SELinux 问题快速分析

*SELinux Overview

1-SELinux Overview

1. SELinux 来源

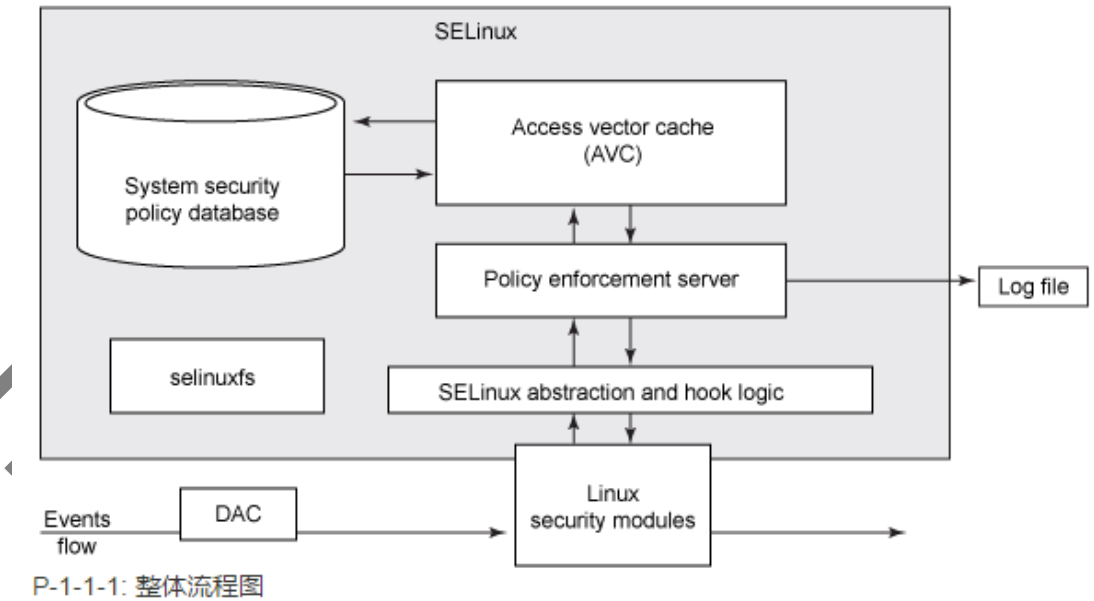
SELinux 即 Security-Enhanced Linux, 由美国国家安全局(NSA)发起, Secure Computing Corporation (SCC) 和 MITRE 直接参与开发, 以及很多研究机构(如犹他大学)一起参与的强制性安全审查机制, 该系统最初是作为一款通用访问软件，发布于 2000 年 12 月(代码采用 GPL 许可发布)。并在 Linux Kernel 2.6 版本后, 有直接整合进入 SELinux, 搭建在 Linux Security Module(LSM)基础上, 目前已经成为最受欢迎，使用最广泛的安全方案.

2. SELinux 基本架构与原理.

SELinux 是典型的 MAC-Mandatory Access Controls 实现, 对系统中每个对象都生成一个安全上下文(Security Context), 每一个对象访问系统的资源都要进行安全上下文审查。审查的规则包括类型强制检测(type enforcement), 多层安全审查(Multi-Level Security), 以及基于角色的访问控制(RBAC: Role Based Access Control).

SELinux 搭建在 Linux Security Module(LSM)基础上, 关于 LSM 架构的详细描述请参见文章 “Linux Security Modules: General Security Support for the Linux Kernel”, 该文章在 2002 年的 USENIX Security 会议上发表。有完整的实现 LSM 的所有 hook function.

SELinux 的整体结构如下图所示:

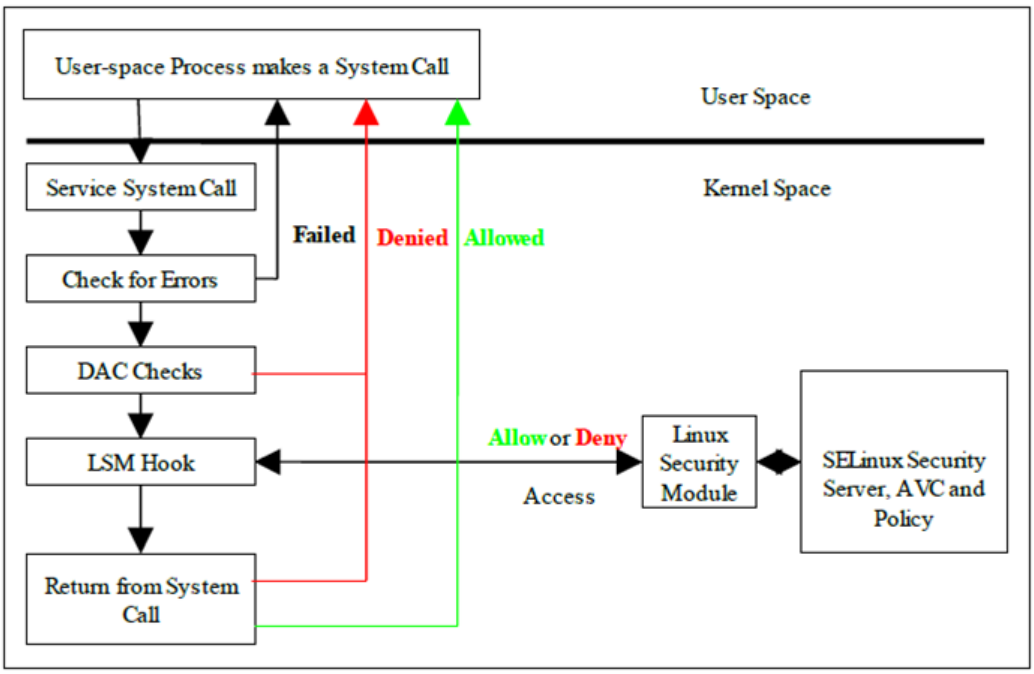


P-1-1-1: 整体流程图

SELinux 包含五个基本组成：

* 用于处理文件系统的辅助模块, 即 SELinuxFS.
* 集成 Linux Security Modules 的 hooks sets.
* Security Policy Database.
* Security Label 验证模块.
* Access Vector Cache (AVC), 访问向量缓存,以便提高验证速度.

基本的访问流程如下图所示:



P-1-1-2: 访问流程图

流程如下：

* 进程通过系统调用(System Call) 访问某个资源, 进入Kernel 后, 先会做基本的检测, 如果异常则直接返回.
* Linux Kernel DAC 审查, 如果异常则直接返回.
* 调用Linux Kernel Modules 的相关hooks, 对接到SELinux 的hooks, 进而进行MAC 验证, 如果异常则直接返回.
* 访问真正的系统资源.
* 返回用户态, 将结构反馈.

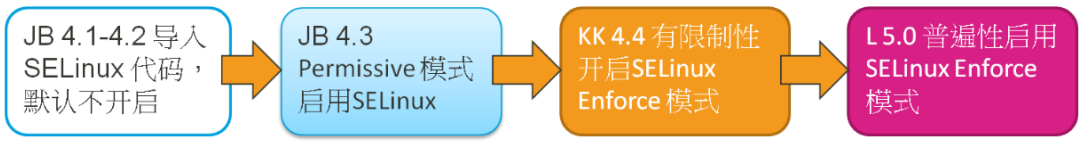
2- SELinux and SEAndroid

1. SELinux and SEAndroid

Android 是建立在标准的 Linux Kernel 基础上, 自然也可以开启 SELinux, 通常在通用移动平台上, 很少开启这样的安全服务, Google 为了进一步增强 Android 的安全性, 经过长期的准备, 目前已经在 Android 5.0(L) 上有完整的开启 SELinux, 并对 SELinux 进行深入整合形成了 SEAndroid.

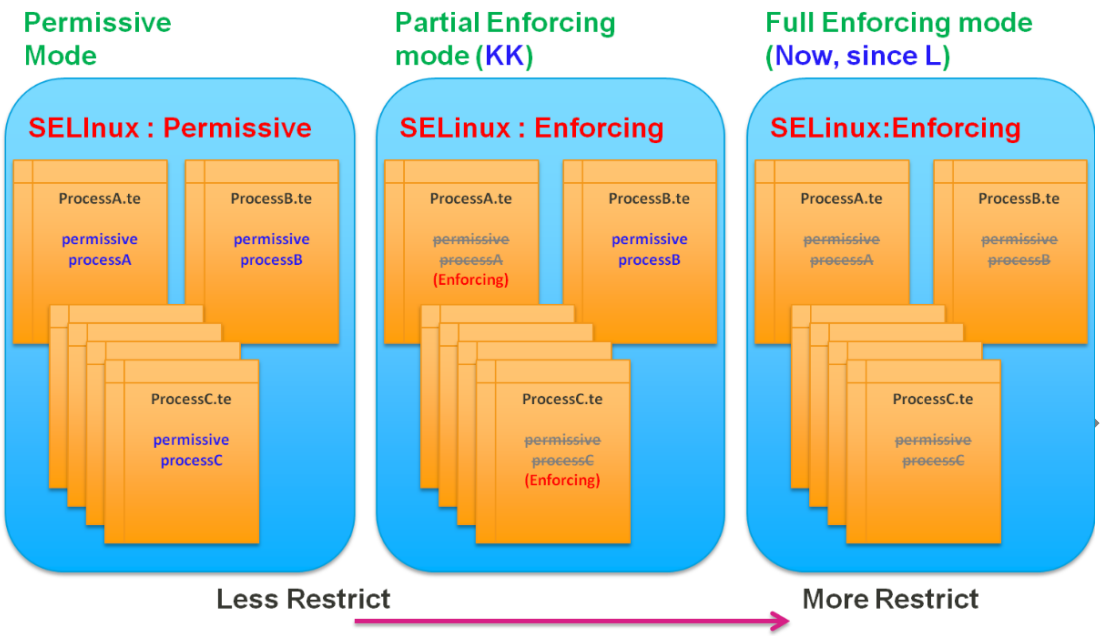
2. SELinux **在** Android **上的更新历史**

如下图所描述:



P-1-2-1 SELinux 在 Android 的更新过程

* KK 4.4 针对 netd, installd, zygote, vold 四个原本具有 root 权限的 process, 以及它们 fork 出的子进程启用 Enforce 模式.
* L 版本普遍性开启 SELinux Enforce mode.
* Permissive 模式，只打印 audit 异常 LOG，不拒绝请求, Enforce 模式，即打印 audit 异常 LOG, 也拒绝请求



P-1-2-2 SELinux 在 Android 启动模式变化
估计后续版本 Google 都会要求强制性开启 SELinux, 以保证手机安全.

