genglei.cuan

不断成长的见证

Android 5.1 SEAndroid分析之启动篇

2015-09-28

Android核心服务 Android底层开发

前面介绍了init进程中启动的核心服务中的zygote和property,接下来就介绍下SEAndroid.init进程中加载了SEAndroid的策略文件,开启了SEAndroid.SEAndroid主要用来提升Android系统的安全体验,基于SElinux实现.在完备的SEAndroid策略的支持下,纵然获取了root权限,也没有多大用处,再也不能像以前那样为所欲为了.因为在SEAndroid中,root用户,也仅仅是一个普通用户而已.

Android sandbox 模型

Android基于linux系统,Android 自然而然的继承了linux的沙箱模型,只不过Android做了稍许修改而已.

linux系统中每个用户都有一个UID,不同用户之间的数据是隔离的,在没有授权的情况下,用户只能访问自己的数据,而不能访问其他用户的数据.同一用户启动的进程,其UID和GID一致,但是可以通过使用setuid和setgid等系统调用改变UID和GID.Linux 支持多用户工作，每个用户之间都使用了因为UID不同,进而达到相互隔离，互不影响,起到沙箱的作用.

Android扩展了Linux内核安全模型的用户与权限机制，将多用户操作系统的用户隔离机制巧妙地移植为应用程序隔离。在linux中，一个用户标识（UID）识别一个给定用户；在Android上，一个UID则识别一个应用程序。在安装应用程序时向其分配UID。应用程序在设备上存续期间内，其UID保持不变。仅限用于允许或限制应用程序（而非用户）对设备资源的访问。如此，Android的安全机制与Linux内核的安全模型完美衔接！不同的应用程序分别属于不同的用户，因此，应用程序运行于自己独立的进程空间，与UID不同的应用程序自然形成资源隔离，如此便形成了一个操作系统级别的应用程序“沙箱”。

Android系统中将UID通常称为APP ID.在手机:

```
1  /data/system/packages.list
```

中记录了当前系统安装的App信息,其中就有UID,也就是APP ID.

```
com.android.providers.telephony 1001 0 /data/data/com.android.providers.telephony platform 1028,1015,3002,3001,3003
com.android.providers.calendar 10002 0 /data/data/com.android.providers.calendar default 3003,1028,1015
com.android.providers.media 10006 0 /data/data/com.android.providers.media default 1028,1015,1023,1024,2001,3003,3007
com.android.wallpapercropper 10018 0 /data/data/com.android.wallpapercropper platform none
com.google.android.launcher.layouts.shamu 10049 0 /data/data/com.google.android.launcher.layouts.shamu default none
com.android.voicedialer 10016 0 /data/data/com.android.voicedialer default 3002
com.android.documentsui 10027 0 /data/data/com.android.documentsui platform none
com.android.galaxy4 10030 0 /data/data/com.android.galaxy4 default none
com.android.externalstorage 10007 0 /data/data/com.android.externalstorage platform 1028,1015,1023
com.android.htmlviewer 10032 0 /data/data/com.android.htmlviewer default 1028
```

其中第二列就是APP ID.Android 系统本身定义了很多UID,比如1000 代表 system用户等.

Android源码/system/core/include/private/android_filesystem_config.h中记录了Android系统定义的用户UID

```
#define AID_ROOT             0  /* tr

#define AID_SYSTEM        1000  /* sy

#define AID_RADIO         1001  /* te
#define AID_BLUETOOTH     1002  /* bl
#define AID_GRAPHICS      1003  /* gr
#define AID_INPUT         1004  /* ir
#define AID_AUDIO         1005  /* au
#define AID_CAMERA        1006  /* ca
#define AID_LOG           1007  /* lo
#define AID_COMPASS       1008  /* co
#define AID_MOUNT         1009  /* mo
#define AID_WIFI          1010  /* wi
#define AID_ADB           1011  /* ar
 . . . . . .
#define AID_SDCARD_RW     1015  /* ex
. . . . .
#define AID_SDCARD_R      1028  /* ex
. . . .
#define AID_APP          10000  /* fi

.................

#define AID_USER        100000  /* of

.........
```

