维普资讯 http://www.cqvip.com

通过扩展LSM框架构建审计支持机制*)

陈 慧 石文昌 梁洪亮
(中国科学院软件研究所 北京100080)

摘 要 LSM是纳入到Linux内核的一个安全模型支持框架,它通过提供钩子机制来支持安全机制的实现。审计机制是安全操作系统的重要组成部分。然而,LSM的现有设计主要考虑的是对访问控制的支持,对审计机制的支持存在着明显的不足。把审计支持纳入到LSM框架中,为审计系统或者是入侵检测系统提供统一的支持接口,对安全操作系统的研究有重要意义。本文提出一个扩展LSM框架的方法,以增强LSM框架对审计机制的支持能力。本文论述了扩展LSM框架实现审计的方法,以及如何在LSM框架中加入审计钩子及在内核函数中插入钩子函数。依照这种方法,作为安全操作系统SECIMOS开发工作的一个重要组成部分,我们在Linux内核中实现了一个审计系统,并对其性能进行了分析。

关键词 LSM,审计系统,Linux

Enhancing the LSM Framework to Build Audit Support Mechanisms

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

Abstract LSM (Linux Security Module) is a security-model-support framework which is now an important part of the Linux kernel. It supports security mechanisms by providing hooks mechanism. Audit is an essential part of secure operating system. Mainly focusing on access control, LSM is lack of ability to support audit. It is significant to extend the LSM framework to support audit mechanism which can provide an interface to audit system or intrusion detection system. This paper presents a method to enhance the LSM framework to support audit functions. It discusses how to add audit hooks into the LSM framework and how to insert hook functions into Linux kernel functions. In this way, an audit system is implemented as a main part of the SECIMOS secure operating system. Also, its performance is tested and analyzed.

Keywords LSM, Audit system, Linux

1 引言

目前,面向实际应用的安全操作系统的研究与开发主要采用安全增强的方法,其中,访问控制增强和审计增强是安全增强的两个重要部分。一个成功的安全操作系统应该既有访问控制功能也有审计功能。

国内外很多团体都在研究安全操作系统,访问控制的增强功能在很多安全操作系统的实现中都得到了高度的重视,如RSBAC、SELinux等[1]。在Linux操作系统领域,随着LSM(Linux Security Module)[2]框架被纳入到Linux的内核结构之中,访问控制增强机制的实现有了一个统一的方法。

另一方面,审计增强的研究还没有形成这种统一的方法。事实上,审计系统设计的关键问题之一就是对系统内核功能的影响[3]。Linux内核是不断升级的,在系统中也可能加载其他的安全模块,因此,审计系统既要能适应内核的升级,也不能对内核的其他功能影响过多。从这一点考虑,审计系统最好做成单独的LKM(Loadable Kernel Module可加载内核模块)的形式。但在实现中有困难,因为审计信息需要从内核系统调用中分散收集,审计系统又是和内核紧密相关的,从这方面来说,很难将审计系统"分"出来。因此,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。

我们可以通过将LSM框架扩展,加入审计功能。将审计函数的钩子插入到内核中,为审计模块提供一个接口,同时,也让审计模块可以和其他基于LSM框架的安全模块共存。

本文首先简要叙述论文的研究背景,接着介绍通过扩展LSM框架实现审计的方法,依此方法,作为安全操作系统SECIMOS[4]开发工作的一个重要组成部分,在Linux内核中实现了一个审计系统(本项工作开始时,Linux内核的最高版本为2.5.72,所以,我们选择该版本为原型),接着对系统性能进行分析,最后总结全文。

2 研究背景

2.1 LSM介绍

LSM是一个已被纳入到Linux内核中的安全支持框架。当然,它本身并不能增强安全性,只是提供一个支持安全模块的基础。使用LSM可以使安全增强模块以可加载内核模块的方式加入内核。