3. SELinux 给 Android 带来的影响.
* 严格限制了 ROOT 权限, 以往 ROOT "无法无天" 的情况将得到极大的改善.
* 通过 SELinux 保护, 降低系统关键进程受攻击的风险, 普通进程将没有权限直接连接到系统关键进程.
* 进一步强化 APP 的沙箱机制, 确保 APP 难以做出异常行为或者攻击行为.
* 将改变 APP 一旦安装, 权限就已经顶死的历史, APP 权限动态调整将成为可能.

3- SELinux Mode.

1. SELinux Mode.
SELinux 分成两种模式, 即 Permissve Mode(宽容模式), 和 Enfocing mode(强制模式).
Permissive Mode 只通过 Audit System 记录 LOG, 但不真正拦截访问.
Enforcing Mode 在打印 LOG 的同时，还会真正的拦截访问.

通常在调试时，我们会启用 Permissive Mode, 以便尽可能的发现多的问题, 然后一次修正. 在真正量产时使用 Enforcing mode, 来保护系统.

*SELinux Basic Theory

4- DAC and MAC

1. DAC and MAC
DAC 即 Discretionary Access control, 自主访问控制, 即系统只提供基本的验证, 完整的访问控制由开发者自己控制。
MAC 即 Mandatory Access control, 强制性访问控制, 即系统针对每一项访问都进行严格的限制, 具体的限制策略由开发者给出.

2. Linux DAC
Linux DAC 采用了一种非常简单的策略, 将资源访问者分成三类, 分别是 Owner, Group, Other; 资源针对这三类访问者设置不同的访问权限. 而访问权限又分成 read, write, execute.
访问者通常是进程有自己的 uid/gid, 通过 uid/gid 和 文件权限匹配, 来确定是否可以访问.
将 Root 权限根据不同的应用场景划分成许多的 Root Capabilities, 其中如果有 CAP_DAC_OVERRIDE 这项的话, 可以直接绕过 Linux DAC 限制.

Linux DAC 有明显的不足, 其中一个重要点就是, Root 权限 “无法无天”, 几乎可以做任意事情, 一旦入侵者拿到 root 权限, 即已经完全掌控了系统. 另外每一个进程默认都拿到对应这个用户的所有权限, 可以改动/删除这个用户的所有文件资源, 明显这个难以防止恶意软件.

3. Linux MAC

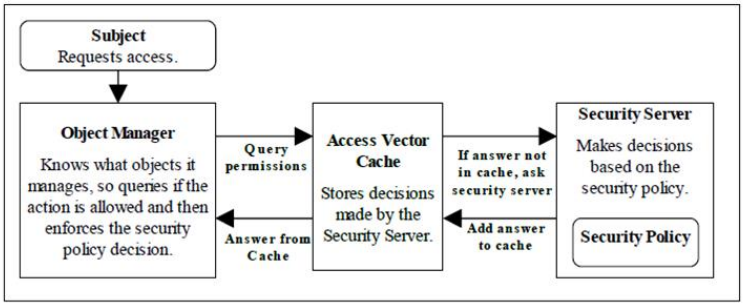
Linux MAC 针对 DAC 的不足, 要求系统对每一项访问, 每访问一个文件资源都需要进行针对性的验证. 而这个针对性的验证是根据已经定义好了的策略进行. 在 Linux Kernel, 所有的 MAC 机制都是搭建在 Linux Security Modules (LSM) 基础上, 包括有：SELinux、Apparmor、Smack 和 TOMOYO Linux 等。目前 SELinux 已经成了事实上的行业标准.

针对 Linux DAC, MAC 可以明显弥补 DAC 的缺陷, 一方面限制 Root 权限, 即使你有 root 权限, 如果无法通过 MAC 验证, 那么一样的无法真正执行相关的操作. 另外对每一项权限进行了更加完整的细化, 可限制用户对资源的访问行为.

5- Core SELinux Components and Security Context

1. Core SELinux Components

SELinux 的核心组件可以参考下面的图:



P-2-1-1 SELinux 核心组件

* Subject 通常是指触发访问行为的对象, 在 Linux 里面通常是一个进程(Process).
* Object Manager 即是对象访问管理器, 即可以知道 Subject 需要访问哪些资源，并且触发验证机制
* Security Server 即安全服务器, 用来验证某个 Subject 是否可以真正的访问某个 Object, 而这个验证机制是基于定义好的 Security Policy.
* Security Policy 是一种描述 SELinux Policy 的语言.
* Access Vector Cache (AVC) 是访问缓存, 用来记录以往的访问验证情况, 以便提供效率，快速处理.

2. Security Context

SELinux 给 Linux 的所有对象都分配一个安全上下文(Security Context)， 描述成一个标准的字符串。

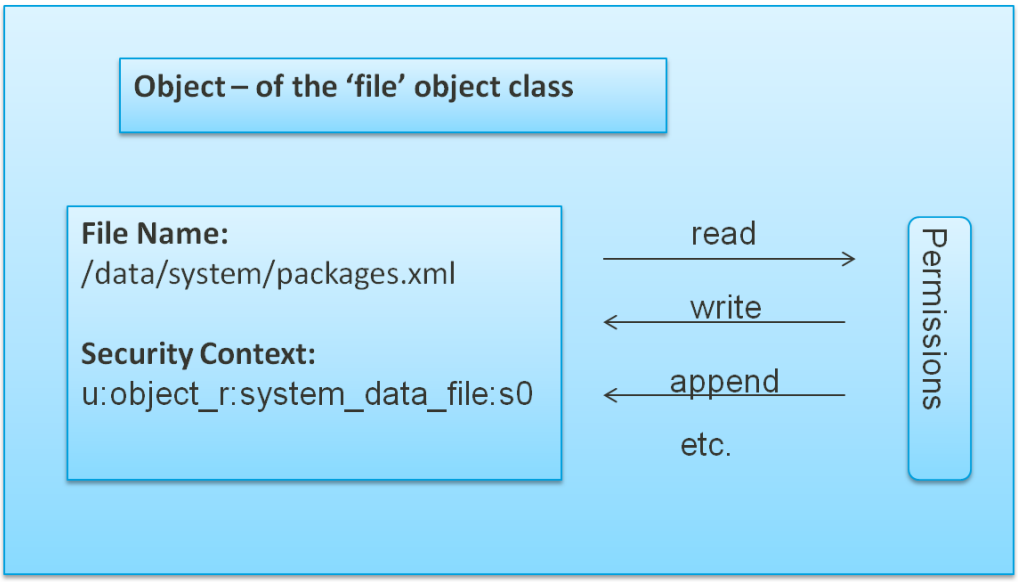
两种类型

* Subject 主体, linux 通常以进程为单位
* Object 访问对象, linux 通常以文件为单位

标准格式

user:role:type[:range]

* User: 用户, 非 Linux UID。
* Role: 角色，一个 user 可以属于多个 role，不同的 role 具有不同的权限。它是 SELinux 中一种比较高层次，更方便的权限管理思路，即 Role Based Access Control（基于角色的访问控制，简称为 RBAC, SELinux 不推荐使用）。
* Type: Subject 或者 Object 的类型。 MAC 的基础管理思路其实不是针对上面的 RBAC，而是所谓的 Type Enforcement Access Control（简称 TEAC，一般用 TE 表示）。对进程来说，Type 就是 Domain。
* Range: Multi-Level Security（MLS）的级别。MLS 将系统的进程和文件进行了分级，不同级别的资源需要对应级别的进程才能访问。



P-2-2-2

对象类型和权限项关系.

每一个对象都有一个 Class, 比如一支文件, 它是 File Class 类型, 每个 Class 类型都会根据实际情况定义权限类型项, 如: read, write, exec, ioctl, append 等等.

Subject 即前面提到的主体，它能够产生访问行为, Linux 里面通常是一个 process, 但 process 本身也可能是一个 Object, 比如另外一个进程对它发送 Signal, 比如去对它进行 Ptrace 操作.

