and Beyond》中提出:硬件和固件将日益成为通过复杂技术进行攻击的主要目标,其中攻击硬件和固件的便是 Rootkit。Rootkit 技术涉及 UNIX/Linux、Windows、Netware 等计算机系统以及 Google Android、Apple iOS、Windows Phone 等移动智能终端操作系统^[3-5]。

Rootkit 技术在一些重大网络攻击中的应用已经不是什么秘密。例如,被称为世界上首个网络“超级破坏性武器”的“震网”(Stuxnet)病毒于 2010 年对伊朗核电设施进行了攻击,导致 1/5 的离心机报废,并使有毒放射性物质泄漏。“震网”病毒是由美国和以色列共同研制,通过采用 Rootkit 技术渗透进 Windows 操作系统后,对 Windows 进行重新编程而进行的破坏性攻击。2013 年披露的美国“棱镜”(PRISM)计划^[9],其核心技术也是利用了 Rootkit 技术植入恶意软件并长期隐匿,伺机发起攻击,获取所需要的信息。

针对 Rootkit 攻防技术的研究,目前已经取得了一些初步成果。其中,参考文献[6]介绍了 Rootkit 的基本概念,分析了 Windows Rootkit 的

相关机制,并对 Rootkit 技术发展给出了一些展望;参考文献[7]提出了利用交叉视力验证技术来检测恶意软件隐蔽对象的方法,并在 VMware 虚拟环境中进行了测试;参考文献[8]在介绍了 Rootkit 技术的基础上,以 2015 年针对乌克兰关键基础设施的 ATP(advanced persistent threat,高级持续式威胁)为例,对 Rootkit 的检测和修复方法进行了分析;参考文献[9]介绍了一种从广泛使用的 Android 设备中获取不稳定的物理内存的工具 AMExtractor,该工具使用/dev/kmem 在内核模式下执行代码,可在运行状态下提取设备的信息,并有效检测 Rootkit 的存在。本文将从网络攻防的角度,首先归纳 Rootkit 的定义和工作机制,重点对攻击实施过程及关键技术进行较为详实的分析,在此基础上,再从防守者的角度,讨论相应的检测方法和防御思路。

2 Rootkit 的特征和工作机制

2.1 Rootkit 的特征

从网络攻击方法来看,Rootkit 是攻击者使用的一个工具集,用于获得对系统的非授权访问,为攻击者获取敏感数据提供特殊权限,并隐藏自己的存在,而且根据需要允许安装其他恶意软件^[10]。从 Rootkit 的组成“Root”(特权用户)和“kit”(工具集)可以看出,Rootkit 是能够获得系统特权并能够控制整个系统的工具集,在安装 Rootkit 之前,攻击者需要获得管理员权限。Rootkit 是最具挑战性的恶意软件,因为它很难通过系统提供的检测机制以及第三方的检测软件发现。

Rootkit 是一种能够同时针对操作系统(包括微内核操作系统)的用户模式和内核模式进行程序或指令修改,达到通过隐藏程序执行或系统对象的变化来规避系统正常检测机制,绕开安全软件监控与躲避取证手段,进而实现远程渗透、深度隐藏、长期潜伏并对整个系统进行控制的攻击技术。与传统的恶意代码不同,Rootkit 攻击的灵

活性更大、破坏性更强、被检测难度更高、技术要求更复杂。

Rootkit 与计算机病毒、蠕虫、木马、后门和僵尸程序等同属于恶意代码，都是由攻击者按照攻击意图植入被攻击系统中的程序或代码，都具有潜伏性和破坏性。但与其他类型的恶意代码不同的是，Rootkit 还会替换或修改被攻击系统中的程序。作为一类特殊形态的恶意代码，Rootkit 能够将自己伪装成系统中的一个合法程序（木马的特征），使得攻击者可以按照自己的方式访问系统（后门的特征），修改或替换 BIOS、显卡等固件中的代码或系统中的正常程序而将自己隐藏起来（高级计算机病毒的特征）。

2.2 Rootkit 的用户模式和内核模式

自底向上分层模型的特点是通过层间接口技术将下层的差异性利用统一的服务模式（标准或协议）封装起来，下层为其上层提供一种抽象一致的按需服务。然而，这种分层模型却为攻击者提供了可乘之机，攻击者通过篡改下层组件结构或劫持并替换下层组件的输出值，达到欺骗上层调用并隐匿自身的目的。目前广泛使用的 Windows、UNIX/Linux 操作系统都采用分层模型，Rootkit 可以工作在操作系统的用户模式和内核模式。

Rootkit 可以攻击操作系统用户模式下的一些内建程序或库文件，攻击者可以用事先编写的替换程序覆盖被访问计算机的程序和服务；替换系统中各种信息查看工具从而向用户隐瞒攻击者在被控系统中的文件、进程和网络使用情况；还可以更改程序运行时间，删除或修改系统日志；也能够解开工具包并自动安装脚本程序等。其中，Rootkit 在 Windows 系统中可以使用开发接口将自身的恶意功能逻辑地插入正常的 Windows 函数，而无需覆盖现有的 Windows 代码，为了能够修改操作系统中的关键文件还可以关闭 Windows 中的 WFP（Windows file protection）机制，或通过 DLL 注入（DLL injection）和 API 挂钩（API hooking）

技术将恶意代码直接加载到已运行进程的内存空间中。

从攻击的有效性来讲，内核模式下的 Rootkit 可以直接对操作系统的内核对象（如事件对象、文件对象、文件映射对象、进程对象、线程对象等）进行攻击，可以通过修改内核代码来欺骗用户模式下的可信软件，能够通过访问底层信息来对整个系统进行控制；能够利用内核权限来对抗一些正常的安全机制；通过修改内核代码来实施深度隐藏（如隐藏自己所在的文件和文件夹、隐藏进程、隐藏通信端口等）等。所以，Rootkit 作用在内核模式时要比作用在用户模式时对系统的破坏性更强，更难被检测和防御。对 Linux 而言，由于其开源性，同时支持在内核中加载 LKM（Linux kernel module）这一内核扩展机制，所以从一定程度上讲，Rootkit 对 Linux 内核的修改更加容易实现。在 Windows 系统中实现 Rootkit 的方式较多，通过修改设备的驱动程序来改变某个特定的系统服务调用，通过修改底层函数使之在执行完 Rootkit 函数后再继续执行原函数以绕过上层软件的检测，通过修改直接内核对象操作（direct kernel object manipulation, DKOM）^[1]来隐藏文件、程序、注册表项等。