LSM框架在Linux内核中增加了两个部分:在内核数据结构中加入了安全域来存放安全属性;在内核代码中加入对

*)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和60373054)资助。陈 慧 硕士研究生,主要研究方向为计算机安全与系统软件;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。

维普资讯 http://www.cqvip.com

钩子函数的调用(这些调用称为"钩子"),这些钩子函数对安全域进行操作并实现安全支持功能。安全域一般就是在数据结构增加的 * void 指针,具体指向的数据结构由设计安全模块的开发者定义。同时,LSM 定义了一个全局表 security_ops,钩子就是该表中的函数指针,该指针具体指向哪个函数,采用什么安全政策也是由具体的安全模块定义的。LSM 仅仅定义了每个钩子的参数和它们的位置。

内核版本2.5.72中提供了155个钩子。因为钩子函数实现的功能主要是访问控制,因此它们和客体资源是有关的,比如对 inode 的操作、文件操作、socket 操作等都有相应的钩子存在。它们在内核中的位置恰好在访问客体资源之前。比如创建一个文件,LSM 提供了一个钩子 security_inode_create,当找到文件所在的目录 dentry 以及完成了其它的工作后,马上就要创建 inode,此时就插入钩子 security_inode_create。

2.2 安全操作系统总体设计

我们的安全操作系统 SECIMOS 是参考 LSM 框架,按照中国国家标准 GB17859-1999"计算机信息系统安全保护等级划分准则"[5]第四级的要求进行设计的。我们将多个安全模块通过一个主模块整合进内核。实现的主要模块有安全审计、自主访问控制、强制访问控制、可信路径、可信进程、特权用户等功能。该安全系统及其相关的国家863计划信息安全技术主题重点课题"操作系统与数据库平台安全核心技术研究"(课题编号:2002AA141080)已经通过了国家863专家组的验收。

3 LSM 的扩展与审计功能的实现

3.1 内核审计要求

根据中国国家标准 GB17859-1999"计算机信息系统安全保护等级划分准则"第四级对审计内容的要求,审计内容主要包括四个方面:事件基本信息、主体信息、客体信息和事件特殊信息。具体如下:

1) 审计事件基本信息:时间,审计号,审计类别等;

2) 主体信息:事件发出的用户 uid,安全级别等;

3) 客体信息:事件作用的客体(可以是文件,进程等),安全级别;

4) 事件特殊信息:参数等。

每个审计都有的部分就是事件基本信息和主体信息。可以看出,这两项审计信息可以从系统信息得到或通过当前进程的信息得到。而后两项信息是与具体的事件相关的。因此我们主要收集后两项信息。

我们可以看看一个具体的审计事件例子 sys_open,它的审计信息如下:

客体信息:文件名(包括路径)、当前路径、打开的文件描述符、文件属主(uid)、文件模式(mode)、客体安全标记;

事件特殊信息:打开标志(flag)、打开模式(mode)。

3.2 审计钩子及其插入点

LSM 提供的钩子机制主要用于访问控制,它是不能完全满足审计需求的。这一方面体现在钩子插入点,LSM 的钩子一般恰好出现在访问客体资源之前,而在此之前有可能出现操作失误而导致访问客体资源失败,因此如果直接利用 LSM 的访问控制钩子,实际上就缩小了审计范围;另一方面体现在传递给钩子函数的参数,审计系统需要收集全面的信息,原有的 LSM 钩子不能完全满足需求。

根据对审计内容的分析,一方面我们要得到事件是否成功的结果,这只能在事件结束时才能得到;另一方面要得到事件的特殊参数,这一般要在事件开始时才能得到,因为在事件处理过程中原始参数可能被改变。因此,至少要增加两个审计钩子。实验证明,增加两个审计钩子可以完成审计功能。我们增加的两个审计钩子是 aud_check 和 aud_to_buf。

(1) aud_check(int aud_num,va_list args)

