LKM 后门综述*)

袁 源 戴冠中
(西北工业大学自动化学院控制与网络研究所 西安 710072)

摘 要 LKM后门作为Linux下危害最大的恶意代码,运行在内核层,比传统技术下的后门更隐蔽,功能更强大。本文分析LKM后门的技术原理与威胁,并在此基础上研究各种后门检测方法。这些方法都有局限性,因此多方法融合、有机组合互补将成为LKM后门检测的发展趋势。

关键词 Linux,可装载内核模块,后门,系统调用,检测

Survey on LKM Backdoors

YUAN Yuan DAI Guan-zhong
(College of Automation, Northwestern Polytechnical University, Xi'an 710072, China)

Abstract LKM backdoors run in kernel as the most dangerous evil code. It is more secluded and stronger than traditional backdoors. This paper analyses the principle and threaten of LKM backdoors. Some detecting methods are researched based on it. These methods have their own limitations, so organically combining several methods will be the development trend for detecting LKM backdoors.

Keywords Linux, Loadable kernel modules, Backdoors, System call, Detecting

1 引言

自从1991年芬兰赫尔辛基大学的Linus Torvalds在新闻组comp. os. minix上发布了大约有一万行代码的Linux v0.01版本以来,在全世界Linux爱好者与程序员的共同努力下,Linux已经发展成为一个功能强大、设计完善的通用操作系统,受到越来越多计算机用户的认同。Linux内核借用了微内核思想的整体式结构,具有很高的运行效率,同时利于移植、定制。而其开放源代码(Open Source)的开发模式,保证很多系统漏洞都能被及时发现与纠正。另外通过对读/写进行权限控制、带保护的子系统、审计跟踪、核心授权等方式,有效提高了系统的安全性。目前,Linux操作系统在高端服务器、桌面应用、嵌入式应用和超级计算机等领域都占有一定市场份额,这一切都源于其高效、稳定、安全的特点。

随着受关注程度的日益增加,黑客也将目光对准了Linux。如今,Linux操作系统再也不能被称为绝对安全的操作系统,已经有许多针对Linux的病毒、蠕虫、木马和后门出现。后门工具中,危害最大的就是LKM(Loadable Kernel Module,可装载内核模块)后门[1]。它利用现代操作系统模块技术,作为内核的一部分运行起来,可以实现隐藏文件、隐藏进程、隐藏网络连接、重定向可执行文件、端口复用等功能。由于并不增加、修改、替换任何可执行文件,它比传统技术上的后门更加隐蔽,一旦被安装并运行在目标机器上,系统就会完全由黑客操纵,管理员很难找到任何存在安全隐患的痕迹[2]。因此,LKM后门及其检测技术的研究已经成为Linux安全性研究的热点。

本文其它部分组织如下:第2节介绍LKM技术,主要是LKM的组成、使用方法和特点。第3节介绍LKM后门原理,分析截获系统调用的实现步骤以及如何通过截获系统调用实现各种后门功能。第4节介绍LKM后门的检测方法,分别讨论各种检测方法的原理与优缺点。最后总结本文的工作并对检测方法的未来发展方向进行展望。

2 LKM技术

LKM是指可装载内核模块,Linux系统自内核版本2.0开始提供对该技术的支持,用于扩展内核功能。LKM被单独编译成为目标代码,形成一个目标文件,它是核心的一部分,但是并没有编译到核心里面去。LKM可以根据用户需要在系统启动后动态地加载到核心中,当其不再被需要时,可以随时卸载出核心。使用LKM的优点是,通过动态地将代码加载到内核可以减小核心代码的规模,使核心配置更为灵活;若在调试新核心代码时采用LKM技术,程序员不必每次修改代码后都重新编译内核和重启系统[3]。由于这些优点,LKM技术常常用在设备驱动程序的开发和测试中。

每个LKM至少由两个基本的函数组成:init_module()和cleanup_module()。init_module()通常是为内核注册一个程序或是利用自身代码来取代某个内核函数;cleanup_module()的任务是清除init_module()所做的工作,以确保该模块可以安全地卸载。

insmod命令用于加载内核模块,执行该命令后,init_module()将被执行,完成初始化操作。若某个模块空闲,用户可将它卸载出内核。rmmod命令用于卸载内核模块,此时cleanup_module()将被执行。只有root用户才能执行这两条指令。另外,显示当前内核中运行的模块命令是lsmod。

