

Unisoc Confidential For hiar

Android 10.0 SELinux 客制化指导手册

文档版本

V1.0

发布日期

2020-10-14

版权所有 © 紫光展锐（上海）科技有限公司。保留一切权利。

本文件所含数据和信息都属于紫光展锐（上海）科技有限公司（以下简称紫光展锐）所有的机密信息，紫光展锐保留所有相关权利。本文件仅为信息参考之目的提供，不包含任何明示或默示的知识产权许可，也不表示有任何明示或默示的保证，包括但不限于满足任何特殊目的、不侵权或性能。当您接受这份文件时，即表示您同意本文件中内容和信息属于紫光展锐机密信息，且同意在未获得紫光展锐书面同意前，不使用或复制本文件的整体或部分，也不向任何其他方披露本文件内容。紫光展锐有权在未经事先通知的情况下，在任何时候对本文件做任何修改。紫光展锐对本文件所含数据和信息不做任何保证，在任何情况下，紫光展锐均不负任何与本文件相关的直接或间接的、任何伤害或损失。

请参照交付物中说明文档对紫光展锐交付物进行使用，任何人对紫光展锐交付物的修改、定制化或违反说明文档的指引对紫光展锐交付物进行使用造成的任何损失由其自行承担。紫光展锐交付物中的性能指标、测试结果和参数等，均为在紫光展锐内部研发和测试系统中获得的，仅供参考，若任何人需要对交付物进行商用或量产，需要结合自身的软硬件测试环境进行全面的测试和调试。

Unisoc Confidential For hiar

紫光展锐（上海）科技有限公司



前言

概述

本文档详细地描述了 SELinux 模块基础知识，提供了 Android 10.0 SELinux 常见问题的解决方法。

读者对象


本文档主要适用于想了解 Android 10.0 SELinux 相关问题的客户和研发人员。

缩略语

缩略语	英文全名	中文解释
SELinux	Security-Enhanced Linux	安全增强型 Linux
AVC	Access Vector Cache	访问向量缓存
DAC	Discretionary Access Control	自主访问控制
MAC	Mandatory Access Control	强制访问控制

符号约定

在本文中可能出现下列标志，它所代表的含义如下。

符号	说明
 说明	用于突出重要/关键信息、补充信息和小窍门等。 “说明”不是安全警示信息，不涉及人身、设备及环境伤害。

变更信息

文档版本	发布日期	修改说明
V1.0	2020-10-14	第一次正式发布。

关键字

SELinux、AVC、SELinux 常见问题。

Unisoc Confidential For hiar

目 录

1 SELinux 基础知识	1
1.1 访问控制	1
1.2 SELinux 模式	1
1.2.1 SELinux 模式查看	1
1.2.2 SELinux 模式切换	2
1.3 AVC 基础语法	2
1.4 代码修改后验证	3
1.5 SELinux 策略	4
2 SELinux 常见问题	5
2.1 SELinux 权限问题判断	5
2.2 添加 SELinux 权限	5
2.3 Neverallow 问题	5
2.4 新增文件配置安全上下文	6
2.4.1 新增设备节点文件	6
2.4.2 新增 proc 文件	7
2.5 新增属性配置安全上下文	7
2.5.1 Android 属性介绍	7
2.5.2 新增属性	7
2.6 手机中查看安全上下文的方法	8
2.7 mls 规则	8
2.8 SELinux 安全上下文的恢复	9
2.9 AVC backtrace	9
2.9.1 功能介绍	9
2.9.2 使用方法	9

1 SELinux 基础知识

1.1 访问控制

Linux 内核资源访问控制分为 DAC（Discretionary Access Control，自主访问控制）和 MAC（Mandatory Access Control，强制访问控制）两类。

DAC

基于“用户-用户组-其他/读-写-执行”的权限检查，进程理论上所拥有的权限与执行它的用户的权限相同，该管理过于宽松，如果获得 root 权限，可以在 Linux 系统内做任何事情。

