Security Enhancements (SE) for Android

Stephen Smalley
Trusted Systems Research
National Security Agency
Agenda

• Motivation/Background
• Current State
• Using SELinux in Android
• What's Next for SELinux in Android
• Beyond SELinux
Trusted Systems Research: Who are We?

• Perform R&D in support of NSA's Information Assurance (IA) mission to protect and defend National Security Information and Information Systems.

• Long history of open source software contribution and participation, starting with release of SELinux in December 2000.
Our Motivation

- Increasing demand to use mobile devices.
- Desire to use commodity solutions.
- Risks posed by currently available solutions.
  - Exploitation over wireless, radio, NFC...
  - Data Leakage
  - Application privilege escalation
Why It Matters for Everyone

• Increasing importance of mobile device security.
  • Payment, banking, remote control.
  • BYOD trend for corporate/enterprise use.
  • Increasing use of mobile platforms in non-traditional venues, including safety-critical.

• It isn't just a problem for government use.
A Step in the Right Direction

- NSA Security Enhancements (SE) for Android project
  - formerly known as Security-Enhanced (SE) Android
- Identify and address critical gaps in the security of Android.
- Why Android?
  - Open source platform: suitable for a reference implementation accessible to anyone.
  - Broad market adoption: opportunity to improve the security of a widely used mobile platform.
SE for Android: Contributions

• Created and released an open source reference implementation of how to enable and apply SELinux in Android.
• Presented the case for adopting SELinux in Android.
• Worked with Android Open Source Project (AOSP) to gain adoption into mainline Android.
SE for Android: Timeline

Jan 6 2012
SE for Android released

Mar 2012
Samsung collaboration begins

Apr 2013
First device w/ SE ships - Galaxy S4

Sep/Oct 2013
2nd device w/ SE ships - Galaxy Note 3

Oct 31 2013
First Android release w/ SE enforcing - Android 4.4

Jan 9 2012
Google invites submission

Feb 2013
Samsung announces KNOX w/ SE for Android

Jul 2013
First Android release w/ SE permissive-Android 4.3

Oct 2013
4.3 update for Galaxy S4 w/ SE enforcing

Feb 2014
Samsung announces KNOX 2.0, Galaxy S5
**SELinux: What is it?**

- Mandatory Access Control (MAC) for Linux.
  - Enforces an admin-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” / uid 0.
  - Protect from misuse, contain damage.
  - Mitigate risks of flawed and malicious programs.
SELinux: Labeling

- Each process and object is labeled with a security context.
  - A string of the form “user:role:type:level”.
  - Only the type field is used in AOSP presently.
- Process types are also called domains.
- Domains and types are security equivalence classes.
  - Identifiers for processes and objects in policy.
  - Same domain/type => same access.
SELinux: Policy

• The security policy configuration defines:
  – how to label processes and objects with domains and types,
  – how domains can interact with each other (e.g. signals, IPC, ptrace), and
  – how domains can access types.

• No processes are exempt from the policy.
  – Not overridden by uid-0 or Linux capabilities.
  – Only notion of “unconfined” is policy-defined.
SELinux: Possible States

- **Disabled**
  - Not enabled in the kernel or disabled via kernel parameter.

- **Permissive**
  - Just logs denials but does not enforce them.

- **Enforcing**
  - Logs and enforces denials for all enforcing domains (processes).
SELinux: Possible States

- Per-Domain Permissive
  - Permissive for specific domains (processes).
  - Specified in policy on a per-domain basis.
  - Enables incremental application of SELinux to an ever increasing portion of the system.
  - Enables policy development for new services and apps while keeping the rest of the system enforcing.
State of SELinux in AOSP

- Android 4.2 or earlier: Disabled.
- Android 4.3: Permissive.
  - With all domains permissive + unconfined.
- Android 4.4: Enforcing.
  - Enforcing for `installd`, `netd`, `vold`, and `zygote`.
  - Permissive for app domains (logging denials).
  - Permissive + unconfined for all other domains.
State of SELinux in Samsung KNOX

- First included in Galaxy S4 (4.2.2) but in permissive by default.
- 4.3 and later updates switched to enforcing mode.
- No permissive domains (all enforcing).
- Only kernel and init domains are unconfined.
- Policy originally derived from our policy, but customized by Samsung.
Using SELinux in Android

- Exploring SELinux.
- Policy configuration files.
- Policy for services.
- Policy for apps.
- Dealing with denials.
- Dealing with neverallow failures.
Exploring SELinux

- **toolbox** built-in commands and options
  - `getenforce`, `setenforce`
  - `ls -Z`, `ps -Z`

- Seeing denials:
  - `dmesg | grep avc:` # current boot
  - `cat /proc/last_kmsg | grep avc:` # prior boot
Policy Configuration Sources

- **external/sepolicy**
  - Device-independent configuration
  - Do not modify for your device!