3. **查看** Security Context

* 查看进程的 Security Context 用: ps -Z 或者 cat proc/PID/attr/current, 查看当前进程可用 id, 如:

```
u:r:system_server:s0aroot@mt6589_phone_720pv2:/ # ps -Z 706
ps -Z 706
LABEL                          USER     PID   PPID  NAME
u:r:system_server:s0           system    706   157   system_server
root@mt6589_phone_720pv2:/ # cat proc/706/attr/current
cat proc/706/attr/current
u:r:system_server:s0aroot@mt6589_phone_720pv2:/ #
```

P-2-2-3 ps 查询进程 Security Context

* 查看文件的 Security Context 用: ls -Z

```
127|root@mt6589_phone_720pv2:/proc/706/attr # ls -Z
ls -Z
-rw-rw-rw- system   system          u:r:system_server:s0 current
-rw-rw-rw- system   system          u:r:system_server:s0 exec
-rw-rw-rw- system   system          u:r:system_server:s0 fscreate
-rw-rw-rw- system   system          u:r:system_server:s0 keycreate
-r--r--r-- system   system          u:r:system_server:s0 prev
-rw-rw-rw- system   system          u:r:system_server:s0 sockcreate
```

P-2-2-4 查看文件 Security Context

6- Type Enforcement Access Control

1. SELinux supports two forms of MAC
* Type Enforcement (TE)
顾名思义, Type Enforcement 是根据 Security Label 中的 type 进行权限审查, 审查 subject type 对 object type 的某个 class 类型中某种 permission 是否具有访问权限，是目前使用最为广泛的 MAC 审查机制, 简单易用。

* Multi-Level Security (MLS)
多层安全机制, 是基于 Bell-La Padula (BLP) 模型, 将 Subject 和 Object 定义成多层次的安全等级, 不同安全等级之间有相关的访问约束, 常见的访问约束是 "no write down" 和 "no read up". 它是根据 Security Label 里面的最后一个字段 label 进行确认的.

目前在 Android 中，重点启用了 Type Enforcement 机制, Multi-Level Security (MLS) 虽然有定义, 但没有真实的投入使用, 这里暂时略过。

2. Type Enforcement Access Control

* 控制语句格式
rule_name source_type target_type : class perm_set
* rule: 控制类型， 分成两方面 allow 以及 audit
* source_type：也叫 subject，通常是 domain。
* Target_type: 代表请求的资源的类型
* class perm_set: 代表对资源访问的操作

3. Rule of Type Enforcement
如下表所示:

Rule Name	Allow if Match	Allow if NOT Match	Audit if Match	Audit if NOT Match
allow	Yes	No	No	Yes
dontaudit	No	No	No	No
auditallow	No	No	Yes	Yes
neverallow	This is a compile time check.			

P-2-3-1 TEAC-Rule

* allow：赋予某项权限。
* auditallow：audit 含义就是记录某项操作。默认 SELinux 只记录那些权限检查失败的操作。 auditallow 则使得权限检查成功的操作也被记录。注意，allowaudit 只是允许记录，它和赋予权限没关系。赋予权限必须且只能使用 allow 语句。
* dontaudit：对那些权限检查失败的操作不做记录。
* neverallow：前面讲过，用来检查安全策略文件中是否有违反该项规则的 allow 语句

4. **简单的案例**
下面一段是截取的 init.te 中的一部分

```
# Create /data/property and files within it.
allow init property_data_file:dir create_dir_perms;
allow init property_data_file:file create_file_perms;

# Set any property.
allow init property_type:property_service set;

# Run "ifup lo" to bring up the localhost interface
allow init self:udp_socket { create ioctl };

# This line seems suspect, as it should not really need to
# set scheduling parameters for a kernel domain task.
allow init kernel:process setsched;
```

```
# The init domain is only entered via setcon from the kernel domain,
# never via an exec-based transition.
neverallow { domain -kernel} init:process dyntransition;
neverallow domain init:process transition;
neverallow init { file_type fs_type }:file entrypoint;
```

7- Domain and Object Transitions

1. Domain Transitions

在 SELinux 中, 我们通常称一个进程是一个 domain, 一个进程 fork 另外一个进程并执行(exec) 一个执行档时, 我们往往会涉及到 domain 的切换. 比如 init 进程, SELinux 给予了它很大的权限, 而它拉起的服务, 我们要限制这个服务的权限，于是就涉及到从一个 domain 切换到另外一个 domain, 不然默认就使用 init 进程的 domain.

在 SELinux 里面有专门的一条语法:

type_transition statement.

在准备切换前我们先要确保有相关的权限操作.

*. The source domain has permission to transition into the target domain.
源 domain 必须有权限切换到这个目标 domain.

*. The application binary file needs to be executable in the source domain.
源 doamin 必须要有执行这个执行档的权限.

*. The application binary file needs an entry point into the target domain.
这个执行档必须是目标 domain 的入口(Entry Point)

如下面的 demo, init 拉起 apache 并且切换到 apache 的 domain.

```
#首先，你得让 init_t 域中的进程能够执行 type 为 apache_exec_t 的文件
allow init_t apache_exec_t : file {read getattr execute};

#然后，你还得告诉 SELinux，允许 init_t 做 DT 切换以进入 apache_t 域
allow init_t apache_t : process transition;

#最后，你还得告诉 SELinux，切换入口（对应为 entrypoint 权限）为执行 apache_exec_t 类型 的文件
allow apache_t apache_exec_t : file entrypoint;

#Domain Transition
type_transition init_t apache_exec_t : process apache_t;
```

2. Object Transitions

一个进程创建在一个目录下创建文件时, 默认是沿用父目录的 Security Context, 如果要设置成特定的 Label, 就必须进行 Object Transitions.

同样是使用

type_transition statement.

对应的必须有两个前提条件:

* The source domain needs permission to add file entries into the directory
这个 process 必须有在这个目录下添加文件的权限.

* The source domain needs permission to create file entries
这个 process 必须有在这个目录下创建以这个 Security Context 为 Label 的文件权限.

下面是一个 demo, ext_gateway_t 这个 domain 在类型为 in_queue_t 的目录下，创建类型为 in_file_t 的文件.

```
#首先，你得让 ext_gateway_t 对 in_queue_t 目录具备访问权限
allow ext_gateway_t in_queue_t : dir { write search add_name };

#然后，你还得告诉 SELinux，允许 ext_gateway_t 访问 in_file_t 的文件
allow ext_gateway_t in_file_t : file { write create getattr };

#Object Transition
type_transition ext_gateway_t in_queue_t : file in_file_t;
```

结果如下图:

```
ls -Za /usr/message_queue/in_queue
drwxr-xr-x root root unconfined_u:object_r:in_queue_t      .
drwxr-xr-x root root system_u:object_r:unconfined_t        ..
-rw-r--r-- root root unconfined_u:object_r:in_file_t       Message-1
-rw-r--r-- root root unconfined_u:object_r:in_file_t       Message-2
```

P-2-4-1 对象domain 切换结果

8- SELinux API

1. SELinux File System.
SELinux 在 Kernel 中实现, 统一对外的接口都集中在 SELinux File System 当中, 即通过读写相关文件的方式来达成。
SELinux File System 在比较新的 Linux Kernel 中被 mount 在/sys/fs/selinux. 老的版本就直接是/selinuxfs
具体每一只文件的说明, 请参考:
The SELinux Notebook: 2.19.2.3 SELinux Filesystem

2. SELinux Lib
所有 SELinux Lib 几乎都是对 selinuxfs 的封装, 具体的 API 可以参考:
The SELinux Notebook: 9. APPENDIX B - LIBSELINUX LIBRARY FUNCTIONS

在 android 上, Google 有进一步对这些 API 进行封装, 具体都在/external/libselinux 下面, 下面对一些重要的 API 进行说明. (参考: http://selinuxproject.org/page/NB_SEforAndroid_1)

selinux_android_setcontext
Sets the correct domain context when launching applications using setcon(3). Information contained in the seapp_contexts file is used to compute the correct context.
It is called by frameworks/base/core/jni/com_android_internal_os_Zygote.cpp when forking a new process and the system/core/run-as/run-as.c utility.

selinux_android_setfilecon
Sets the correct context on application directory / files using setfilecon(3). Information contained in the seapp_contexts file is used to compute the correct context.
The function is used by the package installer within frameworks/native/cmds/installd/commands.c via the package install() and make_user_data() functions.

selinux_android_restorecon
selinux_android_restorecon_pkgdir
Basically these functions are used to label files and directories based on entries from the file_contexts and/or seapp_contexts files. They call a common handler (selinux_android_restorecon_common()) that will then relabel the requested directories and files. It will also handle recursive labeling of directories and files should a new app, file_contexts or seapp_contexts be installed (see the Checking File Labels section for further information).
The selinux_android_restorecon function is used by:
frameworks/native/cmds/installd/installd.c when installing a new app.
frameworks/base/core/jni/android_os_SELinux.cpp for the Java native_restorecon method.
frameworks/native/cmds/dumpstate/utils.c when dumping Dalvik and stack traces to ensure correct label.
The selinux_android_restorecon_pkgdir function is used by:
frameworks/native/cmds/installd/commands.c for the package restorecon_data() and make_user_data() functions.

selinux_android_seapp_context_reload
Loads the seapp_contexts file for frameworks/native/cmds/installd/installd.c when the package installer is loaded.

selinux_android_load_policy
Mounts the SELinux filesystem if SELinux is enabled and then calls selinux_android_reload_policy to load the policy into the kernel. Used by system/core/init/init.c to initialise SELinux.

selinux_android_reload_policy
Reloads the policy into the kernel. Used by system/core/init/init.c selinux_reload_policy() to reload policy after setting the selinux.reload_policy property.