结合当前网络安全现状和发展趋势，针对 Windows 内核的攻击成为研究的重点和热点。操作系统固有机制的不完善性为攻击的实施提供了可乘之机，然而对于任何一款操作系统而言，如果要彻底地改变已有的工作机制和模式，几乎是不现实的。为此，针对 Rootkit 攻击的防御将是一个长期的漏洞发现、利用和修补过程。

2.3 Bootkit 攻击

自从 1999 年 Greg Hoglund 针对 Windows NT 首次提出 Windows Rootkit 概念^[3]以来，随着研究的深入，Windows Rootkit 的内涵和外延虽然一直不断发展变化，但其具有的深度隐藏、长期潜伏、伺机攻击的特征却没有改变。2005 年，eEye Digital



安全公司首次提出 Bootkit 的概念,将其定义为主要针对硬件或固件攻击的 Rootkit。2011 年出现的“BMW”病毒,通过连环感染 BIOS、MBR 和针对 Windows 的 BIOS Rootkit,能够在操作系统之前获取对主机的控制权,并对抗几乎所有的杀毒软件、HIPS (host-based intrusion prevention) 和审计系统,同时在硬盘上不留任何痕迹,即使重新安装了操作系统甚至更换了硬盘也无法清除。“BMW”病毒使安全界对 Bootkit 攻击有了更进一步的认识。

下面围绕图 1 所示的计算机启动流程,讨论 Windows Bootkit 的实现过程。Bootkit 攻击的主要对象是计算机主板上的 BIOS 芯片和磁盘的 MBR,主要针对运行在实模式下的 BIOS 和 MBR

以及其他引导程序。Bootkit 攻击主要利用挂钩技术,将攻击代码插入正常的引导程序,在计算机开始启动进入操作系统之前,通过劫持原有程序的运行,使攻击者在早于操作系统之前获取对系统的控制权,进而规避系统的安全防护措施,实施对系统的破坏并隐藏自己。

计算机系统在从 BIOS 到 MBR 的引导过程中,分别使用了 INT 13h 和 INT 19h 实模式中断, BIOS 通过 19h 中断从启动盘(硬盘、光盘等)中选取一个启动分区,然后通过 13h 中断将该启动分区中的 Bootloader 加载到内存中,并对 CPU 及相关硬件进行初始化,为操作系统的加载提供所需要的运行环境。攻击者利用了系统启动过程

