The Case for Security Enhanced (SE) Android

Stephen Smalley
Trusted Systems Research
National Security Agency

Background / Motivation

- Increasing desire to use mobile devices throughout the US government.
- Increasing interest in Android as an open platform with broad market adoption.
- Need for improved security in mobile operating systems.

What is SE Android?

- A project to identify and address critical gaps in the security of Android.
  - Initially, enabling the use of SELinux in Android.
  - But not limited in scope to SELinux alone.
- A reference implementation.
  - Initially, a worked example of how to enable and apply SELinux to Android.

SE Android: Use Cases

- Prevent privilege escalation by apps.
- Prevent data leakage by apps.
- Prevent bypass of security features.
- Enforce legal restrictions on data.
- Protect integrity of apps and data.
- Beneficial for consumers, businesses, and government.

Android's Not Linux

- Very divergent from typical Linux.
- Almost everything above the kernel is different.
  - Dalvik VM, application frameworks
  - bionic, init/ueventd
- Even the kernel is different.
  - Binder, Ashmem, ...

Android Security Model

- Application-level permissions model.
  - Controls access to app components.
  - Controls access to system resources.
  - Specified by app writers and seen by users.
- Kernel-level sandboxing and isolation.
  - Isolate apps from each other and from system.
  - Prevent bypass of app permissions model.
  - Normally invisible to users and app writers.

Android & Kernel Security

- App isolation and sandboxing is enforced by the Linux kernel.
  - The Dalvik VM is not a security boundary.
  - Any app can run native code.
- Relies on Linux discretionary access control (DAC).

Discretionary Access Control

- Typical form of access control in Linux.
- Access to data is entirely at the discretion of the owner/creator of the data.
- Some processes (e.g. uid 0) can override and some objects (e.g. sockets) are unchecked.
- Based on user & group identity.
- Limited granularity, coarse-grained privilege.

Android & DAC

- Restrict use of system facilities by apps.
  - e.g. bluetooth, network, sdcard
  - relies on kernel modifications
- Isolate apps from each other.
  - unique user and group ID per installed app
  - assigned to app processes and files
- Hardcoded, scattered “policy”.

SELinux: What is it?

- Mandatory Access Control (MAC) for Linux.
  - Enforces a system-wide security policy.
  - Over all processes, objects, and operations.
  - Based on security labels.
- Can confine flawed and malicious applications.
  - Even ones that run as “root” / uid 0.
- Can prevent privilege escalation.

How can SELinux help Android?

- Confine privileged daemons.
  - Protect from misuse.
  - Limit the damage that can be done via them.
- Sandbox and isolate apps.
  - Strongly separate apps from one another.
  - Prevent privilege escalation by apps.
- Provide centralized, analyzable policy.

What can't SELinux mitigate?

- Kernel vulnerabilities, in general.
  - Although it may block exploitation of specific vulnerabilities.
- Anything allowed by security policy.
  - Good policy is important.
  - Application architecture matters.
    - Decomposition, least privilege.

SE Android: Goals

- Integrate SELinux into Android in a comprehensive and coherent manner.
- Demonstrate useful security functionality in Android using SELinux.
- Improve the suitability of SELinux for Android.
- Identify and address other security gaps in Android.

SE Android: Challenges

- Kernel
  - No support for per-file security labeling (yaffs2).
  - Unique kernel subsystems lack SELinux support.
- Userspace
  - No existing SELinux support.
  - Sharing through framework services.
- Policy
  - Existing policies unsuited to Android.

Kernel Support

- Enabled SELinux and its dependencies.
  - AUDIT, XATTR, SECURITY
- Implemented per-file security labeling for yaffs2.
  - Using recent support for extended attributes.
  - Enhanced to label new inodes at creation.
- Analyzed and instrumented Binder for SELinux.
  - Permission checks on IPC operations.

Userspace Support

- Minimal port of SELinux userspace.
- Labeling support in filesystem tools.
  - Labeling at image build time.
- Extensions for init, ueventd, toolbox, installd, dalvik, zygote.
- JNI bindings for SELinux APIs.
- Settings support for managing SELinux.

Policy Configuration

- Small TE policy written from scratch.
- Confined domains for daemons and apps.
- MLS categories for app isolation.
- New configuration for app labeling.
- No policy writing for app writers.
- Normally invisible to users.

SE Android: Size

	Non-SE	SE	Increase
boot	3444K	3584K	+140K
system	161620K	161668K	+48K
recovery	3776K	3916K	+140K

- full_crespo4g-userdebug

Current State

- Working reference implementation
  - originally based on Gingerbread / 2.3.x.
  - now based on Android Open Source Project (AOSP) master branch (4.0.3+)
  - tested on emulator, Nexus S, Motorola Xoom
- Still a long way from a complete solution
  - But let's see how well it does...

Case Study: vold

- vold - Android volume daemon
  - Runs as root.
  - Manages mounting of disk volumes.
  - Receives netlink messages from kernel.
- CVE-2011-1823
  - Does not verify message origin.
  - Uses signed integer without checking < 0.
- Demonstrated by GingerBreak exploit.

