openEuler内核编程技术

课程讲稿

第十章 第2讲

《**LSM框架**》

软件所制

第十章 第2讲 内核审计框架

**学时：**1学时

**教学目的：** 学习理解LSM框架，明白LSM在内核安全中所起到的作用。

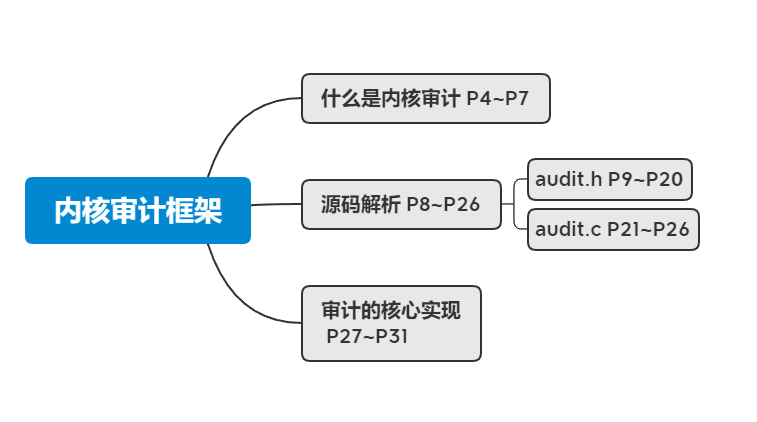
**课程时间线：**



**课外参考读物：**

**《Linux内核安全模块深入剖析》**

**知识框图：**



**PPT讲稿：**



我们来看一下Linux的优缺点。首先Linux作为一个开源操作系统，有众多技术高手为其贡献了高质量的代码，有出色的性能和稳定性。同时开源的形式也为其带来了非常棒的灵活性和可扩展性。最重要的是，Linux可以免费的获取，对于各行各业的人员都有很大的吸引力。但Linux也有其自身的局限性，其只提供了UNIX的自主访问控制，只有root和用户ID两种账户，存在一定的安全风险。



针对Linux的安全问题，也有很多的安全方案以及模型框架被提出来。比如安全增强型Linux，域和类型增强，以及Linux入侵检测系统。多种不同的安全方案也让我们操作系统的维护变得碎片化和不方便。所以在2001年的Linux内核峰会上，Linux作者Linus提出内核需要一个通用的安全访问框架，随后便诞生了LSM框架。



Linux安全模块有以下的四大特点。第一它是一个轻量级的通用访问控制框架。第二对于各种各样不同的安全访问控制模型，可以采用内核模块的形式进行实现。第三根据用户不同的需求，可以自有的选择合适的安全模块加载到内核中。第四，LSM的存在，极大的提高了Linux的安全性以及其灵活和易用性。



对于LSM的设计理念，以Linus为代表的的内核开发人员提出了三点要求。1. 通用。当使用不同的安全模型时，只需加载一个内核模块即可。2.简单。对Linux内核的影响要尽可能的小。3.支持现有的POSIX逻辑。为了实现这些设计要求，LSM在内核中放置了钩子，来实现安全访问，比如任务，文件，inode节点等。



一个典型的LSM框架下的系统调用如下。用户进程执行系统调用，首先游历Linux内核原有的逻辑找到并分配资源，进行错误检查。并经过经典的UNIX自主访问控制，恰好就在Linux内核试图对内部对象进行访问之前，一个Linux安全模块（LSM）的钩子对安全模块所必须提供的函数进行一个调用，从而对安全模块提出这样的问题“是否允许访问执行？”安全模块根据其安全策略进行决策，作出回答：允许，或者拒绝进而返回一个错误。



LSM是以一个Linux内核补丁的形式存在的。其本身不提供任何具体的安全策略，其主要在五个方面对Linux内核进行了修改：1.在特定的内核数据结构中加入了安全域。2在内核源代码中不同的关键点插入了对安全钩子函数的调用。3.加入了一个通用的安全系统调用。4.提供了函数允许内核模块注册为安全模块或者注销。5.将capabilities逻辑的大部分移植为一个可选的安全模块。



