

SELinux / SE for Android Tutorial

Stephen Smalley

Laboratory for Advanced Cybersecurity Research (LACR), NSA



Security-Enhanced Linux (SELinux)

- SELinux, originally created and still actively maintained by LACR, provides flexible, fine-grained mandatory access control for Linux.
- It was NSA's first open-source software release in 2000 and has become a force multiplier through community adoption and partnerships.
- Default-enabled in Fedora, openSUSE, and all downstream distros: CentOS Stream, Red Hat, Oracle, Rocky, Alma, Amazon (≥ 2023), Azure (≥ 3.0), SUSE (≥ 16) and more.
- Today it is running on millions of Linux servers and is still breaking new ground in cloud-native Linux distros.

What is SELinux?

- Mandatory Access Control (MAC) for Linux
 - Enforces an administratively-defined security policy.
 - Over all processes, objects, and operations.
 - Based on security labels / contexts.
- Can confine services and apps.
 - Even services that run as “root” / superuser / uid-0.
 - Protect from misuse, contain damage.

How is SELinux used?

- Sandbox applications/services/containers/virtual machines
 - Most commonly network-facing services, containers, and VMs
- Enforce least privilege
 - General goal of most security policies
 - Even if an attacker can gain control, limit what they can do
- Protect a program from the rest of the system
- Create assured pipelines

Security Enhancements (SE) for Android

- SE (for) Android was LACR's open-source reference implementation for applying SELinux to Android to close critical Android security gaps.
- It demonstrated effective use of SELinux in Android and is now fully integrated into mainline Android.
- Mandatory for all Android devices running 5.0 or later, supported by Google's Android security team and various Android device OEMs.
- Today it is fully enforcing on more than 3.8 billion Android devices (99.9% of all active devices, everything running Android 5.0 or later).

SELinux: Security Models

- Implements four Mandatory Access Control (MAC) models:
 - User-Based Access Control (UBAC)
 - Role-Based Access Control (RBAC)
 - Type Enforcement (TE)
 - Multi-Level Security (MLS) or Multi-Category Security (MCS)
- UBAC, RBAC, and MLS/MCS are effectively optional.
- TE is the Lampson access matrix over “domains” and “types” – security equivalence classes for processes and objects (e.g. files, sockets).
- All models are configuration-driven.

SELinux: Labeling

- Each process and object is labeled with a security context.
 - user:role:**type**[:lowsens[:*lowcats*][-highsens[:*highcats*]]]
- Process types are also called domains.
- Domains and types are security equivalence classes.
 - Same domain/type => same permissions.
- Visible via `id -Z`, `ps -Z` (or `cat /proc/pid/attr/current`), `ls -Z` (or `getfattr -n security.selinux /path/to/file`), ...

Android Process Labeling Examples

1. `u:r:init:s0`
2. `u:r:system_server:s0`
3. `u:r:system_app:s0`
4. `u:r:untrusted_app:s0:c0,c256,c512,c768`
5. `u:r:untrusted_app_25:s0:c512,c768`
6. `u:r:isolated_app:s0:c512,c768`
7. `u:r:ephemeral_app:s0:c0,c256,c512,c768`
8. `u:r:priv_app:s0:c512,c768`
9. `u:r:gmscore_app:s0:c512,c768`

Android File Labeling Examples

1. u:object_r:init_exec:s0
2. u:object_r:system_file:s0
3. u:object_r:system_data_file:s0
4. u:object_r:system_app_data_file:s0
5. u:object_r:app_data_file:s0:c0,c256,c512,c768
6. u:object_r:app_data_file:s0:c512,c768

Linux Process Labeling Examples

1. system_u:system_r:**init_t**:s0
2. system_u:system_r:**auditd_t**:s0
3. system_u:system_r:**system_dbusd_t**:s0-s0:c0.c1023
4. system_u:system_r:**httpd_t**:s0
5. system_u:system_r:**unconfined_service_t**:s0
6. unconfined_u:unconfined_r:**unconfined_t**:s0-s0:c0.c1023
7. user_u:user_r:**user_t**:s0-s0:c0.c1023
8. staff_u:staff_r:**staff_t**:s0-s0:c0.c1023
9. sysadm_u:sysadm_r:**sysadm_t**:s0-s0:c0.c1023