GingerBreak: Overview

- Collect information needed for exploitation.
  - Identify the vold process.
  - Identify addresses and values of interest.
- Send carefully crafted netlink message to vold.
  - Trigger execution of exploit binary.
  - Create a setuid-root shell.
- Execute setuid-root shell.
- Got root!

GingerBreak: Would SELinux help?

- Let's walk through it again with SE Android.
- Using the initial example policy we developed.
  - Before we read about this vulnerability and exploit.
  - Just based on normal Android operation and policy development.

GingerBreak vs SELinux #1

- Identify the vold process.
  - /proc/pid/cmdline of other domains denied by policy
- Existing exploit would fail here.
- Let's assume exploit writer recodes it based on some other means.

GingerBreak vs SELinux #2

- Identify addresses and values of interest.
  - /system/bin/vold denied by policy.
  - /dev/log/main denied by policy.
- Existing exploit would fail here.
- Let's assume that exploit writer recodes exploit based on some other means.

GingerBreak vs SELinux #3

- Send netlink message to vold process.
  - netlink socket create denied by policy
- Existing exploit would fail here.
- No way around this one - vulnerability can't be reached.
- Let's give the exploit writer a fighting chance and allow this permission.

GingerBreak vs SELinux #4

- Trigger execution of exploit code by vold.
  - execute of non-system binary denied by policy
- Existing exploit would fail here.
- Let's assume exploit writer recodes exploit to avoid executing a separate binary.

GingerBreak vs SELinux #5

- Create a setuid-root shell.
  - remount of /data denied by policy
  - chown/chmod of file denied by policy
- Existing exploit would fail here.
- Let's give the exploit writer a fighting chance and allow these permissions.

GingerBreak vs SELinux #6

- Execute setuid-root shell.
  - SELinux security context doesn't change.
  - Still limited to same set of permissions.
  - No superuser capabilities allowed.
- Exploit “succeeded”, but didn't gain anything.

GingerBreak: Conclusion

- SELinux would have stopped the exploit six different ways.
- SELinux would have forced the exploit writer to tailor the exploit to the target.
- SELinux made the underlying vulnerability completely unreachable.
  - And all vulnerabilities of the same type.

Case Study: zygote

- zygote - Android app spawner
  - Runs as root.
  - Receives requests to spawn apps over a socket.
  - Uses setuid() to switch to app UID.
- Did not check/handle setuid() failure.
  - Can lead to app running as root.
- Demonstrated by Zimperlich exploit.

Zimperlich: Overview

- Fork self repeatedly to reach RLIMIT_NPROC for app UID.
- Spawn app component via zygote.
- Zygote setuid() call fails.
- App runs with root UID.
  - Re-mounts /system read-write.
  - Creates setuid-root shell in /system.

Zimperlich vs SELinux

- zygote setuid() would still fail.
- Security context changes upon setcon().
  - Not affected by RLIMIT_NPROC.
- App runs in unprivileged security context.
  - No superuser capabilities.
  - No privilege escalation.

Other Root Exploits

- ueventd / Exploid, vold / zergRush
  - similar to vold / GingerBreak
- adbd / RageAgainstTheCage
  - similar to zygote / Zimperlich
- ashmem / KillingInTheNameOf
  - mprotect PROT_WRITE of property space
- Likewise blocked by SE Android.

Case Study: Skype

- Skype app for Android.
- CVE-2011-1717
  - Stores sensitive user data without encryption with world readable permissions.
    - account balance, DOB, home address, contacts, chat logs, ...
- Any other app on the phone could read the user data.

SELinux vs Skype vulnerability

- Classic example of DAC vs. MAC.
  - DAC: Permissions are left to the discretion of each application.
  - MAC: Permissions are defined by the administrator and enforced for all applications.
- All apps denied read to files created by other apps.
  - Each app and its files have a unique SELinux category set.

Was the Skype vulnerability an isolated incident?

- Lookout Mobile Security
- Symantec Norton Mobile Security
- Wells Fargo Mobile app
- Bank of America app
- USAA banking app

Case Studies: Conclusion

- Android security would benefit from SE Linux.
  - Android needs Mandatory Access Controls (MAC).
  - SELinux would have mitigated a number of Android exploits and vulnerabilities.

Application Layer Security

- SE Android presently limited to kernel-level MAC.
  - + a few permission checks in the zygote.
- Also need MAC for the Android permissions model.
  - Requires extensions to the frameworks.
- Related work:
  - Sven Bugiel et al, *Towards Taming Privilege Escalation Attacks on Android,* NDSS '12*.*

Timeline of Events

- First public release Jan 6 2012.
- First submission to AOSP Jan 13 2012.
- bionic patches merged Jan 20 2012.
- Other patches in progress.
  - Coding Style, minor cleanups.
  - Wrap with HAVE_SELINUX conditionals.

What's Next?

- Finish upstreaming to AOSP.
- MAC for Android permissions.
- Runtime policy management.
- Further integration (kernel and userland).
- Identifying and addressing other security gaps.

Questions?

- http://selinuxproject.org/page/SEAndroid
- SELinux mailing list:
  - selinux@tycho.nsa.gov
- NSA SE Android team:
  - seandroid@tycho.nsa.gov
- My email:
  - sds@tycho.nsa.gov