操作系统内核级安全审计系统的设计与实现*)

陈 慧 石文昌 梁洪亮 孙玉芳
(中国科学院软件研究所 北京100080)

摘 要 随着人们对操作系统的安全要求越来越高,建立安全、完备的审计子系统成为操作系统安全领域的一个重要课题。本文提出一个通过在内核安插钩子函数来实现模块化审计系统的方法,用这种方法设计并实现一个基于Linux操作系统的、遵循国家标准GB17859-1999第四级要求的安全审计子系统。这个审计系统以模块形式连入内核,通过往内核中安插审计钩子来收集审计信息,对内核影响较小,并能适应内核的升级;通过用内核线程代替后台进程将审计记录存入磁盘,实现了审计的完全内核化,增强了系统安全;通过对所有系统调用进行审计,实现了对利用隐蔽存储信道时可能被使用的事件的审计。

关键词 操作系统,内核级,安全,审计系统,Linux

Design and Implementation of Kernel Level Secure Auditing System in Operating System

CHEN Hui SHI Wen-Chang LIANG Hong-Liang SUN Yu-Fang
(Institute of Software, Chinese Academy of Sciences, Beijing 100080)

Abstract With the ever-increasing demands on operating system security, to build up secure auditing subsystems is becoming an important issue in the area of operating system security. This paper presents a method to implement a modular auditing system by inserting hooks into the Linux kernel. In this way, a secure kernel-level auditing subsystem, which is Linux-based and accords with the fourth level requirements of the National Standard of China, is designed and implemented. The auditing subsystem is used as a loadable kernel module. It can be easily upgraded with the development of the Linux kernel. It implements kernel-level audit by using kernel thread instead of user-level daemon. Audit events that might be used in the exploitation of covert storage channels are auditable due to the full coverage of all system calls by the auditing subsystem.

Keywords Operating system, Kernel level, Security, Auditing system, Linux

1 引言

审计系统是安全操作系统的一个重要组成部分。在普通的Linux操作系统中,日志机制担负着类似审计的功能。但是日志机制在审计粒度、审计的安全性、审计灵活性等方面都存在不足。随着人们对操作系统的安全要求越来越高,建立更加完备、安全的审计系统成为操作系统安全领域的一个重要课题。

为了适应不同的Linux用户的需要,避免内核的庞大,审计系统一般都做成单独的LKM(loadable kernel module,可加载内核模块)的形式或者是安全模块的一部分。另一方面,审计信息需要从内核系统调用中收集,现有的模块化审计子系统在收集审计信息的时候,多采用替换系统调用的方法来进行内核审计,如国外的审计系统SNARE(System iNtrusion Analysis & Reporting Environment)就是用的这种方法[1]。随着Linux内核的升级,为了防止LKM(Loadable kernel module)对系统调用进行恶意修改,2.4.18以后的Linux版本中syscall_table已经不是导出符号(export symbol)了,采用这种方式不能实现审计。SNARE系统采取的解决办法(适用于2.4.20的版本)是修改内核,将syscall_table改为导出符号。显然,这种方法是不能被Linux内核所接受的,也是不安全的。如何建立对内核影响较小而又能收集到全面的审计信息的模块化审计系统成为一个难点。

我们通过扩展LSM(Linux Security Modules)框架来解决这个问题。LSM是一个Linux的访问控制框架,它在内核中安插钩子点,当访问信息经过这些钩子点时,将会被钩子函数钩起来判断是否允许它通过[2]。LSM中的钩子主要用于实现访问控制,我们可以在这个框架中添加审计钩子,在内核的审计点插入审计钩子,每到审计点钩子函数就记录审计信息、建立审计记录或完成其他审计功能。这些钩子函数的功能由我们的审计子系统实现,这样不会影响到内核的其它部分,而且即使内核改动,审计系统也能方便地随之升级。

另外,目前的审计系统都不是完全内核化的,一般由一个用户层的应用程序来将审计记录存入磁盘,造成一定的安全隐患。我们采用内核线程来完成这个工作,使得审计完全内核化,更加安全。

