Android 10 系统属性兼容性介绍

文档版本	V1.0
发布日期	2020-4-17

紫光展锐科技有限公司

紫光展锐科技有限公司

前 言

概述

Android8.x 上 system 分区和 vendor 分区是共享系统属性的，即 system 分区可以随意访问 vendor 分区的属性，vendor 分区也可以随意访问 system 分区的属性；但随着 Android Treble 架构的推出，这种无限制的访问形式会造成在不同 Android 版本上不能保证属性的可访问性，即当单独升级某个分区后属性可能会发生改变导致其他分区访问不到，这就是所谓的兼容性问题；且这种无限制的访问形式会给系统带来隐藏的风险，当 vendor 分区可随意修改 system 分区的属性时也就意味着 system 分区的不稳定性。为了解决这一问题，从 Android 9 版本开始限制了不必要的 system 和 vendor 间系统属性共享，并提供了在保持一致性的前提下共享属性的方法。

本文的主要讲解内容是：Android 10 系统属性要求及常见问题。旨在通过本文的介绍可以使读者了解 Android 10 版本的属性相关知识并可以自行添加属性和解决属性违规问题。

读者对象

本文档主要适用于 Android 软件开发技术人员。该人员须具备以下经验和技能：

- 熟悉 Java/C++编程。
- 了解 Android SELinux。

变更信息

文档版本	发布日期	作者	修改说明
V1.0	2020-04-17	方放杰	第一次正式发布。

关键字

Android 属性、 Android prop、属性、prop、属性命名、白名单属性。

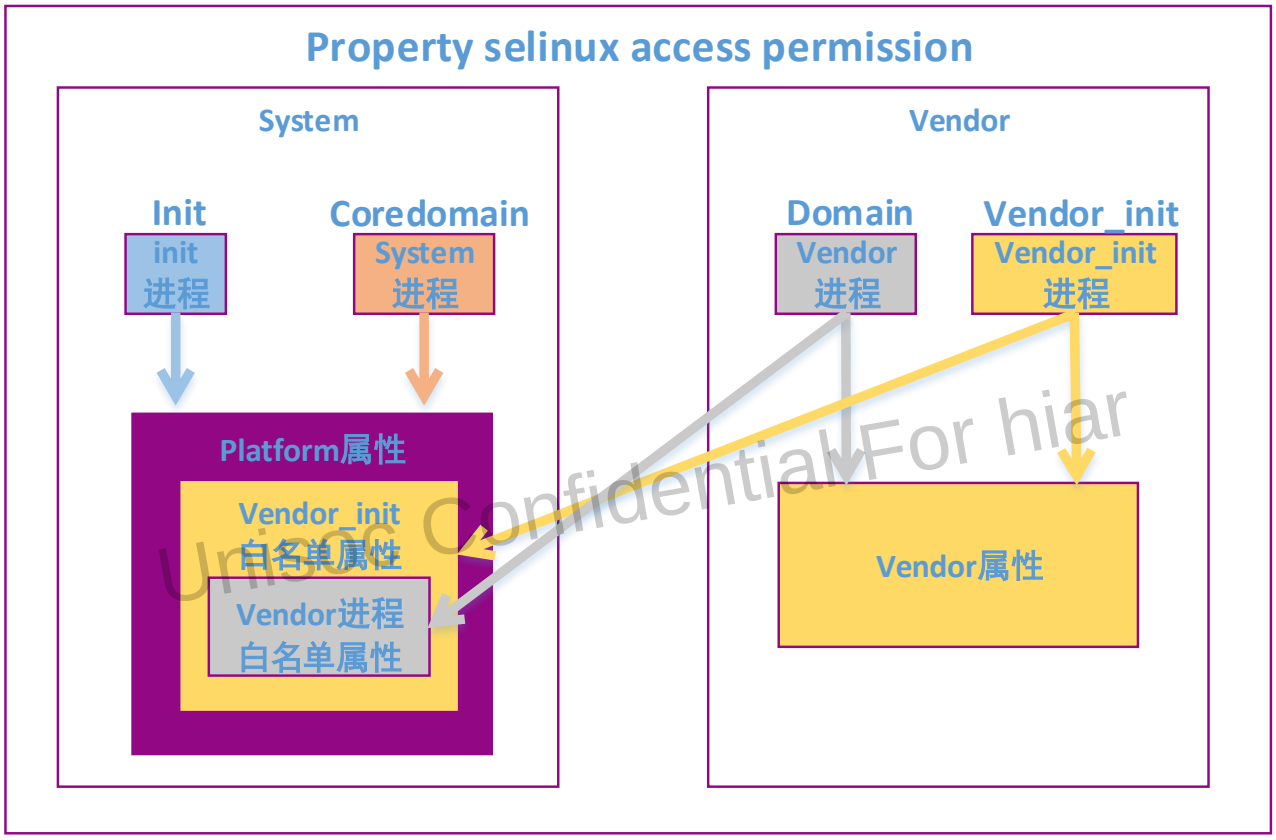
目 录

1 Android 10 属性相关说明

1.1 属性访问限制