可以看到安装的app的 UID是从10000开始分配的.另外要注意的是,Android 已经支持多用户了,多用户机制后面有机会的话,会单独开文章介绍.Android多用户在手机上没有太大的使用场景,因为手机通常属于私人设备,不像PC那样,可能有多个人使用,有必要设置多个账

genglei.cuan

不断成长的见证

主页

所有文章

LINUX基础

ANDROID底层开发

APP开发

项目管理

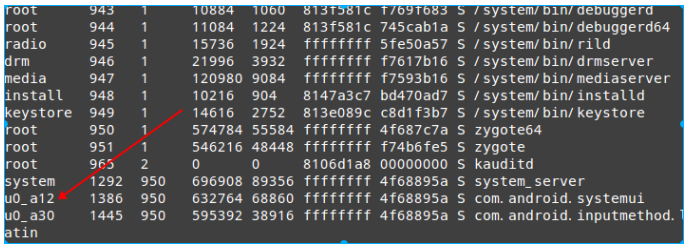
PYTHON

不断成长的见证

随笔

号.所以现在手机厂商默认都将android多用户机制裁剪掉了.

adb shell 登陆手机终端,并执行ps命令:



第一列就是APPID,类似uY_aXXX,如上图所示.其中XXX是相对于AID_APP.Y是Android user id(这里的user id和前面说的UID含义不一样,是Android多用户中的某个用户ID).

前面android_filesystem_config.h中

```
1  #define AID_USER        100000  /* off
```

可以看到每个 Android user id之间相差100000.比如上图中的u0_a12,实际上就是100000+12=100012,也就是说其APPID是100012.

实际上linux 沙箱模型可以归结为一句话,进程的权限,取决于启动他的用户的权限.比如说某个进程是root启动的,那么他就拥有root权限.这种控制模型叫做DAC,全称是Discretionary Access Control，翻译为自主访问控制.

特殊权限标志位

su这个程序,大家应该都不陌生,有了他,就能获取root权限.它是怎么做到的呢?

```
1  root@generic_x86_64:/system/xbin # ll
2  -rwsr-x--- root     shell      338416
```

看第一列,似乎多了一个特殊的权限标志位”s”,这个特殊权限标志位叫做SUID.

当一个设置了SUID 位的可执行文件被执行时，该文件将以所有者的身份运行，也就是说无论谁来执行这个文件，他都有文件所有者的特权。再来看su他的拥有者是谁,是root.所以su才能使其他程序获得root权限.

genglei.cuan

不断成长的见证

Capability

Linux系统分为root用户和非root用户，root用户至高无上，想干嘛干嘛，至于非root用户权限有限，想获得root权限，可以通过setuid系统调用实现。如果一个进程想要一个大于非root用户权限，只能将自己提升到root权限；实际上此进程并不需要这么多权限，这可能会导致很多风险(一个进程权限太大)，所以POSIX制定了Capability机制来细分root的权限。

Android系统中也使用到了这个机制.

Android源码/system/core/include/private/android_filesystem_capability.h

```
#define CAP_CHOWN 0
#define CAP_DAC_OVERRIDE 1
#define CAP_DAC_READ_SEARCH 2
#define CAP_FOWNER 3
#define CAP_FSETID 4
#define CAP_KILL 5
#define CAP_SETGID 6
#define CAP_SETUID 7
#define CAP_SETPCAP 8
#define CAP_LINUX_IMMUTABLE 9
#define CAP_NET_BIND_SERVICE 10
#define CAP_NET_BROADCAST 11
#define CAP_NET_ADMIN 12
#define CAP_NET_RAW 13
#define CAP_IPC_LOCK 14
#define CAP_IPC_OWNER 15
#define CAP_SYS_MODULE 16
#define CAP_SYS_RAWIO 17
#define CAP_SYS_CHROOT 18
#define CAP_SYS_PTRACE 19
#define CAP_SYS_PACCT 20
#define CAP_SYS_ADMIN 21
. . . . . .
```

共有37个权限.

