# Android 15 SElinux
# 开发指南

**版本号: 1.5**
**发布日期: 2024.10.31**

# 版本历史

| 版本号 | 日期 | 制/修订人 | 内容描述 |
| --- | --- | --- | --- |
| 1.0 | 2020.11.11 | AWA0385 | 初始版本文档 |
| 1.1 | 2020.12.30 | AWA0385 | 更新 Android 11 v1.2 方案配置相关路径 |
| 1.2 | 2021.11.22 | AWA0385 | 适配 Android 12 |
| 1.3 | 2022.03.17 | AWA1829 | 适配 Android 13 |
| 1.4 | 2023.12.13 | AWA1829 | 适配 Android 14 |
| 1.5 | 2024.10.31 | AWA1829 | 适配 Android 15 |

# 目　录

# 1 前言

## 1.1 文档简介

本文档用于介绍 Android SDK SElinux 使用配置、规则制定。

## 1.2 目标读者

SDK 平台负责人、版本集成人员、SDK 开发人员。

## 1.3 适用范围

Allwinner Android 13 及以上平台。

## 1.4 名词解释

| key | value |
| --- | --- |
| vendor-name | softwinner |
| platform-name | 芯片平台名称 |
| device-name | 设备名称 |
| product-name | 产品名称 |

# 2 Android SElinux 介绍

## 2.1 Android 安全增强功能

Android 安全增强功能又称为 SEAndroid，一款 Android 安全解决方案。最初，SEAndroid 旨在 Android 中使用 SELinux，以便限制有缺陷或恶意应用造成的损失，并强制隔离应用。但是，SE-Android 所涵盖的范围已更改，可在 SELinux 的基础上包含更多内容。SEAndroid 目前是在 Android 上执行 SELinux 强制访问控制 (MAC) 和中间件强制访问控制 (MMAC) 的整体框架。

简单介绍与 SEAndroid 相关的概念：

- 安全增强型 Linux (SELinux) 是一种强制访问控制的实现。它的做法是以最小权限原则为基础，在 Linux 核心中使用 Linux 安全模块 (LSM)。它并不是一款 Linux 发布版，而是一组可以应用在类 UNIX 操作系统（如 Linux 和 BSD）的修改。
- 自主访问控制 (DAC) 是标准的 Linux 安全模型。在此模型中，访问权限基于用户的身份和对象所有权。
- 强制访问控制 (MAC) 限制了主体（进程）和对象（文件、套接字和设备等）的权限。

## 2.2 SEAndroid 策略

SEAndroid 通过向内核和用户空间添加 SELinux 支持来增强 Android 系统的安全性，Android 从 4.4 版开始引入 SEAndroid 的能力，在 Android 5.1 user 版本被设置成强制打开。SEAndroid 的安全上下文与 SELinux 基本一致。四个组成部分：用户、角色、类型和敏感度，即 u: object_r: system_data_file: s0 说明如下：

- 用户：在 SEAndroid 中第一列安全上下文是用户并且只有一个 u。
- 角色：在 SEAndroid 中第二列表示角色 ，分别为 r 和 object_r。
- 类型：对于第三列类型，SEAndroid 确定了上百项不同的策略类型，如设备类型、进程类型、文件系统类型、网络类型和 IPC 类型等等。
- 安全级别：第四列专为多级安全功能（扩展 MLS）而设计，MLS 是一种访问机制，可增加安全上下文和格式敏感度 [: 类别列表] [-敏感度 [: 类别列表]]，例如 s0 - s15: c0 - c1023，而在当前 Android 版本中不需要类别。敏感度和类别组合表示当前的安全级别，数字根据最低和最高级别的安全功能进行确定。此列参数被用于查看 MLS 限制，其中 “15” 和 “1023” 表示最大敏感度和类别。此参数范围可以在 Android.mk 中进行确定，Android 中默认配置为 “1” 和 “1024”，代表敏感度只定义了 s0，类别列表定义了 c0-c1023。

SEAndroid 策略源位于 SDK system/sepolicy 目录。该策略包括用于生成 SELinux 核心策略文件的源文件：file_contexts 配置、property_contexts 配置、seapp_contexts 配置和 mac_permissions.xml 配置。

- file_contexts 配置用于在构建（例如，系统分区）和运行时（例如，设备节点、服务套接字文件和由 init.rc 创建的/数据目录等）标记文件。
- property_contexts 被用于指定 Android 属性的安全上下文，供查看权限。
- seapp_contexts 配置被用于标记应用进程和应用程序包目录。
- mac_permissions.xml 配置是中间件 MAC 策略。