Linux File Labeling Examples

1. system_u:object_r:init_exec_t:s0
2. system_u:object_r:auditd_exec_t:s0
3. system_u:object_r:dbusd_exec_t:s0
4. system_u:object_r:httpd_exec_t:s0
5. system_u:object_r:bin_t:s0
6. system_u:object_r:etc_t:s0
7. system_u:object_r:shadow_t:s0

SELinux: Policy

- The security policy configuration defines rules governing:
 - how to label processes and objects, and
 - how processes can interact with each other and with objects.
- No processes are exempt from the policy:
 - Not overridden by uid-0 or Linux capabilities.
 - Deny by default; must explicitly allow all actions.

SELinux: Rule Abstractions

- Primarily written in terms of domains, types, **attributes**, **object classes**, and **permissions**. Domains and types already defined.
- **Attribute**: A named set of domains or types, used to express the entire set in a single rule.
- **Object Class**: The kind of target object being acted upon. Separate class for each kind of file or socket.
- **Permission**: Per object-class permission checked for a given operation.
- Permissions for the same class are grouped into **Access Vectors** and cached by the **Access Vector Cache** (AVC).

SELinux: Example Attributes

Attribute Name	Description
domain	All domain types
file_type	All file types
exec_type	All executable file types
coredomain	All core domains (Android)
appdomain	All app domains (Android)
daemon	All daemon domains (Linux)
mlstrustedsubject	MLS-trusted domains (Android)
mcs_constrained_type	MCS-constrained domains (Linux)

SELinux: Example Object Classes

Class Name	Description
process	A process
file	A regular file
dir	A directory
chr_file	A character device e.g. tty
blk_file	A block device e.g. disk
tcp_socket	A TCP socket
udp_socket	A UDP socket
capability	A Linux capability (superuser privilege)

SELinux: Example Permissions

File Permissions

create

read

write

ioctl

execute

unlink

rename

link

Socket Permissions

create

read

write

ioctl

connect

listen

accept

bind

SELinux: Kinds of Rules

- **Allow:** allow *domain(s) permission(s)* for *object class(es)* to *type(s)*.
- **Auditallow:** audit (log) whenever *permission(s)* for *object class(es)* are allowed between *domain(s)* and *type(s)*.
- **Dontaudit:** do not audit (log) whenever *permission(s)* for *object class(es)* are denied between *domain(s)* and *type(s)*.
- **Neverallow:** throw an assertion if the policy ever allows *domain(s) permission(s)* for *object class(es)* to *type(s)*.
- **Constrain:** *boolean expression* on (*user, role, type, or level*) that must be true for *permission(s)* to *object class(es)* to be allowed.
- **Transition:** default-label new processes or objects of *object class(es)* with *type* when created by *domain(s)*.
- **Object Context:** label *object(s)* with *context(s)* – per kind of object.

SELinux: xperms Rules

- xperms = eXtended permissions
- Support finer grained distinctions within a SELinux operation
- Introduced by Android security team for ioctl(2) allow-listing
 - per least-significant-16-bits of ioctl(2) command value
- Backward-compatibility for existing ioctl allow rules
- Later extended for netlink message (nlmsg) allow-listing
 - Requires kernel 6.13+, netlink_xperm policy capability
 - per 16-bit nlmsg_type value
- allowxperm, auditallowxperm, dontauditxperm, neverallowxperm

Android Example Rules

1. `type_transition init zygot_exec:process zygot;`
2. `type_transition init tmpfs:file init_tmpfs;`
3. `type_transition system_server system_data_file:sock_file
system_ndebug_socket "ndebugsocket";`
4. `allow domain binderfs:dir { getattr search };`
5. `allow domain binder_device:chr_file { open read write ioctl ... };`
6. `allowxperm domain binder_device:chr_file ioctl { 0x6201 0x6203
0x6205-0x6209 0x620b-0x620d 0x6210 };`
7. `mlsconstrain process { sigkill sigstop signal setsched setpgid setcap setrlimit
ptrace share } (l1 eq l2 or t1 == mltrustedsubject);`