selinux_android_use_data_policy
Used by system/core/init/init.c to decide which policy directory to load the property_contexts file from.
There is also a new labeling service for selabel_lookup(3) to query the Android property_contexts and service_contexts files.
Various Android services will also call (not a complete list):

selinux_status_updated(3), is_selinux_enabled(3), to check whether anything changed within the SELinux environment (e.g. updated configuration files).

selinux_check_access(3) to check if the source context has access premission for the class on the target context.

selinux_label_open(3), selabel_lookup(3), selinux_android_file_context_handle, selinux_android_prop_context_handle, setfilecon(3), setfscreatecon(3) to manage file labeling.

selinux_lookup_best_match called by system/core/init/devices.c when ueventd creates a device node as it may also create one or more symlinks (for block and PCI devices). Therefore a "best match" look-up for a device node is based on its real path, plus any links that may have been created

3. SELinux Java API

Google 进一步将部分重要的 SELinxu API 包装成了 Java API. 对应的代码是：

Java API: frameworks/base/core/java/android/os/SELinux.java

JNI Code: frameworks/base/core/jni/android_os_SELinux.cpp

对应的 API 描述如(参考: http://selinuxproject.org/page/NB_SEforAndroid_1):

boolean isSELinuxEnabled()
  Determine whether SELinux is enabled or disabled.
  Return true if SELinux is enabled.

boolean isSELinuxEnforced()
  Determine whether SELinux is permissive or enforcing.
  Returns true if SELinux is enforcing.

boolean setSELinuxEnforce(boolean value)
  Set whether SELinux is in permissive or enforcing modes.
  value of true sets SELinux to enforcing mode.
  Returns true if the desired mode was set.

boolean setFSCreateContext(String context)
  Sets the security context for newly created file objects.
  context is the security context to set.
  Returns true if the operation succeeded.

boolean setFileContext(String path, String context)
  Change the security context of an existing file object.
  path represents the path of file object to relabel.
  context is the new security context to set.
  Returns true if the operation succeeded.

String getFileContext(String path)
  Get the security context of a file object.
  path the pathname of the file object.
  Returns the requested security context or null.

String getPeerContext(FileDescriptor fd)
  Get the security context of a peer socket.
  FileDescriptor is the file descriptor class of the peer socket.
  Returns the peer socket security context or null.

String getContext()
  Gets the security context of the current process.
  Returns the current process security context or null.

String getPidContext(int pid)
  Gets the security context of a given process id.
  pid an int representing the process id to check.
  Returns the security context of the given pid or null.

String[] getBooleanNames()
  Gets a list of the SELinux boolean names.
  Return an array of strings containing the SELinux boolean names.

boolean getBooleanValue(String name)
  Gets the value for the given SELinux boolean name.
  name is the name of the SELinux boolean.

Returns true or false indicating whether the SELinux boolean is set or not.

boolean setBooleanValue(String name, boolean value)
Sets the value for the given SELinux boolean name. Note that this will be set the boolean permanently across reboots.
name is the name of the SELinux boolean.
value is the new value of the SELinux boolean.
Returns true if the operation succeeded.

boolean checkSELinuxAccess(String scon, String tcon, String tclass, String perm)
Check permissions between two security contexts.
scon is the source or subject security context.
tcon is the target or object security context.
tclass is the object security class name.
perm is the permission name.
Returns true if permission was granted.

boolean native_restorecon(String pathname)
Restores a file to its default SELinux security context. If the system is not compiled with SELinux, then true is automatically returned. If SELinux is compiled in, but disabled, then true is returned.
pathname is the pathname of the file to be relabeled.
Returns true if the relabeling succeeded.

boolean restorecon(String pathname)
Restores a file to its default SELinux security context. If the system is not compiled with SELinux, then true is automatically returned. If SELinux is compiled in, but disabled, then true is returned.
pathname is the pathname of the file to be relabeled.
Returns true if the relabeling succeeded.
exception NullPointerException if the pathname is a null object.

boolean restorecon(File file)
Restores a file to its default SELinux security context. If the system is not compiled with SELinux, then true is automatically returned. If SELinux is compiled in, but disabled, then true is returned.
file is the file object representing the path to be relabeled.
Returns true if the relabeling succeeded.
exception NullPointerException if the file is a null object.

9- SELinux Commands
1. SELinux Commands
参考(http://selinuxproject.org/page/NB_SEforAndroid_1):

Command	Comment
chcon	Change security context of file: chcon context path
getenforce	Returns the current enforcing mode. getenforce
getsebool	Returns SELinux boolean value(s): getsebool [-a | boolean_name]
id	If SELinux is enabled then the security context is automatically displayed.
load_policy	Load new policy into kernel: load_policy policy-file

ls	Supports -Z option to display security context.
ps	Supports -Z option to display security context.
restorecon	Restore file default security context as defined in the file_contexts or seapp_contexts files. The options are: D - data files, F - Force reset, n - do not change, R/r - Recursive change, v - Show changes. `restorecon [-DFnrRv] pathname`
runcon	Run command in specified security context: `runcon context program args...`
setenforce	Modify the SELinux enforcing mode: `setenforce [enforcing|permissive|1|0]`
setsebool	Set SELinux boolean to a value (note that the cmd does not set the boolean across reboots): `setsebool boolean_name [1|true|on|0|false|off]`

2. Init.rc Commands.

seclabel <securitycontext> service option: Change to security context before exec'ing this service. Primarily for use by services run from the rootfs, e.g. ueventd, adbd. Services on the system partition can instead use policy defined transitions based on their file security context. If not specified and no transition is defined in policy, defaults to the init context.
restorecon <path> action command: Restore the file named by <path> to the security context specified in the file_contexts configuration. Not required for directories created by the init.rc as these are automatically labeled correctly by init.
restorecon_recursive <path> [<path>]* action command: Recursively restore the directory tree named by <path> to the security context specified in the file_contexts configuration. Do NOT use this with paths leading to shell-writable or app-writable directories, e.g. /data/local/tmp, /data/data or any prefix thereof. See the Checking File Labels section for further details.
setcon <securitycontext> action command: Set the current process security context to the specified string. This is typically only used from early-init to

set the init context before any other process is started (seeinit.rc example above).
setenforce 0|1 action command: Set the SELinux system-wide enforcing status. 0 is permissive (i.e. log but do not deny), 1 is enforcing.
setsebool <name> <value> action command: Set SELinux boolean <name> to <value>. <value> may be 1|true|on or 0|false|off

*SELinux Policy

10- SELinux Policy Files.

1. SELinux Policy Path.

两个路径:

Google Original： alps/external/sepolicy/XXX.te (通常不建议修改)

MTK： alps/device/mediatek/common/sepolicy/XXX.te

所有的 Policy 最终采用 Union(合并) 的方式编译在一起.

2. SELinux Policy Files. external/sepolicy/*

access_vectors, security_classes

:SE for Android classes and permissions.

initial_sids, initial_sids_contexts,

:Initial_sid

fs_use, genfs_contexts, port_contexts

: file system label

users, roles

: only user (u) and role (r) used by the policy

mls

: Contains the constraints applied to the defined classes and permissions.

global_macros, mls_macro, te_marcos

: Some macros use to policy define…

attributes

: Contains the attribute names that will be used to group type identifiers defined by the policy.

policy_capabilities

: Contains the policy capabilities enabled for the kernel policy

*.te

: Contains all type enforcement policy for process and resources

file_contexts

: contains default file contexts for setting the filesystem as standard SELinux

property_contexts

: contains default contexts to be applied to Android property services

service_contexts

: Contains default contexts for Android services

seapp_contexts

: contains information to allow domain or file contexts to be computed based on parameters

mac_permissions.xml

: contains information to allow apk install permissions and mappings to selinux domain

selinux-network.sh

:If using iptables, then the information may be configured in this file as part of the build.

3. SELinux Files in Phone

SELinux Policy 手机上关键性的配置文件包括

/sepolicy： 所有 SELinux Policy 编译后的 sepolicy 文件

/file_contexts : 系统文件以及 device 所对应的 security context
/seapp_contexts: zygote 设置 app 的 security context 的配置项
/service_contexts: service 所绑定的 security context
/property_contexts: property 所绑定的 security context
/system/etc/security/mac_permissions.xml 定义 app 的 security context 类型
/selinux_version: 对应 system property: ro.build.fingerprint
/selinux_network.sh 设置 selinux 对网络端的访问

临时性设置/data/security/current/

KK 上 SELinux 是实验性质的, 相关如 file_contexts, seapp_contexts 等都优选/data/security/current 下面的特别文件.
L 版本上, 在 android.c 中使用了特别的判定函数 set_policy_index, 此用来比较
/selinux_version
/data/security/current/selinux_version

两只文件都存在, 并且完全一致, 只有这样才会使用/data/security/current 下面的文件, 其它情况下都会使用根目录的配置文件.
selinux_version 中存储的是 build 的 BUILD_FINGERPRINT 即 system property: ro.build.fingerprint

11-Understand SEAndroid Policy

1. Security Classes and Permissions
针对系统中不同的"文件" 类型(Class), 如普通的 file, socket, 比如 SELinux 使用的 security, 比如针对每个 process 参数的 process 等定义相关的 class. 而每一个 class 都有相对应的 permissions. 比如 file 就有 read, write, create, getattr, setattr, lock, ioctl 等等. 比如 process 就有 fork, sigchld, sigkill, ptrace, getpgid, setpgid 等等.