参数说明:aud_num 是审计事件编号;args 是审计信息,不同的事件的参数不一样,这个参数应该是可变的,因此将 args 设成参数列表的形式。

函数功能:在系统入口处加入,用于判断是否审计和建立审计记录、收集审计信息。

(2) aud_to_buf(long value)

参数说明: value 是该系统调用的返回值,通过这个返回值可以判断事件是否成功。

函数功能: 在系统调用返回之前调用。将审计记录加入审计记录链表。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。

我们还是以 sys_open 为例来说明,通过在函数 sys_open 中加入审计钩子,能够将审计信息收集完整。插入审计钩子前后的系统调用对比见表1。

通过 aud_check 钩子我们可以收集到的信息有:文件名、当前路径、打开标志和模式。打开的文件描述符、文件属主、文件模式不能直接得到,需要经过一些处理才能得到。但这些工作并不需要审计函数来完成,在原来的内核函数处理中可以得到结果,在上面那个例子中,在 open processing 这一步(见表1),就会根据文件名找到文件的上级目录,建立文件描述符,并设定文件属主、文件模式等信息。得到这些结果也不需要增加新的审计钩子,可以利用 LSM 的访问控制钩子。具体方法将在下一小节详细说明。

表1 加入审计钩子前后的 sys_open

加入审计钩子前	加入审计钩子后

sys_open
{
get filename from user space;
 	
if (!filename) {
open processing;
out: ending work;
}
return value;
 	
out_error:
handle err;
goto out;
}
	sys_open
	{
	get filename from user space;
	aud_check(AUD_OPEN,args);
	if (!filename) {
	open processing;
	out: ending work;
	}
	aud_to_buf(value);
	return value;
	out_error:
	handle err;
	goto out;
	}

本审计系统共对85个系统调用和一个系统事件 panic 进行了分析,在内核中找出290个钩子插入点,表2列出了具体情况。

维普资讯 http://www.cqvip.com

表2 内核审计钩子插入点

系统调用	涉及的文件(函数)	新增钩子点
panic	kernel/panic.c	1
execve	arch/i386/kernel/process.c (sys_execve)	2
fcntl	fs/fcntl.c (sys_fcntl)	3
dup	(sys_dup)	2
dup2	(sys_dup2)	2
open	fs/open.c (sys_open)	2
close	(sys_close)	3
chdir	(sys_chdir)	2
fchdir	(sys_ fchdir)	2
chown	(sys_ chown)	2
fchown	(sys_ fchown)	2
lchown	(sys_ lchown)	2
truncate	(sys_ truncate)	2
ftruncate	(sys_ ftruncate)	2
chmod	(sys_ chmod)	2
fchmod	(sys_ fchmod)	2
utime	(sys_utime)	2
statfs	(sys_ statfs)	2
fstatfs	(sys_ fstatfs)	2
read	fs/read_write.c (sys_read)	2
write	(sys_write)	2
readv	(sys_readv)	2
writev	(sys_writev)	2
pread64	(sys_pread64)	3
pwrite64	(sys_pwrite64)	3
sendfile	(sys_sendfile	5
sendfile64	(sys_sendfile64)	5
link	fs/namei.c (sys_link)	3
unlink	(sys_unlink)	3
mkdir	(sys_mkdir)	2
rmdir	(sys_rmdir)	3
rename	(sys_rename)	3
mknod	(sys_mknod)	4
mount	fs/namespace.c (sys_mount)	2
umount	(sys_umount)	2
delete_moudle	kernel/module.c (sys_delete_module)	5
init_module	(sys_init_module)	6
setuid	kernel/sys.c (sys_setuid)	3
setgid	(sys_setgid)	3
setregid	(sys_setregid)	4
setreuid	(sys_setreuid)	5
setresgid	(sys_seresgid)	5
setresuid	(sys_setresuid)	5
setfsgid	(sys_setfsgid)	2
setfsuid	(sys_setfsuid)	1
clone	kernel/fork.c (do_fork)	2
fork	(do_fork)	0
exit	kernel/exit.c (sys_exit,do_exit)	2
exit_group	(sys_exit_group)	1
wait4	(sys_wait4)	3
waitpid	(sys_waitpid)	2
setxattr	fs/xattr.c (sys_setxattr)	3
lsetxattr	(sys_lsetxattr)	3
fsetxattr	(sys_fsetxattr)	3
getxattr	(sys_getxattr)	3
lgetxattr	(sys_lgetxattr)	3
fgetxattr	(sys_fgetxattr)	3
listxattr	(sys_listxattr)	3
llistxattr	(sys_llistxattr)	3
flistxattr	(sys_flistxattr)	3
removexattr	(sys_removexattr)	3
lremovexattr	(sys_lremovexattr)	3
fremovexattr	(sys_fremovexattr)	3
semget	ipc/sem.c (sys_semget)	3
semctl	(sys_semctl)	6
msgget	ipc/msg.c (sys_msgget)	2
msgctl	(sys_msgctl)	16
shmget	ipc/shm.c (sys_shmget)	2
shmctl	(sys_shmctl)	5
socket	net/socket.c (sys_socket)	3
connect	(sys_connect)	2
accept	(sys_accept)	2
listen	(sys_listen)	3
bind	(sys_bind)	3
kill	kernel/signal.c (kill_something_info) (kill_pg_info) (kill_proc_info)	2 1 1
ustat	fs/super.c (sys_ustat)	2
readahead	mm/filemap.c (sys_readahead)	2
ioctl	fs/ioctl.c (sys_ioctl)	2
pipe	arch/i386/kernel/sys_i386.c (sys_pipe)	1
brk	mm/mmap.c (sys_brk)	2
mremap	mm/mremap.c (sys_mremap)	2
sysinfo	kernel/timer.c (sys_sysinfo)	3