MAC

SELinux（Security-Enhanced Linux，安全增强型 Linux）是 MAC 机制的一种实现，基于安全上下文和安全策略的安全机制，用于补充 DAC 检查。访问系统资源时，会先进行 DAC 检查，DAC 检查通过，才能进行 MAC 检查，如果 MAC 检查通过，才能获得资源访问权限。

1.2 SELinux 模式

SELinux 分为 Disable、Permissive 和 Enforcing 三种模式。

Disable

关闭模式，不进行 SELinux 权限检查。

Permissive

宽容模式，代表 SELinux 运作中，违反 SELinux 规则，只会有警告讯息（`avc denied`），而不会实际限制 domain/type 的存取。这种模式可以作为 SELinux 的 debug 之用（查看导致无法访问的原因）。

Enforcing

强制模式，代表 SELinux 运作中，违反 SELinux 规则的行为将被阻止并记录到日志中。

1.2.1 SELinux 模式查看

- 通过命令行查看 SELinux 模式，命令如下：

```
adb shell getenforce
```

- 通过 AVC（Access Vector Cache，访问向量缓存）log 来查看 SELinux 模式，AVC log 结尾会有 permissive=1/0 的标示：
 - permissive=1，说明是 Permissive 模式。

- permissive=0，说明是 Enforcing 模式。

1.2.2 SELinux 模式切换

切换 SELinux 模式有命令行、修改 dts bootargs 参数和修改 init 代码三种方式。

命令行

手机开机后，开启手机 USB debug 选项，用 USB 线将手机与电脑连接，在电脑终端输入如下命令：

```
adb root;  
adb shell setenforce 0
```

说明

该方式仅适用于 userdebug 版本，系统重启会失效。

修改 dts bootargs 参数

board 对应的 dts 文件，在 bootargs 参数里增加 androidboot.selinux=permissive 字段，如在 kernel/common/arch/arm64/boot/dts/sprd/xxx.dts 文件中，如下红色字体所示：

```
bootargs = "earlycon=sprd_serial,0x508d0000,115200n8 console=ttySE0,115200n8 loglevel=1 init=/init root=/dev/ram0 rw  
androidboot.hardware=sl8541e_1h10 vmalloc=360M androidboot.dtbo_idx=0 printk.devkmsg=on androidboot.selinux=permissive  
androidboot.boot_devices=soc/soc:ap-ahb/20600000.sdio"
```

修改完成后重新编译 boot.img 代码，再烧写至手机。

说明

该方式仅适用于 userdebug 版本，系统重启仍然有效。

修改 init 代码

修改 system/core/init/SELinux.cpp 文件里的 IsEnforcing() 函数，将该函数直接返回 false 值，如下红色字体所示：

```
bool IsEnforcing() {  
    return false;  
}
```

修改完成后重新编译代码，再将生成的 super.img 烧写至手机。

说明

该方式适用于 user 和 userdebug 版本，系统重启仍然有效。

1.3 AVC 基础语法

AVC log 举例如下：

```
22:33:11.062 <36>[ 8.755284] c6 type=1400 audit(1325428390.909:11): avc: denied { read } for pid=1834 comm="gnss_download"  
name="mmcblk0p17" dev="tmpfs" ino=10268 scontext=u:r:gnss_download:s0 tcontext=u:object_r:block_device:s0 tclass=blk_file permissive=0
```


AVC log 说明

1. `avc: denied`: 表示当前操作被拒绝。
2. `{ read }`: 表示被拒绝的操作, `{ }`中含有实际尝试的操作。
3. `for pid=1834`: 当前发生 `avc: denied` 的进程的进程 ID。
4. `comm="gnss_download"`: 当前发生 `avc: denied` 的进程的进程名, 即主体进程名称。
5. `name="mmcblk0p17"`: 操作尝试的目标文件或目录的路径, 即客体资源名称。
6. `dev="tmpfs"`: 含有这个文件系统的设备节点, 客体资源在该文件系统中。
7. `ino=10268`: 目标文件或目录的节点号。
8. `scontext=u:r:gnss_download:s0`: 主体进程的安全上下文。
9. `tcontext=u:object_r:block_device:s0`: 客体资源的安全上下文。
10. `tclass=blk_file`: 访问资源所属类别。
11. `permissive=0`: 当前是 Enforcing 模式, `permissive=1` 时为 Permissive 模式。

