**1 Selinux 基础**

**1.1 什么是Selinux**

　　SEAndroid 是 Google 在 Android 4.4 上正式推出的一套以SELinux 为基础与核心的系统安全机制。而 SELinux 则是由美国 NSA（国安局）和一些公司（RedHat、Tresys）设计的一个针对 Linux的安全加强系统。

　　SELinux 是一种访问控制系统，其核心就是对系统中的各种操作进行控制。为了实现这一目的，需要做两个修改：

　　1、在各种操作实际进行之前添加钩子，以判断权限并控制后续操作；

　　2、对不同的操作进行标记，并制定相应的控制策略；

　　对于第一点，是由 SELinux 实现框架来做到的，对于初学者不必深入研究。第二点才是我们要重点研究的内容。

**1.2 如何将系统中的操作进行标记**

**1.2.1 什么是操作**

　　之前也介绍过，Selinux 是一种访问控制系统，说的直白点，就是对访问权限进行控制，允许（不允许）某种主体对某种类型的客体做什么样的动作，如果没有定义权限，那么默认不允许；

**1.2.2 主体、客体及动作**

　　对于任何一个操作，都包含主体、客体、动作这三个部分：

　　1.主体通常就是进程或者内核的线程（后面以进程说明）

　　2.客体是系统中所有可以被操作的资源，包括文件、进程、设备等等

　　3.在操作系统的正常运行中会有大量的动作，因此 SELinux的设计者把这些动作根据客体类别（object class）进行分类，并且把每个类别所可以进行的动作都定义好。例如：file 类别定义了read、write、open 等权限，dir 类别定义了 add\_name、remove\_name、search 等权限

　　除了标准 SELinux 定义的类别，SEAndroid 还定义了特有的类别，包括 binder、zygote、property\_service、service\_manager等。

**1.3 什么是 Scontext？**

　　在 Selinux 中，主体和客体都被赋予了一个属性，叫做Security Context（安全上下文），简称 Scontext。这个属性就是主体和客体的唯一标识。

　　在 SeAndroid 中，使用命令 ps -z 即可查看所有进程的Scontext，使用 ls -Z 可以查看当前目录文件的 Scontext，

如图：



比如 Vendor 文件的 Scontext 就是 u:object\_r:vendor\_file:s0.

　　据 SELinux 规范，SContext 的格式为：

user:role:type:[range]

　　user：表示对象的所属用户。在 SEAndroid 中，目前仅定义一个 user（也就是 u）。

　　role：表示活动对象的角色。在 SEAndroid 中，目前仅定义一个 role（也就是 r）。非活动对象此处均为 object\_r。

　　type：表示对象的类型。

　　range：表示对象的分级。如不同级别的资源需要对应级别的进程才能访问，一般都是 s0。

　　其中 type 是需要重点关注的，其它几个暂时无需关注。

**1.4 什么是 type？**

　　在 SELinux 体系中，所有的东西都要有 type。为了做到精确控制，必须对不同的东西定义不同的 type。这样就会引入大量 type，而规则又都是使用 type 来进行控制的，有些 type 是有共性的（比如，vfat 和 fuse都属于 sd 卡的文件系统），为了同一目的，可能需要需要编写多条规则。因此 SELinux 的设计者引入了attribute，在定义 type 时可以指定 attribute。同一 type 可以指定多个attribute，不同 type 也可以指定同一个 attribute。在 SEAndroid 中，对于进程的 type 都会指定 domain 属性。对于文件的 type 都会指定 file\_type属性，而可执行文件的 type 都会指定 exec\_type 属性。

例如：

type vold, domain;

type vold\_exec, exec\_type, file\_type;

vold type 就是 vold 进程的 type，所以拥有 domain 属性；而vold\_exec 是 vold 可执行文件的 type，所以同时拥有 exec\_type和 file\_type 属性。

type 定义命令的完整格式为：

type type\_id [alias alias\_id] , [attribute\_id];

如果在 type 定义之后再指定属性可以用 typeattribute。

例如：

typeattribute vold mlstrustedsubject;

特别注意：

　　当遇到neverallow问题时，但是在文件中却找不到冲突的语句，可能就是有的type不但使用type type\_id [alias alias\_id] , [attribute\_id]这种格式定义，还有其他地方使用的typeattribute来增