怎么查看进程具备哪些Capability呢?

我现在以 adb shell的方式连接到了 user版本的android设备,我想看 shell终端具有哪些能力,可以如下操作:

```
130|shell@shamu:/ $ ps | grep shell
shell     5960  1     16980  212   ffffffff
```

```
3  shell     5979  5960  9320   764   c01
4  shell     6045  5979  10676  992   006
5  shell     6046  5979  10676  988   c02
6  shell@shamu:/ $
```

找到 shell的进程号为5979.然后就从proc文件系统中查看shell进程具有哪些能力了:

```
1  . . . . .
2  CapInh: 0000000000000000
3  CapPrm: 0000000000000000
4  CapEff: 0000000000000000d
5  CapBnd: 00000000000000c0
6  . . . . .
```

是以位图的形式表示的,前面说过有37个能力 ,所有37个bit来表示,为1说明具有相应的能力;为0说明不具备相应的能力.

其中 CapPrm表示进程具有的最大能力集,CapEff表示当前进程的有效能力集,通常是CapPrm的子集;CapInh表示当前进程启动其他程序,那些可以被继承的能力.

CapBnd很重要,表示边界能力.上面shell进程CapBnd的能力位图为c0,即11000000,从android_filesystem_capability.h可以可知其代表

```
1  #define CAP_SETGID 6
2  #define CAP_SETUID 7
```

意味着从shell启动的程序,就算具备特权标志位,他的能力也会被限制为CapBnd.shell进程CapInh也为0,所以从shell启动的进程的能力被限制为CAP_SETUID和CAP_SETGID capabilities,其他能力都被限制了.

也就是说,你有su,也白搭,因为你的能力被完全限制了,只有37种能力中的两种.倘若你执行的操作不涉及37种能力,则不受影响.当然如果你的设备是 eng或者debug版本的话,默认能力是全打开的.

能力是对root权限的切割,目前只切割出了这37种能力而已.

SEAndroid的引入

DAC太过宽松了，那么SELinux如何解决这个问题呢？原来，它在DAC之外，设计了一个新的安全模

genglei.cuan

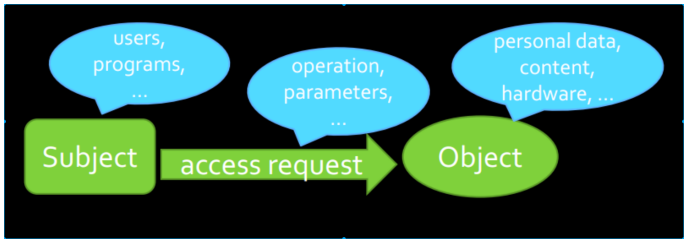
不断成长的见证

genglei.cuan

不断成长的见证

型，叫MAC（Mandatory Access Control），翻译为强制访问控制。MAC的处世哲学非常简单：即任何进程想在SELinux系统中干任何事情，都必须先在安全策略配置文件中赋予权限。凡是没有出现在安全策略配置文件中的权限，进程就没有该权限.

如下图所示:



Subject:主体,通常指的是可执行程序.

Object: 客体,通常指某种资源,如文件,硬件等.

用规则来描述主体是否对客体有某种操作权限.

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。

SELinux架构

SEAndroid可以理解为是SELinux的精简版,更适合移动设备.所以有必要介绍下SELinux的架构.

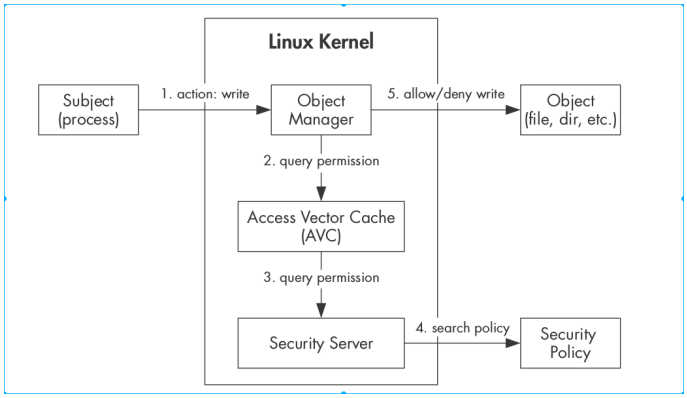
Linux内核中有一个Linux Security Modules,简称LSM框架,允许第三方权限访问机制挂接到内核中来.很类似于linux驱动模型,框架什么的都有了,只要你按照这个框架写的东西,都可以挂接到内核中去.SElinux就是由美国NSA按照linux内核安全模型框架编写的基于MAC访问控制的模块.