一旦LKM被载入核心,它就成为核心代码的一部分,可以访问核心符号表所指定的资源。Linux允许模块堆栈,即

*)基金项目:航空科学基金(01F53031);教育部博士点基金(20020699026)。袁 源 博士研究生,主要研究方向为网络信息安全等;戴冠中 教授,博士生导师,主要研究方向为自动控制、网络信息安全。

一个模块可请求其它模块为之提供服务，通过这种方法，新载入的模块可以访问已装载模块提供的资源[3]。

3 LKM后门原理

3.1 用户空间与内核空间

Linux是一个具有保护模式的操作系统，工作在i386CPU的保护模式之下。内存被分为两个单元：内核空间和用户空间。Linux使用段选择器来区分用户空间和内核空间。核心代码运行在内核空间，用户进程运行在用户空间[4]。作为用户进程来讲，它不能访问内核空间以及其他用户进程的地址空间。核心代码也同样不能访问用户进程的地址空间。例如一个硬件设备驱动试图直接写数据到一个用户空间的程序里是不允许的。但是，它可以利用一些特殊的核心函数来间接完成[1]。同样，当用户进程传递参数到核心时，核心也不能直接地读取该参数，它必须利用一些特殊的核心函数如get_user()，put_user()等来接收参数[1,5,6]。

3.2 系统调用

每个操作系统在内核中都有一些最为基本的函数给系统的其它操作调用。在Linux系统中这些函数就被称为系统调用(System Call)，它们代表从用户级别到内核级别的转换。用户程序通过系统调用向操作系统内核请求服务，操作系统内核完成该服务后，将结果返回给用户进程。系统调用提供了一种访问核心的机制，这种机制提高了系统的安全性，同时保证了应用程序的可移植[7]。在/usr/include/sys/syscall.h中有一个完整的系统调用列表。

每个系统调用都有一个预定义的数字，称为系统调用号，这些数字是sys_call_table(系统调用表，定义在arch/i386/kernel/entry.S中)在内核中的数组结构的索引值。这个结构把系统调用的数字映射到实际使用的函数。内核通过中断0x80来控制每一个系统调用，将所有参数放入寄存器。特别地，系统调用号装入EAX寄存器。

3.3 LKM后门工作原理

当用户进程使用某个系统调用函数时，实际执行的是libc标准库中的封装函数。封装函数将参数装入对应的寄存器并调用int 0x80中断指令进入系统核心。操作系统接收到中断请求后，会根据中断向量找到中断描述符表中对应的描述符，在进行优先级校验后，根据段选择码和段内偏移找到并调用system_call()中断服务子程序。system_call()保存寄存器值，进行系统调用号的有效性检验，根据系统调用号在系统调用表找到并调用相应的系统调用服务程序[1,8]。系统调用服务程序执行结束后，将返回值存入EAX寄存器。操作系统调用ret_from_sys_call()返回。例如，处理sys_write()系统调用的函数执行链如图1所示。

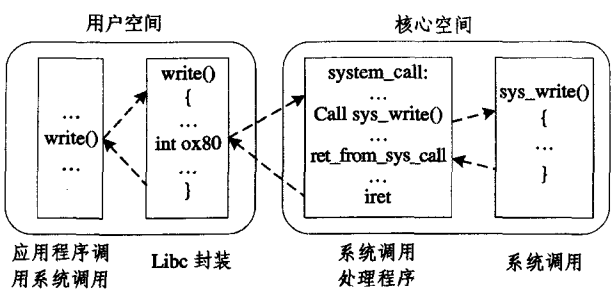

图1 处理sys_write()系统调用的函数执行链

命令ls用于查看当前目录信息，它会调用sys_open()、sys_gendents64()、sys_write()等一系列系统调用，系统核心在处理完每个系统调用后都将结果返回给用户空间的ls命令。如果此时截获sys_write()这个系统调用，修改它的处理函数，那么现在的输出信息就不是真正的系统信息。通过对中断描述符表与系统调用表的分析可以发现，每一个表项中都保存有对应处理函数的地址，只要修改处理函数的地址，使它指向自定义的处理函数，就可以达到函数劫持的目的。LKM后门最大的特点就是截获并修改多个系统调用，从而改变整个系统响应。