- **device/<vendor>/<product>/sepolicy**
  - Device-specific configuration
  - Based on `BOARD_SEPOLICY_*` variables.
  - Documented in `external/sepolicy/README`.
  - Examples for Nexus devices in AOSP, e.g.
    - `device/lge/hammerhead/{BoardConfig.mk,sepolicy/*}`
Type Enforcement (TE) Configuration

• .te files: Domain and type definitions, rules.
  – Typically one .te file per domain, e.g. installd.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.
Labeling Configuration Files

- **file_contexts**: File security contexts
  - Labels for /system (consulted by `make_ext4fs`).
  - Labels for /dev, /sys, /data directories created by `init.rc` files (consulted by `init`, `ueventd`, and others).
  - Labels for `restorecon` (“restore security context”).

- **property_contexts**: Property security contexts
  - Labels for `init` property service permission checks.
App Labeling Configuration Files

- **mac_permissions.xml**
  - Maps app certificate to a *seinfo* string.
  - Used by **PackageManagerService / SELinuxMMAC**.

- **seapp_contexts**
  - Maps app UID and optionally *seinfo* string to domain for app and type for /data/data directory.
  - Used by **zygote** and **installd** via libs selinux.
Policy Build

• Union/replace/ignore files based on BOARD_SEPOLICY_* variables.

• Concatenate and expand macros using m4.
  - For kernel policy, yields policy.conf file.

• For kernel policy, compile policy.conf file to binary sepolicy file using checkpolicy.

• Other configurations checked but not compiled using similar helpers (checkfc, checkseapp).
On-Device Policy Files

- `/sepolicy`: Kernel binary policy
- `/file_contexts`: File security contexts
- `/property_contexts`: Property security contexts
- `/seapp_contexts`: App security contexts
- `/system/etc/security/mac_permissions.xml`: App certificate to seinfo mapping
Policy for Services

- Every service needs a domain.
- `ps -Z | grep :init:` should only list the `init` process.
- Anything else is a service left running in the `init` domain.
- Need to place any such service into its own domain.
- This is enforced by CTS in AOSP master.
Labeling a Service

• Options:
  – Define an automatic domain transition in policy.
  – Use the `seclabel` option in the `init.<board>.rc` file.

• First option is preferred if possible.

• Second option supports services run from rootfs or launched via shell scripts.
Labeling a Service via Transition (1/2)

- **device/lge/hammerhead/sepolicy/netmgrp.d.te:**
  type netmgrp, domain;
  type netmgrp_exec, exec_type, file_type;
  init_daemon_domain(netmgrp)

  ...

- **device/lge/hammerhead/sepolicy/file_contexts:**
  /system/bin/netmgrp   u:object_r:netmgrp_exec:s0
Labeling a Service via Transition (2/2)

- **device/lge/hammerhead/BoardConfig.mk:**
  
  ```
  BOARD_SEPOLICY_DIRS += \
  device/lge/hammerhead/sepolicy
  
  BOARD_SEPOLICY_UNION += \
  netmgrd.te \
  file_contexts \
  ...
  ```
Labeling a Service via seclabel

- **device/asus/flo/init.flo.rc:**
  
  service hciattach /system/bin/sh /system/etc/init.flo.bt.sh
  seclabel u:r:bluetooth_loader:s0

- **device/asus/flo/BoardConfigCommon.mk:**
  
  BOARD_SEPOLICY_DIRS += device/asus/flo/sepolicy
  BOARD_SEPOLICY_UNION += bluetooth_loader.te

- **device/asus/flo/sepolicy/bluetooth_loader.te:**
  
  type bluetooth_loader, domain;
  allow bluetooth_loader shell_exec:file { entrypoint read };

Labeling Apps

- Based on `mac_permissions.xml` and `seapp_contexts`.
- Divides into several categories:
  - System apps by platform UID
  - System apps by certificate
  - Other apps
  - Isolated services
System Apps by Platform UID