本文首先简要叙述论文的研究背景和设计要求,接着,在此基础上详细介绍采用钩子机制设计和实现一个基于Linux的内核级的审计系统,最后对研究工作进行了总结。

2 研究背景

本文研究的审计子系统是国产安全操作系统SECIMOS(Security in Mind Operating System)的一部分。SECIMOS是在符合国家标准GB17859-1999第三级的安全操作系统RS-

*)本文得到国家863高技术研究发展项目(2002AA141080)和国家自然科学基金项目(60073022)资助。陈 慧 硕士研究生,主要研究方向为计算机安全;石文昌 博士,研究员,主要研究方向为计算机安全与系统软件;梁洪亮 博士,主要研究方向为计算机安全与系统软件;孙玉芳 研究员,博士生导师,主要研究方向为系统软件与中文信息处理。

Linux[3]研究工作的基础上开展的安全操作系统项目，它的目标是达到国家标准 GB17859-1999第四级的要求。SECIMOS 在 LSM 安全框架的基础上来实现身份鉴别、自主访问控制、强制访问控制、客体重用、安全审计、数据完整性、隐蔽信道分析和可信路径等安全功能。我们的审计子系统要达到以下四个方面的要求：

(1)审计面广。从系统调用层面上对系统活动进行审计，为了能够审计所有可能产生隐蔽通道的事件，将审计扩大到所有系统调用。因为所有隐蔽信道都是开发者通过内核调用找到并利用的。如果所有的系统调用都被审计了，那么利用隐蔽信道时可能被使用的事件也就被审计了[5]。在用户层审计上向用户程序提供审计接口。

(2)操作方便。向审计者提供了图形菜单和文本菜单两种审计管理界面。审计配置文件的格式和对其操作对审计员都是透明的。有一般计算机操作知识的人都能对其进行管理。

(3)配置灵活。审计事件的配置有五种审计规则，可以从主体、客体、客体的安全级别等各个角度进行审计限定，使审计配置更加灵活。

(4)更加安全。对于审计子系统的管理设置多个管理者权限：开启审计者，关闭审计者，审计配置员等等，权限的细分降低了入侵者获得某个权限后对系统造成的破坏程度。审计文件和审计配置文件不仅受到自主访问控制的保护，还加入了强制访问控制的保护。

3 审计系统设计与实现

3.1 审计原理

实际应用的审计系统分为内核层和用户层两大部分。内核层部分就是要收集审计信息并在磁盘中形成审计记录文件，用户层负责读取审计文件，显示和查询审计记录。本文中所指的审计系统主要是指审计的内核层部分，它的设计和实现是整个审计系统的关键。

我们把审计子系统分为两个部分：审计后台部分和审计部分。前者负责将审计记录存入磁盘文件中；后者负责收集内核中的审计信息，建立审计记录。这两者的结合点就是内存中的审计记录链。它们的工作原理如下图所示。