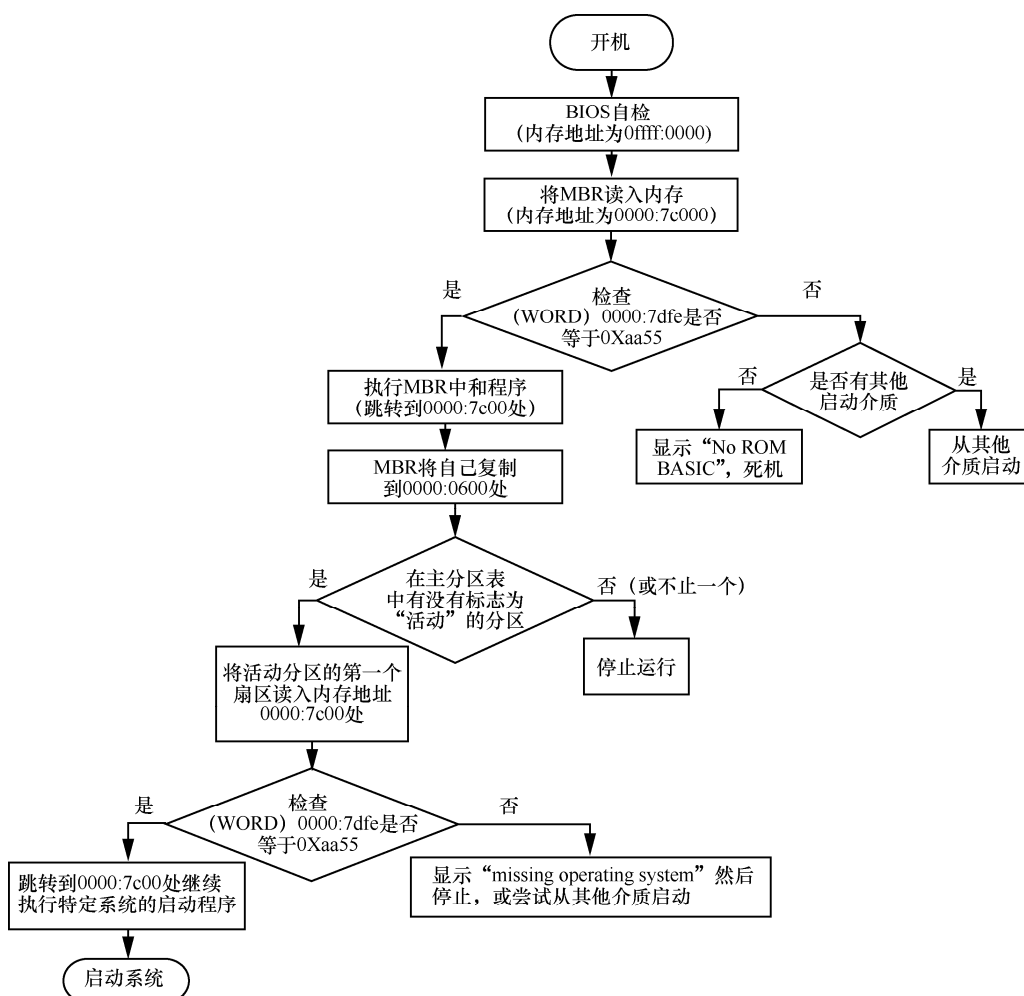


图1 计算机启动流程

中 BIOS 和 MBR 的引导特点: 首先针对具体的 BIOS 文件类型, 将攻击代码插入原有的 BIOS 文件中 (或用新编写的带有攻击代码的 BIOS 文件替换 BIOS ROM 中原有的文件), 在系统启动时将运行带有攻击代码的 BIOS 程序; 然后利用挂钩技术, 系统在调用 19 h 和 13 h 中断时, 首先会执行攻击程序, 结束后再分别返回正常的 19 h 和 13 h 调用。在此过程中, 表面上整个启动过程没有发生任何变化, 一般也不会影响操作系统的正常加载 (由 bootloader 引导操作系统), 但实质上攻击代码通过两次劫持早已运行。更为可怕的是, 攻击者可以通过对两次中断劫持中攻击方式的组合, 实施更复杂、破坏性更强的攻击。例如, 在第 1 次中断劫持后获得对系统底层操作的控制权, 实现对底层环境的控制, 而在第 2 次劫持后可以获取对操作系统的控制权, 实施对内核的 Rootkit 攻击。

3 Rootkit 攻击技术

虽然 Rootkit 可以攻击操作系统的用户模式, 但其核心功能还是以攻击内核模式为主。同时, 为增加技术的针对性, 本文主要结合 Windows Rootkit 进行分析。

3.1 挂钩技术

3.1.1 挂钩技术介绍

Windows 系统采用基于事件驱动的消息传递机制。Windows 系统将事件封闭在消息中, 信息是系统用于告诉应用程序某个事件发生的一个通知。例如, 对于用户移动鼠标、按下键盘等操作, 系统都作为一个事件而产生一个信息, 并将该信息传递给指定的窗口进行处理, 这样基于事件驱动的信息传递机制便实现了 Windows 应用程序 GUI 界面的交互。在此过程中, 系统内部具体执行了哪些操作, 内部的程序又是按什么顺序运行的, 用户并不知道。

挂钩 (Hooking) 技术^[12]是将要执行的具有某种特殊功能的代码 (例如 Rootkit 攻击代码) 作为

外挂程序巧妙地插入目标程序 (被挂钩程序) 中, 当目标程序执行到被挂钩处时强行转向执行外挂程序, 当外挂程序执行结束后再返回目标程序的被挂钩处继续执行目标程序的后续部分。挂钩技术能为用户提供系统或进程中各种事件产生的消息, 并能够根据用户需要改变程序的执行流程, 增加新的功能。也就是说, 挂钩技术为用户访问 Windows 系统程序的结构和执行方式提供了一种途径, 而 Windows 系统的这一工作机制却为实现 Rootkit 攻击创造了条件。攻击者只要能够访问目标对象运行的地址空间, 就可以挂钩并修改其中的任何函数 (例如函数指针、系统调用入口地址等), 在进程打开时这些被修改后的函数被调用执行, 此时将自动跳转到攻击者设置的攻击代码所在的地址去执行, 并实现隐藏进程和端口等功能。例如, 利用挂钩技术, 攻击者可以将木马、后门等恶意代码以驱动程序的形式挂钩到系统的正常启动流程 (如 Windows 的 Winload.exe) 中, 使这些恶意代码在用户根本不知情的情况下随着系统驱动程序的加载而自动运行。

3.1.2 API 函数挂钩攻击

从理论上讲, 不管是用户模式还是内核模式, 只要存在能够挂钩的地方都可以实现基于挂钩的 Rootkit 攻击。API 函数挂钩是最典型的一种挂钩技术^[13]。在 Windows 环境下主要有两种实现 API 函数挂钩的操作: 一种是通过修改可移植执行体 (portable executable, PE) 文件的输入地址表 (import address table, IAT) 使 API 函数地址重定向, 该方式称为基于 IAT 表的挂钩 (IAT Hooking); 另一种是篡改 API 函数地址中的机器码, 用无条件跳转指令 JMP 的机器码来替换 API 函数入口地址中的机器码, 该方式称为 Inline Hooking。

(1) IAT Hooking

输入函数是指允许被程序调用, 但其自身却不在调用程序中的函数。Windows 系统中的输入函数执行体一般位于一个或多个动态链接库



(DLL) 中, 当 PE 文件被调入内存时, Windows 加载程序才会加载 DLL, 即通过 DLL 来调用输入函数。IAT 表中就保存着调用函数与输入函数地址之间的关联信息。如图 2 所示, Rootkit 攻击程序会分析内存中目标程序 (PE 文件) 的结构, 用 GetProcAddress 获取 API 函数的地址, 根据该地址在 IAT 表中查找目标函数 (输入函数) 的地址, 然后用 VirtualProtect 改变内存区域的保护, 之后再用攻击函数地址替换 IAT 表中该条目中目标函数地址。之后, 当目标函数被调用时, 实际执行的是攻击函数, 而非原函数。该攻击方式实现起来较为容易, 但对使用 GetProcAddress 显式调用的 DLL 不起作用。

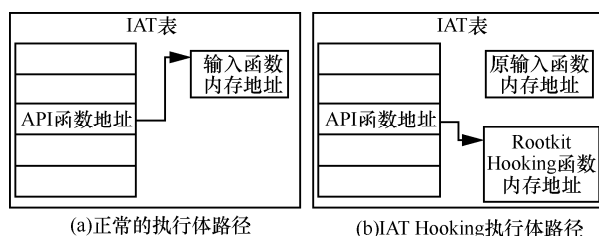


图 2 IAT Hooking 工作过程

(2) Inline Hooking

与 IAT Hooking 通过修改内存中输入函数地址实现到 Rootkit 攻击函数跳转不同的是, Inline Hooking 直接修改 API 函数入口处的机器码使其直接转到 Rootkit 攻击函数。在进行了跳转后, 为了使 API 函数能够顺利执行, 在完成 Rootkit 攻击函数执行后还须返回 API 函数, 接着执行 API 函数后续的代码。Inline Hooking 的工作过程如图 3 所示。该攻击方式的通用性强, 从理论上可以在 API 函数的任何地方把原来指令替换成 Rootkit 攻击者的跳转指令来躲避在线 (Inline) 检测, 但缺点是由于不同操作系统中的机器码可能不同, 所以攻击函数的稳定性和跨平台操作效果较差。