属性的访问限制是通过 SELinux 规则实现的，原则上，Android 10 上 system 和 vendor 属性禁止互相访问，但为了更好的过渡，Google 增加了白名单机制使 vendor 可以访问部分开放的 platform 属性。

访问限制如下图所示(图中的进程和属性都是和 SELinux 关联的)：



system 进程：位于 system 分区的进程

vendor 进程：位于 vendor 分区的进程

platform 属性：被创建供 android platform 使用的属性，位于 system 分区

vendor 属性：被创建供 vendor 进程使用的属性，位于 vendor 分区

白名单属性：一部分开放给 vendor 分区进程使用的 Platform 属性

1.2 属性的 SELinux 规则

如前所述，系统属性访问限制是通过设置 SELinux 规则达成的，SELinux 有专门的属性 context 文件 property_contexts 用以说明该属性所属的属性类型上下文，SELinux 通过控制这些属性类型的权限达到限制访问的目的。

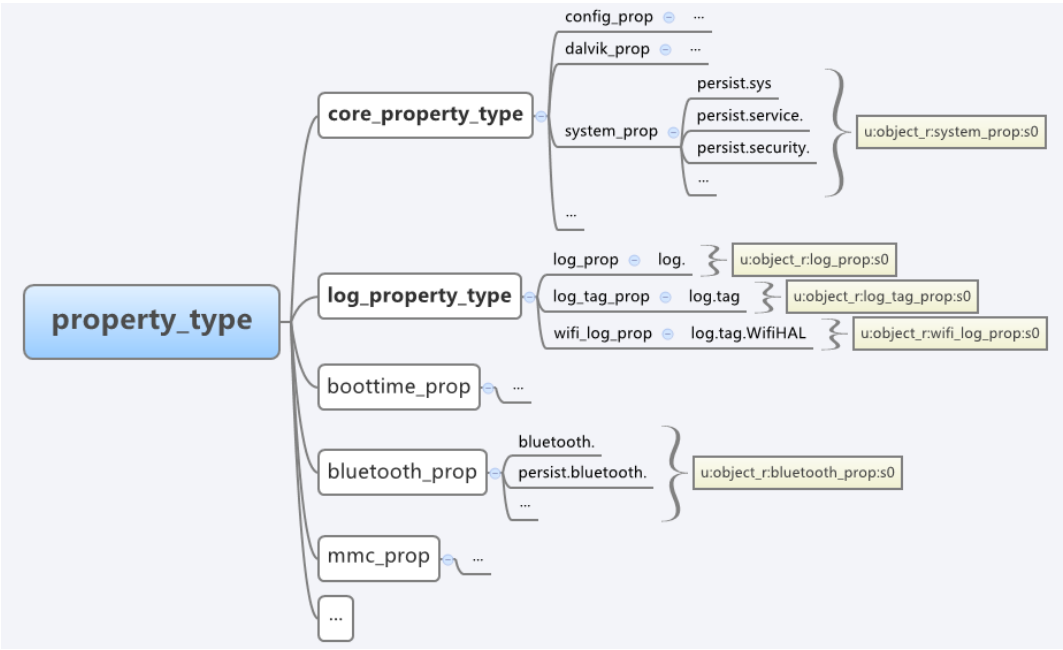
定义属性前缀的 SELinux context：

```
system/sepolicy/private/property_contexts
…
persist.sys.                    u:object_r:system_prop:s0
persist.service.                u:object_r:system_prop:s0
persist.security.               u:object_r:system_prop:s0
…
```