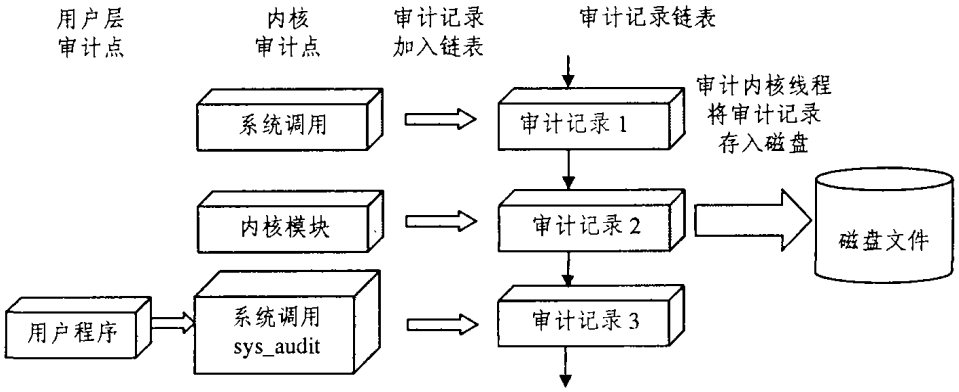

图1 审计原理图

审计系统要解决的问题就是围绕审计记录展开的。下面我们从审计记录的生成到存入磁盘的过程详细介绍。

3.2 审计的事件

审计事件分布在两个层次：内核层和用户层。

内核层的审计包括：与系统安全有关的审计和系统调用。其中，与系统安全有关的审计包括对特权进程的审计和安全模块中的审计。

用户在系统中的一切活动都需要系统调用的支持。在系统调用中(包括原有的系统调用和安全模块新加入的系统调用)设立审计点能够把用户在系统中的所有活动记录下来。

用户层审计包括对 Shell 命令的审计和用户程序中的审计。Shell 命令包括涉及到安全和特权用户的命令。主要有 login、su、sudo、passwd、adduser 等和 SECIMOS 新增命令。为了给用户程序提供审计功能，要新建审计系统调用，作为用户的审计接口。

3.3 内核审计的钩子机制

用户层的审计是通过专门的审计系统调用来实现的，我们设计的内核层审计用分布在众多系统调用中的钩子函数来实现。实现钩子机制面临几个问题：安插多少钩子、安插在何处，以及如何存放收集的信息。首先要确定需要哪些审计信息。

3.3.1 *审计信息* 每个审计事件主要记录五个方面的内容。审计事件基本信息：包括时间、审计号、审计类别、审计记录大小等；主体信息：包括事件发出的用户信息等；客体信息：包括事件作用的客体(可以是文件，进程等)信息、安全级别等；事件特殊信息：包括特殊参数等；事件是否成功信息：包括成功、或者失败原因。

每个审计记录至少有审计事件基本信息和主体信息。主体信息可以从当前进程(current)的 task_struct 结构中找到。审计时间也可以计算得出。因此在内核中要收集的信息主要是客体信息、事件特殊信息和事件结果信息。这就是钩子函数的任务之一。一般客体信息和事件特殊信息是通过系统调用的传递参数来体现的，应该在系统调用入口处收集；事件结果在系统调用出口处收集。

3.3.2 *系统调用中安插的钩子* 我们在每个系统调用中安插两种钩子函数，钩子函数的插入点可以从图2中看出。

(1)aud_check(int aud_num，va_list args)。其中，aud_num 是审计的事件编号，args 是该审计事件所需要的参数列表，这个列表的长度和内容都是可变的，方便针对不同的事件收集不同的信息。

该钩子在系统调用入口处加入，用于判断是否进行审计并收集审计基本信息、客体信息和特殊信息等。审计事件编号传入钩子函数后，相应的审计类别等事件基本信息可以确定。同时，将系统调用的参数作为参数传入钩子函数，可以获得其他信息。

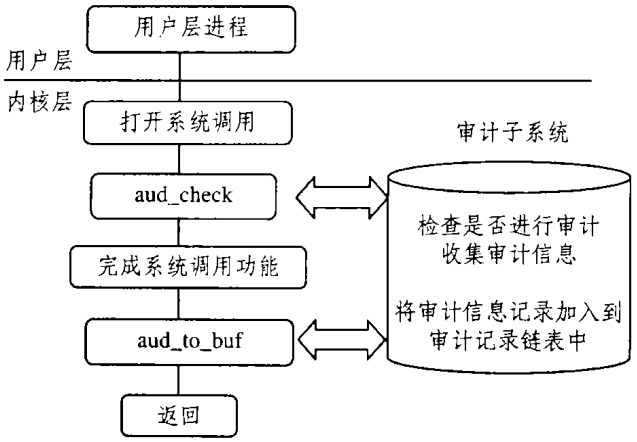

图2 系统调用中的钩子安插

(2)aud_to_buf(int value)。其中,value 是该系统调用完成后的返回值,根据该值来判断事件成功与否。

该钩子在系统调用出口处插入。系统调用有多个出口,成功出口和不同的错误出口,在出口处能确定 value 值。如果该事件是要审计的,则将完整的审计记录放入审计记录链。

在实际的系统实现中,审计钩子的插入是比较灵活的。钩子机制的原则之一就是对内核的改动要尽量小。在某些系统调用中,如果在适当的审计钩子插入点已经有了 LSM 的其他访问控制钩子,那么就可以在执行访问控制的钩子函数时调用审计钩子函数,避免了对内核改动。