- **seapp_contexts:**
  
  user=system domain=system_app type=system_data_file
  user=bluetooth domain=bluetooth type=bluetooth_data_file
  user=nfc domain=nfc type=nfc_data_file
  user=radio domain=radio type=radio_data_file
  user=shell domain=shell type=shell_data_file
System Apps by Certificate

- **mac_permissions.xml:**
  ```xml
  <signer signature="@PLATFORM" >
    <seinfo value="platform" />
  </signer>
  ```

- **seapp_contexts:**
  ```plaintext
  user=_app seinfo=platform domain=platform_app
type= app_data_file
  ```
System Apps by Certificate

• At build time, `mac_permissions.xml` signature tag names (e.g. `@PLATFORM`) are rewritten to the actual certificate value extracted from `.pem` file specified by `external/sepolicy/keys.conf`.

• `build/tools/releasetools/sign_target_files_apks` rewrites `mac_permissions.xml` with updated certificate values for new keys.
Other Apps

- `seapp_contexts`:
  user=_app domain=untrusted_app type=app_data_file

- Assigned to system apps with regular app IDs unless they have a more specific entry that matches.

- Assigned to all third party apps (in AOSP).
Isolated Services

• **seapp_contexts:**

  \[\text{user=\_isolated, domain=isolated\_app}\]

• **isolated\_app** domain is for services with **android:isolatedProcess=’true’** in manifest.
  - e.g. Chrome sandbox process
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.
Fixing Labeling Problems Example

- `device/lge/hammerhead/fstab.hammerhead:`
  
  `/dev/block/platform/msm_sdcc.1/by-name/modem/firmware` 
  `vfat ro,shortname=lower,uid=1000,gid=1000,` 
  `dmask=227, fmask=337,` 
  `context=u:object_r:firmware_file:s0` 
  `wait`

- `device/lge/hammerhead/sepolicy/genfs_contexts:`
  
  `genfscon proc /bluetooth/sleep/lpm u:object_r:proc_bluehooth_writable:s0`
  
  `genfscon proc /bluetooth/sleep/btwrite u:object_r:proc_bluehooth_writable:s0`
Dealing with Denials: dontaudit

• Some denials are harmless – the program will not fail even if not allowed.
  – Can use a **dontaudit** rule to silence the denial.
  – Be careful about using such rules!

• Example: **netmgrp** attempts to load a network driver, triggers sys_module denial. But kernel is not modular!
  – **dontaudit netmgrp** self:capability sys_module;
Dealing with Denials: Linux capabilities

• Consider whether you can avoid the need for the capability.
  – Add a group to the service or change the ownership or mode of a file.
  – Pre-create directories with correct owner/mode in `init.<board>.rc`.

• Consider whether a lesser capability can be allowed.
  – `dac_read_search` rather than `dac_override`.
Dealing with Denials: audit2allow

adb shell su 0 cat /proc/kmsg > dmesg.txt &
audit2allow -p out/target/product/<product>/root/sepolicy < dmesg.txt > allows.txt

• Review allows.txt.
• But do NOT blindly add the rules it generated to your policy!
• Always try to generalize the rule generated by audit2allow.
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 execute, execute_no_trans? Use rx_file_perms.
  - Reduces policy brittleness.
SELinux Denial Example

avc: denied { **execute** } for pid=3849
comm="netmgrd" name="sh" dev="mmcblk0p25"
ino=224 scontext=**u:r:netmgrd:s0**
tcontext=**u:object_r:shell_exec:s0** tclass=file

- netmgrd service attempted to execute sh.
- To allow, add following line to netmgrd.te:
  allow netmgrd shell_exec:file rx_file_perms;
Addressing Hidden Denials

- Fails in enforcing mode but no avc: denied message.
- Remove suspect `dontaudit` rules and re-test.
- Can also use `sepolicy.dontaudit` file.
  - Under `obj/ETC/sepolicy_intermediates`.
  - Copy of policy with all `dontaudit` rules stripped.
  - But do not allow everything logged when using this policy!
Dealing with neverallow failures

- Policy contains a set of neverallow rules to prevent adding unsafe allow rules.
- Checked by checkpolicy during policy build.
  - New CTS test will also check on device.
- 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...
Neverallow Failure Example

- **rmt_storage** reads/writes raw partitions.
  
  ```
  allow rmt block_device:blk_file rw_file_perms;
  ```

- This violates a neverallow rule and will fail to build.
  
