老罗的Android之旅 爱生活，爱Android

个人资料

罗升阳

访问： 7041610次
积分： 36313
等级： BLOG 8
排名： 第61名

原创： 154篇 转载： 0篇
译文： 0篇 评论： 7107条

SEAndroid安全机制简要介绍和学习计划

标签： Android SEAndroid SELinux MAC DAC

2014-06-30 00:58 28169人阅读 评论(29) 收藏 举报

分类： Android（152）

与iOS相比，Android最被人诟病的是其流畅性和安全性。然而，从4.0开始，Android不遗余力地改善其流畅性。特别是在即将发布的L版本中，用ART替换了Dalvik，相信会越来越流畅。至于安全性，Android也没有遗忘。从4.3开始，Android引入了一套基于SELinux的安全机制，称为SEAndroid，来加强系统安全性。接下来我们就对SEAndroid进行简要介绍和制定学习计划。

老罗的新浪微博：http://weibo.com/shengyangluo，欢迎关注！

在介绍SEAndroid安全机制之前，我们要先了解一下Android当前所采用的安全机制是什么。实际上，在前面从NDK在非Root手机上的调试原理探讨Android的安全机制一文中，我们已经介绍过Android系统的安全机制了。因此，这里主要是进行一下总结。

在引进SEAndroid安全机制之前，Android系统的安全机制分为应用程序和内核两个级别。应用程序级别的安全机制就是我们通常说的Permission机制。一个应用如果需要访问一些系统敏感或者特权资源，那么就必须要在AndroidManifest.xml配置文件中进行申请，并且在安装的时候由用户决定是否赋予相应的权限。应用安装过后，一般是通过系统服务来间接使用系统敏感或者特权资源的。这样系统服务在代表应用使用这些资源之前，就会先检查应用之前是否已经申请过相应的权限。如果已经申请过，那么就直接放行，否则的话，就拒绝执行。

内核级别的安全机制就是传统的Linux UID/GID机制。在Linux中，每一个用户都拥有一个用户ID，并且也有一个用户组ID，分别简称为UID和GID。此外，Linux系统的进程和文件也有UID和GID的概念。Linux就是通过用户、进程、文件的UID/GID属性来进行权限管理的。

我们知道，当Linux内核启动完成之后，启动的第一个进程称为init进程。Init进程接着会启动一个login进程，等待用户输入用户名和密码登录。一旦登录成功，就会启动一个shell进程。之后这个shell进程就负责执行用户的命令。注意，在上述启动过程中，init进程是以root用户身份启动的，也就是init进程的UID是0，即root。在默认情况下，由init进程启动的所有进程，也就是fork出来的所有子进程，也同样是以root身份运行的。因此。负责登录的login进程的UID也是0。但是，当用户输入用户名和密码，并且登录成功后，所启动的shell进程就不再是root了，而是成功登入系统的用户。这是如何做到的呢？原来，具有root权限的进程可以通过系统接口setuid来改变自己身份。也就是说，由login进程启动的用户shell进程在开始的时候的身份其实也是root的，不过在它可以执行用户的命令之前，它已经通过setuid将自己的UID修改为登录用户对应的UID了。这样就可以限制每一个成功登入系统的用户的权限。

用户之后在shell进程中执行的各种命令，要么是在本进程中执行，要么在启动的子进程中执行。注意，根据我们上面的分析，由用户shell进程启动的子进程同样是以登录用户的身份运行的，并且这些进程在运行的过程中所创建的文件的默认UID和GID也是和登录用户的UID和GID一致的，并且这些文件只可以被用户自己访问。如果一个用户想将一些自己创建的文件交给另外一个用户访问，那么应该怎么办呢？Linux将文件的权限划分为读、写和执行三种，分别用字母r、w和x表示。每一个文件有三组读、写和执行权限，分别是针对文件的所有者、文件所有者所属的组以及除了所有者以及在所有者所属组的用户之外所有其它用户。这样，如果一个用户想要将一个自己创建的文件