加定义，此点很重要，算是解决selinux问题的一个关键点吧。

**1.5 如何进行规则定义？**

　　SELinux 控制各种操作的策略是通过 TE 语句来完成的。最普遍的就是 allow 语句，例如：

allow vold device:dir write;

　　这句的意思是：允许 type 为 vold 的主体 对 type 为 device的客体 拥有 客体类别为 dir 的write 权限。根据 SELinux 规范，规则定义的格式为：

rule\_name source\_type target\_type : class perm\_set;

　　可以看到规则中只用到了主体和客体的 type，而动作是以操作类别与具体操作的组合体现的。

　　所有 rule\_name 如下：

　　　　allow 表示允许主体对客体执行允许的操作；

　　　　auditallow 表示即便允许操作也要记录访问决策信息(仍然需要有 allow 规则才允许)；

　　　　dontaudit 表示违反规则的决策信息也不记录(便于定位问题，已知此操作会被拒绝但不会引起真正问题)；

　　　　neverallow 表示不允许主体对客体执行指定的操作；

**2 实例分析**

**2.1 章节背景**

　　此章节的目的为以一个在系统中创建服务为例子，帮助同学理解在实际项目中，会遇到的一些实际问题。

**2.2 创建服务**

　　想要创建服务，首先需要修改代码中的 init.rc，代码如下：

　　+service closesourcecode /system/bin/closesourcecode  
　　+ class main  
　　+ user root  
　　+ disabled  
　　+ onshot  
　　+ seclabel u:r:closesourcecode:s0  
　　+  
　　+on property:sys.move.vendor=1  
　　+ start closesourcecode

　　以上即创建了一个名字为 closesourcecode 的服务，当属性sys.move.vendor 为 1 时，即调用此服务，服务执行的程序为/system/bin/closesourcecode

　　Seclabel 是用于在服务在运行之前修改其安全上下文，主要用在位于 rootfs 分区的服务，比如ueventd, adbd。而位于 system分区的服务，则可以通过 transitions 规则进行修改，如果没有指定transition 规则，那么默认就是 init context。

　　由之前的基础知识可知 seclabel u:r:closesourcecode:s0中的 u:r:closesourcecode:s0 指的是closesourcecode 这个服务的 Scontext（安全上下文），seclabel u:r:closesourcecode:s0中的

closesourcecode 指的是服务的类型，即 type，所以接下来我们为这个服务以及服务对应的可执行文件定义 type;

**2.3 为服务以及服务对应的可执行文件定义 type**

　　根据 1.4 章节的内容可知，为 closesourcecode 以及服务对应的可执行文件定义 type 的代码

如下：

　　+type closesourcecode, domain, coredomain;  
　　+type closesourcecode\_exec, exec\_type, file\_type;

　　解析一下:

　　定义一个名字为 closesourcecode 的 type，这个 type 拥有domain(进程)属性，且是一个coredomain（核心进程）。

　　定义一下名字为 closesourcecode\_exec 的 type，这个 type拥有 file\_type（文件）属性，且是一个 exec\_type（可执行文件）。

　　由于 closesourcecode 这个 type 在创建服务的时候已经指定了 Scontext（安全上下文），所以不需要定义 Scontext（安全上下文），但是 closesourcecode\_exec 这个 type 没有对应的 Scontext（安全上下文），所以我们需要自己创建。

**2.4 创建 Scontext（打标记）**

　　在 Linux 系统中，大概可以分成两种东西，一种是死的（比如文件、端口、系统属性等），一种是活的（就是进程）。此处的“死”和“活”是一种比喻，映射到软件层面的意思是：进程能发起动作，例如它能打开文件并操作它。而文件只能被进程操作。

　　给死东西打标记很容易，我们先来看一下这个。

　　在 SEAndroid 系统中 external/sepolicy 下有这么几个文件是  
用来给死东西打标记的

　　　　file\_contexts 给文件打标记

　　　　property\_contexts 给属性打标记

　　　　seapp\_contexts 给应用打标记

　　　　service\_contexts 给服务打标记

　　由于我们的服务 closesourcecode 对应的是一个可执行文件，所以我们只需要在 file\_contexts 文件修改即可，改动代码如下：  
+/system/bin/closesourcecode u:object\_r:closesourcecode\_exec:s0

