Security-Enhanced Linux in Android

As part of the Android security model (/security), Android uses Security-Enhanced Linux (SELinux) to enforce mandatory access control (MAC) over all processes, even processes running with root/superuser privileges (Linux capabilities). Many companies and organizations have contributed to Android's SELinux implementation (https://android.googlesource.com/platform/external/selinux/). With SELinux, Android can better protect and confine system services, control access to application data and system logs, reduce the effects of malicious software, and protect users from potential flaws in code on mobile devices.

SELinux operates on the principle of default denial: Anything not explicitly allowed is denied. SELinux can operate in two global modes:

- *Permissive* mode, in which permission denials are logged but not enforced.
- *Enforcing* mode, in which permissions denials are both logged **and** enforced.

Android includes SELinux in enforcing mode and a corresponding security policy that works by default across AOSP. In enforcing mode, disallowed actions are prevented and all attempted violations are logged by the kernel to `dmesg` and `logcat`. When developing, you should use these errors to refine your software and SELinux policies before enforcing them. For more details, see Implementing SELinux (/security/selinux/implement).

SELinux also supports a *per-domain permissive* mode in which specific domains (processes) can be made permissive while placing the rest of the system in global enforcing mode. A domain is simply a label identifying a process or set of processes in the security policy, where all processes labeled with the same domain are treated identically by the security policy. Per-domain permissive mode enables incremental application of SELinux to an ever-increasing portion of the system and policy development for new services (while keeping the rest of the system enforcing).

Background

The Android security model is based in part on the concept of application sandboxes (/security/app-sandbox). Each application runs in its own sandbox. Prior to Android 4.3, these sandboxes were defined by the creation of a unique Linux UID for each application at time of installation. Android 4.3 and later uses SELinux to further define the boundaries of the Android application sandbox.

In Android 5.0 and later, SELinux is fully enforced, building on the permissive release of Android 4.3 and the partial enforcement of Android 4.4. With this change, Android shifted from enforcement on a limited set of crucial domains (`installd`, `netd`, `vold` and `zygote`) to everything (more than 60 domains). Specifically:

- Everything is in enforcing mode in Android 5.x and higher.
- No processes other than `init` should run in the `init` domain.
- Any generic denial (for a `block_device`, `socket_device`, `default_service`) indicates that device needs a special domain.

Android 6.0 hardened the system by reducing the permissiveness of our policy to include better isolation between users, IOCTL filtering, reduced threat of exposed services, further tightening of SELinux domains, and extremely limited `/proc` access.

Android 7.0 updated SELinux configuration to further lock down the application sandbox and reduce attack surface. This release also broke up the monolithic mediaserver stack into smaller processes to reduce the scope of their permissions. For more details, see Protecting Android with more Linux kernel defenses (https://android-developers.googleblog.com/2016/07/protecting-android-with-more-linux.html) and Hardening the media stack (https://android-developers.googleblog.com/2016/05/hardening-media-stack.html).

Android 8.0 updated SELinux to work with Treble (/devices/architecture#hidl), which separates the lower-level vendor code from the Android system framework. This release updated SELinux policy to allow device manufacturers and SOC vendors to update their parts of the policy, build their images (`vendor.img`, `boot.img`, etc.), then update those images independent of the platform or vice versa.

While it is possible to have higher/newer platform (framework) version running on the device, the opposite case is not supported; the vendor images (`vendor.img/odm.img`) cannot have a newer version than the platform (`system.img`). So, a newer platform version might introduce SELinux compatibility issues because the platform SELinux policy is at a newer version than

vendor SELinux parts of the policy. The Android 8.0 model provides a method to retain compatibility (/security/selinux/compatibility) to prevent unnecessary simultaneous OTAs.

Additional resources

For help constructing useful SELinux policies, refer to the following resources:

- Security Enhancements for Linux (https://events.static.linuxfound.org/sites/events/files/slides/abs2014_seforandroid_smalley.pdf)
- Security Enhanced (SE) Android: Bringing Flexible MAC to Android (http://www.cs.columbia.edu/~lierranli/coms6998-7Spring2014/papers/SEAndroid-NDSS2013.pdf)
- The SELinux Notebook, 4th Edition (http://freecomputerbooks.com/books/The_SELinux_Notebook-4th_Edition.pdf)
- SELinux Object Classes and Permissions Reference (http://selinuxproject.org/page/ObjectClassesPerms)
- Implementing SELinux as a Linux Security Module (https://www.nsa.gov/resources/everyone/digital-media-center/publications/research-papers/assets/files/implementing-selinux-as-linux-security-module-report.pdf)
- Configuring the SELinux Policy (https://www.nsa.gov/resources/everyone/digital-media-center/publications/research-papers/assets/files/configuring-selinux-policy-report.pdf)
- GNU M4 - GNU Macro Processor Manual (https://www.gnu.org/software/m4/manual/index.html)
- Your visual how-to guide for SELinux policy enforcement (https://opensource.com/business/13/11/selinux-policy-guide)

Last updated 2020-09-01 UTC.