SELinux in Android Oreo or:
How I Learned to Stop Worrying and Love Attributes

Dan Cashman
September 15th, 2017
$ whoami

- Dan Cashman - dcashman@google.com
- Android Security since 2013
- Software Engineer
Acknowledgements

- Alex Klyubin
- Jeff Vander Stoep
- Jim Carter
- Nick Kralevich
- Sandeep Patil
- Stephen Smalley
Background: A(n) History of SELinux on Android

- Prehistory: SELinux added to Linux as an LSM
- Jelly Bean (4.3): SEAndroid upstreamed to AOSP and released in permissive mode
- KitKat (4.4): Four critical daemons in enforcing mode
- Lollipop (5.0): Enforcing EVERYWHERE.
- Marshmallow (6.0): extended perms, multi-user, svcmgr object manager, hardening
- Nougat (7.0): hardening + verified boot protection
- Oreo (8.0): Treble
SELinux reduced severity of almost half of kernel bugs
(Android security bulletin data for Jan-Apr 2017)
Introducing Treble

Project Treble is a re-architecture of the Android software to make the stack more modular and facilitate faster platform upgrades and security updates.
Before Treble

Previous Android Release

Previous Android OS framework

Previous vendor implementation

Updated Android Release

Updated Android OS framework

Reworked vendor implementation
With Treble

Previous Android OS framework → Updated Android OS framework

Original vendor implementation

Vendor interface
Treble Key Players

- VINTF - the vendor interface
- HIDL - HAL Interface Definition Language
- VTS - Vendor Interface Test Suite
- VNDK - Vendor Native Development Kit (a la NDK for apps)
- Separate Ownership
SEAndroid vs Treble

- As Android’s mandatory access control (MAC) system, SELinux policy should be all-powerful and control every component of the system.
- Treble seeks to create a modular Android where different owners may update their components independently of others.
Changes for Treble

On-device policy compilation

New public/private split (policy API)

Compatibility attributes and mapping files

HAL policy

Neverallow-driven development

Questions?
Policy Compilation
Approaches Mooted

- Policy hierarchy: odm > vendor > system or LRU (least-recently-updated)
  - Pro: simple implementation
  - Con: Prevents independent update
- Switch from monolithic kernel policy to base policy + modules
  - Pro: modules in the name, so modular?
  - Cons: language limitations, libsemanage deps, policy rewriting
- Cloud-compilation
  - Pro: simple on-device implementation
  - Con: additional update server infrastructure
- On-device compilation (winner!)
  - Pro: each component owner can provide policy alongside code that needs it
  - Con: new work needed at early-boot
On-device Compilation

- Split policy into two components: plat and non_plat (framework and device-specific)
- Added first stage mount of /system and /vendor partitions (all plat on /system and all nonplat on /vendor)
- Added secilc executable and call from init to build policy binary from split components
- Modified configuration file consumers to reflect split
  - libselinux - file_contexts (forked from upstream)
  - PackageManager - mac_permissions.xml
  - libselinux android.c - property_contexts, seapp_contexts, service_contexts (and hwservice_contexts)
  - Defined object ownership according to split
public/private policy split
Global vs. Device-specific Policy

Policy Size (LOC)
- sailfish device: 13.2%
- core policy: 86.8%
Device-Specific Type Usage

Sailfish types

public
21.1%

device-specific
78.9%
Public/Private?

- The public/private split is the SELinux extension to the treble VINTF. Public policy can be relied on by vendor policy.
- **Public policy**
  - Is basically what the global policy was pre-Oreo
  - types and attributes can be used directly in vendor policy
  - types are versioned (more later)
  - avrules are copied to the device policy
- **Private policy**
  - Describes internal Android framework components
  - Does not interface with vendor components
  - Could disappear at any point
Compatibility Attributes
Problem: Labels Change Across Releases

MediaServer

ExtractorService
MediaCodecService
AudioServer
CameraServer
MediaDrmServer
Problem: Labels Change Across Releases

/dev/cam
- camera_device

/video_device

/proc/meminfo
- proc

/proc_meminfo
- proc_meminfo
Problem: Labels Change Across Releases

- Vendor policy is written based on Framework policy
- Framework can be changed with a framework-only update (treble goal)
- Framework policy owner has no knowledge of vendor policy
Solution: Attributes

Every object has a security context
- `u:r:untrusted_app:s0:c512,c768`
  - `u` - user
  - `r` - role
  - `untrusted_app` - domain/type
  - `s0:c512,c768` - mls

Only one type per object, but multiple types per attribute.

Solution: Rewrite vendor policy in terms of attributes. Framework policy needs to map the object types in the new version to their attribute representation from an old version.
Policy in Oreo

- O public
- O private

+ O public (versioned)
  + O vendor
Policy with Framework Update

P public

P private

O public (versioned)

O vendor
SELinux Common Intermediate Language (CIL)
CIL Benefits

- `typeattributeset()` can contain attributes
- Ordering doesn’t matter
- Easier to manipulate
- Designed as basis for higher-level languages
Example: Adding a new type

```c
type sysfs_A;  -> (type sysfs_A) (in CIL)
type sysfs; (type sysfs) (in CIL)
allow ... sysfs: ...; (allow ... sysfs ...) (in CIL)
allow ... sysfs_A: ...; (allow ... sysfs_A ...) (in CIL)
```

New v2 plat/framework policy w/sysfs_A as a new sysfs type.

```c
(typeattribute sysfs_v1 (sysfs sysfs_A))
```

Mapping file linking to v1

```c
v1 nonplat/vendor policy

(typeattribute sysfs_v1)
(allow ... sysfs_v1 ...)
```
HAL policy
HAL policy

- HALs are the main architectural change in Treble
- Multiple HALs could be in same process
- HAL clients can change after update
  - E.g. mediaserver split
- HIDL, the lingua franca of Treble, required over /dev/hwbinder
- Solution: attributes again
Shut the HAL Up

https://android-developers.googleblog.com/2017/07/shut-hal-up.html
Attributes!

- 36 new HAL policy files
- 108 (36 x 3) attributes from HALs alone
- Used to
  - Create stable interface
  - Migrate to Treble using same code base
- Performance hit required CIL change (thanks Jim Carter!)

```c
attribute hal_allocator;
expandattribute hal_allocaltor true;
attribute hal_allocator_client;
expandattribute hal_allocator_client true;
attribute hal_allocator_server;
expandattribute hal_allocator_server false;
```

```
attribute hal_wifi_supplicant;
expandattribute hal_wifi_supplicant true;
attribute hal_wifi_supplicant_client;
expandattribute hal_wifi_supplicant_client true;
attribute hal_wifi_supplicant_server;
expandattribute hal_wifi_supplicant_server false;
```
Attribute Performance

... { fs_type -rootfs } ...

```
ebitmap_for_each_positive_bit(sattr, snode, i) {
  ebitmap_for_each_positive_bit(tattr, tnode, j) {
    (typeattribute base_typeattr_26)
    (typeattributeset base_typeattr_26 (and (fs_type ) (not (rootfs ))))
  }
}
```
Never allow driven development
Large Re-architecture Projects are Hard

- SELinux can help!
- New attributes created to catch bugs and guide development
  - binder_in_vendor_violators
  - socket_between_core_and_vendor_violators
  - vendor_executes_system_violators
  - coredomain, vendor_file_type
- 74 bugs found and fixed violating new architecture
The Future (Why I’m Here)

- Upstream necessary changes
- SELinux tools now performance-critical!
- “If all you have is a hammer, everything looks like an attribute” - explore alternatives with other stakeholders
- Clean up existing policy
QUESTIONS ?