Linux Example Rules

1. `type_transition init_t httpd_exec_t:process httpd_t;`
2. `type_transition init_t tmp_t:file init_tmp_t;`
3. `type_transition init_t var_run_t:sock_file rpcbind_var_run_t "rpcbind.sock";`
4. `allow domain device_t:dir { getattr search };`
5. `allow domain null_device:chr_file { open read write ioctl ... };`
6. N/A
7. `mlsconstrain process { sigkill sigstop signal setsched setpgid setcap setrlimit ptrace share } (h1 dom h2 or t1 != mcs_constrained_type);`

Example Domain Transition Rules

1. `type_transition init_t httpd_exec_t:process httpd_t;`
2. `allow init_t httpd_exec_t:file { read getattr map execute open };`
3. `allow init_t httpd_t:process { transition };`
4. `allow httpd_t httpd_exec_t:file { read getattr map execute open entrypoint };`
5. `dontaudit init_t httpd_t:process { noatsecure siginh rlimitinh };`

SELinux: Conditional Policy

- Policy booleans can be defined with an initial true/false value.
- Blocks of rules can be made conditional on a boolean expression.
- The policy boolean values can be changed at runtime to enable/disable rules dynamically.
- Originally introduced to support a "kill switch" feature.
 - Cut off access to peripherals, shut off a network flow, etc
- Now widely used in Linux / refpolicy
 - e.g. httpd_use_nfs, httpd_can_connect_ldap
- Prohibited in Android policy, checked by CTS
 - Prevent any runtime tampering with policy state

SELinux: Modes

- Global Modes
 - Disabled – Disabled in the kernel
 - Permissive – Logs what would be denied but does not enforce
 - Enforcing – Logs and enforces denials for all enforcing domains
 - Check with 'getenforce' or 'cat /sys/fs/selinux/enforce'
- Per-Domain Modes
 - Per-Domain Permissive – Permissive for individual domain(s)
 - Defined in the policy itself
 - Check with 'seinfo –permissive [/path/to/policy]'
 - Often used for new domains to enable learning/testing policy

SELinux: Tools

- Fuller list on <https://github.com/selinuxproject/selinux/wiki/Tools>
- Analysis tools described in subsequent slides.
- Some other commonly used tools:
 - `sestatus [-v]` : report the status and config of SELinux
 - `getenforce` : report the SELinux mode
 - `setenforce [0|1]`: change the active SELinux mode
 - `setsebool [-P] name value | name=value` : set policy boolean
 - `semodule` : install/remove/list policy modules
 - `semanage` : manage all the non-module configuration
 - `audit2allow` : generate allow rules from logged denials
 - `audit2why` : identify the cause of logged denials

Linux: Setting SELinux Modes

- To disable, boot kernel with selinux=0.
 - Or use lsm=lockdown,yama,integrity,bpf,landlock,ipe (everything but selinux!).
- To boot permissive, boot kernel with enforcing=0.
 - Or set SELINUX=permissive in /etc/selinux/config and reboot.
- Similarly for booting enforcing (enforcing=1, SELINUX=enforcing).
- At runtime, use setenforce 0 | 1 to switch modes temporarily.
 - Will reset upon reboot to kernel and /etc/selinux/config defaults
- Kernel command-line takes precedence over /etc/selinux/config
 - Protect your bootloader configuration!

Linux: Configuration Fallbacks

- If there is no `/etc/selinux/config`, the latest SELinux userspace (3.12+) will automatically fall back to `/usr/lib/selinux/config` if it exists.
 - Likewise for all the `/etc/selinux` configuration files
 - Allows vendors to ship read-only policy defaults under `/usr/lib`, only use `/etc` for admin overrides.

