OpenEuler内核编程技术

课程讲稿

第十章 第1讲

《**内核审计框架**》

软件所制

第十章 第1讲 内核审计框架

**学时：**2学时

**教学目的：** 学习理解内核审计框架，明白内核审计在内核安全中所起到的作用。

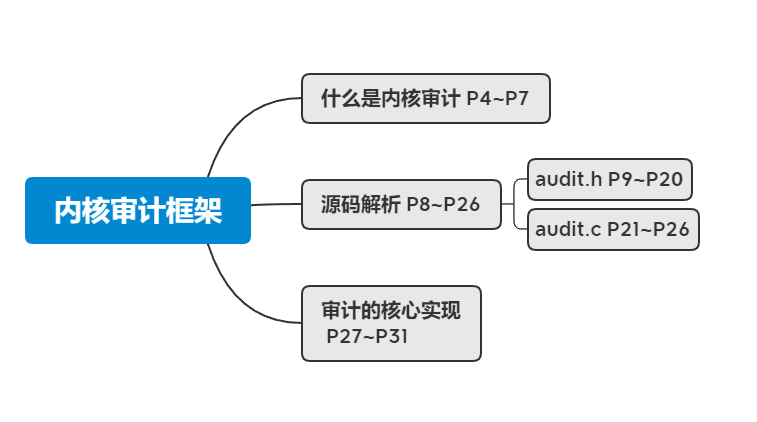
**课程时间线：**



**课外参考读物：**

**《Linux内核安全模块深入剖析》**

**知识框图：**



**PPT讲稿：**

1. 同学们好，今天我们开始进入第十章的学习
2. 第十章中我们分为四个小节，第1讲：内核审计框架，第2讲：LSM框架，第3讲：AppArmor，第4讲：内核中key的管理。本次我们进行第一讲的内容讲解：内核审计框架。
3. 在本讲中我们主要从三个方面进行介绍：内核审计是什么，源码解析和审计的核心实现，我们首先来看一下内核审计它到底是什么。
4. 在Linux内核中，审计日志是安全模块的重要组成部分。举一个贴近使用的例子，假设我做了一个网站并发布到了网上，我的网站服务器使用了自己搭建的Linux服务器，然后就开始了网站的运营。然后某一天，发生了某些特殊的情况，（例如被篡改了主页/其他人获取了网站管理权限等等情况），证明我自己搭建的网站服务器遭到了入侵。那么接下来我应该怎么办？黑客现在获取了我的网站的权限这个事情暂且不表，现在我们来讨论一下另一个事情：服务器的安全问题。
5. 出现了被入侵的情况，作为服务器的主人第一反应是当前损失是什么，而第二反应就应该是他(黑客)是怎么做到的？我的服务器是不是有安全漏洞？那么，这个时候，我们的主角，内核审计(audit)就出场了。简单一点讲的话，audit的作用就相当于是它是一个摄像头，不停的保存着系统内核的情况，所记录的信息格式大概是这样：“某某时间：某某用户进行了某某操作。”

严肃一点描述的话，就是audit保存了在内核态中进行的关键操作日志。

1. 注意区分一下“审计”和“日志”的区别：“审计”等义于“审计日志”，与“日志”的区分在于：记录的主体不一样。审计是“在内核中根据设定的规则生成审计消息“，而日志是“基于一种自觉行为，系统的多个守护进程在执行过程中发送日志消息给日志服务进程，后者将消息记录到日志文件中。”换句话说就是审计是在核心态进行的记录，而日志是在用户态。
2. 那么回到我们的小故事，我现在就可以查看审计内容来一步步查看黑客是如何获取连接、提升自己的权限以完成网站的入侵修改操作的。然后通过这一步步的操作来看关键点在哪出现了漏洞，哪是本不应该出现的情况使其钻了空子——也就是寻找系统漏洞。

那么聪明的你已经明白了，‘审计’在安全模块的作用并不是说能够阻止安全问题的发生，而是在安全问题发生后能够及时找到问题所在，甚至可以寻找到是谁进行了安全攻击——当然这在一定程度上是可以阻止安全问题的发生的，就像商场安装了摄像头可以避免很多的盗窃行为一样。