利用LKM技术截获系统调用的通常步骤如下：(1)找到需要的系统调用在sys_call_table[]中的入口；(2)保存sys_call_table[x]的旧入口指针(x代表所想要截获的系统调用号)；(3)将自定义的新的函数指针存入sys_call_table[x][9]。完成上述步骤之后，原系统调用被截获。例如，sys_write()系统调用被截获后的函数执行链如图2所示。此时，hack_write()的执行结果将代替sys_write()的执行结果返回给用户空间。

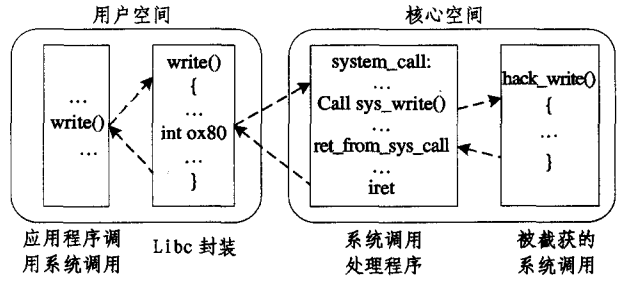

图2 截获系统调用后的函数执行链

需要指出的是，出于安全性的考虑，Linux在内核版本2.4.18以后不再导出sys_call_table[]，原先的模块在编译时会提示找不到sys_call_table[]的入口地址。不过，可以通过读/dev/kmem设备文件计算出sys_call()函数的地址，然后在sys_call()函数内通过模式匹配获得sys_call_table[]的入口地址，从而实现在高于内核版本2.4.18的情况下对系统调用的劫持[10]。

对于众多的系统调用，不需要全部劫持，只需要劫持系统管理员用于查看文件、进程、网络连接等命令用到的关键系统调用即可。下面分析LKM后门在设计时主要截获哪些关键的系统调用。

3.3.1 隐藏文件

strace命令用于跟踪某命令使用了哪些系统调用。为了隐藏文件，需要跟踪ls命令。通过“strace ls”可以发现ls命令利用sys_getdents64()系统调用来获取某指定目录下文件的信息，然后将结果传回给用户空间的ls命令。因此这里需要劫持sys_getdents64()系统调用。在伪造的sys_getdents64()中首先调用原系统调用，在其返回的文件目录信息中删除需要隐藏的文件或目录信息，再将修改后的文件目录信息返回给用户。这样，所有使用sys_getdents64()系统调用命令的输出中将不再有指定文件的信息。

3.3.2 隐藏进程

LKM后门不会在运行过程中产生任何进程，但是黑客往往需要在目标主机上运行一些恶意程序，如Sniffer，Trojan Horse等等，这些进程需要隐藏。Linux中不存在直接查询进程信息的系统调用，类似ps这样查询进程信息的命令是通过

查询/proc目录来实现的。/proc目录下有许多由十进制数字组成的目录,它们包含系统中所有运行进程的信息,数字对应于PID(Process ID,进程标识符)。由于/proc应用文件系统的接口实现,因此同样可以用隐藏文件的方法来隐藏进程,只需要加入对/proc文件系统的判断即可。另外一种简单的方法是在内核进程链中给指定进程加入标记(token),然后在sys_getdents64()系统调用中通过查看该标记位来决定是否将进程信息返回给ps。

3.3.3 隐藏网络连接

对于Trojan Horse来说,除了隐藏进程,还必须隐藏网络连接,这样才不会给系统管理员留下任何痕迹。Linux系统使用netstat命令查看网络连接状态。通过"strace netstat"可以发现它是读取/proc/net/tcp和/proc/net/udp两个文件来获取网络连接信息的。因此,只需要替换sys_read()系统调用,屏蔽从这两个文件中读取的指定网络连接信息,从而实现对指定IP、指定通信端口的隐藏。还有一种隐藏方法是截获sys_write()系统调用,屏蔽指定信息的输出[2]。