　　以上，我们就完成了基础工作，接下来就会遇到一些具体的解析 log，调试功能的工作。

**2.5 如何抓取完整的 Selinux log**

　　编译完版本后（需要编译 boot 以及 vendor），抓取 kernel以及 logcat，可以发现会出现 selinux log，但是此时千万不要根据一个 log 修改一次代码，再编译版本，出现 log，再次解析出 allow 语句，再编译版本...无限循环，这是非常错误的做法。

　　Selinux 启用后有两种运行模式：

　　1.强制模式(Enforcing) 正常运行模式，遇到权限检查无法通过会拒绝操作并打出日志;

　　2.宽容模式(Permissive) 规则检查不变，遇到权限检查无法通过会打出日志，但不会拒绝;

　　实际发布时都会使用强制模式，但是运行在宽容模式可以方便调试。因为只有这样才能收集到所有的权限拒绝信息，保证一次性添加所有规则。

**2.6 关闭 selinux**

　　关闭 Selinux 的方法有两种，一种是在代码中修改，还有一种是使用命令进行修改，当服务是在开机时即需要调用，那么需要使用在代码中关闭 selinux 的方法，如果服务可以通过属性控制并不需要在开机时即调用，那么可以使用命令关闭 selinux。

**2.6.1 代码中关闭 selinux 的方法**

　　在 高 通 平 台 ， 只 需 要 修 改 device/qcom/ 对 应 的 平 台/BoardConfig.mk 文 件 添 加 如 下 代 码 （ 在 最 后一 个BOARD\_KERNEL\_CMDLINE 接着添加）：

BOARD\_KERNEL\_CMDLINE += androidboot.selinux=permissive

**2.6.2 使用 shell 命令关闭 selinux**

　　Selinux 提供了命令用来查看或者设置运行模式：

　　　　#getenforce ——用来查看运行模式，1 为强制模式，0 为宽容模式

　　　　#setenforce 1 ——设置运行模式为强制模式即 Enforcing

　　　　#setenforce 0 — — 设 置 运 行 模 式 为 宽 容 模 式 即Permissive

**2.7 解析 selinux log**

　　抓取完整 Selinux log 之后，我们就需要解析 log，log 示例如下：

[ 70.343292] type=1400 audit(257168.459:53): avc: denied { open } for pid=1340 comm="cp" path="/system/vendor/bin/btconfig" dev="mmcblk0p6" ino=1905 scontext=u:r:closesourcecode:s0 tcontext=u:object\_r:vendor\_file:s0 tclass=file permissive=1

其中关键的信息有这么几个：

avc denied { open } 拒绝了 open 操作  
scontext u:r:closesourcecode:s0 主体 SContext，其  
type 为 closesourcecode  
tcontext u:r:vendor\_file:s0 客体 SContext，其 type  
为 vendor\_file  
tclass file 操作类别为 file

如果要解决这个问题，只要将这几项信息组合成前面说的allow 规则然后添加策略文件中就可以了。规则如下：  
allow closesourcecode vendor\_file:file open;

任何selinux权限拒绝错误都会包含这几项信息，一般情况下，都可以通过添加 allow 规则来解决这种问题。

**2.8 Neverallow 问题解决方法**

　　有时，当我们添加完对应的.te 规则后，在编译时会报nerverallow 错误，导致不能编译通过，针对这种问题，其实就是我们自己添加的规则和系统源生的 nerverallow 语句冲突导致的。这里提供一下解决此类问题的思路：

　　1.首先我们需要分析编译 log，在 log 中一定会提示我们添加的 allow 语句与系统中的哪个文件中的哪一个语句有冲突，我们要  
根据冲突，找到自己添加的 allow 语句（当然如果您只添加了一个  
语句就不用查了）。  
　　2.假如说 allow closesourcecode vendor\_file:file open;这个 allow 语句与系统中的 nerverallow 冲突，那么您就需要查清楚主体 closesourcecode 到底是对哪个 vendor\_file 类型的文件操作，并查一下系统是否有定义 closesourcecode 允许对某种类型的文件进行 open 操作，如果有那么直接把 closesourcecode 操作的文件的 type 修改为 closesourcecode 允许的类型即可。当然思路也可以转换一下，可以到系统中查看是否有 allow 某种主体对vendor\_file 这种客体的文件有 open 权限，修改主体的 type 亦可。如果以上两种方式均没有解决，那么直接把closesourcecode 具体操作的文件的标记改为自己定义的就可以啦。  
3.某些同学可能自作聪明，认为直接把冲突的语句删掉或者修改一下不就可以了嘛，答案肯定是不行的，这样会影响后期的 cts认证。  
4.由于理论看起来比较复杂，所以后续第 3 章会有具体的实例分析。