下面说一下访问控制的概念。



访问控制分为主体和客体两大基本组成部分。主体通常指用户，或代表用户意图运行进程或设备。主体是访问操作的主动发起者，它是系统中信息流的启动者，可以使信息流在实体之间流动。客体通常是指信息的载体或从其他主体或客体接收信息的实体。主体有时也会成为访问或受控的对象，如一个主体可以向另一个主体授权，一个进程可能控制几个子进程等情况，这时受控的主体或子进程也是一种客体。

客体不受它们所依存的系统的限制，可以包括记录、数据块、存储页、存储段、文件、目录、目录树、库表、邮箱、消息、程序等，还可以包括比特位、字节、字、字段、变量、处理器、通信信道、时钟、网络结点等。



自主访问控制分有不同的管理方式，管理的方式不同形成了不同的访问控制方式。一种方式是由客体的属主对自己的客体进行管理，由属主自己决定是否将自己客体的访问权或部分访问权授予其他主体，这种控制方式是自主的，我们把它称为自主访问控制（DAC）。在自主访问控制下，一个用户可以自主选择哪些用户可以共享他的文件。Linux系统中有两种自主访问控制策略：一种是9位权限码（User-Group-Other），另一种是访问控制列表ACL（Access Control List）。



我们看一下9位权限码。我们在命令行中输入ls -l，可以看到当前目录下的文件，以及其9位权限码。但显示的一共有10位，第一位表示的是文件类型，后9位是权限。第一位是可变字符，有以下的一些可选项，代表着不同的文件类型。



剩下的九位，每三位一组，分成了三组。从左至右分别是，文件所有者权限，文件所属组权限，其他人权限。每一组有三位，第一位是读权限，r表示可读，-表示不可读。第二位是写权限，w表示可写，-表示不可写，第三位是执行权限，x表示可执行，-表示不可执行。



再来看下强制访问控制。强制访问控制（Mandatory Access Control——MAC），用于将系统中的信息分密级和类进行管理，以保证每个用户只能访问到那些被标明可以由他访问的信息的一种访问约束机制。通俗的来说，在强制访问控制下，用户与文件都被标记了固定的安全属性（如安全级、访问权限等），在每次访问发生时，系统检测安全属性以便确定一个用户是否有权访问该文件。其中多级安全（MultiLevel Secure, MLS）就是一种强制访问控制策略。



访问控制作为一种最为基本和重要的安全机制，对于保护操作系统中资源免受非法访问起着关键的作用。

与DAC相比，MAC中的安全策略由安全管理员根据安全威胁和安全假设预先定义，用户或代表用户的进程即使拥有客体也不能修改其安全属性，从而能够有效防止特洛伊木马的攻击，这对于保证整个系统的安全具有重要意义，因此，MAC在高等级安全操作系统的设计中被强制要求。

然而，当今主流的操作系统并未提供MAC机制，难以为系统提供充足的安全保证。为此，学术界对如何增强操作系统的MAC机制进行了大量研究。Linux的创始人Linus认为Linux内核需要一个通用的访问控制框架，响应他的号召，DARPA等开展了LSM的研发。



下面来看Hook点分布和security\_ops结构 。



Linux安全模块（LSM）提供了两类对安全钩子函数的调用，一类管理内核对象的安全域，另一类仲裁对这些内核对象的访问。对安全钩子函数的调用通过钩子来实现，钩子是全局表security\_ops中的函数指针，这个全局表的类型是security\_operations结构：这个结构定义在include/linux/security.h这个头文件中



钩子函数分为两种，一种是任务钩子，另一种是程序装载钩子。Linux安全模块（LSM）提供了一系列的任务钩子使得安全模块可以管理进程的安全信息并且控制进程的操作。

任务钩子提供了控制进程间通信的钩子，例如kill()；还提供了控制对当前进程进行特权操作的钩子,例如setuid()；还提供了对资源管理操作进行细粒度控制的钩子，例如setrlimit()和nice()。



进程间通信IPC钩子，安全模块可以使用进程间通信IPC钩子来对System V IPC的安全信息进行管理，以及执行访问控制。

文件操作钩子，定义了三种钩子：文件系统钩子，inode结点钩子，以及文件钩子。Linux安全模块（LSM）在对应的三个内核数据结构中加入了安全域，分别是：super\_block结构，inode结构，file结构。