交给另外一个用户访问，那么只需要相应地设置一下这个文件的其它用户权限位就可以了。

我们知道，Android是一个基于Linux内核的系统，但是它不像传统的Linux系统，需要用户登录之后才能使用。然而，Android系统又像传统的Linux系统一样有用户的概念。只不过这些用户不需要登录，也可以使用Android系统。具体来说，就是Android系统将每一个安装在系统的APK都映射为一个不同的Linux用户。也就是说，每一个APK都有一个对应的UID和GID。这些UID和GID是在APK安装的时候由系统安装服务PackageManagerService分配的。

我们知道，APK所运行在的进程是由另外一个系统服务ActivityManagerService负责启动的。ActivityManagerService在启动APK进程之前，会先向PackageManagerService查询APK安装时分配到的UID和GID。有了APK的UID和GID后，ActivityManagerService就向另外一个以root身份运行的zygote进程发出创建APK进程的请求。Zygote进程收到请求之后，就会fork出一个子进程来作为请求创建的APK进程。APK进程的创建过程的详细分析可以参考Android应用程序进程启动过程的源代码分析一文。

注意，我们上面提到，zygote进程是以root身份运行的。因此，它fork出来的子进程，也就是APK进程，在一开始的时候也是以root身份运行的。不过，APK进程在可以执行APK代码之前，会通过系统接口setuid将自己的UID设置为APK安装时分配到的UID。这个过程与传统的Linux系统通过login进程启动用户shell进程的过程非常类似。通过这种方式，就可以保证每一个APK进程都以不同的身份来运行，从而保证了相互之间不会受到干扰。这就是所谓的沙箱了，这完全是建立在Linux的UID和GID基础上的。

以上分析的UID/GID机制可以通过图1来描述：

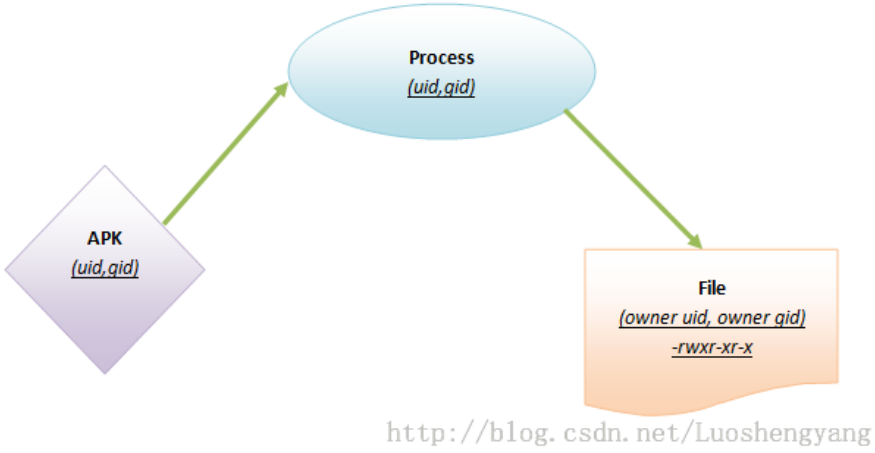

图1 Android系统基于UID/GID的安全机制

这种基于Linux UID/GID的安全机制存在什么样的问题呢？注意我们前面提到的，当一个用户想将授予另外一个用户访问自己创建的文件的时候，它只需要修改一下该文件的访问权限位就行了。也就是说，在Linux系统中，文件的权限控制在所有者的手中。因此，这种权限控制方式就称为自主式的，正式的英文名称为Discretionary Access Control，简称为DAC。

在理想情况下，DAC机制是没有问题的。然而，在现实中，会产生严重的安全问题。例如，一个用户可能会不小心将自己创建的文件的权限位错误地修改为允许其它用户访问。如果这个用户是一个特权用户，并且它错误操作的文件是一个敏感的文件，那么就会产生严重的安全问题。这种误操作的产生方式有三种：

1. 用户执行了错误的命令

2. 负责执行用户命令的程序有BUG

3. 负责执行用户命令的程序受到攻击

由此可见，DAC机制只能在理想情况下没有问题，但是在现实中是防不胜防！例如，GingerBread漏洞就是通过攻击以root身份运行Android磁盘管理守护进程vold来获得root权限，从而实现对设备进行root的。