说明

此 AVC log 说明 `gnss_download` (进程) 缺少对标签为 `block_device`、类型为 `blk_file` 和名称为 `mmcblk0p17` 文件的 `read` 权限, 权限的添加可参考 2.2 添加 SELinux 权限。

1.4 代码修改后验证

如果只是修改 `selinux` 相关文件, 可以参考如下步骤进行快速验证:

步骤 1 `lunch` 工程后, 输入如下命令单独编译 `sepolicy` 模块:

```
cd system/sepolicy
mma
```

步骤 2 将编译产物 `out/target/product/xxx/system/etc/selinux` 和 `out/target/product/xxx/vendor/etc/selinux` 导入手机对应目录, 重启手机验证修改是否生效。具体执行指令如下:

```
cd out/target/product/xxx
adb root
adb remount
adb push system/etc/selinux /system/etc/
adb push vendor/etc/selinux /vendor/etc/
```

说明

在 `push` 之前需保证手机已解锁。

----结束

1.5 SELinux 策略

SELinux 策略类型

- **Public** 公共策略（基础部分）：导出的策略是非平台策略，开发人员可以在该策略上编写附加策略。类型和属性被版本化的策略包含在已交付的非平台策略中，非平台策略将与平台策略组合在一起。
- **Private** 私有策略（基础部分）：平台功能需要的策略，不会导出给其他供应商策略开发人员，因此可以假定不存在。
- **Vendor** 供应商通用组件策略（外扩组件部分）：供应商功能需要“仅供应商策略”。该政策可以参考公共政策，但不能参考私人政策。此策略适用于从核心/非供应商树生成并放置到供应商分区的组件。

谷歌原生策略

- `/system/sepolicy`: 谷歌原生策略，不要在此路径下进行修改
- `/system/sepolicy/private`: 谷歌原生 system 分区 private sepolicy
- `/system/sepolicy/public`: 谷歌原生 system 分区 public sepolicy
- `/system/sepolicy/vendor`: 谷歌原生 vendor 分区 sepolicy
- `/system/sepolicy/prebuilts`: 版本兼容 sepolicy

展锐补充策略

- `device/sprd/xxx(具体项目)/common`: 展锐一般在此路径配置修改 SELinux 策略
- `device/sprd/xxx(具体项目)/common/plat_sepolicy/private`: system 分区 private sepolicy
- `device/sprd/xxx(具体项目)/common/plat_sepolicy/public`: system 分区 public sepolicy
- `device/sprd/xxx(具体项目)/common/sepolicy`: vendor 分区 sepolicy

2 SELinux 常见问题

2.1 SELinux 权限问题判断

当在 Enforce 模式下出现代码执行异常时，可同步在 Permissive 模式下验证该异常是否依旧存在：

- 如果异常消失，则说明该异常与 SELinux 权限问题有关，可参考 2.2 添加 SELinux 权限进行相应权限的添加。
- 如果异常仍然存在，则说明该异常与 SELinux 权限问题无关，需排查其它原因。

2.2 添加 SELinux 权限

由于缺少 SELinux 权限导致如下 “avc: denied”，需要根据 AVC log 信息添加相应权限。

```
avc: denied { read write } for pid=3483 comm="Binder_6" name="rtc0" dev="tmpfs" ino=4451 scontext=u.r:system_server:s0  
tcontext=u.object_r:refnotify_device:s0 tclass=chr_file permissive=0
```