网络钩子，指的对网络的应用层访问使用一系列的socket套接字钩子来进行仲裁，这些钩子基本覆盖了所有基于socket套接字的协议。



安全域的内核数据结构各自所代表的内核内部对象。如表中所示。



下面讲Capabilities机制



Capabilities机制主要思想在于分割root用户的特权．即将root的特权分割成不同的capability，每种权能代表一定的特权操作。例如权能CAP\_SYS\_MODULE表示用户能够加载(或卸载)内核模块操作的特权操作。而CAP\_SETUID表示用户能够修改进程用户身份的特权操作。

在capability中，只有进程和可执行文件才具有权能．每个进程拥有三组权能集，分别称为cap\_effective、cap\_inheritable、cap\_permitted,其中cap\_permitted表示进程所能拥有的最大权能集。cap\_effective表示进程当前可用的权能集。而cap\_inheritable则表示进程可以传递给其子进程的权能集。系统根据进程的cap\_effective权能集来进行访问控制,cap\_effective为cap\_permitted的子集。进程可以通过取消cap\_effective中的某些权能来放弃进程的一些特权。



可执行文件也拥有三组权能集,对应于进程的三组权能集，分别称为cap\_effective、cap\_aIlowed和cap\_forced,cap\_allowed表示程序运行时可从原进程的cap\_inheritable中继承的权能集，cap\_forced表示运行文件时必须拥有才能完成其服务的权能集而，cap\_effective则表示文件开始运行时可以使用的权能。



目前．Linux定义了29种权能。因此在进程控制结构task\_struct中用三个32位的整数：cap\_effective，cap\_inheritable，cap\_permitted来分别表示进程的三组权能集，整数的每位代表一种权能(高3位没有定义)。

在task\_struct中还引进了1个控制位keep\_capabilities。用于控制进程的用户身份由root特权用户变为普通用户时是否保持其权能信息不变。Keep\_capabilities的值可以通过系统调用prctl()来改变。



那么 Capabilities机制是如何实现的呢？Linux使用一个全局变量cap\_bset．甩来限定系统中所有进程所能拥有的权能．将cap\_bset中的某权能位清0则系统所有进程不会再拥有此权能，cap\_bset的值只能在内核编译前改变．



进程权能集的计算和调整（1）init进程的具体赋值：init进程是系统启动后运行的第一个用户进程．它的各权能集和keep\_capabilities的值在宏定义INIT\_TASK中具体赋值（2）进程调用fork()、vfork()或cIone()等函数生成子进程时．子进程复制父进程的各权能集；（3）改变进程映象后的新进程各权能集的重新计算



下面讲一下smack机制。



Smack（Simplified Mandatory Access Control Kernel）是于2007年在LSM基础上实现的Linux强制访问控制安全模块，它以内核安全补丁的形式存在于Linux操作系统中。其设计思想是利用LSM安全域将Linux内核中所有主体与客体都打上安全标签，并规定安全策略，只有符合安全策略的访问方式才被容许。与SELinux和DTE相比，Smack安全策略要简单得多，但却能实现它们相似的强制访问控制功能，并且Smack对内核性能损耗比较低，因此，Smack也逐渐被业界看好和推广。



Smack主体是指Linux内核进程。Smack客体是指Linux内核客体对象，如文件、消息队列、套接字、共享内存、信号量等，客体也可以是Linux进程或者IPC。Smack安全标签是C语言的字符串，但最多包含24个字符（包括‘/0’），Smack修改了进程task\_struct安全域，在进程被初始创建时，其安全标签是“\_”。同样，Smack修改了虚拟文件系统的inode和super\_block安全域，使得文件系统被创建时所有文件的安全标签是“\_”。Smack最初版本的访问方式只有四种，即读（r或R）、写（w或W）、执行（x或X）、盲写（a或A）。其中，在进程之间通信中，一个进程发送消息或者数据包给另一个进程时，这样的操作属于写操作。