  ```
  neverallow on line 223 of external/sepolicy/domain.te (or line 7284 of policy.conf) violated by allow rmt block_device:blk_file { read write open };
  ```
Never allow Failure Resolution

- Only allow access to specific partitions.
- `device/lge/hammerhead/sepolicy/device.te`:
  type modem_block_device, dev_type;
- `device/lge/hammerhead/sepolicy/file_contexts`:
  /dev/block/mmcblk0p1[23] u:object_r:modem_block_device:s0
- `device/lge/hammerhead/sepolicy/rmt.te`:
  allow rmt modem_block_device:blk_file rw_file_perms;
Analyzing Policy

- Compiled policy file
  - `out/target/product/<product>/root/sepolicy`
  - `/sepolicy` (on device)

- SELinux tools available in Linux distributions
  - `yum install "setools*"` (Fedora)
  - `apt-get install setools` (Ubuntu >= 12.10)
  - `seinfo, sesearch, sediff, apol`

- Some tools included in AOSP master
  - `dispol, sepolicy-analyze`
What's Next for SELinux in Android?

- Disclaimer: Speculative, merely based on what is presently merged in the Android Open Source Project (AOSP) master branch.
- Some of these changes may not have been merged in time for the next Android release or may be reverted before release.
- We have no insight into what Google is doing in their internal tree, so there may be other SELinux changes coming in the next release.
What's Next for SELinux in Android?

- All domains will be enforcing (in -user builds).
- Many more domains have been confined.
- Unconfined is no longer all powerful.
- mmap/mprotect PROT_EXEC is more restricted.
- Recursive restorecon support has been added.
- New CTS tests for SELinux have been added.
- Denials available via logcat.
- Fewer app domains by default.
All Domains Enforcing

- New permissive_or_unconfined() policy macro.
- Per-domain permissive if -userdebug or -eng.
- Unconfined but enforcing if -user.
- Enables policy debugging in debug/eng builds.
- Makes domain enforcing with unconfined rules in user builds.
- Use this instead of direct permissive <domain>; declarations in your .te files.
- Remove permissive_or_unconfined() call once all denials have been addressed in your policy.
Confined+Enforcing Domains

- 4 (out of 48) in Android 4.4.2 for Nexus 5.
- 43 (out of 61) in current AOSP master for Nexus 5.
- Primarily domains for services.
- Also includes shell (ADB shell) and isolated_app (isolatedProcess, e.g. Chrome sandbox) domains.
- Also includes domains for recovery.
  - Requires updating init.rc for recovery.
  - See bootable/recovery/etc/init.rc in AOSP master.
Unconfined Domain Lockdown

• Only init can load SELinux policy or change enforcing mode.
• Nothing can read/write /dev/kmem or /dev/mem.
• Only init can set kernel usermodehelpers and proc security settings.
• Nothing can ptrace init.
• Nothing can map low memory.
Unconfined Domain Lockdown

• No (re)mounting filesystems (*) except as allowed by policy.
• No raw I/O or mknod (*).
• No kernel module loading (*).
• No ptrace attach or access to sensitive /proc/pid files (*).
• No execute to files outside of rootfs or /system (*)
• No transitions to other domains (*).
mmap/mprotect PROT_EXEC lockdown

• No PROT_EXEC anonymous mappings or (modified) private file mappings except as allowed by policy.
• Still must be allowed for the Dalvik-based components (Java) for JIT.
• But can be locked down for system services.
Recursive restorecon

- New `restorecon_recursive` init built-in command.
- `restorecon_recursive /data` called by `init.rc`.
  - Fixes labels on existing userdata.
  - Only runs once per change to `file_contexts`.
- Similar support in `PMS/installld` for `/data/data`.
  - Only runs once per change to `seapp_contexts`.
- `init.<board>.rc` files can call `restorecon_recursive` for other partitions (e.g. `/persist`, `/factory`).
- No more unlabeled files!
New CTS tests

• SELinuxTest
  − Policy must not contain any booleans.
  − Policy must pass a core set of neverallow & allow checks.

• SELinuxDomainTest
  − Running services must have the correct domain, executable, and cardinality.
  − No processes other than init in the init domain.
  − No non-kernel threads in the kernel domain.
Denials via logcat

• logd
  – New userspace log daemon created by Google.

• Includes audit support.
  – Derived from SE for Android auditd code.