Linux: Setting Permissive Domains

- Only useful if your global mode is enforcing.
 - Otherwise all domains are already permissive.
- semanage permissive -a httpd_t
 - Adds httpd_t as a permissive domain
 - httpd_t can now do anything but denied permissions will still be logged
- semanage permissive -d httpd_t
 - Deletes httpd_t from being a permissive domain
 - httpd_t returns to being enforcing (if global mode is enforcing)

Android: Setting SELinux Modes

- Disabled is never an option; SELinux is mandatory in Android.
- Permissive is only an option for userdebug/eng builds.
 - androidboot.selinux=permissive
 - adb root && adb shell setenforce 0
 - Not supported for user (production) builds.
- Enforcing (both global and per-domain) is mandatory for Android user builds, tested by CTS.
 - No domain is allowed setenforce or load_policy permissions
 - Only init can do so before first setting enforcing mode to 1
 - Some device kernels omit permissive mode support altogether

Linux: SELinux Policy Types

- Only a concept in Linux, no parallel in Android
- Just a name for a particular policy configuration
- Default policy type is the "targeted" policy, i.e. confine/protect a targeted set of processes while leaving the rest unconfined.
- Other types include:
 - "mls" - Confine everything and use MLS rather than MCS.
 - "minimum" - a minimal policy
- The policy type to load at boot is specified via SELINUXTYPE= in {/etc,/usr/lib}/selinux/config.
- The policy files live under {/etc,/usr/lib}/selinux/\$SELINUXTYPE

Linux: Strict and Other Policies

- Originally there was also a "strict" policy – confine everything, like "mls" but still using MCS rather than MLS.
- With modern Linux, "strict" policy can be achieved just by mapping all users to confined roles via semanage and removing:
 - The unconfineduser module (no more unconfined users)
 - The unconfined module (no more unconfined processes)
 - This configuration might not be tested by your distro.
- Refpolicy has standard, mcs, and mls policy types instead.
 - standard = no-MCS/MLS; mcs = targeted in Fedora
- Since policy types are just names, other distros may use different names to mean the same thing as "targeted" e.g. "default".

Permission Denials

- Denials show up as “avc: denied...” log / audit messages.
 - **avc:** prefix is historical; message is logged by the AVC.
- View with dmesg / journalctl by default on Linux, logcat on Android.
- If running auditd, use ausearch -i -m AVC to view them.
- Some denials may be hidden via dontaudit rules in policy.

avc: denied { *execute* } for pid=4866 comm="netmgrd" name="sh"
dev="mmcblk0p25" ino=138 scontext=u:r:**netmgrd**:s0
tcontext=u:object_r:**shell_exec**:s0 tclass=*file*

- The **netmgrd** domain was denied permission to *execute* a *file* with the **shell_exec** type.
- IOW your network manager tried to spawn a shell!

Getting Full Pathnames

- Many SELinux permission checks happen at a point where we only have the inode information or the last component of the pathname.
- Full pathnames can be collected and audited with the Linux audit subsystem.
- Add any audit filter to turn on the machinery, e.g.
 - `auditctl -w /etc/shadow -p wa`
 - Subsequent denials will also report separate PATH records
- This can be set persistently across reboots:
 - `echo "-w /etc/shadow -p wa" > /etc/audit/rules.d/99-shadow.rules`
 - `augenrules --load`

Fixing Mislabeled Files

- Many SELinux denials are a result of mislabeled files.
 - File has the wrong label => permission denied!
- First thing to try: restore the file to the policy default
 - `restorecon [-r] -v /path/to/file`
- Still not working? Assign a different *type* in the policy and on-disk:
 - `semanage fcontext -a -t type /path/to/file or /path/to/dir(/.*)?`
 - `restorecon [-r] -v /path/to/file`
- What *type* to use?
 - Look at the type of similar files, e.g. `ls -Z /path/to/related/file`
 - Read the man page for the daemon's SELinux policy, e.g. `man httpd_selinux`, or look at that policy module's `.te` and `.fc` files