注意，上面我们说的DAC问题虽然是针对内核级别的Linux UID/GID机制，然而同样适用于应用级别的Permission机制。这个问题通过MasterKey漏洞表现得淋漓尽致。我们知道，MasterKey漏洞可以在不改变签名的情况下对APK进行篡改。这会导致什么后果呢？假如被篡改的APK申请有特殊的Permission，那么就意味着嵌入的恶意代码可以任意地使用这些特殊的Permission。

更为严重的是被篡改的APK是一个系统APK。Android系统的Permission分为两种，一种是所有APK都可以申请的，另一种是系统APK才可以申请的。只有系统APK才可以申请的Permission更为敏感，例如用来安装APK的Permission--android.permission.INSTALL_PACKAGES。这意味着一个具有android.permission.INSTALL_PACKAGES的系统APK被篡改后，恶意代码就可以在设备上安装任意的恶意APK了。

事实上，应用级别的Permission机制也是建立在Linux UID/GID基础上的。当我们在APK的AndroidManfest.xml配置文件中申请某一个Permission的时候，Android系统的安装服务PackageManagerService除了会记录它申请有相应的Permission之外（以便APK调用需要权限的API接口时进行验证），还会将APK加入到相应的某个Linux用户组去。这是因为在Android系统中，并不是所有的特权操作都是间接地通过系统服务来执行的，例如网络访问。一旦一个APK申请网络访问的Permission，那么它就会加入到Linux的网络用户组去，这时候APK就可以通过创建socket来访问网络了。

这样，在Android系统中，无论是应用级别的Permission机制，还是内核级别的Linux UID/GID机制，都同样会受到DAC问题的困扰。这时候我们就需要一种更为强有力的安全机制来保证系统的安全。

在访问控制模型中，与DAC机制相对的是MAC机制。MAC的全称是Mandatory Access Control，翻译为强制访问控制。在MAC机制中，用户、进程或者文件的权限是由管理策略决定的，而不是由它们自主决定的。例如，我们可以设定这样的一个管理策略，不允许用户A将它创建的文件F授予用户B访问。这样无论用户A如何修改文件F的权限位，用户B都是无法访问文件F的。这种安全访问模型可以强有力地保护系统的安全。我们在这个系列的文件中要介绍的SEAndroid就是一种MAC机制。

在SEAndroid中，每一个进程和文件都会关联有一个安全上下文。这个安全上下文由用户、角色、类型、安全级别四个部分组成，每一部分通过一个冒号来分隔。例如，u:r:t:s0描述的就是一个SEAndroid安全上下文。当每一个进程和文件都关联上一个安全上下文之后，系统管理员就可以基于这些安全上下文制定一个安全访问策略，用来规定什么样的进程可以访问什么样的文件。