Avc log 含义

system_server（进程）缺少对标签为 refnotify_device、类型为 chr_file 和名称为 rtc0 文件的读写权限。

添加权限参考公式

打开 device/sprd/xxx(具体项目)/common/sepolicy/scontext.te 文件，添加如下代码：

```
allow scontext tcontext:tclass permission;
```

添加权限

打开 device/sprd/xxx(具体项目)/common/sepolicy/system_server.te 文件，添加如下代码：

```
allow system_server refnotify_device:chr_file{ read write };
```

📖 说明

可以设置 SELinux 为 permissive，使所有相关 avc 暴露，添加权限。

2.3 Neverallow 问题

例如，在添加如下代码进行编译时，编译失败并报 neverallow 错。

```
allow system_app sysfs:file {write};
```

原因是 Google 不允许应用进程写 sysfs 类型的文件，其代码如下：

```
neverallow { appdomain -bluetooth -nfc }sysfs:dir_file_class_set write;
```

这种情况下需要自定义 type，并更改客体的安全上下文。

根据操作文件类型选择放入 vendor 或 system 文件夹下，下面以 vendor 举例，步骤如下：

步骤 1 参考原来的 sysfs 类型定义一种新 type，type 名字可以自定义。

打开 device/sprd/xxx(具体项目)/common/sepolicy/file.te 文件，添加如下代码：

```
type sysfs_xxx, fs_type,sysfs_type;
```

步骤 2 使用新定义的 type 为目标文件配置安全上下文。

打开 device/sprd/xxx(具体项目)/common/sepolicy/file_context 文件，添加如下代码：

```
/xxx/xxx (目标文件路径) u:object_r:sysfs_xxx:s0
```

步骤 3 重新赋予主体进程访问新类型文件的权限。

打开 device/sprd/xxx(具体项目)/common/sepolicy/system_app.te 文件，添加如下代码：

```
allow system_app sysfs_xxx:file { write };
```

----结束

2.4 新增文件配置安全上下文

新增文件配置主要包含虚拟文件配置写入 genfs_contexts 文件和磁盘文件配置写入 file_contexts 文件。

2.4.1 新增设备节点文件

以新增的设备节点/dev/autotst为例，配置安全上下文步骤如下：

步骤 1 确认新增的设备节点类型是否在 system/sepolicy/public/device.te 文件中定义。

a 已有定义，跳至步骤 2。

b 没有定义，需要自定义类型，打开 device/sprd/xxx(具体项目)/common/sepolicy/device.te 文件，添加如下代码：

```
type autotest_device, dev_type;
```

步骤 2 file_contexts 中配置文件安全上下文。

打开 device/sprd/xxx(具体项目)/common/sepolicy/file_contexts 文件，添加如下代码：

```
/dev/autotst u:object_r:autotest_device:s0
```

步骤 3 te 文件中赋予对应进程访问该类型文件的权利。

打开 device/sprd/xxx(具体项目)/common/sepolicy/autotest.te 文件，添加如下代码：

```
allow autotest autotest_device:dir { search read };
```

----结束

📖 说明

这节主要是以 vendor 举例说明，system 的也可参考。

2.4.2 新增 proc 文件

proc 文件与特殊文件系统的安全上下文标记相关，比较常见的是 Bluetooth 会创建一个 proc 文件系统的文件。新增 proc 文件配置安全上下文步骤如下：

步骤 1 确认新增 proc 文件的类型是否为已有类型。

- a 是已有类型，跳至步骤 2。
- b 不是已有类型，则需重定义标签，打开 device/sprd/xxx(具体项目)/common/sepolicy/file.te 文件，添加如下代码：

```
type proc_bluetooth_writable, fs_type, proc_type;
```

步骤 2 genfs_contexts 中配置 proc 文件安全上下文。

打开 device/sprd/xxx(具体项目)/common/sepolicy/genfs_contexts 文件，添加如下代码：

```
genfscon proc /bluetooth/sleep/btwrite u:object_r:proc_bluetooth_writable:s0
```