File Context Equivalence

- What if you just want to reuse all of the existing file_contexts assignments but with a re-mapped prefix?
 - e.g. alternate web root directory
- Define the two paths as equal in policy:
 - `semanage fcontext -a -e /var/www /path/to/alternate/root`
- And then restorecon the directory and its files:
 - `restorecon -rv /path/to/alternate/root`
- With selinux userspace 3.12+, you can also use a '*' whole-component wildcard to match exactly one non-empty component, e.g.
 - `semanage fcontext -a -e / "/var/lib/extensions/*"`

Fixing Mislabeled Ports

- Some SELinux denials are due to mislabeled ports.
 - Port has the wrong label => permission denied!
- Assign a specific *type* in the policy:
 - `semanage port -a -t type -p protocol port-number`
- Or if you need to override a pre-existing assignment:
 - `semanage port -m -t type -p protocol port-number`
- What *type* to use?
 - Look at the type of similar ports, e.g. `semanage port -l`
 - Read the man page for the daemon's SELinux policy, e.g. `man httpd_selinux`, or look at that policy module's `.te` file

Fixing Denials

- If running setroubleshoot, check its suggestion first.
- Is the process running in the right domain?
 - No => Fix the label on its executable file.
- Is the file or port that was being accessed labeled correctly?
 - No => Fix the label on the file or port.
- Does an existing boolean cover this situation?
 - Run audit2why on the avc: denied message to see.
 - Yes => Set the boolean to the correct value.
- Otherwise, run audit2allow and/or audit2why to fix or further diagnose.

Audit2allow Examples

- Create and insert a local policy module that allows all logged denials:
 - `audit2allow -a -M mypolicy`
 - `semodule -i mypolicy.pp`
- Create and insert a local policy module that allows all httpd denials:
 - `audit2allow -t httpd -a -M myhttpd`
 - `semodule -i myhttpd.pp`
- Create and insert a local policy module that allows all httpd denials and tries to use refpolicy interfaces when possible:
 - `audit2allow -R -t httpd -a -M myhttpd`
 - `semodule -i myhttpd.pp`

Audit2allow Caveat

- audit2allow blindly generates allow rules for every logged denial the kernel reported.
- You might not want to allow some of those permissions!
 - Some might be harmless to deny – use dontaudit instead.
 - Some might have safer variants, e.g. if you see dac_override, try allowing dac_read_search instead and see if it works.
 - dac_override is only required for write access but always gets checked
 - Or see if you can chown/chmod the directory/files to avoid needing either one
 - Always review audit2allow's generated .te file before using in production.

Silent Denials

- What if you don't see any log messages but your program still fails?
- First, make sure you can't reproduce the failure in permissive mode:
 - Run `setenforce 0` or `semanage permissive -a domainname`.
 - Retry, if it still fails it likely isn't SELinux's fault!
 - Run `setenforce 1` or `semanage permissive -d domainname`.
- If it only fails in enforcing mode, it might be silenced via dontaudit:
 - Run `semodule -DB` to strip all dontaudit rules.
 - Retry and see if you get log messages.
 - Run `semodule -B` when finished to restore the dontaudit rules.
 - Otherwise, your logs will quickly fill with noise!

Analyzing SELinux Policy

- Extract policy files from host/device or firmware.
 - /sys/fs/selinux/policy on a running system/device
 - /etc/selinux/\$SELINUXTYPE/policy/policy.NN on Linux
 - /{odm,vendor}/etc/selinux/precompiled_sepolicy on Android
- Binary policy can be analyzed using setools.
 - setools: seinfo, sesearch, sediff, sedta, seinfoflow
 - setools-gui: apol
- Binary policy can also be decompiled using the SELinux policy compiler/decompiler, checkpolicy.
 - checkpolicy -M -F -b /path/to/sepolicy -o policy.conf
 - Not quite the same as original source; post-processed and concatenated.

Analysis: A Sample Walkthrough

Is source S allowed to perform operation O on target T?

1. Find out the context in which S runs.

- Running device: `ps -Z` or `cat /proc/pid/attr/current`
- Firmware only: Determine from policy files.

2. Find out the context assigned to T.

- Running device: Same as above if T is a process, `ls -Z` if a file
- Firmware only: Determine from policy files, or mount firmware image and `getfattr -n security.selinux` on the relevant file.