具体在 SEAndroid 当中, 在 external/sepolicy/access_vectors 中定义相关的 class, 以及他们具有那些 Permissions. 在 security_classes 中声明这些 classes.

比如 file:
external/access_vectors
```
#
# Define a common prefix for file access vectors.
#
common file
{
 ioctl
 read
 write
 create
 getattr
 setattr
 lock
 relabelfrom
 relabelto
 append
 unlink
 link
 rename
 execute
 swapon
 quotaon
 mounton
}

==========>

class file
inherits file
{
 execute_no_trans
 entrypoint
 execmod
 open
```

audit_access
}

==========>
external/security_classes
class file

注意的是, 这些定义和 Kernel 中相关 API 是强相关的，普通用户严禁修改.

2. SE for Android Classes and Permissions
除了 Kernel 默认定义的 Class/Permissions 之外, Google 还为 SEAndroid 定义了几个 UserSpace 会使用到的 Class/Permissions, 它们是:

binder class - This is a kernel object to manage the Binder IPC service.	
Permission	**Description** (4 unique permissions)
call	Perform a binder IPC to a given target process (can A call B?).
impersonate	Perform a binder IPC on behalf of another process (can A impersonate B on an IPC?). Not currently used in policy but kernel (selinux/hooks.c) checks permission in selinux_binder_transaction call.
set_context_mgr	Register self as the Binder Context Manager aka servicemanager (global name service). Can A set the context manager to B, where normally A == B. See policy module servicemanager.te.
transfer	Transfer a binder reference to another process (can A transfer a binder reference to B?).

zygote class - This is a userspace object to manage the Android application loader. See Java SELinux.checkSELinuxAccess() inframeworks/base/core/java/com/android/internal/os/ZygoteConnection.java	
Permission	**Description** (4 unique permissions)
specifyids	Peer may specify uid's or gid's.
specifyrlimits	Peer may specify rlimits.
specifyinvokewith	Peer may specify --invoke-with to launch Zygote with a wrapper command.
specifyseinfo	Specify a seinfo string for use in determining the app security label.

property_service class - This is a userspace object to manage the Android Property Service. See check_mac_perms() in system/core/init/property_service.c	
Permission	**Description** (1 unique permission)
set	Set a property.

service_manager class - This is a userspace object to manage Android services. See check_mac_perms() in frameworks/native/cmds/servicemanager/service_manager.c	
Permission	**Description** (3 unique permission)
add	Add a service.
find	Find a service.
list	List services.

keystore_key class - This is a userspace object to manage the Android keystore (see system/security/keystore/keystore.cpp).	
Permission	**Description** (16 unique permissions)
test	Test if keystore okay.
get	Get key.
insert	Insert/update key.
delete	Delete key.
exist	Check if key exists.
saw	Search for matching string.
reset	Reset keystore.
password	Generate new keystore password.
lock	Lock keystore.
unlock	Unlock keystore.
zero	Check if keystore empty.
sign	Sign data.
verify	Verify data.
grant	Add or remove access.
duplicate	Duplicate the key.
clear_uid	Clear keys for this uid.

rd class - This is a userspace object to allow file dumps (see system/core/debuggerd/debuggerd.cpp).	
Permission	**Description** (2 unique permissions)
dump_tombstone	Write tombstone file.
dump_backtrace	Write backtrace file.

drmservice class - This is a userspace object to allow finer access control of the Digital Rights Management services (see frameworks/av/drm/drmserver/DrmManagerService.cpp).	
Permission	**Description** (8 unique permissions)
consumeRights	Consume rights for content.
setPlaybackStatus	Set the playback state.

openDecryptSession	Open the DRM session for the requested DRM plugin.
closeDecryptSession	Close DRM session.
initializeDecrypSession	Initialise the decrypt resources.
decrypt	Decrypt data stream.
finalizeDecryptUnit	Release DRM resources.
pread	Read the data stream.

3. Multi-Level Security and Multi-Category

Google 目前虽然已经定义好了 MLS/MCS 的框架，但并没有真正使用. 相关的 policy code 主要定义在 mls, 以及 mls_macros 当中.

比如:

```
# Process read operations: No read up unless trusted.
mlsconstrain process { getsched getsession getpgid getcap getattr ptrace share }
   (l1 dom l2 or t1 == mlstrustedsubject);
# Process write operations:  No write down unless trusted.
mlsconstrain process { sigkill sigstop signal setsched setpgid setcap setrlimit ptrace share }
   (l1 domby l2 or t1 == mlstrustedsubject);
```

但目前 mls_num_sens=1, mls_num_cats=1024; 所有 process 的 level 都是 0 并且, 所有的 process 都还没有真正使用 level 属性, 所以 MLS/MCS 目前我们可以不关注.

4. Attribute and Type

这个是 SELinux 的重点之一, 每一个文件有唯一的 Security Context, 而目前版本 type 是 Security Context 中最重要的栏目. 即可以说，每一个文件对应一个 type, 而每一个 type 都对应有一个或几个 Attribute.

所有常见的 Attribute 定义在: external/attrubites 下面, 如；

```
# All types used for devices.
attribute dev_type;
# All types used for processes.
attribute domain;
# All types used for filesystems.
attribute fs_type;
# All types used for context= mounts.
attribute contextmount_type;
```

Type 的定义就比较分散, 主要有:

* 普通文件 type 定义在 file.te 如

```
type labeledfs, fs_type;
type pipefs, fs_type;
type sockfs, fs_type;
type rootfs, fs_type;
type proc, fs_type;

.....
# File types
type unlabeled, file_type;
# Default type for anything under /system.
type system_file, file_type;
# Default type for anything under /data.
type system_data_file, file_type, data_file_type;
# /data/.layout_version or other installd-created files that
# are created in a system_data_file directory.
type install_data_file, file_type, data_file_type;
```

* 设备文件 type 定义在 device.te 如；

```
# Device types
type device, dev_type, fs_type;
type alarm_device, dev_type, mlstrustedobject;
type adb_device, dev_type;
type ashmem_device, dev_type, mlstrustedobject;
```

type audio_device, dev_type;
type binder_device, dev_type, mlstrustedobject;
type block_device, dev_type;
type camera_device, dev_type;
type dm_device, dev_type;

* 执行档文件, 如果需要 domain 切换, 通常直接定义在以这个执行档文件命令的扩展名为 te 的文件中, 如 vold.te 会定义执行档类型, 以及这个 process 的 domain.

```
# volume manager
type vold, domain;
type vold_exec, exec_type, file_type;
init_daemon_domain(vold)
```

5. SELinux User and Role.

Google 使用了下面的两个语句来定义 user 和 role

```
user u roles { r } level s0 range s0 - mls_systemhigh;    #(s0 - s0:c0.c1023)
role r;
role r types domain;
```

即: android 还只定义了一个 user u, 一个 role r , 以及 file system 使用 object_r; user u 只有一个 role r; mls 的 low_level 是 s0, high level 是 s0:c0.c1023; android 因为只有一个 role r 所以并没有进行角色访问约束(RBAC).

6. File system labels

所有 Process 都使用 domain attribute, 对于系统中的文件, 则不同，默认是根据不同的文件系统类型来定义。

对于存储型的 file system. 比如 ext2/3/4, yaffs2 等支持 fs_use_xattr, fs_use_task or fs_use_trans 操作, 默认标签是 labelfs, 参考 fs_use

```
# Label inodes via getxattr.
fs_use_xattr yaffs2 u:object_r:labeledfs:s0;
fs_use_xattr jffs2 u:object_r:labeledfs:s0;
fs_use_xattr ext2 u:object_r:labeledfs:s0;
fs_use_xattr ext3 u:object_r:labeledfs:s0;
fs_use_xattr ext4 u:object_r:labeledfs:s0;
fs_use_xattr xfs u:object_r:labeledfs:s0;
fs_use_xattr btrfs u:object_r:labeledfs:s0;
fs_use_xattr f2fs u:object_r:labeledfs:s0;
```

对于非真实存储型的 file system 并且无法支持 fs_use_xattr, fs_use_task or fs_use_trans, 比如 proc 此类需要使用 genfscon 来定义, 参考 genfs_contexts, 如

```
# selinuxfs booleans can be individually labeled.
genfscon selinuxfs / u:object_r:selinuxfs:s0
genfscon cgroup / u:object_r:cgroup:s0
# sysfs labels can be set by userspace.
genfscon sysfs / u:object_r:sysfs:s0
genfscon inotifyfs / u:object_r:inotify:s0
genfscon vfat / u:object_r:vfat:s0
genfscon debugfs / u:object_r:debugfs:s0
genfscon fuse / u:object_r:fuse:s0
genfscon pstore / u:object_r:pstorefs:s0
genfscon functionfs / u:object_r:functionfs:s0
genfscon usbfs / u:object_r:usbfs:s0
```