步骤 3 te 文件中赋予对应进程访问该类型文件的权利。

打开 device/sprd/xxx(具体项目)/common/sepolicy/bluetooth.te 文件，添加如下代码：

```
allow bluetooth proc_bluetooth_writable:file write;
```

---结束

说明

如需查看文件安全上下文配置情况，可参考 2.6 手机中查看安全上下文的方法。

2.5 新增属性配置安全上下文

Android 属性在系统中存放在一块共享内存中，每个进程都能读，但是只有 init 进程能够写，请求 init 进程帮助写属性值是需要 SELinux 权限的。

2.5.1 Android 属性介绍

Android 属性名称的前缀必须用 system/core/init/property_service.c 中定义的：

- vendor 属性名前缀
- persist.vendor.**
- vendor.**
- ro.vendor.**

属性名不包含 vendor，则为系统属性。

2.5.2 新增属性

根据属性的命名规范，选择定义在 system 分区还是 vendor 分区，以 vendor 为例，新增属性步骤如下：

步骤 1 在 property.te 中添加属性的 type 定义。

打开 device/sprd/xxx(具体项目)/common/sepolicy/property.te 文件，添加如下代码：

```
type vendor_xxx_prop, property_type;
```

步骤 2 property_contexts 中对属性设定安全上下文。

打开 device/sprd/xxx(具体项目)/common/sepolicy/property_contexts 文件，添加如下代码：

```
xxx.xxx.      u:object_r:vendor_xxx_prop:s0
```

说明

xxx.xxx 后的“.”表示匹配以 xxx.xxx 开头的属性。

步骤 3 te 文件中赋予对应进程访问该类型属性的权利。

打开 device/sprd/xxx(具体项目)/common/sepolicy/xxx.te 文件，添加如下代码：

```
set_prop(xxx, vendor_xxx_prop)
```

---结束

说明

如需查看文件安全上下文配置情况，可参考 2.6 手机中查看安全上下文的方法。

2.6 手机中查看安全上下文的方法

- 在手机中，可通过如下指令查看文件安全上下文：

```
ls -lZ
```

- 在手机中，可通过如下指令查看属性安全上下文：

```
getprop -Z
```

- 在手机中，可通过如下指令查看进程安全上下文：

```
ps -Z
```

2.7 mls 规则

如果 te 文件已经添加 SELinux 权限，但没有生效，查看 AVC log 信息出现 “s0:c512,c768” 字眼，则可判断是由于 mls 规则导致。说明主体和客体安全级别不同，可通过使主体进程关联 mlstrustedsubject 或客体关联 mlstrustedobject 来解决。

举例

已经在 platform_app.te 中添加了 SELinux 权限，但 log 中依然有如下报错：

```
avc: denied { connectto } for pid=2002 comm="slogHandlerThre" path=00736C6F676D6F64656D scontext=u:r:platform_app:s0:c512,c768
tcontext=u:r:slogmodem:s0 tclass=unix_stream_socket permissive=0
```

这是因为 Google 在文件 system/sepolicy/private/mls 中进行了安全级的限制，限制代码如下：

```
#Stream connect: Client must be equivalent to server unless one of them is trusted.

mlsconstrain unix_stream_socket { connectto }
(l1 eq l2 or t1 == mlstrustedsubject or t2 == mlstrustedsubject);
```

这种情况需要主体进程或者客体进程中的一个 mlstrustedsubject，这里 platform_app 最好不要修改，所以要修改客体 slogmodem。具体修改方法如下：

打开 device/sprd/xxx(具体项目)/common/plat_sepolicy/system/public/slogmodem.te 文件，添加如下代码：

```
type slogmodem, domain, mltrustedsubject;
```

2.8 SELinux 安全上下文的恢复

SELinux 安全上下文的恢复可以使用 `restorecon` 命令来实现，格式如下：

```
[root@localhost ~] # restorecon [选项] 文件或目录
```