上面描述的SEAndroid安全机制如图2所示：

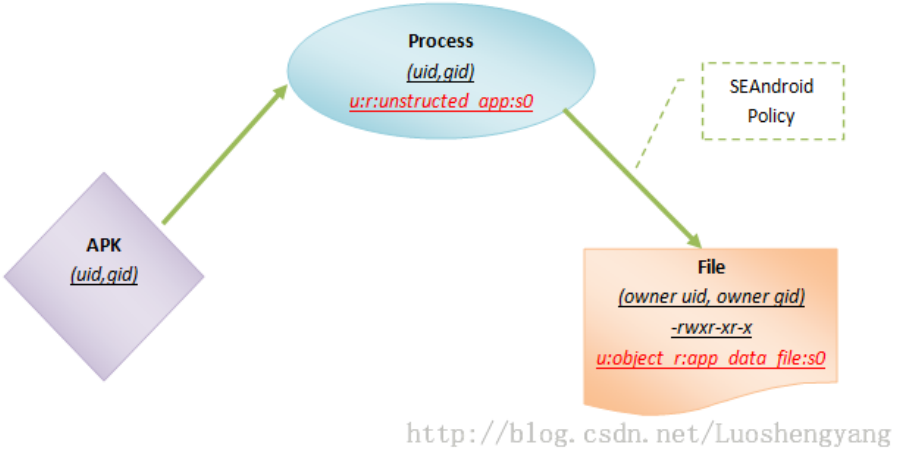

图2 SEAndroid安全机制

在图2中，红色标注的即为SEAndroid安全上下文。其中，u:r:unstructed_app:s0描述的是用户安装的APK所运行在的进程的安全上下文，而u:object_r:app_data_file:s0描述的是用户安装的APK在运行过程中生成的数据文件的安全上下文。

从图2还可以看到，SEAndroid安全机制与传统的Linux UID/GID安全机制是并存关系的，也就是说，它们同时用来约束进程的权限。当一个进程访问一个文件的时候，首先要通过基于UID/GID的DAC安全检查，接着才有资格进入到基于SEAndroid的MAC安全检查。只要其中的一个检查不通过，那么进程访问文件的请求就会被拒绝。上述的安全检查过程如图3所示：

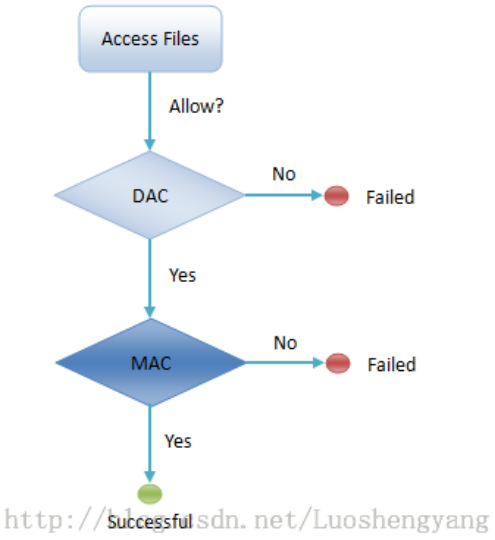

图3 基于Linux UID/GID和SEAndroid的安全访问流程

我们通过一个例子来说明上述的安全访问流程。当我们想从手机上下载一个文件到电脑上时，我们使用adb pull命令来实现。当我们执行adb pull命令的时候，实际上是由手机上的守护进程adbd来读出指定的文件，并且将读出来的内容发送给在电脑上运行的adb进程的。接下来，我们就按照以下步骤尝试从启用了SEAndroid的三星Note II上下载文件/system/bin/gpsd到电脑上来。

1. 执行ls -l命令检查手机上存在/system/bin/gpsd文件，以及它基于传统的Linux UID/GID的权限位：

```
[plain]
01. $ ./adb shell ls -l /system/bin/gpsd
02. -rwxr-xr-x root     shell     2822268 2014-02-11 03:27 gpsd
```

从命令的输出可以看到，如果只考虑传统的Linux UID/GID安全机制，手机上的/system/bin/gpsd文件是所有用户均可以读取的。

2. 执行adb pull命令下载手机上的/system/bin/gpsd文件：

```
[plain]
01. $ ./adb pull /system/bin/gpsd ./gpsd
02. failed to copy '/system/bin/gpsd' to './gpsd': Permission denied
```

从命令的输出可以看到，我们没有按照预期那样将手机上的/system/bin/gpsd文件下载电脑上来，原因是“Permission denied”，也就是权限不够。

3. 分别通过ls -Z和ps -Z命令检查文件/system/bin/gpsd和进程adbd的安全上下文：

```
[plain]
01. ./adb shell ls -Z /system/bin/gpsd
02. -rwxr-xr-x root     shell             u:object_r:gpsd_exec:s0 gpsd
03. $ ./adb shell ps -Z | grep 'adbd'
04. u:r:adbd:s0                   shell     1978  1     /sbin/adbd
```

从命令的输出可以看到，文件/system/bin/gpsd的安全上下文为u:object_r:gpsd_exec:s0，而进程adbd的安全上下文为u:r;adbd:s0，因此我们可以断定，在三星Note II运行的系统上，一定存在一个访问策略不允许安全上下文为u:r;adbd:s0的进程访问安全上下文为u:object_r:gpsd_exec:s0的文件。