1. 接下来我们进入源码解析的部分。
2. 我们首先来看audit.h这个文件。audit.h头文件中主要存储了一些在（audit.c及其他文件中）具体实现时用到的数据结构，因此首先介绍一下这些在此定义的数据结构，为解读具体实现audit.c文件做基础。其实就是在头文件之中进行了一些数据结构的定义。
3. **enum audit\_state**

在这里使用enum枚举声明了audit审计模块可能的活动级别，分别是：

AUDIT\_DISABLED：不记录任何系统调用

AUDIT\_BUILD\_CONTEXT：在进入系统调用时进行记录

AUDIT\_RECORD\_CONTEXT：在进入和退出系统调用时进行记录

分别代表了三种不同的安全级别，逐层提高。

1. 在这里声明了一个数据结构audit\_entry，实际上是一个用于保存审计规则的双向链表。

list\_head：

用于创建双向循环链表的结构。其在内核中被定义见下一页。

struct list\_head {

　struct list\_head \*next, \*prev;

};

可以看到，在此引入list\_head，其实就是实现了双向链表的前后指针。

虽然linux内核是用C语言写的，但是list\_head的引入，使得内核数据结构也可以拥有面向对象的特性，通过使用操作list\_head 的通用接口很容易实现代码的重用，有点类似于C++的继承机制



**struct audit\_entry {**

**struct list\_head list;**

**struct rcu\_head rcu;**

**struct audit\_krule rule;**

**};**

在这里声明了一个数据结构audit\_entry，实际上是一个用于保存审计规则的双向链表。



rcu\_head：

RCU(Read-Copy Update)，用于实现读-拷贝修改。深入理解其机制在此不表，仅描述其作用如下：通过引入此数据结构，实现了对此数据结构（audit\_entry）的RCU保护，使得读者需要访问该信息时无需获取任何锁，但是写者在访问时首先拷贝一个副本，然后对副本进行修改，最后使用一个回调（callback）机制在适当的时机把指向原来数据的指针重定向至被修改后的数据。简单描述就是用于控制访问冲突，对读者开放，对写者进行访问控制（避免冲突）。



struct audit\_entry {

struct list\_head list;

struct rcu\_head rcu;

struct audit\_krule rule;

}

在这里声明了一个数据结构audit\_entry，实际上是一个用于保存审计规则的双向链表。

audit\_krule：

具体的data域，保存一条条具体的规则。这里的规则就是指在具体什么样的前提下进行审计记录。对于不同的情况下规则表不同，因此需要使用这样的一个数据结构来进行记录。



struct audit\_cap\_data {

kernel\_cap\_t permitted;

kernel\_cap\_t inheritable;

union {

unsigned int fE;

kernel\_cap\_t effective;

};

kernel\_cap\_t ambient;

kuid\_t rootid;

};

audit\_cap\_data是用于保存能力数据的数据结构，在此简要介绍其内容。

kernel\_cap\_t类型是无符号整数类型，一共32位，每一位表示系统细分的一命特权。

effective表明了进程当前使用的特权情况；permitted表明进程可拥有的最多特权；inheritable包含进程下一步exec新文件时所拥有的特权集。



struct audit\_names {

struct list\_head list;

struct filename \*name;

int name\_len;

bool hidden;

unsigned long ino;

dev\_t dev;

umode\_t mode;

kuid\_t uid;

kgid\_t gid;

dev\_t rdev;

u32 osid;

struct audit\_cap\_data fcap;

unsigned int fcap\_ver;

unsigned char type;

bool should\_free;

};

简单理解为是一个用于存储文件名及文件属性信息的数据结构。

其中list\_head list上文讲过，是用于实现双向链表的结构；

filename\* name用于存储文件名（实际上存储的是保存着文件名的数据结构的指针）；

audit\_cap\_data fcap也是上文中提过的用于保存能力数据。



struct audit\_proctitle {

int len;

char \*value;

};

字面意思理解：是一个用于存储审计进程名的数据结构。内含一个长度和一个字符数组(其实就是字符串)罢了。



核心数据结构：audit\_context。

如果审计规则成立(被启用)，则在系统调用进入时创建此数据结构用于记录信息，在系统调用退出时（如果需要记录）维护此数据结构中信息。

1. 关于audit\_context:

其中部分信息的解释如下：

\* int in\_syscall：当前进程是否在系统调用中。

\* enum audit\_state state, current\_state：记录audit的状态。前文有介绍。

\* unsigned int serial：audit的序列号，用于唯一标识此记录。

\* struct timespec64 ctime：生成此记录的时间。

\* long return\_code：系统调用的返回值。

\* int return\_valid：系统调用的返回值是否合法。

1. 接下来我们来看audit.c这个文件，它提供了核心的审计机制。
2. 首先是init函数，审计子系统的初始化由audit\_init执行。除了设置数据结构之后，该函数还创建了一个netlink套接字，与用户层通信。

在初始化函数中，涉及的操作并不多，在此简要介绍：一开始的

是一个明显的判断，判断当前audit的state是否是disable状态，如果是的话无需审计，直接返回退出。

1. 接下来的一系列操作是在存储audit的数据结构（事实上是队列，用来存储一系列audit）中初始化其在多个队列（audit\_queue、audit\_retry\_queue、audit\_hold\_queue）中的位置。

这里便是在创建与用户层通信的套接字。

1. 这里是audit\_receive函数
2. audit\_receive函数是用于处理auditd发送过来的数据。

与当前我们分析的audit工作在内核态不同，auditd是工作在用户态的进程，主要负责消息传递。内核audit向外传送消息使用了上文提到了netlink套接字，套接字连接的就是auditd进程。同时auditd也负责向内核传送用户态的一些消息。

我们在这里提到的audit\_receive函数就是用于处理auditd通过netlink套接字向内核传送的用户态信息。



注意一下函数的参数`struct sk\_buff \*skb`就是我们需要的message buffer.

后续首先通过`nlmsg\_hdr`函数获取skb的netlink报头，获取skb的长度，然后开始进行循环处理：

通过skb消息与其netlink套接字的报头nlh经过`audit\_receive\_msg`函数处理得到返回值err，主要就是是进行判断二者是否匹配（核查从用户态发来的数据）。

如果发生了不匹配（err返回值=1）或者是netlink报头中指明需要回复的情况时，就需要通过`netlink\_ack`函数进行回复：回复内容主要就是err信息。

1. 接下来我们来看一下内核审计的代码上的具体实现
2. **审计的核心实现分为三步：首先，用`audit\_log\_start`开始记录过程。然后，用`audit\_log\_format`格式化一个消息，最后用`audit\_log\_end`结束该审计记录，消息通过netlink发送到auditd。**

**struct audit\_buffer \*audit\_log\_start(struct audit\_context \*ctx, gfp\_t gfp\_mask,**

**int type)**

**{**

**struct audit\_buffer \*ab;**

**struct timespec64 t;**

**unsigned int uninitialized\_var(serial);**

**if (audit\_initialized != AUDIT\_INITIALIZED)**

**return NULL;**

**if (unlikely(!audit\_filter(type, AUDIT\_FILTER\_EXCLUDE)))**

**return NULL;**

**//……**

**}**

用`audit\_log\_start`函数开始审计过程，其中包括初始化检查、缓冲区限制检查等一系列限制问题的判断，在判断未出错的前提下即可开始审计过程。



**void audit\_log\_format(struct audit\_buffer \*ab, const char \*fmt, ...)**

**{**

**va\_list args;**

**if (!ab)**

**return;**

**va\_start(args, fmt);**

**audit\_log\_vformat(ab, fmt, args);**

**va\_end(args);**

**}**

此函数用于向一个给定的审计缓冲区(ab)写入格式化消息（fmt），参数中的…是可选的对fmt字符串的限制。



**void audit\_log\_end(struct audit\_buffer \*ab)**

**{**

**struct sk\_buff \*skb;**

**struct nlmsghdr \*nlh;**

**if (!ab)**

**return;**

**if (audit\_rate\_check()) {**

**skb = ab->skb;**

**ab->skb = NULL;**

**nlh = nlmsg\_hdr(skb);**

**nlh->nlmsg\_len = skb->len - NLMSG\_HDRLEN;**

**skb\_queue\_tail(&audit\_queue, skb);**

**wake\_up\_interruptible(&kauditd\_wait);**

**} else**

**audit\_log\_lost("rate limit exceeded");**

**audit\_buffer\_free(ab);**

**}**

在所有必要的记录消息都已写入到审计缓冲区之后，需要调用audit\_log\_end确保将审计记录发送到用户空间守护进程auditd。

1. 本节内容就是这样。