3.3 与LSM其他钩子的结合

LSM原有的访问控制钩子在审计中也得到了运用，这些钩子主要有以下几个作用：

(1) *用作审计钩子* 在内核中加入审计钩子的原因是因为现有钩子不能满足审计需要，但是也有例外。如在系统调用setuid中，两个访问控制钩子security_task_setuid和security_task_post_setuid恰好分别处在系统调用的入口和出口处，而且它们的参数也满足审计要求，就可以利用这两个钩子来完成审计功能而不需再另外插入审计钩子。

(2) *判断是否进行客体审计* 这主要用在和文件、用户等客体有关的系统调用中。系统的审计规则定义了五种，包括全局审计、主体审计、客体审计、安全级别审计和无条件审计。在判断这些审计事件是否审计时，首先判断是否进行全局审计和主体审计。如果这两个条件都不满足，还不能立即将这个记录舍弃，还应该看这个事件涉及的客体是否有审计要求。LSM的钩子插入点都是在即将访问客体资源之前，因此，在这一步客体已经可以确定了。相应的客体安全信息(如安全级别、审计事件向量等)都可以确定了，在此可以方便地取出这些信息来进行判断。

(3) *完善审计记录* 这主要是用在和文件等客体有关的系统调用中。还是让我们来看看上一小节sys_open的例子，我们还需要收集文件描述符、文件属主和访问模式的信息。sys_open系统调用可以有几种功能，它可以完成创建新文件或打开已有文件。如果是打开文件，当内核函数找到该文件*i*节点后，如果成功了，则要进入钩子函数*security_inode_permission*来检查是否有访问权限。到这一步的时候，*i*节点已经找到了，*i*节点中的文件属性就可以得到了，我们就可以在这个钩子函数中收集关于客体的审计信息，完善审计记录。当然，如果没有找到*i*节点，这些信息是没有的，审计的返回值表明该事件失败。如果是创建一个文件，则由LSM钩子函数*security_inode_post_create*来完成相应的信息收集。