从上面这个例子就可以看出，在启用SEAndroid之前，原本可以访问的文件，到启用SEAndroid之后，就变得不可以访问了！如果我们确实是需要访问这些文件，例如我们需要将这些文件打包在我们自己制作ROM里面，那么有没有其它办法访问呢？当读完SEAndroid安全机制这个系列的文章之后，你就会发现答案是肯定的，并且是在遵循SEAndroid安全策略的前提下实现的！

关于SEAndroid安全机制，还有一个关键点是值得提及的。那就是SEAndroid安全机制的目的不是为了完全杜绝别人攻击我们的设备，而是为了保证我们的设备受到攻击时，受到的损害减少到最少的程度。例如，SEAndroid安全机制并不能完全阻止我们的设备被root，但是它能保证我们的设备被root之后，一些敏感的文件仍然是不可访

问，这样就可以最大程度地保护我们的设备。这是因为只要程序是由人类来编写的，就或多或少地存在BUG，或者说漏洞，特别是复杂的程序，进而就会被黑客利用，并且成功地侵入到我们的系统中来。这是防不胜防的。当然，我们并不是说SEAndroid对阻止设备被侵入毫无用处，它在一定程度上还是能加大侵入的技术难度的。

由于SEAndroid的内 写一本很厚很厚的书来描述，但是在接下来的文章中，老罗并不打算逐一逐一地介绍，而是主要抓住 洋细的分析，因此希望同学们在继续学习接下来的文章之前，可以读读以下的一本书以及一篇论文：

1. SELinux by Exar ecurity Enhanced Linux

2. Security Enhanc id: Bringing Flexible MAC to Android

事实上，接下来介绍 文章都是基于上面的论文来Security Enhanced (SE) Android: Bringing Flexible MAC to Android展开的，目的是从Android系统源码分析的角度来阐述该论文的内容。

好了，依照惯例，接下来我们按照以下的情景来深入学习SEAndroid安全机制：

1. SEAndroid安全机制框架分析

2. SEAndroid安全机制的文件安全上下文分析

3. SEAndroid安全机制的进程安全上下文分析

4. SEAndroid安全机制的Binde IPC保护支持分析

关闭

支持分析

对Android系统的安全机制有一个深刻的认识，以帮助我们更好地保护可以关注老罗的新浪微博：http://weibo.com/shengyangluo。

顶 28 踩 2

上一篇 从CM刷机过程和原理分析Android系统结构
下一篇 SEAndroid安全机制框架分析

我的同类文章

Android（152）

- Chromium网页Render Obj… 2016-02-15 阅读 1264
- Chromium网页URL加载过… 2016-01-25 阅读 7381
- Chromium网页加载过程简… 2016-01-04 阅读 3684
- Chromium硬件加速渲染的O… 2015-12-07 阅读 5013
- Chromium硬件加速渲染的O… 2015-11-09 阅读 5953
- Chromium网页DOM Tree创… 2016-02-01 阅读 1909
- Chromium网页Frame Tree… 2016-01-11 阅读 3856
- Chromium硬件加速渲染的U… 2015-12-21 阅读 5139
- Chromium硬件加速渲染的G… 2015-11-23 阅读 5977
- Chromium硬件加速渲染的O… 2015-11-02 阅读 3997

更多文章

主题推荐 安全

猜你在找

Android底层技术：Linux驱动框架与开发
Linux操作系统及常用基础命令深入讲解
Linux操作系统及常用命令实战和进阶
Linux核心必备技能-用户及权限详解
Android移植基础
SEAndroid安全机制框架分析
SEAndroid安全机制对Android属性访问的保护分析
SEAndroid安全机制框架分析
SEAndroid安全机制对Android属性访问的保护分析
SEAndroid安全机制对Android属性访问的保护分析