Smack的安全策略分为Smack内置的安全策略和用户可定制的安全策略。Smack内置的安全策略是指原本就已经被Smack访问控制代码所规定的，包括如下几条：

（1）安全标签是“\*”的进程发起的任何形式的访问都被拒绝；

（2）安全标签是“^”的进程发起的读或执行的请求都被容许；

（3）任何进程对安全标签是“\_”的客体发起的读或执行的请求都被容许；

（4）任何进程对安全标签是“\*”的客体发起的任何形式的请求都被容许；

（5）如果主体和客体的安全标签相同，那么该主体对该客体发起的任何形式的访问都被容许



除此之外，用户可以根据自己的安全需求，在主、客体安全标签都已经存在的前提下，通过“smackload”工具写入安全策略。当用户连续写入以上三条规则后，Smack会将相同主、客体的规则进行覆盖，最终，abc对xyz没有任何权限。



2007版本的smack源码并不庞大；它被组织成了smack.h、smack\_access.c、smack\_lsm.c和smackfs.c四个源文件；下面本小节就针对这四个文件来分析Smack的实现原理。



LSM的安全域是“void \*security”，它可以指向任意类型的指针，正因为此，Smack定义了几个重要的结构体，使得内核对象的安全域指向它们，下表总结了LSM安全域和Smack结构体之间的对应关系如表中所示。



Smack访问控制函数smk\_access定义如下所示。

smk\_access首先根据Smack默认的五条访问规则进行判断，然后在smack内核规则链表中去查找和参数主客体标签一致的结点，判断此结点的权限是否包含请求的权限request。

访问控制函数smk\_curacc用于判断当前进程对目标客体是否有访问权限，smk\_curacc\_on\_task用来判断当前进程对目标进程是否有访问权限，它们最终是调用smk\_access来进行访问控制。

需要指出的是，smk\_curacc可以被特权进程绕过，正因为此，Smack不能阻止超级用户或特权进程的一切行为。



Smack是在LSM的基础上实现钩子函数以达到强制访问控制的目的，下表总结了Smack访问控制对象及其相应的钩子函数。



如之前的表所示，Smack是在LSM基础上实现了对进程、文件、套接字、共享内存、消息队列、信号量等的控制。Smack的访问控制非常简单，就是检查主体对客体有没有Smack访问方式，除此之外，Smack使用了钩子函数为进程和文件设置安全标签。当把Smack编译进Linux内核后，Linux操作系统上所有文件和进程默认的安全标签都是“\_”。

只有特权进程才能改变自身的安全标签，只有特权进程才能设置文件的安全标签。一个进程无法改变另一个进程的安全标签。

当父进程创建子进程时，子进程的安全标签就是父进程的安全标签。

当一个进程创建文件时，这些文件的安全标签就是进程的安全标签。

当一个进程创建套接字时，套接字的安全标签就是该进程的安全标签。

当两个进程要通过网络进行通信时，发送和接收IP包的进程必须对彼此都有写权限。



虚拟文件系统：

smack\_access.c和smack\_lsm.c构成了Smack强制访问控制机制，但强制访问控制机制是需要安全策略数据库的支持，为此，Smack采取了虚拟文件系统作为其安全策略存储系统，这样Smack决策时间会很短，

但这也就意味着，一旦计算机系统重新启动，需求人为得再次设定安全策略。Smack文件系统位于“/smack”目录下，包含了load、cipso、doi、direct、ambient、netlabel、onlycap和logging这几个虚拟文件，它们被组织成Linux内核链表形式，其中最常用到的是load、cipso和netlabel。

load文件存放了Smack的安全策略，cipso存放cipso值，包括安全级别和安全分类，netlabel存放了主机的IP地址和其相关的Smack标签，cipso和netlabel用于Smack网络通信控制。

Smack重写文件系统操作函数，并使用注册文件系统、挂载文件系统和初始化系统调用等内核API来实现虚拟文件系统。



Linux安全模块（LSM）的起因有两个方面，一方面Linux内核现有的安全机制是不足够的；另一方面现存的安全增强系统又各自为战并且难以使用。

Linux安全模块（LSM）比较好的解决了这个问题，一方面补丁比较小，对内核源代码的修改影响不多，所带来的负载也不大，另一方面对现存的安全增强系统提供了比较好的接口支持，并已经有不少很好的安全模块可以使用。

LSM目前仍然是作为一个Linux内核补丁的形式提供，我们期待着那一天：LSM被Linux内核接受成为Linux内核安全机制的标准，在各个Linux发行版中提供给越来越多的用户使用。