openEuler内核编程

课程讲稿

第10章 第4讲

《内核中key的管理》

软件所制

第10章 第4讲 内核中key的管理

**学时：**1学时

**教学目的：**学习内核中key的设计原理和管理机制，理解key的应用场景，能够理解keyctl工具的作用和内核服务对key的使用场景。

**课程时间线：**

概述

相关系统调用

相关文件

用户态操作接口库

25 min

45 min

**课外参考读物：**

**知识框图：**

****

**PPT讲稿：**



这一节课，我们介绍一下内核中的key管理。



**略**。



**主要从一下几个方面来介绍，首先是对key这个概念与实现的简单描述，然后我们将介绍key实现过程中添加的系统调用及内部逻辑，最后，从用户态程序的角度，我们介绍Linux发行版中看到的与key相关的文件的含义，封装了用户态key操作接口的keyutils库**。



**首先介绍一下key这一概念。在本讲中，我们介绍内核中的一种用户凭证——key，一般来说，key是可以是一段数据，在某些场景下这段数据作为用户进行某些操作的凭证，比如进行任务管理时。另外一方面，出于安全性考虑，key可以作为用户请求内核进行某些重要操作的用户凭证，不适合直接存储在静态文件系统中。所以，Linux为key进行了一些更加安全的管理机制，在内核实现中，key数据存在于内存中，实现代码主要为Key结构体**。



Key结构体定义了多个成员，其中包括每个key独有的、仅在内核中使用的32位ID，该key的类型，用于描述该key的字符串，同时可用于检索。该key的控制权限信息，以及key结构体维护的核心数据，放在payload成员中。



在内核服务中，key类型可以被灵活定义和删除，与之相反的是，用户态程序只能够使用已有的key类型。默认存在三类特殊的key类型为keyring即钥匙环类型，user类型和logon类型。其中keyring类型支持管理其他key，从而实现key的分层管理。



除开三类内置的特殊Key类型之外，Linux还默认实现了两种安全性更高的Key类型，Trusted类型和Encrypted类型。Trusted类型在内核中创建，并且默认加密payload内容，从用户态接口只能够看到加密后的内容，在实现上，Trusted类型依赖于硬件模块TPM。Encrypted类型与Trusted类型的作用相似，但不依赖于硬件TPM，从软件实现上使用AES算法进行加解密。



对于一个key，可以设置不同的处理权限，包括是否允许查看、读取内容、修改内容，是否允许搜索或链接入另一个钥匙环等。



接下来，我们对Key相关的系统调用进行简单介绍。



我们在这里简单列举了key添加的三个系统调用以及内核服务可见的一些内核态编程接口。系统调用包括添加一个key、请求一个key和对某一个key或keyring进行操作。内核态编程接口则允许进行key的释放、key类型的注册等更多细粒度的操作。



整个key的管理体系对用户程序可见的只有三个系统调用所描述的功能，系统调用的内部逻辑通过内核态接口来进行实现，内核态接口可以在编写内核服务的时候使用，从而内核服务可以使用完整的、更加贴近底层的key管理体系。



对key的操作与内核中的相关函数直接对应起来，能够直接对应到内核中的Key结构体实例。



相比而言，暴露给内核态服务的key操作接口更加丰富，不仅有释放和搜索等常规功能，还有注册类型、取消类型等更灵活的操作。



在一系列接口的实现中，request操作的实现是值得学习的，这一系统调用的实现过程中包含了若干重的用户态回调，当请求的key不存在时，内核可以根据传入的callout\_info信息进行用户态回调，由指定的用户态程序来新建一个key，最终作为结果返回。



详细的创建过程与我们在上一页概述的过程大致相同，但多了一个认证密钥V的创建和回收，认证密钥V的作用是让用户态回调时创建的用户进程helper，与进行系统调用的caller进程有相同的权限，这是/sbin/request-key所要求的合法权限。后续的key初始化程序由/sbin/request-key根据合理的callout\_info来选择并执行。



在后续创建的新进程中，可能会需要获取进程A持有的某些密钥，此时该新进程可以通过keyring中V的存在，接触到进程A的上下文中的密钥，正确给出搜索结果。正确完成初始化之后，当前程序将会exit 0退出，执行权回到内核，内核中将V废弃，将U作为结果返回到caller，回到进程A继续执行，进程A看到request操作成功。



分析到这里，也许有些同学已经反应过来了，这样的回调创建过程可能会需要进行递归，这也是Key V存在的另一个重要原因，如果Key U初始化过程中要求的某一个密钥W在进程A的keyring中同样不存在，那么需要对W进行创建的操作，此时新的创建过程将被启动，有一种类似递归的执行流程。由于V的存在，在创建W的过程中，使用的上下文仍然可以使用进程A的上下文。同时，由于进程A的keyring中key的数量是有限的，可以限定递归层数是有限的。

使用这种方式而不是简单地将A的keyring交给/sbin/request-key的一个重要原因是，/sbin/request-key中执行execv等系统调用时，并不会默认传递keyring，如果要这么做势必会复杂化进程创建操作的实现。



Keyctl是Key相关操作的一个集成型系统调用，具有较强的可拓展性，同时也较为复杂，这里列举了一部分调用参数。Keyctl的设计原则可以参考ioctl的相关设计原则，相当于将一些杂项集成到一个系统调用。



接下来，我们简单介绍一下用户态可读写的一些key操作的相关文件。



这里列举了一部分key相关的文件，都挂载在/proc虚拟目录下



Keys文件提供了访问当前进程可见的密钥的信息，包括我们之前介绍的串行码和状态、UID、GID等。



这里是在某个linux发行版上的示例



Key-users文件记录着当前系统中持有密钥的用户的配额情况，包括各个用户的已有秘钥数目和可配置密钥数目，限定了不同用户可以使用的key资源。



另外，还有一些与key管理状态相关的文件，包括root用户和非root用户所能够使用的最大数量和key数据大小。



最后，我们介绍一下进行key的用户态操作的一个接口库，keyutils库。



Keyutils库提供了三个系统调用的简单封装，可以供C语言和其他语言的程序直接使用，同时面向用户提供了一个可执行程序keyctl。



除此之外，keyutils还包括我们前面提到过的/sbin/request-key用于key的用户态回调创建。以及一些配置文件位于/etc文件夹下。



最后，展示一个在C语言中使用keyutils库的例子，在代码中关联对应的头文件，并在链接时链接keyutils库，就可以在C语言中使用对应接口。同时，使用可执行文件的例子可以使用help选项进行查看。



**略**