定义属性 SELinux context 的类型：

```
system/sepolicy/public/property.te
…
type net_radio_prop, property_type, core_property_type;
…
```


合在一起如下图，属性前缀、property type 和 SELinux context 就关联起来了：



通过设置进程对属性的 SELinux 规则即可以达到管理属性权限目的：

```
device/sprd/sharkle/common/sepolicy/sprd_engineermode_app.te

set_prop(sprd_engineermode_app, vendor_default_prop)
```

1.3 白名单属性

白名单属性是开放给非 system 分区访问的 platform 属性，此类属性不会因为 android 的版本升级而改变，vendor 应用可以只读方式访问此类 platform 属性。

类型和位置	详情
system/sepolicy/public/property_context vendor-init-settable	vendor init 可设置的 platform 属性, 即 vendor 分区的. rc 和. prop 可设置的 platform 属性
system/sepolicy/public/property_contexts vendor-init-readable	vendor init 可读取的 platform 属性，即 vendor 分区的.rc 可 platform 属性
system/sepolicy/public/property_contexts public-readable	public 只读白名单属性，所有分区皆可以只读方式访问此类属性

1.4 vendor 属性命名规则

vendor 添加的属性必需按照以下规则命名：

属性类型	可使用的前缀
read-write	vendor. odm.
read-only	ro.vendor. ro.odm. ro.boot. ro.hardware.
persist	persist.vendor. persist.odm.

相关 SELinux 规则添加时也要遵守类似的命名规则，增加 vendor_前缀，如

```
persist.vendor.ylog.       u:object_r:vendor_ylog_prop:s0
vendor.ylog.          u:object_r:vendor_ylog_prop:s0
```

1.5 新增属性的位置摆放

原则上，新增的属性需要放至其所属的分区中：

- 新增的 vendor 属性需要放至 vendor 分区
- 新增的 platform 属性需要放至 system 分区

新增属性需要加到对应宏和文件的选择请参考 1.9 属性宏的区别

1.6 属性合规检查

白名单机制和 vendor 属性命名规则的要求会通过 vts 测试来保证，build 阶段和 runtime 阶段都会进行属性违规检查。

1.6.1. build 阶段

属性相关的 SELinux neverallow 检查，违反规则会报错，比如：hal_power_default 属于 domain 进程，没有权限设置 system_prop，编译时有如下打印


```
neverallow check failed at plat_pub_versioned.cil:6946
   (neverallow base_typeattr_248_10000_0 base_typeattr_252_10000_0 (file (ioctl read write create setattr lock relabelfrom append unlink link rename open)))
      <root>
      allow at vendor_sepolicy.cil:2482
         (allow hal_power_default system_prop_10000_0 (file (ioctl read getattr lock map open)))
```

1.6.2. runtime 阶段

对属性访问进行 SELinux 权限检查，如果没有权限，会报错，参考 2.1. SELinux 权限问题处理流程

1.6.3. vts 测试

vts 测试文件位置：test/vts-testcase/security/system_property/VtsTrebleSysPropTest.py

包含以下三项检查

1） ro.actionable_compatibility_property.enabled 是否为 true，

即需要 board 配置中 PRODUCT_COMPATIBIE_PROPERTY 为 true

2） 白名单是否被篡改，白名单参考 1.3. 白名单属性
3） vendor 属性是否符合 vendor 命名规则，命名规则参考 1.4. vendor 属性命名规则
4） odm 属性是否符合 odm 命名规则，命名规则与 vendor 属性一致，可参考 1.4. vendor 属性命名规则