7. File labels.

目前统一全部定义在 file_contexts 当中. 这个文件也是我们会经常会修改的, 比如:

```
#############################
# Devices
#
/dev(/.*)?  u:object_r:device:s0
/dev/akm8973.*  u:object_r:sensors_device:s0
/dev/accelerometer u:object_r:sensors_device:s0
/dev/adf[0-9]*  u:object_r:graphics_device:s0
/dev/adf-interface[0-9]*\.[0-9]* u:object_r:graphics_device:s0
/dev/adf-overlay-engine[0-9]*\.[0-9]* u:object_r:graphics_device:s0
/dev/alarm  u:object_r:alarm_device:s0
/dev/android_adb.* u:object_r:adb_device:s0
```

```
##############################
# System files
#
/system(/.*)?  u:object_r:system_file:s0
/system/bin/sh  -- u:object_r:shell_exec:s0
/system/bin/run-as -- u:object_r:runas_exec:s0
/system/bin/bootanimation u:object_r:bootanim_exec:s0
/system/bin/app_process32 u:object_r:zygote_exec:s0
/system/bin/app_process64 u:object_r:zygote_exec:s0
/system/bin/servicemanager u:object_r:servicemanager_exec:s0
/system/bin/surfaceflinger u:object_r:surfaceflinger_exec:s0

##############################
# Data files
#
/data(/.*)?  u:object_r:system_data_file:s0
/data/.layout_version  u:object_r:install_data_file:s0
/data/backup(/.*)?  u:object_r:backup_data_file:s0
/data/secure/backup(/.*)? u:object_r:backup_data_file:s0
/data/security(/.*)? u:object_r:security_file:s0
/data/system/ndebugsocket u:object_r:system_ndebug_socket:s0
/data/drm(/.*)?  u:object_r:drm_data_file:s0

##############################################
# Root
/   u:object_r:rootfs:s0
# Data files
/adb_keys  u:object_r:adb_keys_file:s0
/default\.prop  u:object_r:rootfs:s0
/fstab\..*  u:object_r:rootfs:s0
/init\..*  u:object_r:rootfs:s0
/res(/.*)?  u:object_r:rootfs:s0
/ueventd\..*  u:object_r:rootfs:s0
```

8. Type Enforce

类型审查, 通常涉及到:

* domain 类型定义
* 执行档 入口定义
* SELinux 模式设置
* domain 切换
* 类型访问规则

下面我们就以 mediaserver 这个进程的定义来解析, 对应文件是 mediaserver.te

* domain 类型定义

```
type mediaserver, domain;
```

* 执行档 入口定义

```
type mediaserver_exec, exec_type, file_type;
```

* SELinux 模式设置

在 L 版本上，默认都是 enforcing mode , Google 严禁 process 在正式版本中使用 permissive mode. 设置成 permissive mode 通常都是为了 debug 需要, 在 debug 完成后都需要设置成 enforcing mode.

* domain 切换

```
init_daemon_domain(mediaserver)
```

这个是一个复杂的宏, 简单来说就是当 init fork 子进程执行 mediaserver_exec 这个类型的执行档时, 其 domain 从 init 切换到 mediaserver.

具体这个宏展开是:

```
########################################
# init_daemon_domain(domain)
# Set up a transition from init to the daemon domain
# upon executing its binary.
```

```
define(`init_daemon_domain', `
domain_auto_trans(init, $1_exec, $1)
tmpfs_domain($1)
')
======>
########################################
# domain_auto_trans(olddomain, type, newdomain)
# Automatically transition from olddomain to newdomain
# upon executing a file labeled with type.
#
define(`domain_auto_trans', `
# Allow the necessary permissions.
domain_trans($1,$2,$3)
# Make the transition occur by default.
type_transition $1 $2:process $3;
')
======>
define(`domain_trans', `
# Old domain may exec the file and transition to the new domain.
allow $1 $2:file { getattr open read execute };
allow $1 $3:process transition;
# New domain is entered by executing the file.
allow $3 $2:file { entrypoint open read execute getattr };
# New domain can send SIGCHLD to its caller.
allow $3 $1:process sigchld;
# Enable AT_SECURE, i.e. libc secure mode.
dontaudit $1 $3:process noatsecure;
# XXX dontaudit candidate but requires further study.
allow $1 $3:process { siginh rlimitinh };
')
```

此类的宏还有很多, 可以参考 te_macros

* 类型访问规则

```
#允许 mediaserver 访问 init 的 property socket
unix_socket_connect(mediaserver, property, init)
#允许 mediaserver 读取 sdcard_type 类型的目录和文件
r_dir_file(mediaserver, sdcard_type)
#允许 mediaserver 使用 binder 服务 和发起 binder call
binder_use(mediaserver)
binder_call(mediaserver, binderservicedomain)
binder_call(mediaserver, appdomain)
binder_service(mediaserver)
# 允许 mediaserver 做其他的一些操作
allow mediaserver self:process execmem;
allow mediaserver kernel:system module_request;
allow mediaserver media_data_file:dir create_dir_perms;
allow mediaserver media_data_file:file create_file_perms;
allow mediaserver app_data_file:dir search;
allow mediaserver app_data_file:file rw_file_perms;
allow mediaserver sdcard_type:file write;
allow mediaserver gpu_device:chr_file rw_file_perms;
allow mediaserver video_device:dir r_dir_perms;
.....
```

12-SELinux Policy limited by Google

1. 维持 external/sepolicy 与 Google AOSP 一致, 尽量不要修改. 特别是不要去除任何的 neverallow 语句.

2. 严禁修改 untrusted_app.te, 放开 untrusted_app 的权限.

3. 系统关键进程启动长时间运行的 process, 必须进行 domain 切换. 比如 netd, installd, vold, zygote 等.

4. Native thread 严禁使用 kernel 标签, 而 kernel thread 除 init 外, 都应当是 kernel 标签.

5. 在 user build 中不能存在如 su, recovery, init_shell 标签的进程.

6. 在 user build 中所有的 process 都必须是 Enforce Mode

* SELinux Policy Demo

13-Add new service started by init

- 情景: 定义一个 init 启动的 service, demo_service, 对应的执行档是 /system/bin/demo.

- (1). 创建一个 demo.te 在/device/mediatke/common/sepolicy 目录下, 然后在/device/mediatke/common/BoardConfig.mk 的 BOARD_SEPOLICY_UNION 宏中新增 demo.te
- (2). 定义 demo 类型，init 启动 service 时类型转换, demo.te 中
  - type demo, domain;
  - type demo_exec, exec_type, file_type;
  - init_daemon_domain(demo)
- (3). 绑定执行档 file_contexts 类型
  - /system/bin/demo u:object_r:demo_exec:s0
- (4). 根据 demo 需要访问的文件以及设备, 定义其它的权限在 demo.te 中.

14-Set System Property

- 情景: 一个 native service demo, 需要设置一个自定义的 system property, demo.setting.case1.
- (1). 定义 system property 类型. 在 property.te
  - type demo_prop, property_type;
- (2). 绑定 system property 类型. 在 property_contexts
  - demo. u:object_r:demo_prop:s0
- (3). 在 demo.te 中新增
  - unix_socket_connect(demo,property,init);
  - allow demo demo_prop:property_service set;

15- Use Binder and Add System Service

- 情景: 一个 native service demo, 创建了一个 binder service demo, 并对外提供服务.

- (1). 在 service.te 中定义 service 类型
  - type demo_service service_manager_type;
- (2). 在 service_contexts 中绑定 service
  - demo u:object_r:demo_service:s0
- (3). 在 demo 中声明 binder 权限 demo.te
  - binder_use(demo)

•binder_call(demo, binderservicedomain)
•biner_service(demo)
▪(4). 允许 demo 这个进程添加 demo 这个 service, 在 demo.te
•allow demo demo_service:service_manager add;

16- Access system device
▪情景: 一个 native service 需要访问一个专属的 char device /dev/demo
▪(1). 定义 device 类型, device.te
•type demo_device dev_type;
▪(2). 绑定 demo device, file_contexts
•/dev/demo u:object_r:demo_device:s0
▪(3). 声明 demo 使用 demo device 的权限 demo.te
• allow demo demo_device:chr_file rw_file_perms;

17-Use Local Socket

▪情景: 一个 native service 通过 init 创建一个 socket 并绑定在 /dev/socket/demo, 并且允许某些 process 访问.
▪(1). 定义 socket 的类型 file.te
•type demo_socket, file_type;
▪(2). 绑定 socket 的类型 file_contexts
•/dev/socket/demo_socket u:object_r:demo_socket:s0
▪(3). 允许所有的 process 访问.
•# allow app connectto & write
•unix_socket_connect(appdomain, demo, demo)
•# more..

*Quickly Debug

18-**如何问题是否与** SELinux **相关？**

1. **将** SELinux **调整到** Permissive **模式测试.**
将 SELinux 模式调整到 Permissive 模式，然后再测试确认是否与 SELinux 约束相关.
ENG 版本:
adb shell setenforce 0
如果还能复现问题，则与 SELinux 无关, 如果原本很容易复现, 而 Permissive mode 不能再复现, 那么就可能关系比较大.

2. **查看** LOG **中是否有标准的** SELinux Policy Exception.
在 Kernel LOG / Main Log 中查询关键字 "avc:" 看看是否有 SELinux Policy Exception, 并进一步确认这个异常是否与当时的逻辑相关.

19-**如何设置确认** SELinux **模式？**

1. **如何确认** SELinux **模式**？
在 ENG/USER 版本中，都可以使用 getenforce 命令进行查询，如：
```
root@mt6589_phone_720pv2:/ # getenforce
getenforce
Enforcing
```

2. **如何设置** SELinux **模式**？
在 ENG/USERDEBUG 版本中, 可以使用 setenforce 命令进行设置:
```
adb shell setenforce 0    //设置成 permissive 模式
adb shell setenforce 1    //设置成 enforce 模式
```
在 USER 版本中无法设置.