**2.9 代码规范建议**

1.在没有产品线的前提下，所有的 selinux 修改目录为device/qcom/sepolicy/对应平台/

　　2.针对每一个主体的权限修改都要有一个主体名字对应的.te 文件，如果没有则 touch 一个

**3 Selinux工作中的问题总结**

**3.1 编译版本刷到手机后 selinux 没生效原因总结**

**3.1.1 编译问题**

　　一定不要偷懒只编译 boot，当修改 device/qcom/sepolicy/对应平台/目录下的文件时，是需要编译 boot 以及 vendor 的，且都要烧写。

**3.1.2 代码问题**

　　代码如下：  
　　+type closesourcecode, coredomain;  
　　+type closesourcecode\_exec, exec\_type, file\_type;  
　　如果代码按照如上模式写的话即不能生效，coredomain，并不代码它修饰的 type 就是 domain 类型，文件也一样，如果形容为可执行文件，

系统并不知道它是一个文件，道理如此，此问题很难查到，希望大家能引起重视。

**3.2 工具简介**

　　在某些情况下，可能会出现大量的 selinux log，那么此时就需要小工具进行辅助，即 audit2allow。

　　使用环境：Linux 操作系统

　　安装方法：sudo apt-get install policycoreutils

　　使用方法：audit2allow -i \*.log（此工具还有很多使用方法，具体其他用法请百度），这样即可根据 Log 自动生成 allow 语句。

**3.3 Nerverallow 实例分析**

**3.3.1 问题背景介绍**

　　项目中想要添加与 audio 相关的功能，添加完后，编译版本，在关闭 selinux 的情况下，功能正常，打开 selinux 即有问题，且  
在 log 中有 selinux 相关的 log，所以判断此问题为 selinux 问题，根据关闭 selinux 的 log 我们解析出两句 allow 语句：  
allow hal\_audio\_default default\_prop:property\_service set;  
allow audioserver default\_prop:property\_service set;

**3.3.2 编译问题**

　根据上文生成的.te 语句，我们在代码中创建对应的.te 文件（没有则创建），编译代码，发现代码编译不过，log 如下：

　　libsepol.report\_failure: neverallow on line 447 of system/sepolicy/public/domain.te (or line 8935 of policy.conf) violated by allow hal\_audio\_default default\_prop:property\_service { set };  
libsepol.report\_failure: neverallow on line 447 of system/sepolicy/public/domain.te (or line 8935 of policy.conf) violated by allow audioserver default\_prop:property\_service { set }; libsepol.check\_assertions: 2 neverallow failures occurred

　　分析可知：我 们 添 加 的 代 码 与system/sepolicy/public/domain.te 中的如下代码有冲突：  
　　　　Nerverallow { domain -init } default\_prop:property\_service set;

　　我们来对比一下：  
　　allow hal\_audio\_default default\_prop:property\_service set;  
　　allow audioserver default\_prop:property\_service set;  
　　结果很显然，是因为我们的主体对 default\_prop 这个 type 进行了 set 操作，所以我们需要绕过去。

**3.3.3 查清源头**

　　因为 default\_prop 是一个很宏观的概念，所以我们需要查清我们的主体到底是对什么属性进行了 set 操作，所以需要查代码，看代码中具体做了什么操作，代码如下：

　　property\_set("pmc.audio.spkoff\_w\_bths\_status", "true");

　　property\_set("pmc.audio.spkoff\_w\_bths\_status", "false");

　　由此便得知，我们的主体是对 pmc.audio.这个属性进行了 set操作，所以我们需要给 pmc.audio.重新打一个标记，给属性打标记需要在 property\_contexts 中添加，代码如下：

　　pmc.audio. u:object\_r:audio\_prop:s0

**3.3.4 修改对应的.te 语句**

　　属性是比较特殊的我们可以直接通过 set\_prop 来添加权限，代码如下:

　　set\_prop(audioserver, audio\_prop)；

　　set\_prop(hal\_audio\_default, audio\_prop)；

**3.3.5 验证功能**

　　编译代码，验证功能。

Ps：源码是最好的老师，感兴趣的同学可以下载一套android的代码，查看selinux代码的规则，进行比较学习；