可以将LSM框架简单的理解为一系列的Hook方法,这些方法会在访问资源的时候被调用,用来简称是否有权限.

下图所示中间部分,即为LSM模块.

genglei.cuan

不断成长的见证



当某一进程主体发起对某一个文件客体写操作的时候,LSM模块Hook到了此次请求,LSM内部的Object Manager首先在 权限访问缓存中查找是否有对应的规则,如果没有的话,Security server就会去查询安全策略中是否有该规则.如果没有的话,就拒绝此次操作;有的话,就将查询到的规则放到访问向量缓存中去,然后向Object Manager报告,允许此次操作,那么最后进程主体就可以对客体文件进行写操作了.

DAC机制和MAC机制可以并存,执行的时候先检查DAC是否满足,如果不满足,那就不需要进行MAC检查了.只有当满足DAC之后,才会进行MAC检查.

SELinux的三种状态:

Disable: 关闭

Permissive: 宽松状态,违反策略,紧紧是提示而已,操作仍能进行

Enforcing: 执行状态,违反策略,操作会被拒绝执行.

adb shell进入android 设备终端,执行如下命令,可以查看当前状态:

```
1  shell@shamu:/ $ getenforce
2  Enforcing
```

可以使用 setenforce命令,在Permission和Enforcing状态切换,如果是Disable的话,不能切换状态.

```
1  # getenforce
2  Enforcing
3  # setenforce 0
4  # getenforce
5  Permissive
6  # setenforce 1
7  # getenforce
8  Enforcing
```

Android 4.2之前的系统都是Disable.

Android 4.3: Permissive.

– With all domains permissive + unconfined.

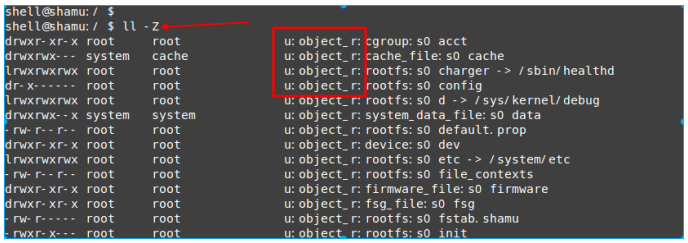
Android 4.4: Enforcing.

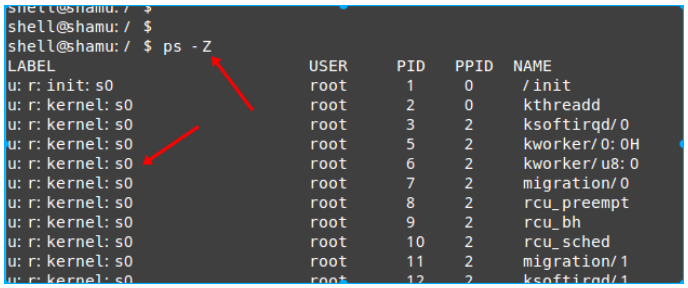
– Enforcing for installd, netd, vold, and zygote.
– Permissive for app domains (logging denials).
– Permissive + unconfined for all other domains.

Android 5.0: Enforcing.从此版本开始,有了较完备的支持.

查看SELinux 安全上下文

在相关命令前,加-Z即可查看SElinux 安全上下文,如下图所示





< Android 5.1 SEAndroid分析之安全上下文语法篇
Android 5.1 property属性系统分析下篇 >

genglei.cuan

不断成长的见证

分享到：