与设备相关的策略文件可通过BOARD_SEPOLICY_DIRS，SYSTEM_EXT_PUBLIC_SEPOLICY_DIRS和SYSTEM_EXT_PRIVATE_SEPOLICY_DIRS指定，位于 device/{vendor-name}/common/sepolicy目录。
与产品相关的策略文件可通过PRODUCT_PUBLIC_SEPOLICY_DIRS，PRODUCT_PRIVATE_SEPOLICY_DIRS指定，位于device/{vendor-name}/{platform-name}/common/sepolicy目录。

# 3 SElinux 使用和配置

本节主要介绍如何开启 SElinux 功能，并介绍如何配置策略。

## 3.1 Android SElinux 开关

下图详细列出了 Android 启动过程 SElinux 加载逻辑。

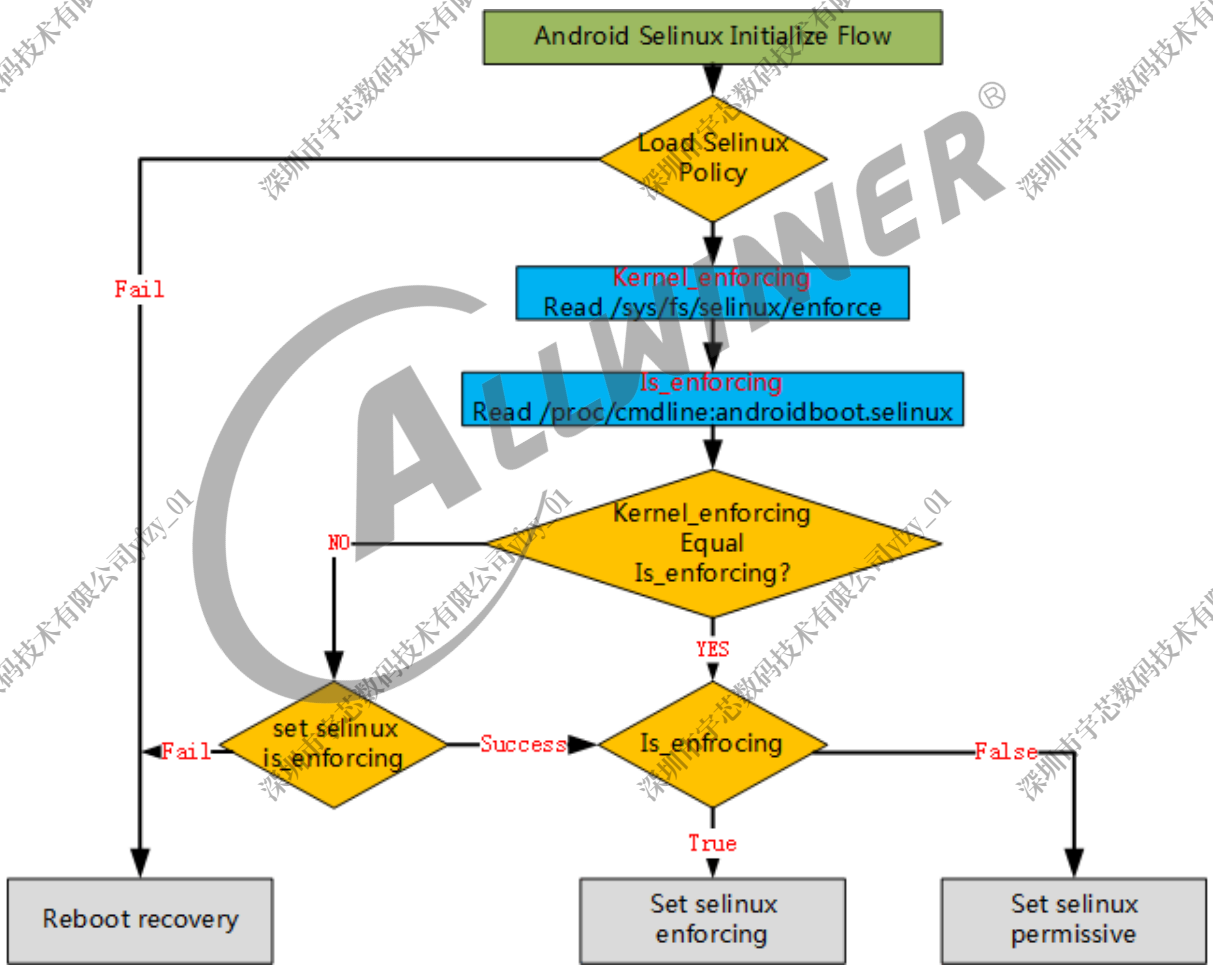


图 3-1: Android SElinux Initialize Flow

在 Android 上将 SELinux 关闭并保证系统正常功能较为困难，因此稍微调整 Android SElinux 开启策略如下：

- 在产品目录中的 BoardConfig.mk 文件中配置内核 boot command，如下：

```
//android/device/{vendor-name}/{platform-name}/{device-name}/BoardConfig.mk
BOARD_KERNEL_CMDLINE := selinux=1 androidboot.selinux=enforcing
```