选项：

- `-R`：递归，当前目录和目录下所有的子文件同时恢复。
- `-V`：恢复过程详细信息显示到屏幕上。

例如，想要根据 `file_contexts` 中的安全上下文配置，可使用如下指令恢复标签：

```
restorecon /mnt/vendor
```

- 如果在手机上使用 `restorecon` 命令可以成功恢复正确标签，则可在所需函数或 `rc` 文件中添加 `restorecon` 命令恢复标签，以此完善函数或 `rc` 的功能。

```
restorecon -R 文件路径
```

- 如果在 `rc` 文件中添加 `restorecon` 命令，不能恢复成功，则需要抓 `log` 查看是否缺少相关权限，继而添加相应权限。

2.9 AVC backtrace

2.9.1 功能介绍

在解决“`avc: denied`”权限问题时，可使用 `backtrace` 打印 AVC log 对应函数调用栈信息，定位对应的代码。AVC backtrace 主要功能如下：

- 抓取 log，堆栈会打印到 kernel log，根据堆栈信息进行定位。
- 在 AVC log 的附近会有类似如下 log，根据 `pid` 和 `comm` 信息找到对应进程的 log，根据该 log 可以知道要访问的客体的 `uid` 和 `gid`（log 中的 `uid` 和 `gid` 是要访问的客体的）：
 - log 样例：check supplement group pid:xxx comm:xxx, uid:xxx, gid:xxx
 - 用户、用户组和 `uid`、`gid` 的对应关系参见文件：
system/core/libcutils/include/cutils/android_filesystem_config.h

2.9.2 使用方法

AVC backtrace 功能为 debug feature，默认为关闭。可以设置“`persist.vendor.avc.backtrace.enabled`”属性为 1 来打开。该功能默认打印所有进程的 AVC backtrace，具体打印哪些进程的，用户可以自定义。

开启 AVC backtrace

打开 AVC backtrace 命令如下：

```
adb root
adb shell setprop persist.vendor.avc.backtrace.enabled 1
```

一般情况下执行以上命令，AVC backtrace 功能就打开了。但若属性触发打开时机较晚，导致关心的 AVC log 堆栈没有打印出来，那么需要将触发时机提前到 early-init 阶段，方法如下：

- 直接修改手机中的文件：

将手机中/vendor/etc/init/avc_backtrace.rc 文件导出来，把该文件中代码 “on property:persist.vendor.avc.backtrace.enabled=1” 替换为 “on early-init”，再导回到手机中（该操作需要手机处于解锁状态）。

- 修改源码：

打开 vendor/sprd/generic/misc/system/core/init/avc_backtrace.rc 文件，把代码 “on property:persist.vendor.avc.backtrace.enabled=1” 替换为 “on early-init”，编译代码，再将编译产物烧写到手机。

指定进程的 AVC backtrace

当打开 AVC backtrace 的开关之后，默认所有进程的 AVC backtrace 都会被打印，如果只想打印自己关心进程的，可以采用命令和修改文件的方法。

- 命令

如打印 ylog/slogmodem/rild 三个进程的 AVC backtrace 命令如下：

```
adb shell echo ylog,slogmodem,rild > /sys/fs/SELinux/backtrace_filter
```

- 修改文件

- 直接修改手机中的文件：

将手机中/vendor/etc/init/avc_backtrace.rc 文件导出来，把该文件中代码 “write /sys/fs/selinux/backtrace_filter "all"” 中的 “all” 改为自己关心的进程的 comm 值，再导回到手机中（该操作需要手机是解锁的）。

- 修改源码：

打开/vendor/sprd/generic/misc/system/core/init/avc_backtrace.rc 文件，把代码 “write /sys/fs/selinux/backtrace_filter "all"” 中的 “all” 改为自己关心的进程的 comm 值，编译代码，再将编译产物烧写到手机。

说明

- 命令方法：手机重启会失效。
- 修改文件：手机重启不会失效。
- 最多可以输入 5 个进程的 comm 值，该值可以通过 cat /proc/pid/comm 来获取，各个 comm 名之间以逗号分割开。