1.7 Board mk 加载顺序

由于同名的属性生效和加载顺序有关系，而属性的初始值大部分是在 board mk 中定义，所以 Board mk 的加载顺序在一定意义上直接决定了 prop 生效的顺序。

board mk 繁多，若逐一查看会非常繁琐，好在 Android 提供了一个原生手段去查看 board mk 的加载顺序，方法如下：

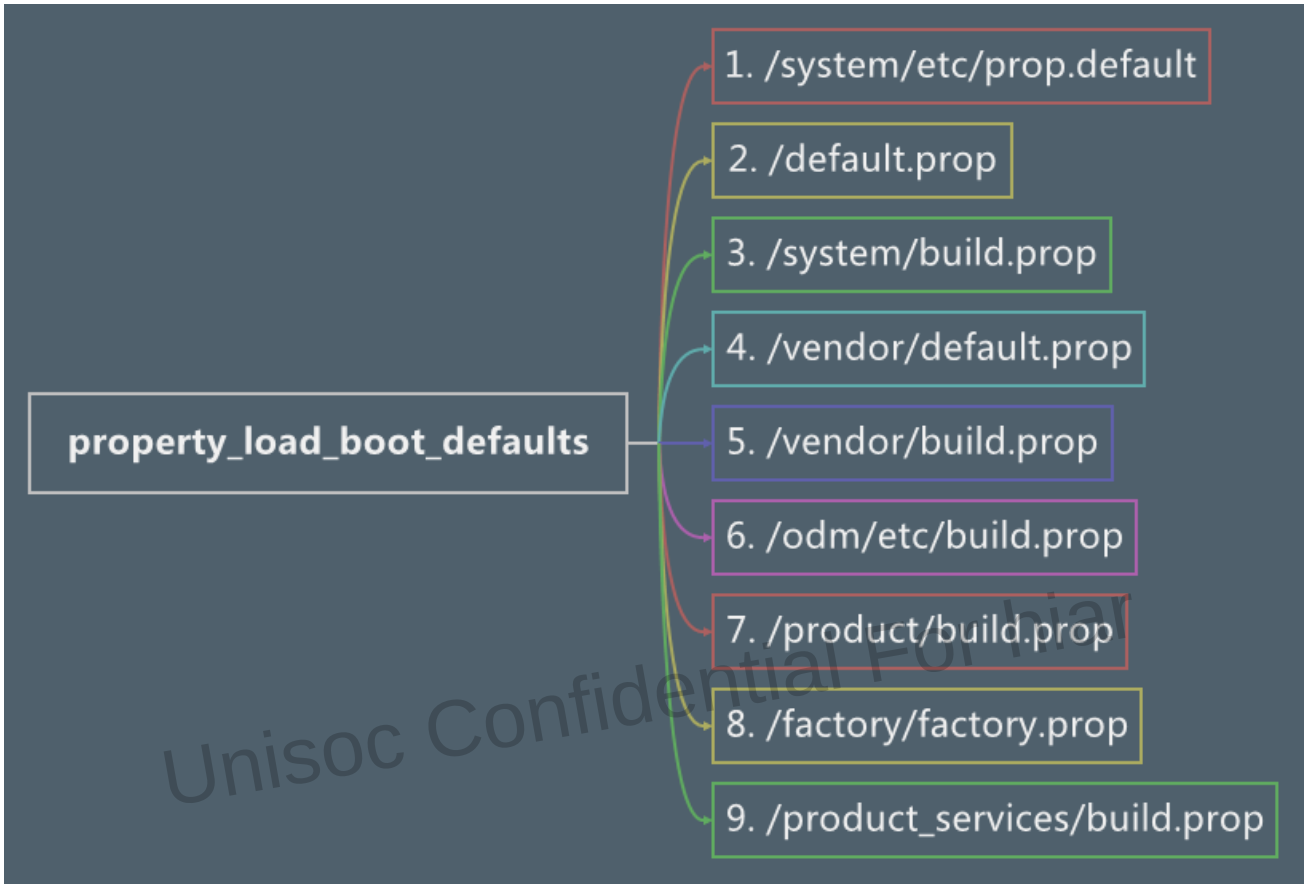
lunch 相关工程后，执行“make product-graph”可以自动生成 product mk 继承关系图 out/product.pdf，out/product.dot，out/product.svg

- 需安装 graphviz：sudo apt-get install graphviz
- 拷贝 product.dot 到本地后运行：dot -Tpdf -Nshape=box -o products.pdf products.dot