查看评论

27楼 叶桐 2015-06-18 14:10发表

还有一点，比MTK的文档看起来消化的要快得多。

26楼 叶桐 2015-06-18 14:08发表

通俗易懂这是最重要的，也佩服老罗的学识、研究精神和分享精神。

25楼 wangjian1937 2015-06-11 08:39发表

老罗 最近在研读你的源码分析 确实写的不错 超赞

24楼 u012568263 2015-05-27 09:21发表

一點小問題。

1. “GingerBreak漏洞”， 應是"GingerBread漏洞"。
2. "因此希望同学们在继续学习接下的文章之前"，這裡"接下的"宜改為"接下來的"。
2. “好了，废话不说了。”， 我閱讀這篇文章不下十遍， 沒讀過一句廢話，處處是振聾發聵之語，因此 “废话不说了”，最好修飾成“言歸正傳“， 或”依照慣例“，比較合理。

Re: 罗升阳 2015-05-28 09:38发表

回复u012568263：这次第三点也按照你说的改了:)

23楼 a343649190 2015-01-13 10:38发表

罗老师 你好 想问一下SEAndroid对内核有什么要求？ 用3.0的内核Selinux的状态是disabled

22楼 edept 2014-07-02 23:06发表

顶一个~我们公司正在做SEAndroid和Linux的东西呢，期待以下几节课

21楼 edept 2014-07-02 23:05发表

顶一个~我们公司正在做SEAndroid和Linux的东西呢，期待以下几节课

20楼 wudiisss 2014-07-02 09:44发表

罗老师你好，三星Note II上含有seandroid安全机制吗？不是4.3才有吗，Note II是4,1的，需要刷机吗？

Re: 罗升阳 2014-07-07 09:41发表

回复wudiisss：你把官方Push的固件更新都装上去，系统就会升级到4.3的了，也就会有SEAndroid的了

19楼 mylove167 2014-07-01 16:47发表

zan yi ge

18楼 yfmlj 2014-07-01 14:43发表

看老罗的文章就像教科书那般条理和严谨，让我等IT屌丝发自内心的佩服，更佩服老罗这种我为人人的分享精神。

17楼 yfmlj 2014-07-01 14:40发表

顶老罗，老罗的博客充实了无数IT屌丝的程序人生。

16楼 ghb602731501 2014-07-01 13:29发表

给力啊

15楼 airk000 2014-07-01 13:04发表

老罗又发力了！！！说的还是那么井井有条！！！赞！

14楼 ps_andy 2014-07-01 10:31发表

感谢，感谢分享。

13楼 echoJiang 2014-07-01 10:26发表

你的小说看完了哦，呵呵，讲的很有条理也很透彻，L系统只是略有耳闻，我们还有产品时2.3的--！持续关注中。

12楼 盘锦--志强 2014-07-01 10:15发表

定一下！感谢分享！

11楼 Ritter_Liu 2014-06-30 20:07发表

感谢老罗，犹如灯塔一般的存在，如果不是老罗的分享，我对Android的学习，还不知道停留在啥地方呢。。。

10楼 soledadzz 2014-06-30 18:22发表

您的文章已被推荐到CSDN首页，感谢您的分享。

9楼 Bludge0n 2014-06-30 16:28发表

坐等接下来的分享！

8楼 李启林liqilin 2014-06-30 16:01发表

顶老罗！

7楼 _一统_ 2014-06-30 15:57发表

http://www.j1758.com/Info/d/52
不是因为有了希望才坚持,而是因为坚持才有了希望!

6楼 bluebirddm 2014-06-30 14:16发表

冒泡，顶一顶

5楼 zbl_zbl 2014-06-30 13:12发表

老罗你回来了！！

4楼 jltxgcy 2014-06-30 11:48发表

顶起，必须的。安全我太想学习了，期待。老罗。

3楼 zhxjun123456 2014-06-30 09:27发表

又开始更新了啊，顶一个！
Selinux确实很有用处！尤其是到了android-l上，改动好大..

2楼 BadPattern 2014-06-30 09:09发表

佩服老罗的研究精神,学习计划制定的非常棒!

1楼 BadPattern 2014-06-30 08:41发表

顶,谢谢分享!