3. **如何开机设置** SELinux **模式**？
3.1 更新配置
bootable/bootloader/lk/platform/mt6xxx/rules.mk
```
    # choose one of following value  -> 1: disabled/ 2: permissive /3: enforcing
     SELINUX_STATUS := 3
```
可直接调整这个 SELINUX_STATUS 这个的值为 2 或者 1

3.2 允许 USER 版本 disable SELinux (如果你需要在 USER Build 上开启)
修改 system/core/init/Android.mk 新增
```
    ifeq ($(strip $(TARGET_BUILD_VARIANT)),user)
    LOCAL_CFLAGS += -DALLOW_DISABLE_SELINUX=1
    Endif
```

20-**如何分析** SELinux Policy Exception

1. SELinux Policy Exception Log 格式.
SELinux Policy Exception 的 LOG 关键字是 "avc: denied" 或者 "avc: denied", 下面就是一句典型的 LOG
```
<5>[   27.706805] (3)[304:logd.auditd]type=1400 audit(1420041991.220:17): avc: denied { execute } for pid=2182
comm="app_process" path="/data/dalvik-cache/arm64/system@framework@boot.oat" dev="mmcblk0p18" ino=15109
scontext=u:r:root_channel:s0 tcontext=u:object_r:dalvikcache_data_file:s0 tclass=file permissive=0
```

具体说明如下:
```
<5> : kernel log level
[   27.706805] : kernel time
(3) : cpu number
[304:logd.auditd] : 表示此 LOG 是通过 auditd 打印的.
type=1400  : SYSCALL
  type=AVC - for kernel events
  type=USER_AVC - for user-space object manager events
audit(1420041991.220:17) : audit(time:serial_number)
avc: denied { execute }:  field depend on what type of event is being audited.
pid=2182 comm="app_process"  : If a task, then log the process id (pid) and the name of the executable file (comm).
path="/data/dalvik-cache/arm64/system@framework@boot.oat" dev="mmcblk0p18" ino=15109: The information of target.
subject context : u:r:root_channel:s0
target context : u:object_r:dalvikcache_data_file:s0
tclass : the object class of the target class=system
permissive:  permissive (1) or enforcing (0)
```

2. 基本分析流程
1). 将 SELinux 调到 permissive mode, 然后一次性抓取出所有的"avc:" 的 LOG.
 * 如果问题与开机流程无关, 并且 eng 版本能够复现.
  adb shell setenforce 0
 * 如果问题与开机流程相关, 或者只有 user build 能够复现. 按下列 FAQ 设置 permissive mode
2). 确认访问是否是必须的？是否是违法恶意访问?
3). 如果是正常访问，确认访问的目标类型是否太过广泛，如果太过广泛，细化具体的文件目标。
4). 添加对应的 SELinux Policy 到对应的 Policy 文件。
5). 如果是违法恶意访问, 追查上层 feature owner 的责任。

21 **快速验证** SELinux Policy **问题**

1. **快速编译测试**
在已经编译过的版本上, 首先编译出新的 selinux policy, 然后打包 boot image.
KK: ./mk project_name mm external/sepolicy
./mk project_name bootimage

L:
mmm external/sepolicy
make -j24 ramdisk-nodeps
make -j24 bootimage-nodeps

然后再重新刷 bootimage 测试.

因为目前 L 版本默认使用加密, 直接用/data/security/current 中的文件替换根目录下的文件，可能直接受影响，目前不推荐使用.

特别注意点: 如果更新了 file_contexts, 给 system 下面的文件重新设置了 SELinux Label, 那么你需要将 system image 重新打包, 不然不会起到作用.
make -j24 snod

22-如何处理与 Google 定义 neverallow 冲突

1. **总体原则**
当我们修改或者增加一些 SELinux Policy 后, 发现无法编译通过，或者与 CTS 测试相冲突, 此时我们需要如何处理呢。

Google 在 external/sepolicy 中有使用相关 neverallow 规则, 对 SELinux Policy 的更新进行了限制, 以防止开发者过度开放权限，从而引发安全问题。并且通过 CTS Test 检测开发者是否有违法相关的规则.

为此，我们要求开发者，严格遵守 Google 定义的规则，具体即是:
* 严禁直接修改 external/sepolicy 下面的相关 SELinux Policy, 使之保持与 Google 一致, 特别是严禁直接删除或者修改 Google 定义的 neverallow 规则.
* 遇到与 Google 定义相违背之处, 只能绕道, 或者修改设计.

下面，我们就一些典型问题进行说明.

2. CTS fail of android.security.cts.SELinuxDomainTest # testInitDomain
说明: Google 要求只有一个 init domain, init 启动任何 service 都要求进行 domian 切换.
解决方式: 请开发者为新建的 service 新增 process domain, 并设置相关权限
http://online.mediatek.com/_layouts/15/mol/topic/ext/Topic.aspx?id=158

3. Android L APP **如何获取** sys **或者** proc file system **中节点的写权限**
说明: Google 默认禁止 app , 包括 system app, radio app 等直接写/sys 目录以 u:object_r:sysfs:s0 为标签的文件以及/proc 目录以 u:object_r:proc:s0 文件, 认为这个是有安全风险的。如果直接放开 SELinux 权限, 会导致 CTS 无法通过.

解决方式:
通常遇到此类情况，你有两种做法:
(1). 通过 system server service 或者 init 启动的 service 读写操作, 然后 app 通过 binder/socket 等方式连接 APP 访问. 此类安全可靠, 并且可以在 service 中做相关的安全审查, 推崇这种方法.

(2). 修改对应节点的 SELinux Security Label, 为特定的 APP, 如 system app, radio, bluetooth 等内置 APP 开启权限, 但严禁为 untrsted app 开启权限. 具体的做法下面以 system app 控制/sys/class/leds/lcd-backlight/brightness 来说明.

* 在 device/mediatek/common/sepolicy/file.te 定义 brightness SELinux type
type sys_lcd_brightness_file, fs_type,sysfs_type;

* 在 device/mediatek/common/sepolicy/file_contexts 绑定 brightness 对应的 label, 注意对应的节点是实际节点，而不是链接.
/sys/devices/platform/leds-mt65xx/leds/lcd-backlight/brightness u:object_r:sys_lcd_brightness_file:s0

* 在 device/mediatek/common/sepolicy/system_app.te 中申请权限.
allow system_app sys_lcd_brightness_file:file rw_file_perms;

* 为其它的 process 申请相关的权限，如 system_server, 在 device/mediatek/common/sepolicy/system_server.te
allow system_server sys_lcd_brightness_file:file rw_file_perms;

原则上我们都推崇使用第一种方式处理.

4. Process 无法访问某个新增 device
说明: Google 默认禁止除 unconfineddomain 的进程以及 ueventd 之外的进程直接访问某个普通定义的 device, 所谓普通定义即 device 对应的 SELinux Label 是 u:object_r:device:s0.
解决方式: 需要为新增的 device 定义具体的 Label, 然后再给对应的 process 开启相关的权限.
http://online.mediatek.com/_layouts/15/mol/topic/ext/Topic.aspx?id=158

5. Native Process 运行 java 程序
说明: 以往我们在上下层通讯中, 比如某个 native process/service 通常会借用如 am 命令, 向 ActivityManagerService 发送 broadcast, 或者 start activity 等. 通知上层某个事件已经发生了. 在 android 5.0 以后， Google 严禁非 app 以及几个特别的 domain 的程序执行非 rootfs or /system 分区的文件, 具体定义如:
neverallow { domain -appdomain -dumpstate -shell -su -system_server -zygote } { file_type -system_file -exec_type }:file execute;

解决方式:
(第一种方法).借用 am 命令，实质上还是通过获取 AMS 的 binder 引用，再通过 binder 向 AMS 发送命令, 同样的你可以直接 binder 来操作, 代表案例有: mediaserver 中使用的/frameworks/av/media/libmediaplayerservice/ActivityManager.cpp. 先获取 servicemanager, 然后再获取 activity binder 接口, 然后再封装 parcel 后发送相关的命令给 AMS.
具体对应的 function cmd 以及 parcel 的封装格式可以参考:
frameworks/base/core/java/android/app/ActivityManagerNative.java.
注意的是如果这个 process 没有申请 binder 权限, 则需要先申请 binder 权限, 可以使用 binder_use, binder_call, binder_service

(第二种方法). 将这个 process 纳入 appdomain, 但这个带来的影响是, 这个 process 的权限会受到更大的限制, 因为 Google 对 appdomain 有非常严格的限制. 比如某个 process 是 demo
device/mediatek/common/sepolicy/demo.te 里面新增
app_domain(demo)

6. system app 读写 nvram 操作
说明: 以往我们可以通过 jni 直接读写 nvram, 目前 Google 已经严禁 app 去读写系统块设备文件.
解决方式： 参考 FAQ, 已经做好了 nvram_agent_binder 这样一个服务进程，贵司使用时就是去访问它这个 service 即可.
请参考:FAQ ID: FAQ04542 NvRAM】APK（应用层）读写 NvRAM

7. UEventObserver 在 APP 中使用.
说明:
(1). Google 禁止 UEventObserver API 在第三方 APP 使用，即任何不可信任的 APP 都禁止使用此 API.
(2). 如果是系统 APP，比如 phone app, settings 等使用，那么就要根据 app 使用的 domain, 添加相关的 SELinux 权限.
解决方式: 确保你的 APP 是系统的 APP, 非 untrusted app.
如针对 settings app, 在 device/mediatek/common/sepolicy/system_app.te 里面新增
allow system_app self:netlink_kobject_uevent_socket { create ioctl read getattr setattr bind connect getopt setopt shutdown };