3.3.4 隐藏后门模块本身

为了防止系统管理员通过lsmod命令发现异常的模块,后门模块本身也需要隐藏。有两种方法实现该功能。一是通过"strace lsmod"可以发现查询系统中加载的模块命令使用了sys_query_module()系统调用,因此截获这个系统调用,屏蔽指定的输出即可。另一种方法是由于Linux中的模块组织方式是单向链表,新加载的模块位于module.list的表头,把对应的模块从链表中除去即可达到隐藏的目的。但是操作系统没有将module.list在内核中输出,无法直接操纵module.list。因此比较流行的方法是在加载后门模块后,立刻再加载一个新的模块,此时新加载模块的next指针本应指向后门模块。不过编程时让新模块的next指针指向后门模块的next指针,就可以实现跳过后门模块。将新模块卸载后,由于后门模块不在module.list中,因此lsmod命令看不见后门模块的存在。但是它并没有被卸载,其功能仍然存在。

3.3.5 可执行文件重定向

通过劫持系统调用sys_execve()可以实现可执行文件的重定向。其主要思想是修改系统调用表中的某一空项,使其指向原系统调用sys_execve(),然后截获sys_execve()使其指向自定义的hack_execve()。hack_execve()根据输入参数判定是否是准备重定向的可执行文件,如果是,则修改参数,指向欲执行的可执行文件,完成后重新调用sys_execve(),从而实现执行重定向后的可执行文件。利用这种技术,可以重定向常用的可执行文件ls,ps等等到指定的后门程序。这种后门技术可以逃避Tripwire等文件完整性校验工具的检测,因为原有的文件根本没有被改变。

3.3.6 端口复用

端口复用的概念是在系统已经开放的端口上进行通信,并且不影响系统的正常工作和服务的正常运行,该技术可以有效地穿越防火墙的限制。通过截获sys_read()系统调用,如果发现有特征字符串(LKM后门自己定义的通讯标记),就调用sys_clone()复制已经开放的端口(常用的如21,23,80等)的套接字描述符。将该描述符传递给用户空间的恶意程序后,恶意程序就可以利用该端口进行通信。

3.4 常见LKM后门工具

当前较为流行的LKM后门工具有knark rootkit和adore rootkit。knark是sekure.net组织的成员creed开发的一个基于Linux2.2内核的后门工具,它并不是一个稳定的版本,在后面的开发中,已经有针对2.4内核的版本出现。其主要功能是隐藏文件,隐藏进程,重定向可执行文件[11],隐藏网络连接,以root身份运行命令等,同时它为了入侵者查询方便,把一些隐藏起来的文件、进程等信息放在/proc/knark里。

adore是teso小组成员开发的一个LKM后门工具,其主要功能和knark差不多,不过它提供了卸载功能(需要提供密码验证),支持2.2和2.4内核,同时易于配置使用。其核心部分是一个叫做adore.o的后门模块,除此之外,还有一个用于隐藏adore.o的模块cleaner.o,一个控制工具ava,以及一个启动脚本startadore。

vlogger是一个高级的基于Linux内核的键盘记录器。它可以记录所有用户通过控制台、串口和远程会话(Telnet,SSH)的击键,并且能够把记录安全地传送到远程服务器。它的智能模式可以自动监测口令输入状态,然后只记录敏感用户的口令信息[12,13]。

4 LKM后门的检测方法

LKM后门是在内核级隐藏目录、文件、进程和通信连接等,它不修改程序二进制文件,因此基于MD5校验的检测工具Tripwire,Aide和Rkhunter无效[14]。按照内核级后门的隐藏特点,已经出现了一些不同类型的检测方法。传统方法注重防患于未然,在系统被入侵前就做好各种准备工作。例如,记录原始每个系统调用的入口地址[2],检测时将现在的系统调用地址与原始记录作比较,如果发生改变,系统就很可能被入侵。传统方法快速、直观,但是其缺点是所有准备工作必须在系统安装初期完成,这样记录下来的系统调用入口地址以及预先截获的某个系统调用都是原始的、干净的。如果在系统被入侵后再采用这种方法就显得无能为力,因为关键的系统调用已经被改变。新的检测方法强调在无论在系统被入侵前或者被入侵后,都能正确发现后门的存在。

4.1 THC的LKM检测理论

THC(The Hacker's Choice)的LKM后门检测方法是:1)记录每一个模块的加载,把模块加载的情况记录在syslog或其它地方,检测的时候用于比较以及查对;2)加载模块的验证,加载模块时会用到sys_create_module()系统调用,那么可以先截获该系统调用,设置一个加载模块时需要密码验证的步骤,任何用户在加载模块时必须通过密码验证,否则不予加载[1]。