3.3.3 审计安全域 建立审计记录后,从钩子函数中返回系统调用时需要将记录和系统调用联系起来。我们可以用当前进程作为联系的桥梁。在进程的 task_struct 结构中加入一个审计指针,即一个类型为 void 的指针,在钩子函数中建立了审计记录后,将审计指针指向这个记录,这样返回到系统调用中后,也能根据审计指针找到记录。

如果该事件不被审计,则审计指针设为 NULL。在系统调用出口处插入的钩子函数中检查该指针是否为空。若为空,则什么都不做。如果指向某个记录块则将该记录块挂入审计缓冲区链。

3.4 用户层审计的系统调用

用户层审计通过审计系统调用来实现。我们新建一个系统调用 sys_audit。这是为所有与审计有关的操作提供的一个统一的系统调用接口。

通过审计系统调用可以完成的操作有:审计用户层事件;读审计文件和审计配置文件;写审计文件和审计配置文件;开启审计和关闭审计。

为了避免系统调用的滥用,造成大量恶意产生的记录,在审计系统调用中会检查该事件是否具备审计特权。

3.5 审计记录链表

如果系统中指定的审计事件较多,审计记录产生的速度非常快,系统不可能在每条记录产生后马上将其记录放入审计文件,而是采取批处理的方式进行处理,审计子系统因而设定一个引起内核线程进行批处理操作的审计记录阈值 AUD_MAX,生成的审计记录先挂接在内存中的审计记录链表中,达到这个阈值才存入磁盘。审计记录链表带有一个读指针、一个写指针、一个审计记录计数器和一个信号计数器。读指针指向链表头,写指针指向链表尾。审计钩子 aud_to_buf 将生成的审计记录加入到链表中,审计内核线程负责将审计记录从链表中取出存入磁盘,它们之间的通信由信号完成。

3.5.1 审计钩子 aud_to_buf 每当将新的审计记录加入到链表尾,都要让计数器增加相应值。然后,检查计数器的值,如果达到阈值 AUD_MAX,则发送信号给审计内核线程,提醒它将 AUD_MAX 个审计记录存入磁盘,信号计数器的值增1。考虑到在下一个审计记录到来时,之前的 AUD_MAX 个记录可能还没有完全存入磁盘,为了避免计数上的重复,发送信号后还要将审计记录计数器的值减去 AUD_MAX。在审计记录加入到链表之前,要检查信号数是否达到系统允许的最大值,如果达到了,则表示内核线程处理信号的速度太慢,因为一个信号表示有 AUD_MAX 个审计记录等待存入磁盘,此时等待存入磁盘的记录太多,达到了允许挂接在审计记录链表中记录数的极限。为了避免记录的无限增长,这条记录只能丢弃。

3.5.2 审计内核线程 kauditd 现在大部分审计系统通过后台进程(deamon)来完成这部分功能,它实际上还是一个用户进程,通过一个 proc 文件来对内核空间的审计记录链表进行读取,比如日志系统所用的 syslogd。这种非内核级审计方式存在安全隐患[6]。在我们的审计系统中,将后台进程改成了内核线程。这样使得对审计记录的操作都在内核级进行,更加安全。

在审计系统初始化的时候,启动内核线程。首先该线程从审计配置文件中读出审计配置,初始化信号队列,然后自动睡眠,隔一定时间醒来查看是否接收到信号。

内核线程收到审计钩子发来的信号后,连续地将 AUD_MAX 个记录存入磁盘文件。接着将信号记录数减1,表示完成了这个信号传达的任务。此时若信号队列中没有新信号则接着睡眠,若有则按信号工作。

内核线程同时要负责监控磁盘中的审计文件大小。在每次将审计记录存入磁盘文件之前,要检查该文件是否达到规定的大小。如果达到了,就关闭该文件,创建新的审计文件,继续工作。