Analysis: A Sample Walkthrough (cont'd)

3. Map O to SELinux class(es) C and permission(s) P.
 - https://github.com/selinuxproject/selinux-notebook/blob/main/src/object_classes_permissions.md
4. Search for a matching allow rule.
 - `sesearch -A -s <S> -t <T> -c <C> -p <P>`
 - Just drop selectors if you want to be broader in your search.
5. No allow rule => not allowed.
6. Allow rule => might still be denied by a constrain/mlsconstrain rule.
 - `audit2why` can help with these.

Sample Searches

1. `sesearch -A -s zygote /path/to/sepolicy`
2. `sesearch -A -t dalvikcache_data_file -c file -p create,write /path/to/sepolicy`
3. `sesearch -A -t vold_socket -c sock_file -p write /path/to/sepolicy`
4. `sesearch -A -t vold -c unix_stream_socket -p connectto /path/to/sepolicy`
5. `sesearch -A -c keystore_key -p add_auth /path/to/sepolicy`
6. `sesearch -A -t radio_prop -c property_service /path/to/sepolicy`
7. `sesearch -A -t gatekeeper_service -c service_manager -p find /path/to/sepolicy`

Search Results and Type Attributes

- Some rules returned by search may be written in terms of type attributes rather than individual types.
- Reminder: A type attribute is a named type set.
- Common attributes used in source policy:
 - *domain, appdomain, file_type, exec_type, ...*
- The attribute can be expanded to individual types via seinfo:
 - `seinfo -adomain -x /path/to/sepolicy`

Domain Transition Analysis

- Supported via sedta command-line or apol GUI
- Analyzes allowed transitions from or to a given type.
 - Reachability analysis from a given source to a given sync
- Two kinds of transitions:
 - *execve*-based (*transition*); also requires associated file *entrypoint* and *execute* permissions
 - *setcon*-based (*dyntransition*); also requires an associated process *setcurrent* permission

Sample Domain Transition Analyses

1. `sedta -p /path/to/sepolicy -s untrusted_app`
2. `sedta -p /path/to/sepolicy -s system_server -r`
3. `apol /path/to/sepolicy -> New Analysis -> Domain Transition Analysis`

Information Flow Analysis

- Supported via seinfoflow command-line or apol GUI.
- Analyzes information flow paths through policy by mapping all permissions to read, write, both, or none, constructing graph.
- Requires a separate permission map file that specifies rwnb and a weight for each permission.
- Base perm_map file can be obtained from upstream setools project (setools/setools/perm_map) or installed setools (e.g. /usr/lib/python3/dist-packages/setools/perm_map). DO NOT USE /var/lib/sepolicy/perm_map.
- perm_map needs to be augmented with Android-specific classes/permissions. Would also benefit from tailoring the weights.

Sample Information Flow Analyses

1. `seinfoflow -m perm_map -p /path/to/sepolicy -s untrusted_app`
2. `apol /path/to/sepolicy -> New Analysis -> Information Flow Analysis`
-> Flows into the target type

SELinux: Resources

- Website: <https://selinuxproject.github.io>
- Public mailing list: selinux@vger.kernel.org
- Free online books: [*The SELinux Notebook*](#), [*The SELinux Coloring Book*](#), [*Red Hat Enterprise Linux Using SELinux*](#), [*SUSE Linux Enterprise Server Understanding SELinux Basics*](#)
- SELinux Books for \$ (no endorsement): *SELinux By Example: Using Security Enhanced Linux*, *SELinux Cookbook*, *SELinux: NSA's Open Source Security Enhanced Linux*, *SELinux System Administration*
- SE (for) Android Books for \$ (no endorsement): *Android Security Internals*, *Exploring SE for Android*

Android-Specific



App Labeling Configuration Files

- **mac_permissions.xml**
 - Maps app certificate and optionally package name to a *seinfo* string.
- **seapp_contexts**
 - Maps app UID, *seinfo* string, boolean selectors (*isPrivApp*, *isEphemeralApp*, ...), and *minTargetSdkVersion* to domain for app process, type for app data, and optionally category set for both.