> selinux=0，内核关闭 selinux；selinux=1，内核打开 selinux；androidboot.selinux = permissive，android 将 selinux 模式设为 permissive androidboot.selinux = enforcing，android 将 selinux 模式设为 enforcing

- SDK 默认所有版本 SElinux = 1 都是打开，保证系统可以正常启动。
- 在使用过程中，允许串口输入 setenforce 0 临时将 SElinux 置为 Permissive 模式。
- 在使用过程中，可以在串口输入 getenforce 获取当前 SElinux 的工作模式，1 为 enforcing，0 为 permissive。

小结：如果需要编译出 selinux 的工作模式为 permissive 的固件，只需要向 BoardConfig.mk 中的 BOARD_KERNEL_CMDLINE 修改为

```
androidboot.selinux=permissive
```

# 3.2 SElinux 规则修改

在上一章节中介绍，Android 中有 2 个地方定义 SElinux 策略文件。system/sepolicy 存放整体原生规则，device/{vendor-name}/common/sepolicy 存放和设备关联的策略。SDK 中已经配置有一套合理的 SElinux policy。在 SDK 开发过程中，可以根据需要，编辑规则文件，增加自定义规则。

## 3.2.1 Step 1. 查看违反规则的方法

> dmesg | grep avc 或者在串口 log 中查找关键字 denied，即可看到有违反规则的行为。

## 3.2.2 Step 2. 生成规则

- 自动生成规则：使用 linux 工具 audit2allow 可以将违反规则的 avc 记录生成放行规则 (适合于 dmesg 所有输出，初期的开发阶段)。
- 手动生成规则: 比如违反 avc 记录如下：

> type=1400 audit(1386760471.880:7): avc: denied { entry-point } for pid=1227 comm= “init” path= “/sbin/healthd” dev= “rootfs” ino=4396 scontext=u:r:healthd:s0 tcontext=u:object_r:rootfs:s0 tclass=file

关键字对应的属性名：

> scontext_name: healthd tcontext_name: rootfs tclass_type: file rules: entrypoint

根据放行公式：

> allow scontext_name tcontext_name:tclass_type rules;

生成 rule 如下：

> allow healthd rootfs:file entrypoint;

### 3.2.3 Step 3. 添加规则

- 编辑 device/{vendor-name}/common/sepolicy 目录下面 TE 格式文件，添加 Step 2 中生成的 rule 到以 scontext_name 命名的 TE 文件。
- 重新编译 Android 就可以生成新的 SElinux policy 固件。
- 如在编译过程出现类似如下错误，说明添加的规则和 system/sepolicy 设置的规则冲突。

> neverallow on line 269 of system/sepolicy/domain.te (or line 5193 of policy.conf) violated by allow platform_app unlabeled:dir { create };

### 3.2.4 Step 4. 解决规则冲突

system/sepolicy 定义通用规则，如果出现 device 规则与其冲突的问题，即 system/sepolicy 定义 neverallow 规则，但是 device 规则又要允许访问。这种情况初学者本能的认为需要在 system/sepolicy 中的 neverallow 中增加例外规则，如下图所示。实际上这种做法不妥，在添加 selinux 规则的时候尽量不要违背 system/sepolicy 的 neverallow，system/sepolicy 的 neverallow 都是 google 工程师经过深思熟虑设置的规则检验关卡，违背了其规则，证明添加的 device 规则不合理。这种情况下一般是没有为 scontext 或者 tcontext 定制化标签导致的，请检查 avc 报错中是否有 default 或者 unlabel 字样。

```
neverallow { domain -unconfineddomain -recovery -untrusted_app} unlabeled:dir_file_class_set create;
```

```
neverallow { domain -unconfineddomain -recovery -untrusted_app -platform_app} unlabeled:dir_file_class_set create;
```