1.8 属性文件的加载顺序

属性文件的加载顺序决定了属性生效的值，在启动过程中加载属性的时候，后加载的属性才是最终设置的属性值，包括只读属性。

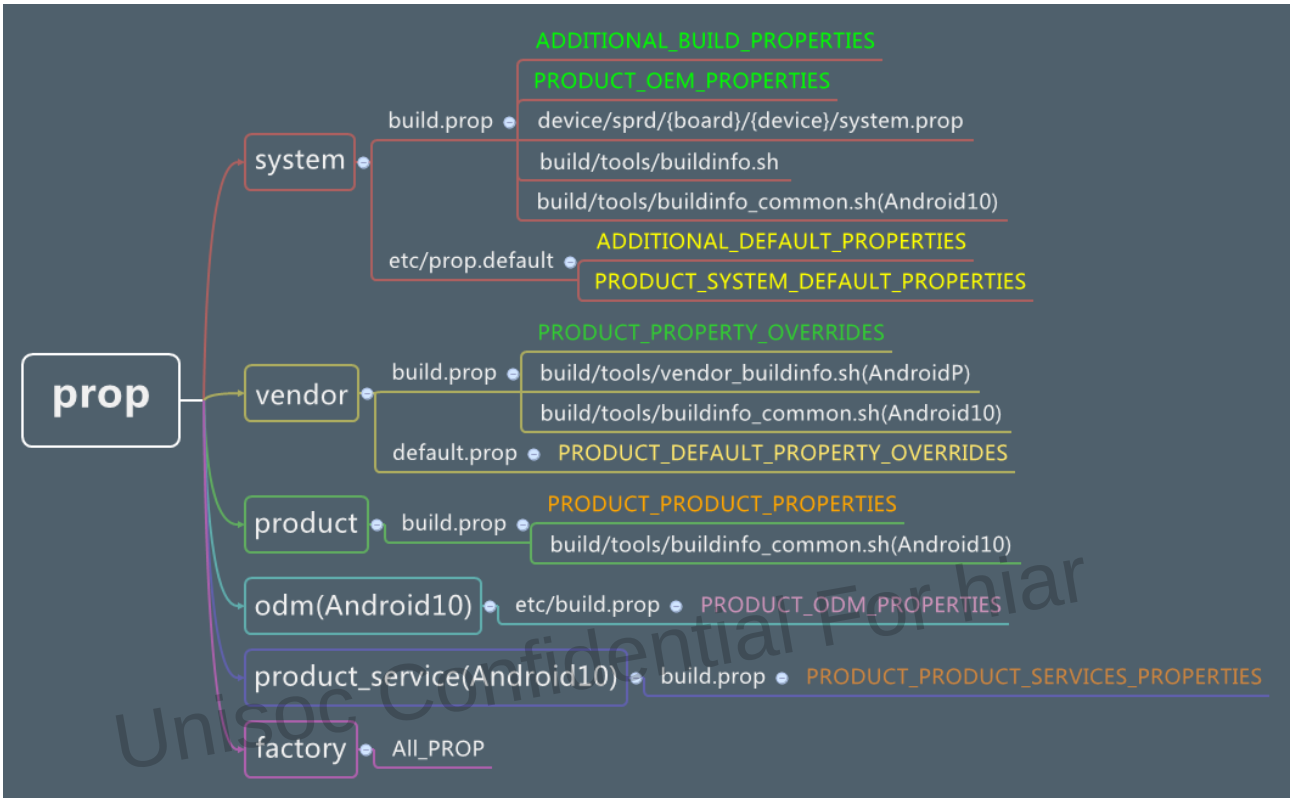
下面是 Android 10 属性文件的加载顺序图：



1.9 属性宏的区别

在为属性做初始值定义的时候，把属性加到不同的宏或者文件中，意味着这个属性会添加到不同的分区中，在不同分区下的 prop 具有不同的访问权限，所以，属性初始值的定义就显得重要，与之相关的，属性宏和文件的选择就是重中之重。