*SELinux vs ROOT

23- ADB root permission

1. adb root permission
* ENG Build/userdebug Build 不受影响，SELinux 给 adb 使用 u:r:su:s0 的 permissive mode label.
** adb root uid/capabilities.
** su permissive SELinux mode

* User Build 维持 u:r:adbd:s0 和 当使用 shell 时, 切换到 u:r:shell:s0 的 label. 权限受到严格限制.
** Root ===> shell
** No any capabilities. Retain CAP_SETUID/CATP_SETGID boundset bits for run-as
** shell use SELinux Enforcing mode

2. adb root in User Build
尽可能的保持 User Build 的完整性. 只开启 adb 完整 root 权限.
(1). alps/build/core/main.mk, 设置 system property ro.secure = 0
ifneq (,$(user_variant))

Target is secure in user builds.
ADDITIONAL_DEFAULT_PROPERTIES += ro.secure=1
将 ADDITIONAL_DEFAULT_PROPERTIES += ro.secure=1 改成
ADDITIONAL_DEFAULT_PROPERTIES += ro.secure=0

(2). alps/system/core/adb/android.mk 放开权限控制宏. ALLOW_ADBD_ROOT, 如果没有打开这个选项，那么 adb.c 中将不会根据 ro.secure 去选择 root 还是 shell 权限，直接返回 shell 权限。因此您必须需要 Android.mk 中的第 124 行：
ifneq (,$(filter userdebug eng,$(TARGET_BUILD_VARIANT)))
===>
ifneq (,$(filter userdebug user eng,$(TARGET_BUILD_VARIANT)))

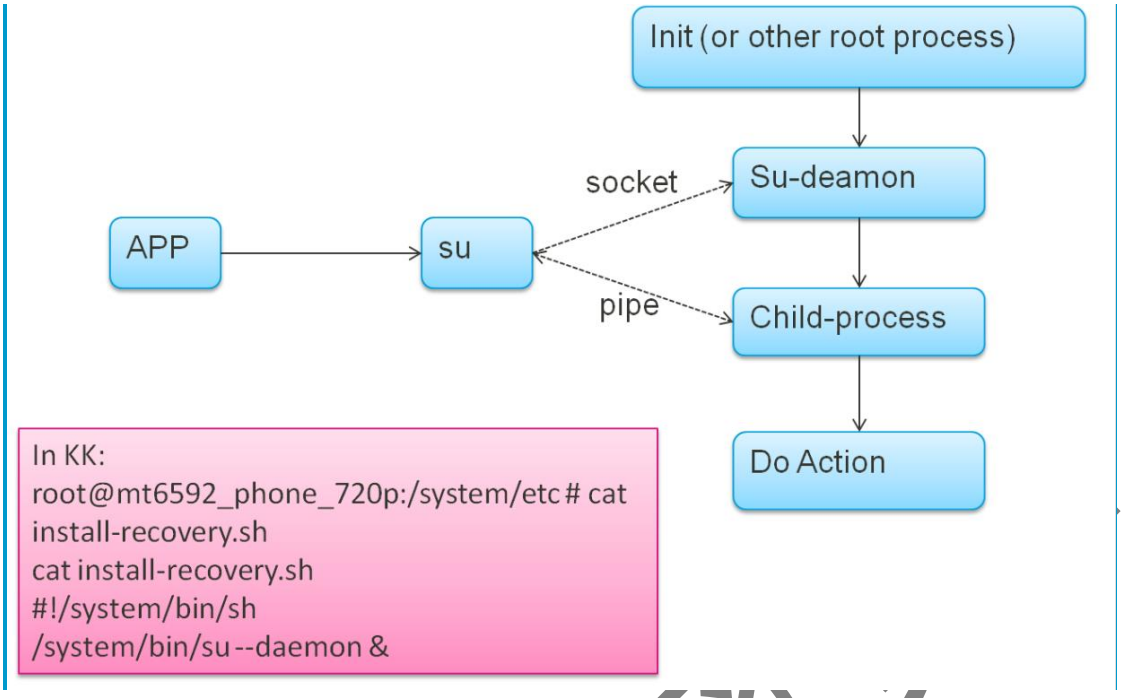
(3). 放开 SELinux 的限制. 更新 alps/external/sepolicy/Android.mk 115 行，将 su label 默认编译进入 sepolicy.
include $(BUILD_SYSTEM)/base_rules.mk
sepolicy_policy.conf := $(intermediates)/policy.conf
$(sepolicy_policy.conf): PRIVATE_MLS_SENS := $(MLS_SENS)
$(sepolicy_policy.conf): PRIVATE_MLS_CATS := $(MLS_CATS)
$(sepolicy_policy.conf) : $(call build_policy, $(sepolicy_build_files))
@mkdir -p $(dir $@)
$(hide) m4 -D mls_num_sens=$(PRIVATE_MLS_SENS) -D mls_num_cats=$(PRIVATE_MLS_CATS) \
-D target_build_variant=$(TARGET_BUILD_VARIANT) \
-D force_permissive_to_unconfined=$(FORCE_PERMISSIVE_TO_UNCONFINED) \
-s $^ > $@
$(hide) sed '/dontaudit/d' $@ > $@.dontaudit
将 -D target_build_variant=$(TARGET_BUILD_VARIANT) \ 改成
-D target_build_variant=eng \

3. adb root and permissive SELinux Mode
目标: adb 默认 root 权限, 并且将 SELinux 调整到 permissive mode.
在 build 时导入特别的参数:
MTK_BUILD_ROOT=yes
I.E
MTK_BUILD_ROOT=yes make –j24 2>&1 | tee build.log
重新 download lk, bootimage

24-APP root permission

1. app root permission
* 从 KK 开始 Google 不遗余力的禁止 APP 使用 root 权限, 通常如执行 su.
** 去除 app 的 root capabilities of boundset.
com_android_internal_os_zygote.cpp drop capabilities bounding set
** 去除 app 的执行 SUID/SGID 文件权限.
app_main.cpp prctl set no_new_privs = 1
* L 版本进一步从 SELinux 上严禁执行
** User build 上无 su label, app label 无法切换到 su.
** SELinux Poicy 严格限制 app 的访问行为.

2. 目前 ROOT 后门通用做法.
因为 app 直接 fork SU 的做法, Google 已经限制死, 目前通用做法换成了使用原本具有 root 权限的进行 fork 出 SU, 然后 app 再使用 SU Client 通过 socket 连接到 SU Server 的做法来达成. 常见的都是利用 init 来达成, 第一个 init 具有强大的 root 权限，第二个就是 init 非常好扩展。如下图所示。



3. L 规避 SELinux

L 版本上, 在 android.c 中使用了特别的判定函数 set_policy_index, 此用来比较

/selinux_version

/data/security/current/selinux_version

两只文件是否都存在, 并且完全一致, 只有这样才会使用/data/security/current 下面的文件, 其它情况下都会使用根目录的配置文件.

selinux_version 中存储的是 build 的 BUILD_FINGERPRINT 即 system property：ro.build.fingerprint

*Reference

25- MTK Online FAQ List

- [FAQ11484] [SELinux] 如何设置确认 selinux 模式
  - https://online.mediatek.com/Pages/FAQ.aspx?List=SW&FAQID=FAQ11484
- [FAQ11486] [SELinux Policy] 如何设置 SELinux 策略规则 ? 在 Kernel Log 中出现"avc: denied" 要如何处理?
  - https://online.mediatek.com/Pages/FAQ.aspx?List=SW&FAQID=FAQ11486
- [FAQ11485] [SELinux Debug]权限(Permission denied)问题如何确认是 Selinux 约束引起?

- https://online.mediatek.com/Pages/FAQ.aspx?List=SW&FAQID=FAQ11485

- [FAQ11483] 如何快速 Debug SELinux Policy 问题
  - https://online.mediatek.com/Pages/FAQ.aspx?List=SW&FAQID=FAQ11483
- [FAQ11414] android KK 4.4 版本后，user 版本 su 权限严重被限制问题说明
  - https://online.mediatek.com/Pages/FAQ.aspx?List=SW&FAQID=FAQ11414

26 Internet Links

- SELinux
  - http://docs.fedoraproject.org/en-US/Fedora/13/html/Security-Enhanced_Linux/chap-Security-Enhanced_Linux-SELinux_Contexts.html
  - http://www.centos.org/docs/5/html/Deployment_Guide-en-US/ch-selinux.html
  - http://debian-handbook.info/browse/stable/sect.selinux.html
  - http://selinuxproject.org/page/AVCRules
  - http://selinuxproject.org/page/ObjectClassesPerms#common_file
- Android security mechanism
  - http://source.android.com/tech/security/#android-application-security
  - Asaf Shabtai, Yuval Fledel, Uri Kanonov, Yuval Elovici, Shlomi Dolev, Chanan Glezer: Google Android: A Comprehensive Security Assessment. IEEE Security & Privacy 8(2): 35-44 (2010)
  - W. Enck, M. Ongtang, and P. McDaniel. Understanding Android security. IEEE Security and Privacy, 7, 2009.
- SE Android
  - http://selinuxproject.org/page/SEAndroid
  - A. Shabtai, Y. Fledel, and Y. Elovici. Securing Android-Powered mobiledevices using SELinux. IEEE Security and Privacy, 2009.