4 系统的实现与性能

基于扩展LSM框架的方法，我们按照中国国家标准GB17859-1999第四级的要求实现了一个审计系统[6]。这个系统是安全操作系统SECIMOS的一部分，通过向内核加入审计钩子，然后加载审计模块来实现。目前，审计系统已经成功移植到2.6.4内核下。

我们对加入审计功能的系统进行了性能测试，用lmbench[7]来从时间花费和带宽两方面分析，使用的机器cpu是Pentium 4，2.4GHz，内存为256M，cache为512K。进程进入系统调用后，收集审计信息并整理成审计记录必然要花费一定的时间，就单个系统调用来说，这种时间上的花费增长是不一样的。从我们测试的数据来看，open + close系统调用的时间增长是没加入审计前的6.75倍，而fork + exit系统调用的时间增长不足1%。这是因为他们要收集的信息不同，收集方式也有差异。要改善性能，一方面，我们可以对审计代码进行优化，减少这种时间花费。但更重要的是从整个系统来看，设置合理的审计配置，在保证审计要求的同时使整个系统的性能达到较好的水平。审计对系统的带宽影响不超过10%。

结束语 LSM是Linux内核中支持的一个安全模型支持框架，它为不同的安全模块提供统一的支持基础。审计是安全操作系统的一个重要组成部分。把审计支持纳入到LSM框架中，为审计系统或者是入侵检测系统提供一个统一的支持接口，对安全操作系统的研究有重要意义。

鉴于LSM的现有设计对审计机制的支持存在着明显的

维普资讯 http://www.cqvip.com

不足，本文提出一个扩展 LSM 框架的方法，以增强 LSM 框架对审计机制的支持能力。论文通过对 LSM 的扩展，讨论了审计功能的实现方法，论述了如何在 LSM 框架中加入审计钩子及如何在内核函数中插入钩子函数。依照这种方法，作为安全操作系统 SECIMOS 开发工作的一个重要组成部分，我们在 Linux 内核中实现了一个审计系统，取得了很好的效果。

参 考 文 献

1 訾小超，姚立红，曾庆凯，茅兵，谢立. 操作系统安全增强技术研究进展. 高技术通信，2003

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

3 Edward G A. Fundamentals of computer Security Technology. Prentice-Hall International Inc. 1994

4 国家863计划信息安全技术主题课题验收文件，操作系统与数据库平台安全核心技术研究技术报告. 中国科学院软件研究所，2004. 3

5 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，1999-9-13

6 陈慧，石文昌，梁洪亮，孙玉芳. 操作系统内核级安全审计系统的设计与实现. 计算机科学，2004，31(8)

8 McVoy L，Staelin C. lmbench: Portable tools for performance analysis. In: Proc. Winter 1996 USENIX，San Diego，CA，Jan. 1996. 279～284

(上接第143页)

例角色扮演。

用例角色扮演的过程如下：指定两个开发小组成员，分别扮演用户和系统两个角色。他们以本质用例对话过程为脚本，模拟用户与系统之间的实际交互。其他的开发小组成员则作为观众监督整个角色扮演过程。

在监督过程中需要注意以下问题。首先是连续性：确保用户和系统都能正确理解对方意图并作出适当的响应，不会因为误解而造成交互过程的中断；其次是避免假定信息：有时用户或系统会假定自己已获得某些信息，而实际上这些信息并未被提供。这两种情况一定要注意避免。