3.6 审计记录文件的存放

审计记录在磁盘中是以文件形式存在。为了节约磁盘空间和保证审计文件的安全性,审计文件用二进制形式存储。审计子系统限定每个审计记录文件的大小。文件大小达到一定值则关闭该文件,开始另一个新的记录文件。每次开启审计系统时,建立新的审计记录文件;关闭审计系统时,关闭当前记录文件。为了查看方便,每个记录文件名都以其开始时间为标记,如:aud_ddmmyy_hhmmss(ddmmyy—日月年,hhmmss—时分秒)。

为了避免审计文件占用磁盘太多空间,还应该配置一个审计磁盘监控后台进程。该进程定时查看磁盘中审计目录下的审计文件,超过一定数目则向审计管理员发邮件通知,并可根据审计配置采取一定措施:可将文件转存到审计管理员指定的其它磁盘空间。

结束语 审计系统是安全操作系统的重要组成部分。为了适应不同的 Linux 用户的需要,避免内核的庞大,审计系统一般做成 LKM(loadable kernel module,可加载内核模块)的形式或是安全模块中的一部分。目前,做得比较成功的模块化审计系统是 SNARE 系统。但随着 Linux 内核的升级以及 LSM 框架嵌入到内核中,SNARE 采取的通过替换系统调用来将审计功能加入内核的方法不再适用。同时,目前的审计系统采用用户层的应用程序作为审计后台进程来将审计记录存入磁盘也存在安全隐患。

针对以上问题,本文讨论了用钩子机制实现安全操作系统 SECIMOS 的内核级审计子系统的方法,通过在系统调用

(下转第184页)

offering2:numStudents=10,hasProfessor=true

输入:学生通过控制台调用 closeRegistration 操作。

方法调用序列:

controller.closeRegistration,controller.isRegistrationOpen,
controller.getNextOffering,(offering.closeRegistration) offering1,
schedule.commit,(offering.closeRegistration)offering2,
schedule.commit,controller.isAllCommitted

预期输出:返回"committed"消息,表示注册被成功提交。

表2

范畴	选择	约束	取值
open	true false	Controller.in(regOpen)	true
numStudents	0～3 4～9 10	(numStudents>=3)$_{offering1}$ offering.in(Full)$_{offering2}$	(4～9)$_{offering1}$ (10)$_{offering2}$
hasProfessor	true false	offering.in(Assigned)$_{offering1}$ (hasProffessor=true)$_{offering2}$	(true)$_{offering1}$ (true)$_{offering2}$

在这个测试用例中,offering1的 numStudents 可取4～9中的任意一个值,此处取为6。方法调用序列就是该场景中的消息序列,但去掉了其中所有返回消息,因为它们并不是真正的方法调用操作。可用同样的方法为所有场景生成测试用例。

结论 UML 是面向对象分析和设计时的标准建模语言,使用 UML 图模型来生成测试用例不仅能充分利用已有的设计结果,减少测试的费用,而且由于 UML 是一种半形式化的建模语言,测试用例的生成可以部分地实现自动化,因此基于 UML 图模型生成测试用例的研究[2~12]近年来受到越来越广泛的关注。

本文提出了一个根据 UML 顺序图生成场景测试用例的方法,包括场景的生成和使用范畴-划分方法生成测试用例。整个方法具有如下几方面的特点:1)完全基于 UML,使它容易被已使用 UML 的工业界采用。2)对顺序图进行了形式化定义,能够通过对顺序图中事件的遍历方便地得到所有的场景,而且加入了对分支和循环的处理,使得该方法的适用范围较为广泛。3)生成测试用例的过程中考虑了对象的状态信息,因此生成的测试用例较为充分。文[2]同样提出了一个使用范畴-划分方法生成测试用例的方法,但是它没有给出生成场景的算法。文[12]提出了一个可测试的顺序图模型,此模型无法对分支情况进行处理,限制了它的适用范围。文[8,9]都提出了对 UML 图进行测试的基本的策略和覆盖标准,文[9]中还给出了一个插桩算法用于跟踪协作图中的方法调用序列,可用来对测试结果进行验证,但它们都没有说明如何生成测试用例。然而我们的工作也存在一些限制,生成的测试用例只能覆盖所有可能的消息序列(场景),无法覆盖所有的事件序列,这是需要进一步改进的地方。