3.1.3 描述符表挂钩攻击

Rootkit 攻击者可以针对 SSDT、IDT、GDT、LDT 等描述符表 (descriptor table, DT) 的挂钩机

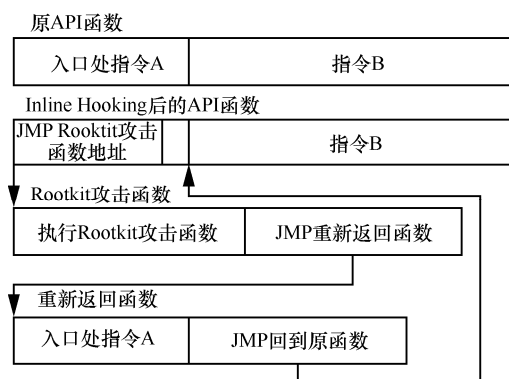


图 3 Inline Hooking 工作过程

制, 对操作系统内核进行攻击。

(1) SSDT 挂钩攻击

系统服务描述符表 (system service descriptor table, SSDT) 是 Windows 系统中实现用户程序调用系统服务的查询表, 即关联用户模式下的 Win32 API 与内核模式下的系统服务函数的关联表。可以被应用程序调用的系统服务函数的服务号及对应的入口地址存放在 SSDT 表中, 这样当应用程序向操作系统发送了一个调用系统服务函数的请求指令后, 操作系统便从 SSDT 表中查询该服务对应的入口地址, 并根据查询结果执行对应的代码。

SSDT Hooking 的实现方法如图 4 所示, 即将 SSDT 表中的某系统服务号对应的入口地址修改为外挂钩子函数的地址, 从而改变系统服务函数原有的执行路径。SSDT Hooking 攻击的实现方法较为简单, 攻击者在确定了某系统服务函数后, 通过修改 SSDT 表中的对应项, 使其指向 Rootkit 代码。根据攻击的需要, SSDT Hooking 可以进行嵌套。

(2) IDT 挂钩攻击

中断描述符表 (interrupt descriptor table, IDT) 挂钩是对系统内核的 Rootkit 技术^[14]。CPU 根据中断号获取对应服务程序的入口地址 (中断向量值), IDT 表就是在内存中建立的一张让 CPU 由中断号查找到对应服务程序入口地址的查询表。IDT 表最大支持 256 个表项, 但每个表项占用的

字节数在操作系统的不同模式下有所不同，实地址模式下为 4 个字节，保护模式下为 8 个字节。根据中断号对异常类型（faults/traps/aborts）的不同，每个表项的组成及功能描述也不相同。

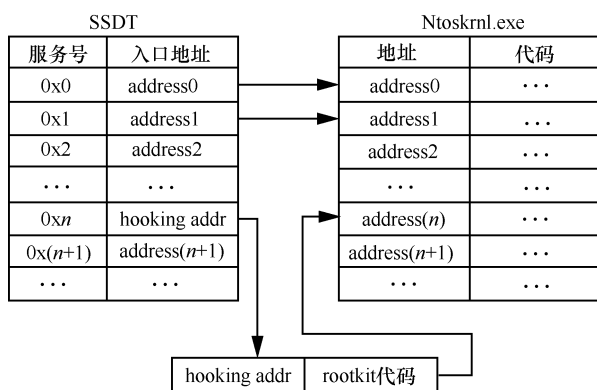


图 4 SSDT Hooking 实现方法

IDT 表由专门的 IDTR 寄存器保存，操作系统可用 SID 指令读出 IDTR 寄存器中的记录信息，用 LIDT 指令将新信息写入 IDTR 寄存器中。IDT Hooking 的实现方法就是用新的表项来替换 IDT 表中已有的表项。Rootkit 攻击者通过用构造的表项替换寄存器中的相应表项，或根据攻击要求写入新的表项，以此来改变系统服务调用的执行流程，并通过劫持和篡改信息达到隐藏自身的目的。

(3) GDT/LDT 挂钩攻击

全局描述符表（global descriptor table, GDT）用于存放描述内存区域的地址以及访问特权的段描述符（如任务状态段 TSS 描述符、数据和代码段描述符等）。GDT 在系统中是唯一的（即一个处理器对应一个 GDT），它由所有程序和任务共享，并可存放在内存的任意位置。Intel x86 处理器用一个 GDTR 寄存器存放 GDT 的入口地址和表长度（GDT 表占用的字节值），分别用 LGDT 和 SGDT 指令向 GDTR 寄存器中加载和读取 GDT 的入口地址。局部描述符表（local descriptor table, LDT）的本质与 GDT 一致，只是每一个进程对应一个 LDT 表，LDT 表的入口地址保存在 LDTR 寄存器中，分别通过 LLDT 和 SLDT 指令加载和

读取。

在 Windows 系统默认状态下，GDT 和 LDT 都包含有调用门（call-gates）入口，GDT/LDT Hooking 的实现方式是在描述符表保留字段中插入一个调用门描述符来改变内存段的执行特权，再借助相关技术实现进程隐藏。由于每个实际运行的处理器分别对应一个 GDT 表，每个线程可以获得其对应进程所包含的 LDT 表的相应内容，在 GDT/LDT Hooking 攻击中，攻击者首先会设计一个包含 Rootkit 攻击代码的调用门，然后将该调用门插入 GDTR 寄存器的 GDT 指定位置。同时，当一个任务发生切换时，包含 Rootkit 攻击代码的线程管理器用攻击程序线程的 LDT 表替换正在运行线程的 LDT 表，实现对线程的隐藏。