Context for Regular Third Party App

- No matching entry in `mac_permissions.xml`.
 - seinfo tag set to *default*
- App assigned a regular app ID.
- Matches *user=_app minTargetSdkVersion=37 domain=untrusted_app ... levelFrom=all* line in `seapp_contexts`.
- Will run in **untrusted_app** domain with category set that encodes both user and app identities.

Context for sharedUserId=system App

- Matches platform signer entry in mac_permissions.xml.
 - seinfo tag set to *platform*.
- App assigned system UID.
- Matches *user=system seinfo=platform domain=system_app ...* entry in seapp_contexts.
- Will run in **system_app** domain with no categories.

Context for Platform-Signed, Non-Priv App

- Matches platform signer entry in mac_permissions.xml.
 - seinfo tag set to *platform*.
- App assigned regular app ID.
- Matches *user=_app minTargetSdkVersion=37 seinfo=platform domain=platform_app ... levelFrom=user* entry in seapp_contexts.
- Will run in **platform_app** domain with a category set that encodes user identity.

Context for Platform-Signed, Priv App

- Matches platform signer entry in `mac_permissions.xml`.
 - `seinfo` tag set to *platform*.
- App assigned regular app ID.
- Matches *user=_app isPrivApp=true domain=priv_app ... levelfrom=user* entry in `seapp_contexts`.
- Will run in **priv_app** domain with a category set that encodes user identity.

Discovering Service Context

- Check `init*.rc` entry for the service for a **seclabel** option; if present, use that, else...
- Look up context of service pathname in *file_contexts*.
 - `grep` or `matchpathcon/selabel_lookup` (`libselinux/Utils`)
- Search for a process type transition on the file type:
 - `seseach -T -t <TYPE> -c process /path/to/sepolicy`

Context for Service Using seclabel

- `init*.rc` contains *seclabel u:r:ueventd:s0* in the *service ueventd* stanza.
- Will run in the **ueventd** domain.
- **seclabel** is normally only used for services run from the rootfs (/), since it doesn't have per-file labeling, or for scripts.
- Technically though can be used for any service.

Context for Service Using Policy Transition

- *service zygot*e stanza has no **seclabel** option, path is `/system/bin/app_process64`.
- *file_contexts*: `/system/bin/app_process64 u:object_r:zygote_exec:s0`
- `seseach -T -t zygote_exec -c process /path/to/sepolicy`
 - `type_transition init zygote_exec:process zygote;`
- Will run in the **zygote** domain.

File Labeling Examples

- file_contexts
 - /dev/binder u:object_r:**binder_device**:s0
 - /system(/.)*? u:object_r:**system_file**:s0
 - /system/bin/init u:object_r:**init_exec**:s0
- seapp_contexts
 - ... type=privapp_data_file levelFrom=user
 - u:object_r:**privapp_data_file**:s0:c512,c768
 - ... type=app_data_file levelFrom=all
 - u:object_r:**app_data_file**:s0:c0,c256,c512,c768
- genfs_contexts
 - genfscon binder /binder u:object_r:**binder_device**:s0

Userspace Object Labeling

- property_contexts
 - persist.radio u:object_r:radio_prop:s0
 - net.dns u:object_r:net_dns_prop:s0
- service_contexts, hwservice_contexts
 - media.camera u:object_r:cameraserver_service:s0
 - android.hardware.camera.provider::ICameraProvider
u_object_r:hal_camera_hwservice:s0
- keystore2_contexts
 - 120 u:object_r:resume_on_reboot_key:s0

MLS policy

- Reinforces Android user and app UID-based separation by assigning separate category sets.
- Key things to know:
 - Only applied to (some) apps, based on levelFrom= in seapp_contexts.
 - mltrustedsubject and mltrustedobject exceptions.
 - Can prevent two apps in the same domain from interacting, sharing files, etc.
 - Only per-app for target SDK ≥ 28 , Android ≥ 9 .