这些属性宏和文件与分区的对应关系如下：



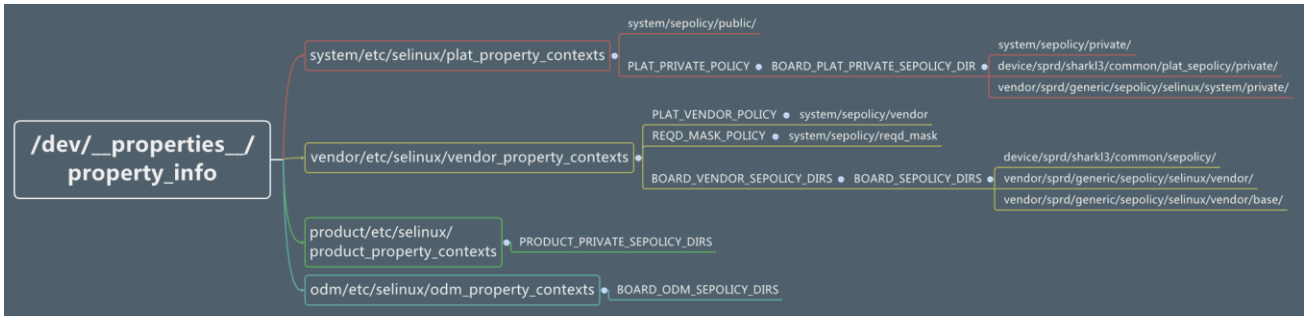
各开发者可依据以上 prop 关系对照图进行选择属性宏和文件，对于 ADDIAIONAL_BUILD_PROPERTIES 添加的限制请参见 1.11 System 属性宏添加位置限制。

1.10 property_contexts 文件的选择

在代码仓库中存在很多的 property_contexts 文件，在添加属性的时候如何选择这个文件呢？先来看下各个 property_contexts 文件的对应关系，尔后可根据这个对应关系添加 property contexts。

一般涉及的路径有：

- system/sepolicy/private/
- system/sepolicy/public/
- device/sprd/{ board }/common/sepolicy/
- device/sprd/{ board }/common/plat_sepolicy/private/
- vendor/sprd/generic/sepolicy/selinux/vendor/
- vendor/sprd/generic/sepolicy/selinux/system/private/



可以看到，每一个 property_contexts 与 system 和 vendor 下的 plat_property_contexts 文件的对应关系，实际使用中可依据 BOARD_PLAT_PRIVATE_SEPOLICY_DIR 和 BOARD_SEPOLICY_DIRS 分别添加 system 和 vendor 的 property_contexts，具体编译规则可参考“system/sepolicy/Android.mk”。

1.11 System 属性宏添加位置限制

System 属性对应的宏添加比较特殊，并不是将 system 属性的定义对应的宏加到每个 board mk 都可以生效，在编译的时候对 system 属性对应的宏生效的文件做了限制。主要规则可参考 vendor/sprd/build/core/apply_product_revision.mk。

主要规则如下：

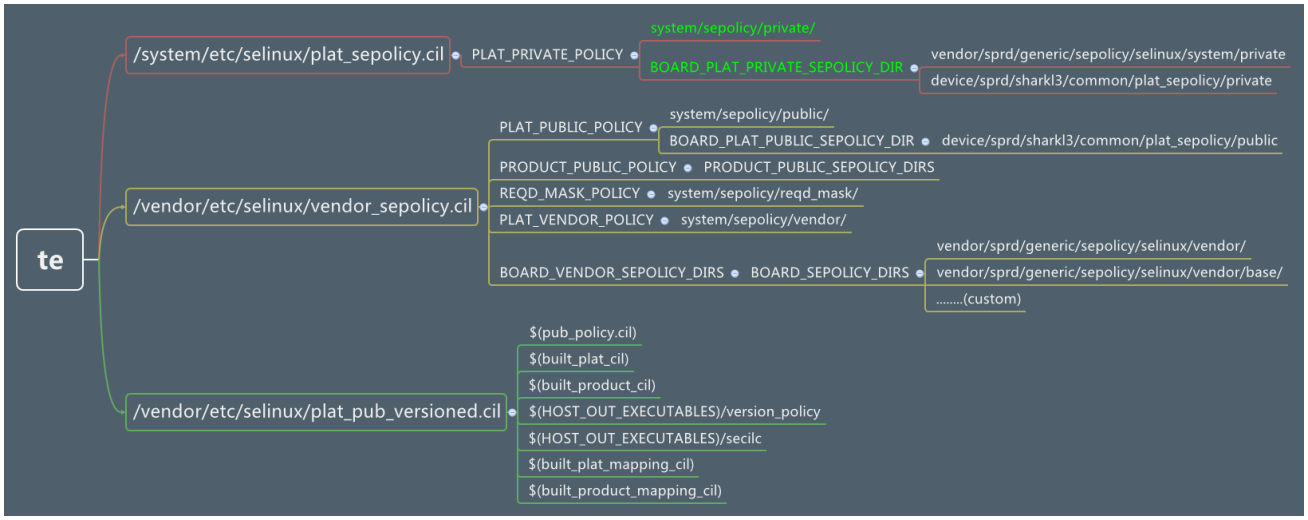
```
PRIVATE_FEATURE_LIST_PATH := \
        $(BOARDDIR)/features/$(carrier_feature_dir) $(BOARDDIR)/features \
        $(PLATDIR)/features/$(carrier_feature_dir)   $(PLATDIR)/features \
        $(PLATDIR)/common/features/$(carrier_feature_dir) $(PLATDIR)/common/features \
        $(PRODUCT_REVISION_COMMON_CONFIG_PATH)/$(carrier_feature_dir) \
        $(PRODUCT_REVISION_COMMON_CONFIG_PATH)
```