(4) IRP 挂钩攻击

I/O 请求包（I/O request package, IRP）是 Windows 内核中一个预定义的数据结构，一个 IRP 由一个固定的首部和不定数目的 IRP 栈单元块组成，IRP 栈为先进后出的向下生长的栈。当上层应用程序需要调用底层的 I/O 设备时，应用程序便发送一个 I/O 请求，I/O 管理器首先将其转换为一个 IRP，然后传送到合适的驱动程序栈中的不同派遣例程进行处理。IRP 用 Io Get Current Irp Stack Location 函数获取指向当前栈单元的指针，使用 Io Copy Current Irp Stack Location To Next 或 Io Skip Current Irp Stack Location 函数把当前栈单元复制到下一个栈单元。

基于 Windows 系统的层次模型和 IRP 栈先进后出的工作机制，驱动程序使用 Io Set Completion Routine 函数挂接一个完成例程，而该完成例程信息存储在对应驱动程序的 IRP 栈单元中。攻击者将 Rootkit 驱动程序插入驱动程序栈中，在截获合法驱动程序后对其信息进行修改和过滤，从而达到隐藏文件、嗅探击键和鼠标操作等目的。

3.2 DKOM 技术

前文介绍的 API 函数挂钩攻击和描述符表挂



钩攻击都是利用被攻击对象的工作机制,通过修改程序执行流程或重定向指令等方式来实现 Rootkit 攻击。而基于 DKOM^[15]的 Rootkit 攻击技术通过直接修改 Windows 系统的设备驱动程序或可加载内核模块,以实现进程、文件和网络连接的隐藏和进程提权^[20]。

以 Rootkit 攻击者实现 Windows 系统进程隐藏为例。当创建一个进程时,系统会给该进程建立一个对应的内核 EPROCESS 结构,该结构的第一个成员为 KPROCESS;当创建一个线程时,系统同样会建立一个对应的内核 ETHERAD 结构,该结构的第一个成员为 KTHREAD。在 EPROCESS 结构体中,有一个类型为 LISTENTRY 结构体的成员 ActiveProcessLinks,它是一个拥有 FLINK 和 BLINK 两个指针成员的双链表节点,两个指针成员分别指向前后两个进程 EPROCESS 结构体的 ActiveProcessLinks 成员。当需要隐藏 Rootkit 攻击代码的进程时,只需要把该进程的 EPROCESS 结构体从双向链表中移除即可,如图 5 所示。

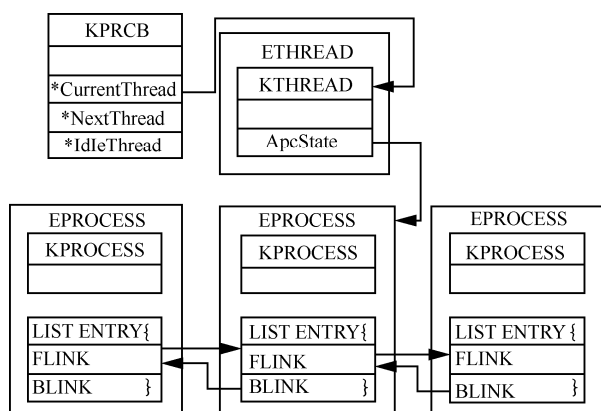


图 5 基于 DKOM 的 Rootkit 攻击原理

3.3 虚拟化技术

早期的 Rootkit 一般通过 Hooking 技术改变程序的执行路径、过滤系统调用的返回信息,或直接采用 DKOM 技术篡改系统内核对象来实现攻击过程,对操作系统的运行具有较大的依赖性,

所以无法彻底消除 Rootkit 攻击过程产生的痕迹,容易被基于操作系统的检测技术发现。硬件虚拟化技术的出现导致了与操作系统无关的恶意软件的出现,如基于虚拟机的 Rootkit (virtual machine based Rootkit, VMBR)。虚拟化 (virtualization) 技术分离了传统的软件与硬件,可支持多种同构或异构的软件(如操作系统)共享同一硬件设施。基于虚拟化技术的 Rootkit 可以将攻击代码插入到原有的软件与硬件之间,使传统的基于操作系统调用的检测方式失效,从而实现深度隐藏的目的。

3.3.1 VMM Rootkit

在虚拟机中,虚拟机管理器 (virtual machine manager, VMM)^[16]是一个运行在硬件与操作系统之间的中间件层,主要功能是对底层物理主机平台的资源进行仲裁,这样多个操作系统 (VMM 客户机) 可以共享同一硬件设施。VMM 为每个虚拟客户机操作系统提供一组虚拟平台接口,构成虚拟机 (virtual machine, VM)^[17]。根据实现方式的不同,可以将 VMM 分为宿主模型 (host-based model)、监控模型 (hypervisor model) 和混合模型 (hybrid model) 3 种类型。其中,在宿主模型中 VMM 运行在宿主操作系统上并利用其提供的设备驱动及底层服务,客户操作系统需要借助宿主操作系统的 VMM 而运行;而在监控模型中 VMM 直接运行在物理硬件上,且拥有特权级;混合模型是对宿主模型和监控模型的集成,以发挥 VM 的应用功能。

基于宿主模型的 VMM Rootkit 通过在宿主操作系统上安装一个 VMM,然后将攻击代码构建在一个 VM 上,并利用 VMM 的底层优势实现对宿主操作系统的监视和完全控制,如截获网络数据分组、读取硬盘或内存中的任意信息等。由于运行有攻击代码的 VM 与宿主操作系统之间完全隔离,致使基于宿主操作系统的安全检测工具无法检测到 VM 上攻击代码的存在,实现了深度隐藏。Subvirt^[18]是一种典型的基于

宿主模型的 VMM Rootkit，主要攻击对象为 Windows 和 Linux 系统，它在重启宿主操作系统时优先加载攻击代码，并长期驻留于硬盘的引导分区。

基于监控模型的 VMM Rootkit 是在硬件虚拟化技术的支持下，实现对客户操作系统内核驱动程序的修改，根据攻击需要插入中断或改变某一程序的执行流程、建立隐蔽通信通道等，甚至直接访问客户操作系统的 I/O 和内存并对信息进行过滤、监视或记录，也可构建一些敏感指令对运行于 VMM 上的客户操作系统进行完全控制并实现隐藏。目前，已经提出了多种 VMM Rootkit，例如 BluePill^[19]在目标操作系统（Windows 或 Linux）运行时攻击代码以驱动程序方式加载，而 Vitriol 主要针对 MacOS 操作系统，并在系统运行时以可装载模块方式加载。