Device SELinux Policy Files

- All policy files, including sepolicy, split per major OS partition.
 - /{system,system_ext,product,vendor,odm}/etc/selinux
- sepolicy split via SELinux Common Intermediate Language (CIL) policy modules.
- Load precompiled_sepolicy from {odm,vendor}/etc/selinux if it was built against a matching platform (system) policy.
- Otherwise, run secilc on all CIL policy modules, load resulting compiled kernel policy.
 - Only necessary upon Android update of system partition without update of {odm,vendor} to match.

Precompiled Policy Requirements

- Find precompiled_sepolicy from odm or vendor (odm takes precedence).
- Compare sha256 files from system, system_ext, and product policies against corresponding files in odm or vendor.
- Only loads precompiled_sepolicy if all match.

Use the Source, Luke!

- Policy sources are available in AOSP.
- Core AOSP policy, inherited by all devices.
 - `system/sepolicy`
- Device-specific policy
 - `device/vendor/board/sepolicy`
 - Only for a few devices with AOSP repos but often other devices reuse bits for similar boards, so may still be useful for reference
- OEMs also sometimes provide their source code and/or policy from their respective website or GitHub – look for it!

Type Enforcement (TE) Configuration

- **.te** files: Domain and type definitions, rules.
 - Typically one **.te** file per domain, e.g. **install.d.te**.
 - Device and file types declared in **device.te**, **file.te**.
 - Shared rules in certain files (**domain.te**, **app.te**).
- Written using macros from **global_macros**, **te_macros** and attributes (type sets) from **attributes**.

Dealing with Denials: Labeling Problems

- Most denials are due to labeling problems.
 - Wrong domain for process or wrong type for file.
- Fix the labeling and the rest will typically follow.
 - Define a domain transition for the service.
 - Define type transitions for service-created files.
 - Update **file_contexts** for:
 - service sockets, /data directories, /dev nodes, /sys files

Other Labeling Problems

- /proc files
 - Label using **genfs_contexts** (part of kernel policy).
- Filesystems that do not support labeling.
 - Default assigned via **genfs_contexts**.
 - Per-mount label can be assigned using **context=** mount option.

Generalizing audit2allow rules

- Allow for all domains?
 - Rewrite using **domain** attribute, add to **domain.te**.
- Allow for all app domains?
 - Rewrite using **appdomain** attribute, add to **app.te**.
- Consider whether the rule should be written using an attribute from **attributes**.

Generalizing audit2allow rules

- Use macros (from **global_macros**, **te_macros**).
 - Common groupings of classes, permissions, rules.
 - Needs create? Use **create_file_perms**.
 - Needs open + read? Use **r_file_perms**.
 - Needs open + write? Use **rw_file_perms**.
 - Needs execute, execute_no_trans? Use **rx_file_perms**.
 - Reduces policy brittleness.

Dealing with neverallow failures

- Policy contains a set of **neverallow** rules to prevent adding unsafe **allow** rules.
- Checked by **checkpolicy** during policy build and by CTS.
- Do not remove or comment out **neverallow** rules!
- Whenever possible, eliminate the need for the **allow** rule.
- As needed, can craft narrow exceptions for specific domains, types or permissions by amending the **neverallow** rule.
 - A good idea to propose to AOSP first!
 - Otherwise you may fail CTS in the future...

Android-specific policy tools

- No endorsement of any of these, and upstream setools might be fine for your use case
- android.googlesource.com/platform/system/sepolicy/+refs/heads/android17-release/tools
- github.com/phhusson/sepolicy-inject
- github.com/fics/BigMAC
- github.com/Heydarchi/SELinux-Explorer

Android-Specific Resources

- Security-Enhanced Linux in Android, source.android.com/docs/security/features/selinux
- The SELinux Notebook SE (for) Android section, github.com/selinuxproject/selinux-notebook/blob/main/src/seandroid.md
- The original paper: *Security Enhanced Android: Bringing Flexible MAC to Android*, 2013 NDSS.
- SE (for) Android presentations, github.com/selinuxproject/selinux/wiki/Presentations-and-Papers
- SE (for) Android Books for \$ (no endorsement): *Android Security Internals*, *Exploring SE for Android*