THC的LKM检测理论属于传统的检测方法,如果系统已经加载了LKM后门,它的检验将无效。不过作为最早提出的针对LKM后门的检测理论,它仍然具有学习的价值。

4.2 KSTAT

KSTAT利用/dev/kmem设备文件,实现对修改系统调用表及中断描述符表进行系统调用劫持的检测[8]。kmem是一个字符设备文件,它保存了系统中所有虚存的一个映像。读取该文件,查找系统调用表与中断描述符表,可以得到当前系统中各个表项的值。System.map文件是编译系统内核时生成的,它包含了系统的内核符号信息。利用kmem字符设备文件检测系统调用劫持的一般步骤如下:

1)按照文件方式打开/dev/kmem;

2)在kmem中查找sys_call_table[],获取系统调用表首地址、中断描述符表在内存中的首地址,从而得到当前内存中的内核表项的值;

3)比较内存中内核表项的值与 System.map 中系统调用表项与已保存的中断描述符表项的值,可以发现被劫持的系统调用。

KSTAT 有多个选项来帮助用户检测 LKM 后门的存在,最常用的就是-s 选项,使用这个选项,KSTAT 将导出核心系统调用的所有地址。如果探察出的地址和原来的地址不同,就会给出一个警告。另一个常用的选项是-p,这个选项列出当前在系统中运行的所有进程,包括被 LKM 后门隐藏的进程,并给出提示信息。

KSTAT 属于传统方法,它不可避免地带有传统方法的缺陷。另外,研究人员在 2001 年提出了逃避 KSTAT 检测的 LKM 后门实现方法。其原理是对原始系统调用加一个跳转指针指向新的调用,这样并不需要添加新的系统调用到系统调用列表中,也不需要修改原始系统调用的入口地址,从而可以逃避 KSTAT 的检测[15]。

4.3 EPA 方法

EPA(Execution Path Analysis,可执行路径分析)方法的原理是:假如有人修改了内核函数,内核的指令执行路径将改变,在系统调用过程中指令的数目将和原始的内核不同。恶意代码在返回到用户空间时肯定有一些附加行为。这意味着将有许多与未被感染系统不同的代码被执行。可以通过测试系统调用的可执行指令的数目变化来确定系统是否被侵入[16,17]。EPA 方法不属于传统方法,因为系统被入侵后它也能正确分析,它的应用成为传统 LKM 后门检测方法的有力补充。不过 EPA 类检测工具自身也容易受到攻击和欺骗,目前尚在研究之中。

4.4 其它

另外也有针对 LKM 某个方面进行的检测。例如 chkrootkit 对 ps 的输出与/proc 下的数字目录做比较,如果发现有不符,则报警。但是假如入侵者有针对性地将 chkrootkit 使用的系统命令也做修改,chkrootkit 将无法监测到 LKM 后门的存在。文献[18]在 chkrootkit 的基础上,利用隐藏的文件可以被访问以及进程数量有限两个特点,提出了基于穷举和对比的 Linux 隐藏进程查找方法。这两个方法的特点都是在系统被入侵后也能正确运行,但是检测结果的完备性存在一定的不足。例如文献[18]只能检测到隐藏的进程名和进程文件信息,至于被加载的 LKM 后门信息则无法确定。

结束语 本文分析了 LKM 后门的原理以及威胁,并在此基础上对其检测方法进行了讨论。在目前的技术条件下,还没有通用的、精确的方法能检测到 LKM 后门,这就需要管理员未雨绸缪、加强管理,防止任何能让别人加载 LKM 的机会。文献[2,19]提到了内核固化的思想,就是让系统不支持 LKM,虽然杜绝了 LKM 后门的存在,但是对系统上的开发却不利。在将来的检测上,更多地趋向于混合利用上述几种检测方法,以达到互相补充、互相弥补、提高检测精度的目的。

参考文献

[1] Pragmatic / THC. Complete Linux Loadable Kernel Modules (v1. 0). http://packetstormsecurity. nl/docs/hack/LKMHACKING. html, 1999