• SELinux denials now visible in logcat!
  – Look in logcat rather than dmesg.
App Domain Reduction

• Dropped separate app domains for build keys other than platform certificate (shared_app, media_app, release_app).

• Coalesced to untrusted_app domain.

• Can still split out specific apps via mac_permissions.xml and seapp_contexts.
App Labeling by Certificate + Package

• **mac_permissions.xml:**

```xml
<signer signature="@BROWSER" >
  <package name="com.android.browser" >
    <seinfo value="browser" />
  </package>
</signer>
```

• **seapp_contexts:**

```
user=_app seinfo=browser domain=browser_app type=app_data_file
```
Beyond SELinux

• Middleware MAC
• TrustZone and Virtualization
Middleware MAC

• Install-time MAC: Whitelist/disable apps.
  – Even pre-installed ones.

• Enterprise Ops: Control app operations.
  – Extension to AppOps mechanism introduced in 4.3.
  – Obsoletes our older permission revocation mechanism.

• Intent Firewall: Control app interactions.
  – Introduced in Android 4.3.
  – Obsoletes our older intent MAC mechanism.
TrustZone and Virtualization

- Leveraging TrustZone to enable trusted boot, sealed storage and remote attestation.
- Leveraging hardware virtualization to confine driver vulnerabilities and to enable protection and assured invocation of critical services.
- See my NDSS'13 keynote: *Laying a Secure Foundation for Mobile Devices*
Questions?

• Send email to seandroid-list-join@tycho.nsa.gov to join the public SE for Android mailing list.
• Private email just to our SE for Android team: seandroid@tycho.nsa.gov
• Source code: https://bitbucket.org/seandroid
• Wiki: http://selinuxproject.org/page/SEforAndroid
Other Resources

- NSA SELinux docs, http://www.nsa.gov/research/selinux/docs.shtml
- SELinux community wiki, selinuxproject.org
Reference Material

• Extra slides that may be helpful for reference.
Type Enforcement (TE) Allow Rules

- **allow** `<domains>` `<types>`:`<classes>` { `<permissions>` };
  - `<domains>`: process domains
  - `<types>`: object types
  - `<classes>`: kind of objects, e.g. process, file, dir (directory), ...
  - `<permissions>`: operations on `<classes>`, e.g. read, write, create, execute, ...

- Classes and permissions defined by **security_classes**, **access_vectors**.

- Common groupings provided by **global_macros**, **te Macros**.
Type Enforcement (TE) Transition Rules

• **type_transition** `<domains> <types>:<classes> <new-type> <optional-component-name>`;
  - `<domains>`: process domains
  - `<types>`: types of related objects (e.g. executable, parent directory)
  - `<classes>`: kinds of object, e.g. process, file, dir (directory), ...
  - `<new-type>`: new type to assign to process or object
  - `<optional-component-name>`: optional file name for name-based transition

• Helper macros in **te_macros** (*init_daemon_domain*, *domain_auto_trans*, *file_type_auto_trans*).
File Type Transition Example

• Label /data/misc/wifi/sockets with \texttt{wpa\_socket} type when created by \texttt{wpa\_supplicant} (wpa.te):
  \begin{verbatim}
  type_transition wpa wifi_data_file:dir wpa_socket "sockets";
  \end{verbatim}

• Preserve upon a \texttt{restorecon\_recursive} (file\_contexts).

\begin{verbatim}
/data/misc/wifi/sockets(./.*)?  u:object_r:wpa_socket:s0
\end{verbatim}
Other Policy Source Files

- **fs_use**
  - Tells SELinux how to label filesystem types.
  - Kernel code and configuration must support the specified behavior or it will not work!

- **genfs_contexts**: Generic filesystem security contexts
  - Labels for filesystems that do not support labeling.
  - Per-file labeling for /proc files.
Other Policy Source Files

• **mls**: Multi-level Security (MLS) configuration
  - Only relevant if assigning levels using `level=` or `levelFrom=` in `seapp_contexts`.
  - Not relevant in AOSP policy.

• **roles, users**
  - Role and (SELinux) user declarations.
  - Only one of each in AOSP policy.
Other Policy Source Files

- Do **NOT** modify any of the following files!
  - They are linked to kernel definitions.

- **security_classes, access_vectors**
  - Define class and permission definitions.

- **initial_sids, initial_sid_contexts**
  - Predefined security contexts used by kernel.

- **policy_capabilities**
  - Enables optional kernel/policy features.