3.3.2 SMM Rootkit

系统管理模式（system management mode, SMM）^[20]是 Intel 系列 CPU 支持的 4 种模式之一，它拥有最高权限，独立于操作系统，专门用于电源管理、温度调节等底层硬件控制。SMM 为 CPU 不同模式之间的切换提供了一种透明机制：当任一程序或硬件发出系统管理中断（system management interrupt, SMI）时，CPU 首先将当前寄存器的值临时存储到 SMRAN 中，然后切换到 SMM 模式，并在一个分离的地址空间执行 SMM 指定的代码；当 SSM 需要返回时将发出 RSM 指令，回到被系统管理中断之前的状态，同时恢复 CPU 寄存器的值。

SMM 其实是 Intel 系列 CPU 预留的一种特权模式，它拥有外部程序不可见的独立的内存区域和隔离的运行空间，而且不会受到内存保护模式和优先级的限制，操作系统无法感知到 SMM 中断的发生，这就为 SMM Rootkit 攻击提供了深度隐藏所需要条件，同时可以对网卡、硬盘、键盘等外围设备进行控制。

4 Rootkit 检测和防御技术

4.1 Rootkit 检测方法

作为恶意代码家族中的新成员，目前针对 Rootkit 的检测方法和防御技术研究还是一个具有挑战性的领域，需要在继承已有研究成果的基础上，针对新的攻击手段和对象不断改进和完善。根据研究对象主体特征的不同，针对 Rootkit 的检测方法可分为基于软件的检测法和基于硬件的检测法两种类型。

4.1.1 基于软件的检测法

基于软件的检测法分为行为检测法、完整性检测法、执行时间检测法、执行路径检测法和差异检测法 5 种类型^[21-22]。

（1）行为检测法

行为检测法通过对比分析已有 Rootkit 行为特征与当前提取到的特征信息来决断是否发生了 Rootkit 攻击行为。该方法是基于对已有 Rootkit 攻击行为特征库的建立，检测过程首先对目标对象（内存、内核文件等）进行扫描，然后在库中进行比对，看是否存在相吻合的项。行为检测法的特点是对已有 Rootkit 攻击行为的检测精确度较高，但前提是需要完善的特征库为支撑，所以对于一些新型或未知的 Rootkit 攻击很容易被绕过。

（2）完整性检测法

完整性检测通过检测系统文件或内核文件的完整性来判断是否存在 Rootkit 攻击。该方法需要建立完善的针对系统不同版本在可信状态下的重要文件特征库（Hash 值），在检测时可提取当前状态下系统对应文件的特征值，并与库中对应的特征值进行比对，进而确定是否发生了 Rootkit 攻击。该方法几乎可以检测出任何一种 Rootkit 攻击行为，而且简单易行，但由于需要事先计算所有可能遭受 Rootkit 攻击对象的特征值，实现起来相对比较困难。

（3）执行时间检测法

执行时间检测法是对系统运行时所启用的服



务指令的执行时间进行计数,以判断是否有其他的代码执行。该方法需要在系统创建时备份原始系统调用函数,并统计其正常状态下的执行时间。如果发生了针对内核的 Rootkit 攻击,不管该攻击采用 Hooking 或 DKOM 方式,原始服务指令的执行时间就会增长,而且代码的执行路径也会发生变化。该方法可以检测出已知和未知的 Rootkit 攻击,但会因计时器(如本地时钟、外部计时器等)的误差产生误报和漏报。

(4) 执行路径检测法

执行路径检测法是通过比较分析某个系统服务在可信系统(未被 Rootkit 攻击的系统)和待检测系统上运行指令数的差异来判断是否存在 Rootkit 攻击。该方法的实现原理是:当某一系统服务隐藏了 Rootkit 攻击对象时,系统在调用该被攻击服务时将会执行一些额外的指令,导致总的指令数必然会增加。该方法从理论上讲是可行的,但在指令执行时 CPU 需要处于单步模式,频繁的中断操作会严重影响系统的性能,可操作性较差。

(5) 差异检测法

差异检测法是通过分析比较从系统不同层次获得的进程、网络连接、文件目录等列表信息,再根据相互间存在的差异来检测出是否存在 Rootkit 攻击。例如,一条信息(如进程)在底层列表中出现,但是却没有出现在高层列表中,就说明该信息已经被 Rootkit 隐藏。检测对象的确定和列表信息的获取是差异检测法的关键。由于攻击代码在运行时一般都需要创建对应的进程,而绝大多数 Rootkit 攻击行为都要将隐藏进程作为必备的手段,所以被隐藏的进程必然是差异检测的主要对象;列表信息的获取需要在确保对象可信的前提下,尽可能获取到最底层的信息。例如,基于交叉视图的差异检测法从理论上讲可以检测出已知和未知的 Rootkit 攻击,但前提是系统信息的获得必须是全面和可信的^[22]。

4.1.2 基于硬件的检测法

近年来,随着 Rootkit 技术越来越向底层渗透,并实现了优先于操作系统和检测软件启动等深度攻击行为,致使基于软件的检测方法失去了功效。在此情况下,基于硬件的 Rootkit 检测和防御方法成为一个关注领域和研究方向。

目前,基于硬件 Rootkit 攻击的检测研究已经取得了一些成果。例如,利用直接内存访问(direct memory access, DMA)技术让硬件检测设备直接对物理内存数据进行复制,然后对复制品分析是否存在异常行为^[23]。根据检测需要设计的内存控制器可以实现特定的外部设备利用 DMA 访问物理内存并进行 Rootkit 检测。该技术的实现需要相应条件的支持(如 CPU 等硬件需要支持 DMA 操作),同时为了防止 DMA 物理内存存在复制时被 Rootkit 攻击,在复制过程中目标主机的 CPU 需要处于暂停状态。另外,基于硬件的实时扫描技术可以对主机的文件系统和物理内存进行实时监控,及时发现系统的运行异常^[24]。不过,现有研究成果多为理论探讨,具体的实践应用尚少。