可以让项目风险承担者和领域专家共同参与到监督过程中来。整个角色扮演过程重复执行，直至整个开发小组对用例对话过程的理解都达成一致。

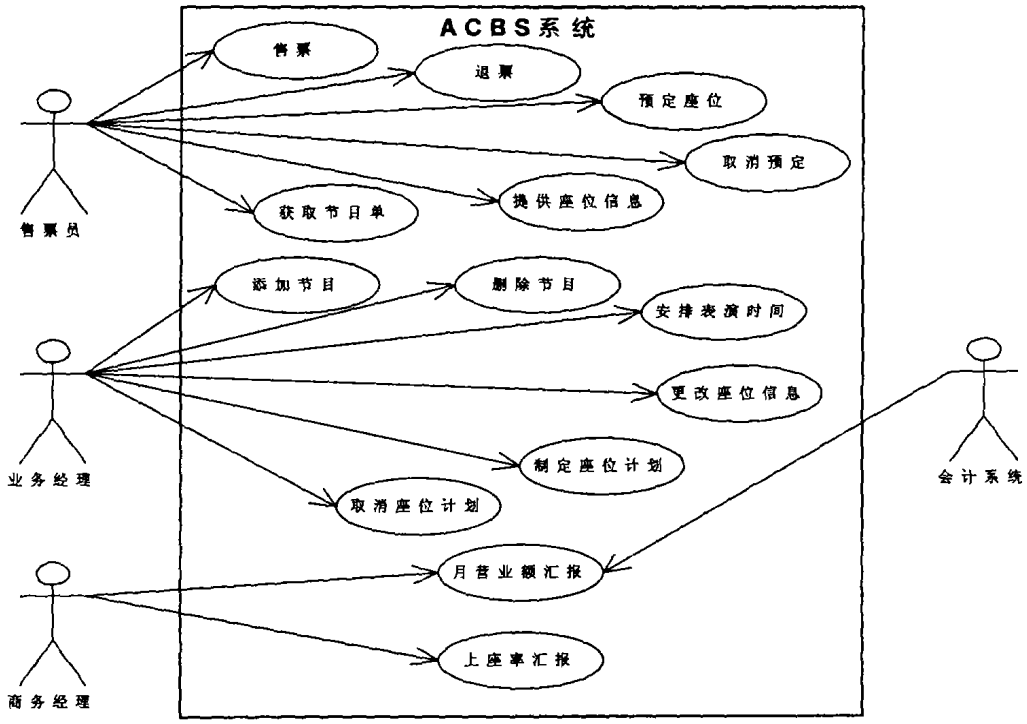

图3 艺术中心订票系统(ACBS)用例图

结束语 用例在软件需求的分析与获取中有其独有的优越性。本质用例作为用例概念的发展，在保留了这些优越性的同时，克服了传统用例所具有的局限性。相对于传统用例，本质用例更为简洁抽象。由于不涉及任何实现细节，可以支持更快速的开发过程。

Constantine 和 Lockwood 提出本质用例的最初目的是为了支持用户界面设计。随着对本质用例认识的深入，人们开始尝试将其应用到其他领域。在研究中我们发现，本质用例可以用来支持更为普遍的软件需求分析与获取。

本文首先介绍了本质用例的相关概念，并结合实例说明了基于本质用例的需求分析与获取过程。在整个的软件系统开发过程中，需求的分析与获取无疑是相当关键的，而本质用例为我们提供了一种行之有效的手段。

参 考 文 献

1 Jacobson I，et al. Object-Oriented Software Engineering: A Use Case Driven Approach. Addison-Wesley，1992

2 Constantine L L，Lockwood L A D. Software for Use: A Practical Guide to the Models and Methods of Usage Centered Design. Addison-Wesley，1999

3 Constantine L，Lockwood L. Structure and Style in Use Cases for User Interface Design，2000

4 Jacobson I，Booch G，Rumbaugh J. The Unified Software Development Process. Addison-Wesley，1999

5 Wirfs-Brock R J. Designing Scenarios: Making the Case for a Use Case Framework. The Smalltalk Report，3(3)，1993

6 Biddle R，Noble J，Tempero E. Patterns for Essential Use Cases: [Technical Report CS-TR-01/02]. 2000

7 Biddle R，Noble J，Tempero E. Essential Use Cases and Responsibility in Object-Oriented Development，2001

8 Oberg R，Probasco L，Ericsson M. Applying Requirements Management with Use Cases，Rational Software White Paper TP505，2000