只有在这些路径下的文件或指定文档定义 system 属性才可以生效编译到手机中，否则不会生效。对应到项目，这个宏展开后如下，供参考：

```
PRIVATE_FEATURE_LIST_PATH := \
        device/sprd/{board}/{board_hw}/features/carriers \
        device/sprd//{board}/{board_hw}/features \
        device/sprd/{ board }/features/carriers \
        device/sprd/{ board }/features \
        device/sprd/{ board }/common/features/carriers \
        device/sprd/{ board }/common/features \
        vendor/sprd/feature_configs/carriers \
        vendor/sprd/feature_configs
```


1.12 Te 文件的选择

和 property_contexts 一样，在代码仓库中存在很多的 te 文件，在添加属性的时候如何选择这个文件呢？由于所有的 Te 文件最终都会汇总到各自分区的 cil 文件中，所以我们只需要理清楚看下各个路径 te 文件的对应关系，尔后可根据这个对应关系添加到相应路径下的模块 te 即可，优先加到各自模块中的 te，以便于后续维护。



2 常见问题

2.1. SELinux 权限问题处理流程

2.1.1 属性读取失败

log 如下：

```
01-01 12:57:41.490 W/HwBinder:294_4(  294): type=1400 audit(0.0:2397): avc: denied { read } for name="u:object_r:media_prop:s0" dev="tmpfs" ino=8581 scontext=u:r:mediacodec:s0 tcontext=u:object_r:media_prop:s0 tclass=file permissive=0

01-01 12:57:41.501 E/libc ( 294): Access denied finding property "media.mediadrmservice.enable"
```

2.1.2 属性设置失败

log 如下

```
01-01 13:44:32.986 W/libc (23479): Unable to set property "persist.audio.test" to "test": connection failed; errno=13 (Permission denied)

01-01 13:44:32.980 W/ndroid.systemui(23479): type=1400 audit(0.0:3642): avc: denied { connectto } for path="/dev/socket/property_service" scontext=u:r:platform_app:s0:c512,c768 tcontext=u:r:init:s0 tclass=unix_stream_socket permissive=0
```


读取和设置失败的 Log 处理方法类似：

首先从 log 查看属性类型和进程类型，然后按以下分类处理：

a. platform 进程访问 vendor 属性
   --- 已禁止
b. platform 进程访问 paltform 属性
   --- 检查进程是否有权限访问属性，然后在进程对应的 te 文件中增加 get_prop 或 set_prop 语句。
   如：2.1.1 属性读取失败的问题解决为添加 get_prop(mediacodec, media_prop)
   如：2.1.2 属性设置失败的问题解决为添加 set_prop(platform_app, audio_prop)
c. vendor 属性访问 platform 属性
   1. 该属性确实是 platform 属性
      --- 检查该属性是否为白名单属性，若非白名单属性则不能访问，若为白名单属性，则添加对应的访问权限
   2. 该属性是 vendor 属性，只是命名成了 platform 属性
      --- 参考 1.4 vendor 属性命名规则，对属性重新命名
d. vendor 进程访问 vendor 属性
   --- 在进程对应的 te 文件中增加 get_prop 或 set_prop 语句

set_prop 已包含 get_prop 规则，所以如果增加了 set_prop 后，不用再加 get_prop 规则

2.2. 新增属性

新增属性的详细规则请参见 1.5 新增属性的位置摆放。

属性的设置有效性请参考 2.3 属性设置有效性。

下面做一些简要说明：

2.2.1. 存放到 system 分区的属性

分三种情况:

- 单独项目特有的属性添加到项目特定的 system.prop 中，如 device/sprd/{board}/{board_hw}/system.prop
- borad common 的属性添加到 device/sprd/{board}/common/features/base/config.mk 中的 ADDITIONAL_BUILD_PROPERTIES
- feature common 的属性，添加到 vendor/sprd/feature_configs/[feature name]/config.mk

  或项目指定的 feature 目录的 config.mk，如

device/sprd/{board}/{board_hw}//features/[featurename]/config.mk 中的 ADDITIONAL_BUILD_PROPERTIES

2.2.2. 存放到 vendor 分区的属性

建议在 device 目录下的 mk 文件中添加 PRODUCT_PROPERTY_OVERRIDES

比如在 device/{board}/{board_hw}/common/DeviceCommon.mk 中添加

PRODUCT_PROPERTY_OVERRIDES += vendor.drm.service.enabled=false

以上是以 build.prop 举例，default.prop 是类似的

2.3. 属性设置有效性

2.3.1. build 阶段

build 阶段 mk 文件读入顺序是有先后的，如果不同的 mk 文件中包含了相同的属性，哪个会起作用呢？

以 persist.sprd.test 举例（其他前缀属性也遵从这个规则）

a.mk 定义 PRODUCT_PROPERTY_OVERRIDES += persist.sprd.test=1

b.mk 定义 PRODUCT_PROPERTY_OVERRIDES += persist.sprd.test=0

系统编译时，先加载 a.mk

结果在/vendor/build.prop 里 persist.sprd.test=1，也就是先加载的决定最后的属性值。

2.3.2. runtime 阶段

runtime 加载属性文件设置属性阶段，最后一次的属性赋值决定最后的属性值，包括 ro 属性。

2.3.3. build 和 runtime 综合例子

以 2.2 小节中新加 platform 属性做例子，会涵盖 build 阶段和 runtime 阶段：

build 阶段 mk 加载顺序如下：

vendor/sprd/feature_configs/[feature name]/config.mk（ro.sprd.test=vendor persist.sprd.test=vendor）

device/sprd/{board}/common/features/base/config.mk（ro.sprd.test=device persist.sprd.test=device）

mk 中的属性合并后去重复项，得到

ro.sprd.test=vendor

persist.sprd.test=vendor

然后 system.prop 与其合并生成/system/build.prop，这里不会去重复项

device/sprd/{board}/{board_hw}/system.prop（ro.sprd.test=prop persist.sprd.test=prop）

编译生成的/system/build.prop 文件中会包含以下

```
…
ro.sprd.test=prop
persist.sprd.test=prop
…
ro.sprd.test=vendor
persist.sprd.test=vendor
…
```


runtime 阶段 init 会读取/system/build.prop 中的属性进行初始化

开机后执行

adb shell getprop ro.sprd.test 返回 prop

adb shell getprop persist.sprd.test 返回 vendor

2.4. 添加 sepolicy 属性权限

当有违反 SELinux 的行为出现时，我们需要根据需要进行添加 sepolicy 权限，一般情况下都是通过 allow 的方式去添加相应权限，如 allow user_t bin_t:file {execute, xxx};

值得注意的是 te 文件中的 set_prop 和 get_prop 是宏定义,其包含了多条 allow 语句；

举例

set_prop(platform_app, storage_prop)最终展开成以下四句

```
allow platform_app property_socket:sock_file write;
allow platform_app init:unix_stream_socket connectto;
allow platform_app storage_prop:property_service set;
allow platform_app storage_prop:file {ioctl read getattr lock map open};
```

get_prop(platform_app, storage_prop)展开成一句

allow platform_app storage_prop:file {ioctl read getattr lock map open};

可以看出 set_prop 已经包含了 get_prop。