[2] 袁源,戴冠中,罗红,等. 基于 LKM 的 Linux 安全检测器的设计与实现. 计算机应用研究,2005,22(7):131-133

[3] 李海刚,崔杜武,姚全珠,等. Linux 模块技术分析及应用. 计算机工程,2003,29(1):120-122

[4] Rubini A,Corbet J. Linux Device Drivers[M]. O'REILLY & Associates,Inc,2001

[5] Plaguez. Weakening the Linux Kernel[J]. Phrack Magazine, 1998,52

[6] 王永杰,刘京菊,孙乐昌. Linux 可装载模块的开发和应用. 计算机应用研究,2002,19(7):143-146

[7] 毛德操,等. Linux 内核源代码情景分析. 浙江:浙江大学出版社,2001:191-256

[8] 时金桥,方滨兴,胡铭曾,等. Linux 系统调用劫持:技术原理、应用及检测. 计算机工程与应用,2003,39(32):167-170

[9] 王斌,须文波,冯斌. 利用 LKM 的 Linux 审计功能实现. 计算机工程,2004,30(3):136-138

[10] Xpiloveyou. Linux2. 4. 18 内核下基于 LKM 的系统调用劫持. http://dev. csdn. net/Develop/article/28/28152. shtm

[11] Palmers. Advances in Kernel Hacking[J]. Phrack Magazine, 2002,59

[12] 兰薇薇,贾卓生. 利用 LKM 记录键盘事件的研究与实现. 网络安全技术与应用,2005,3:24-26

[13] Rd. Writing linux kernel keylogger[J]. Phrack Magazine, 2002,59

[14] 孙淑华,马恒太,张楠,等. 后门植入、隐藏与检测技术研究. 计算机应用研究,2004,7:78-81

[15] E4gle. 逃避 kstat 的检测的 lkm 程序的实现方法(linux). http://www. xfocus. net/articles/200108/258. html

[16] 王松涛,吴灏. Linux 下基于可执行路径分析的内核 rootkit 检测技术研究. 计算机工程与应用,2005,41(11):121-123

[17] Rutkowski J K. Execution path analysis:finding kernel rk's[J]. Phrack Magazine,2002,59

[18] 袁源,戴冠中,罗红. 利用 Perl 实现 Linux 下隐藏进程的查找. 计算机工程与应用,2007,43(3):102-105

[19] Stealth. Kernel Rootkit Experiences[J]. Phrack Magazine, 2003,61

(上接第 4 页)

[23] Sheikh H R,Sabir M F,Bovik A C. A Statistical Evaluation of Recent Full Reference Image Quality Assessment Algorithms. IEEE Transactions on Image Processing, 2006,15:3441-3451

[24] Wang Zhou,Bovik A C,Evans B L. Blind Measurement of Blocking Artifacts in Images // IEEE International Conference on Image Processing. 2000,3:981-984

[25] Horita Y,Arata S,Murai T. No Reference Image Quality Assessment for JPEG/JPEG2000 Coding // The 12th European Signal Processing Conference. 2004

[26] Muijs R,Kirenko I. A No-Reference Blocking Artifact Measure for Adaptive Video Processing// The 13th European Signal Processing Conference. 2005

[27] Marziliano P, Dufaux F,Winkler S,et al. A No-reference Perceptual Blur Metric// IEEE International Conference on Image Processing. 2002,3:57-60

[28] Ong E P,Lin W S,Lu Z K,et al. No-reference JPEG2000 Image Quality Metric // International Conference on Multimedia and Expo. 2003,1:6-9

[29] Sheikh H R, Bovik A C, Cormack L. No-Reference Quality Assessment Using Natural Scene Statistics:JPEG2000. IEEE Transactions on Image Processing, 2005,14:1918-1927

[30] Vasconcelos N. Feature Selection by Maximum Marginal Diversity:Optimality and Implications for Visual Recognition. Computer Vision and Pattern Recognition,2003(1):762-769

[31] Gastaldo P, Zunino R. No-Reference Quality Assessment of JPEG Images by Using CB PNeural Networks // IEEE International Symposium on Circuits and Systems. 2004,5:772-775

[32] Liu Chunmei, Wang Chunheng, Dai Ruwei. Low Resolution Character Recognition by Image Quality Evaluation// IEEE International Conference of Pattern Recognition. 2006,1:864-867