4.2 Rootkit 防御技术

网络攻击与防御是一个既对立又统一的过程,其中检测是防御的前提和重要组成部分。下面在 Rootkit 检测方法的基础上,继续讨论其防御技术。

4.2.1 固件级防御

对于类似于针对 BIOS 的固件级 Rootkit 攻击,由于该类攻击最接近系统的底层,而且在操作系统启动之前就已完成了攻击代码的加载和隐藏,致使在重装系统或格式化硬盘后仍然无法清除植入的攻防代码。针对此类攻击行为,最有效的防御方法是与 Rootkit 攻击之间争夺启动的优先权,即让 Rootkit 防御代码优先于攻击代码加载,使 Rootkit 防御代码优先取得对整个系统的控制权,进而拦截 Rootkit 攻击代码的加载,阻止攻击

行为的发生。

可信计算平台^[25]分别将 BIOS 引导模块和 TPM 作为完整性度量 and 完整性报告的可信根,通过创建一条“CRTM→BIOS→OSLoader→OS→Application”的信任链,通过先度量再移交控制权的方式,确保了每一个环节的可信性,为防御固件级和用户级的 Rootkit 攻击提供了一种行之有效的解决方案。

4.2.2 用户级防御

由于用户级 Rootkit 攻击对象主要是运行在用户模式下的系统程序和用户应用程序,包括系统 DLL 文件和应用程序的二进制文件,以及一些跳转表等。这一类攻击处于系统较低的特权级,攻击代码的隐藏较为困难,已有的针对文件型恶意代码的检测和防御技术能够有效地应对。例如,针对应用程序或文件的 Rootkit 攻击,检测和防御程序只需要对比分析原始对象和疑似被攻击对象之间的异同就可以作出判断和处理。此类 Rootkit 攻击现象目前已很少发生。

4.2.3 内核级防御

由于内核级 Rootkit 攻击行为发生在操作系统的内核空间中,其攻击代码多以 Windows 驱动程序或 Linux LKM 形式以 Hooking 或 DKOM 等方式加载,并成为操作系统的一部分,且拥有系统的最高特权。与用户级 Rootkit 不同的是,内核级 Rootkit 不但能够实现自身的深度隐藏,而且能够对系统内核及运行在操作系统上的检测工具进行任意修改,破坏性更大,而且常规的检测方法难以检测到其存在,防御难度大。

针对内核级 Rootkit 攻击的防御思路是实现操作系统内核模块与外部调入程序(如 Rootkit 攻击代码)之间的有效隔离。这样,当 Rootkit 攻击代码通过 Windows 驱动程序或 Linux LKM 方式试图加载系统内核时,就可以快速地进行判断和清除 Rootkit 攻击代码。例如,微软公司在其 64 位 Windows 操作系统中就引入了 PatchGuard 技术,

它是 Windows 操作系统的一个安全保护层,可对内核关键位置进行检测,并引入了 DES 实现对驱动程序的数字签名。该机制有效防止了 Hooking 和 DKOM,进而避免了任何非授权试图修改 Windows 内核行为的发生。

4.2.4 虚拟级防御

虚拟化技术的出现和虚拟机的广泛应用为 Rootkit 攻击提供了一条新的途径。由于虚拟级 Rootkit 运行在物理硬件与操作系统之间,拥有比操作系统内核更高的特权级,可以对操作系统进行完全控制和操作,其危害程度可想而知。

针对虚拟级 Rootkit 攻击的防御研究,目前已经取得了一些成果。例如,参考文献[26]提出了基于硬件辅助虚拟化(hardware assisted virtualization, HAV)技术,通过引入新的操作系统模式,使客户操作系统以较低的特权级运行,有效保护系统的内存等资源,防止 Rootkit 攻击行为的发生;参考文献[27]针对现有内核级 Rootkit 防御具有的局限性,提出了一种在云环境下的端到端防御技术,位于客户机内核空间中的底层管理程序通过识别可疑的代码区域,对被检测的 Rootkit 代码进行诊断,可有效检测和防御针对客户机内核的 Rootkit 攻击。

5 结束语

作为一类特殊的恶意代码,Rootkit 自出现之日起就紧随着软硬件技术的发展而不断推陈出新。基于计算机系统的层次模型,Rootkit 技术实现了从软件到硬件、从用户模式到内核模式、从外部存储空间到物理内存的自顶向下的参透;基于恶意代码的特征,Rootkit 技术在继承了家族成员的程序复制、信息隐藏、数据伪造与清除等功能的基础上,其攻击过程的复杂性对防御带来了极大的挑战;基于应用领域的不同,Rootkit 技术不仅攻击 Windows、Linux 等桌面和服务器操作系统,而且正在向着 Android、iOS、Windows Phone



等移动智能终端操作系统快速扩散, 攻击目标也从传统的互联网扩展到物联网、工业控制系统等环境。综上, Rootkit 攻击与防御技术之间的博弈, 还将随着技术的发展和演进不断深入, 并伴随着系统越来越复杂和应用越来越多样这一趋势而更加难以预测和控制。

参考文献:

- [1] JERLIN M A. A review on advanced evasion techniques[J]. International Journal of Pharmacy & Technology, 2016, 8(4): 4917-4924.
- [2] XIA H, XU Y, XIA H, et al. Design and research of safety test model based on advanced evasion techniques[C]//Global Conference on Mechanics and Civil Engineering (GCMCE2017), April 21-23, 2017, Nanjing, China. [S.l.:s.n], 2017: 92-96.
- [3] GURI M, POLIAK Y, SHAPIRA B, et al. JoKER: trusted detection of Kernel Rootkits in android devices via JTAG interface[C]//IEEE Trustcom/BigData/ISPA, Aug 20-22, 2015, Helsinki, Finland. Washington DC: IEEE Computer Society, 2015: 65-73.
- [4] DAMOPOULOS D, KAMBOURAKIS G, GRITZALIS S. iSAM: an iPhone stealth airborne malware[J]. IFIP Advances in Information & Communication Technology, 2017: 17-28.
- [5] VETTER J, JUNKER-PETSCHICK M, NORDHOLZ J, et al. Uncloaking Rootkits on mobile devices with a hypervisor-based detector[C]// International Conference on Information Security and Cryptology, March 10, 2016, Seoul, South Korea. Berlin: Springer Press, 2016: 262-277.
- [6] 张瑜, 刘庆中, 李涛, 等. Rootkit 研究综述[J]. 电子科技大学学报, 2015, 44(4): 563-578.
ZHANG Y, LIU Q Z, LI T, et al. Research and development of Rootkit[J]. Journal of University of Electronic Science and Technology of China, 2015, 44(4): 563-578.
- [7] RICHER T J, NEALE G, OSBORNE G. On the effectiveness of virtualisation assisted view comparison for Rootkit detection[C]//The 13th Australasian Information Security Conference July 7-9, 2008, Sydney, Australia. Berlin: Springer Press, 2008: 35-44.
- [8] DAWSON J A, MCDONALD J T, SHROPSHIRE J, et al. Rootkit detection through phase-space analysis of power voltage measurements[C]//International Conference on Malicious and Unwanted Software, Oct 11-14, 2017, Fajardo, Puerto Rico. Piscataway: IEEE Press, 2017: 19-27.
- [9] YANG H Y, ZHUGE J W, LIU H M, et al. A tool for volatile memory acquisition from android devices[C]//12th IFIP WG 11.9 International Conference, September 20, 2016, New Delhi, India. Berlin: Springer, 2016: 365-378.
- [10] JOY J, JOHN A, JOY J. Rootkit detection mechanism: a survey[J]. Communications in Computer & Information Science, 2011(203): 366-374.
- [11] SYED R, GABRIEL L, MATT G, et al. Advocating for hybrid intrusion detection prevention system and framework improvement[J]. Procedia Computer Science, 2016(95): 369-374.
- [12] SUN H M, WANG H, WANG K H, et al. A native APIs protection mechanism in the Kernel mode against malicious code[J]. IEEE Transactions on Computers, 2011, 60(6): 813-823.
- [13] SHAID S Z M, MAAROF M A. In memory detection of Windows API call hooking technique[C]//International Conference on Computer, Communications, and Control Technology, April 21-23, 2015, Kuching, Malaysia. Piscataway: IEEE Press, 2015: 294-298.
- [14] WANG Y, GU D, LI W, et al. Virus analysis on IDT hooks of Rootkits Trojan[C]//International Symposium on Information Engineering and Electronic Commerce, May 6-17, 2009, Ternopil, Ukraine. Washington DC: IEEE Computer Society, 2009: 224-228.
- [15] PAN M R, CAO T J. Research on process hiding technology based on direct kernel object manipulation[J]. Computer Engineering, 2010, 36(18): 138-140.
- [16] GRANISEWSKI W, ARCISXEWKI A. Performance analysis of selected hypervisors (virtual machine monitors-VMMs)[J]. International Journal of Electronics & Telecommunications, 2016, 62(3): 231-236.
- [17] UHLIG R, NEIGER G, RODGERS D, et al. Intel virtualization technology[J]. Computer, 2005, 38(5): 48-56.
- [18] KING S T, CHEN P M. Sub Virt: implementing malware with virtual machines[C]//2006 IEEE Symposium on Security and Privacy, May 21-24, 2006, Berkeley/Oakland, CA, USA. Washington DC: IEEE Computer Society, 2006: 314-327.
- [19] SERGEEV A, MINCHENKOV V, BASHUN V. Malicious hypervisor and hidden virtualization of operation systems[C]//International Conference on Application of Information and Communication Technologies, Oct 14-16, 2015, Rostov on Don, Russia. Piscataway: IEEE Press, 2015: 178-182.
- [20] EMBLETON S, SPARKS S, ZOU C. SMM Rootkits: a new breed of OS independent malware[J]. Security & Communication Networks, 2013, 6(12): 1590-1605.
- [21] LUCKETT P, MCDONALD J T, DAWSON J. Neural network analysis of system call timing for Rootkit detection[C]// Cyber Security Symposium, April 18-20, 2016, Coeur d'Alene, ID, USA. Piscataway: IEEE Press, 2016.
- [22] CASE A, III G G R. Advancing mac OS X Rootkit detection[J]. Digital Investigation, 2015(14): S25-S33.
- [23] ECKERT M, PODEORAD I, KLAUER B. Hardware based security enhanced direct memory access[Z]. 2017.

- [24] ZHU J, ZHOU T, WANG Q. Towards a novel approach for hidden process detection based on physical memory scanning[C]//Fourth International Conference on Multimedia Information Networking and Security, Nov 2-4, 2012, Nanjing, China. Washington DC: IEEE Computer Society, 2012: 662-665.
- [25] MAENE P, GOTZFRIED J, CLERERCQ R D, et al. Hardware-based trusted computing architectures for isolation and attestation[J]. IEEE Transactions on Computers, 2018, 67(3): 361-374.
- [26] DESNOS A, FILIOL E, LEFOU I. Detecting (and creating!) a HVM Rootkit (aka blue Pill-like)[J]. Journal in Computer Virology, 2011, 7(1): 23-50.
- [27] ZHANG L, SHETTY S, LIU P, et al. Rootkit Det: practical end-to-end defense against kernel Rootkits in a cloud environment[C]//European Symposium on Research in Computer Security, September 10, 2014, Wroclaw, Poland. Berlin: Springer, 2014: 475-493.

[作者简介]



李馥娟 (1974-), 女, 江苏警察学院副教授, 主要研究方向为计算机网络技术与应用、物联网、信息安全等。



王群 (1971-), 男, 博士, 江苏警察学院教授, 主要研究方向为网络体系结构与协议、信息物理融合系统、信息安全等。