图 3-2: 解决规则冲突

# 3.3 添加 sepolicy 示例

下面举例说明如何为 Android 的二进制程序 qw 制定 sepolicy 规则，相关规则见

> device/{vendor-name}/common/sepolicy/qw.te。

1. type qw domain,domain_deprecated；为 qw 进程定义专属 type qw，关联属性 domain。
2. type qw_exec,exec_type,file_type；为 qw 的二进制文件定义 type qw_exec。
3. init_daemon_domain(qw)；进行进程的 domain transition，当 init 进程 fork 并执行 qw_exec 标签对应的可执行文件 qw 时，进程的 domain 转为 qw。
4. 在对 qw 进行 sepolicy 的策略配置时，发现 qw 会创建文件夹/dev/com.koushikdutta.superuser.daemon 和 sock:/dev/com.koushikdutta.superuser.daemon/server 用于通信。因此需要对这两个对象打上自定义标签，防止发生 neverallow 规则冲突。
   - type qw_daemon_device,dev_type 在 device.te 中定义 qw_daemon_device 用于给文件夹打标签。
   - type qw_socket,file_type,mlstrustedobject; 在 qw.te 中定义 qw_sock 用于给 server 打标签。
   - file_type_auto_trans(qw,device,qw_daemon_device)；当 qw 在/dev（context：device）目录下创建文件夹 com.koushikdutta.superuser.daemon 时，给该文件夹打上 qw_daemon_device 标签。
   - file_type_atuo_trans(qw,qw_daemon_device,qw_socket)；当 qw 在/dev/com.koushikdutta.superuser.daemon/(context:qw_daemon_device) 下创建文件 server 时，给 server 打上 qw_socket 标签。
5. 接下来编译出固件，根据 dmesg 中的一些 avc 报错添加一些 allow 规则，反复编译几次即可。

按照以上规则不出意外的话就可将 qw 的相关 sepolicy 配置起来。但事与愿违，在添加策略进行验证时，发现一些规则，如：allow qw qw_socket:sock_file create_file_perms; 这些规则

- 没有编译出错
- 没有 neverallow 冲突
- 成功编译进 policy.conf

但是在验证时，仍然报出相关的 avc denied。经过一番查找，发现该规则需要相应的 mls 权限才可以被授予对应权限。mls 中：

```
mlsconstrain{sock_file} {write setattr...}(t2==app_data_file or l1 eq l2 or t1 == mlstrustedsubject or t2 ==
    mlstrustedobject)
```

简单起见，在定义 qw 时，加上 type qw,domain,domain_deprecated,mlstrustedsubject. 注：mlstrustedsubject 和 mlstrustedobject 在 mls 权限控制中拥有至高无上的权利。

# 3.4 SElinux 中的特殊符号

1. te 文件每行末尾以分号 “;” 结束
2. contexts 文件每行末尾没有分号 “;”
3. 通配符

   *：对应集合内的所有内容。

   -：表示集合 A 内去除某项内容 b

   ~：表示 ~ 后元素所在集合去除 ~ 后元素剩余的子集

**著作权声明**



本文档及内容受著作权法保护，其著作权由珠海全志科技股份有限公司（“全志”）拥有并保留一切权利。

本文档是全志的原创作品和版权财产，未经全志书面许可，任何单位和个人不得擅自摘抄、复制、修改、发表或传播本文档内容的部分或全部，且不得以任何形式传播。

**商标声明**





（不完全列举）均为珠海全志科技股份有限公司的商标或者注册商标。在本文档描述的产品中出现的其它商标，产品名称，和服务名称，均由其各自所有人拥有。

**免责声明**

您购买的产品、服务或特性应受您与珠海全志科技股份有限公司（“全志”）之间签署的商业合同和条款的约束。本文档中描述的全部或部分产品、服务或特性可能不在您所购买或使用的范围内。使用前请认真阅读合同条款和相关说明，并严格遵循本文档的使用说明。您将自行承担任何不当使用行为（包括但不限于如超压，超频，超温使用）造成的不利后果，全志概不负责。

本文档作为使用指导仅供参考。由于产品版本升级或其他原因，本文档内容有可能修改，如有变更，恕不另行通知。全志尽全力在本文档中提供准确的信息，但并不确保内容完全没有错误，因使用本文档而发生损害（包括但不限于间接的、偶然的、特殊的损失）或发生侵犯第三方权利事件，全志概不负责。本文档中的所有陈述、信息和建议并不构成任何明示或暗示的保证或承诺。

本文档未以明示或暗示或其他方式授予全志的任何专利或知识产权。在您实施方案或使用产品的过程中，可能需要获得第三方的权利许可。请您自行向第三方权利人获取相关的许可。全志不承担也不代为支付任何关于获取第三方许可的许可费或版税（专利税）。全志不对您所使用的第三方许可技术做出任何保证、赔偿或承担其他义务。