我们今后的研究工作是:1)提出一种与面向对象软件开发过程集成的测试过程。2)针对 UML 单个模型图,研究其在不同测试层次生成测试用例的方法,提出相应可行的测试评价标准,尽量能够直接使用 UML 模型文档,减少形式化的工作量。3)针对某一测试层次的测试用例生成时,研究综合利用待测试系统的各种模型图生成测试用例的方法。4)为上述方法提供自动的工具支持,并希望能够与主流建模工具、测试工具集成。

参考文献

1 Pressman R S 著,梅宏 译. 软件工程－实践者的研究方法(第5版). 机械工业出版社,2002

2 Basanieri F, Bertolino A. A Practical Approach to UML-based Derivation of Integration Tests. In: Proc. of QWE2000, Bruxelles, November 20-24, 3T

3 Ostrand T J, Balcer M J. The Category Partition Method For Specifying and Generating Functional Tests. Communication of the ACM,1988,31(6):676～686

4 Booch G,Rumbaugh J,Jacobson I,著,邵维忠等译. UML 用户指南. 机械工业出版社,Addison-Wesley ,2001

5 UML Specification 1.5. available at http://www.omg.org/uml

6 Binder R V 著,华庆一,王斌君,陈莉 译. 面向对象系统的测试. 人民邮电出版社,2001

7 Offutt A J, Abdurazik A. Generating Tests from UML specifications. In: Proc. 2nd Intl. Conf. on the Unified Modeling Language (UML'99), Fort Collins, CO, Oct. 1999. 416～429

8 Wu Y, Chen M-H, Offutt J. UML-based Integration Testing for Component-based Software. In: The 2nd Intl. Conf. on COTS-Based Software Systems (ICCBSS). Ottawa, Canada, Feb. 2003. 251～260

9 Offutt A J, Abdurazik A. Using UML Collaboration Diagrams for Static Checking and Test Generation. In: Proc. 3rd Intl. Conf. on the Unified Modeling Language (UML'00), York, UK, Oct. 2000. 383～395

10 Briand L, Labiche Y. A UML-Based approach to system testing. In: Proc. of the 4th intl. Conf. in Unified Modeling Language (UML 2001), volume 2185 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, Germany, 2001. 194～208

11 Hartmann J, Imoberdof C, Meisenger M. UML-Based Integration Testing. In: ISSTA 2000 conf. proc. Portland, Oregon, 2000. 60～70

12 Fraikin F, Leonhardt T. Sequence Diagram Based Test Automation. http://www.pi.informatik.tu-darmstadt.de/publikationen/technische%20Berichte/2002/pi2002-2.pdf

(上接第175页)

中安插审计钩子来实现审计模块化,对内核影响较小,在内核升级时也能方便地将审计系统进行升级。我们还采用内核线程作为审计后台进程,使得审计完全内核化,更加安全。该审计子系统按照 Posix.1e 标准和中国国家标准 GB17859-1999 "计算机信息系统安全保护等级划分准则"第四级对审计的要求进行设计。目前,原型系统已在支持 LSM 框架的 Linux 2.5.72版本上实现,同时能够对 LSM 的安全模块进行审计。

参考文献

1 Alliance I. SNARE-System intrusion Analysis&Reporting Environment. http://www.intersectalliance.com/projects/Snare/index.html

2 Morris J, Smalley S, Kroah-Hartman G. Linux Security Modules: General Security Support for the Linux Kernel. In Linux Security Modules: General Security Support for the Linux Kernel, 2002. http://www.citeseer.nj.nec.com/wright02linux.html

3 石文昌,孙玉芳,等. 安全 Linux 内核安全功能的设计与实现. 计算机研究与发展,2001,38(10):1255～1261

4 中华人民共和国国家标准. 计算机信息系统安全保护等级划分准则 GB17859-1999,中国,1999

5 Final Evaluation Report-Trusted Information Systems-Trusted XENIX version 4.0, ReportNo. CSC-EPL-92/001.A, National Computer Security Center, USA, Jan 1994

6 贾春福,徐伟,郑辉. Linux 系统内核级安全审计方法研究. 计算机工程与应用,2002(6):53～55