RESEARCH DIRECTORATE
ADVANCING INTELLIGENCE THROUGH SCIENCE

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.
- A reference implementation produced by the project.
- Initially, enabling and applying SELinux in Android.
- Not limited in scope to SELinux alone.

SE Android is not...

- A government-specific Android.
- A fork of Android.
- A complete solution for all security concerns.
- A product.
- Specially evaluated or approved for use.

SE Android is...

- Security enhancements to Android.
- Addressing platform security.
  - Focused on critical gaps not otherwise being addressed.
- Designed for wide applicability.
- Targeting mainline Android adoption.

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.
  - All apps forked from the same process (zygote).
  - 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

- xattr and AT_SECURE support in bionic.
- Minimal port of SELinux libraries and tools.
- Labeling support in filesystem tools.
- Policy loading, device & socket labeling (init).
- App security labeling (zygote, dalvik, installd).
- JNI bindings for SELinux APIs.
- Management app.

Policy Configuration

- Confined domains for system services.
- Small number of discrete domains for apps.
- MLS categories for app isolation.
- Key properties:
    - Small, fixed policy.
    - No policy writing for app developers.
    - Invisible to users.

Recent Advances

- Recovery console / updater support.
- Runtime policy management support.
- SELinux/MAC permission checks for init property service.
- Install-time MAC.
- Update to Android 4.1/JellyBean.

Current State

- Working reference implementation.
  - Based on Android Open Source Project (AOSP).
  - Tracking ICS (4.0.4), JB (4.1.1), & master.
- Demonstrable on real devices.
  - Nexus S, Galaxy Nexus phones
  - Xoom and Nexus 7 tablets

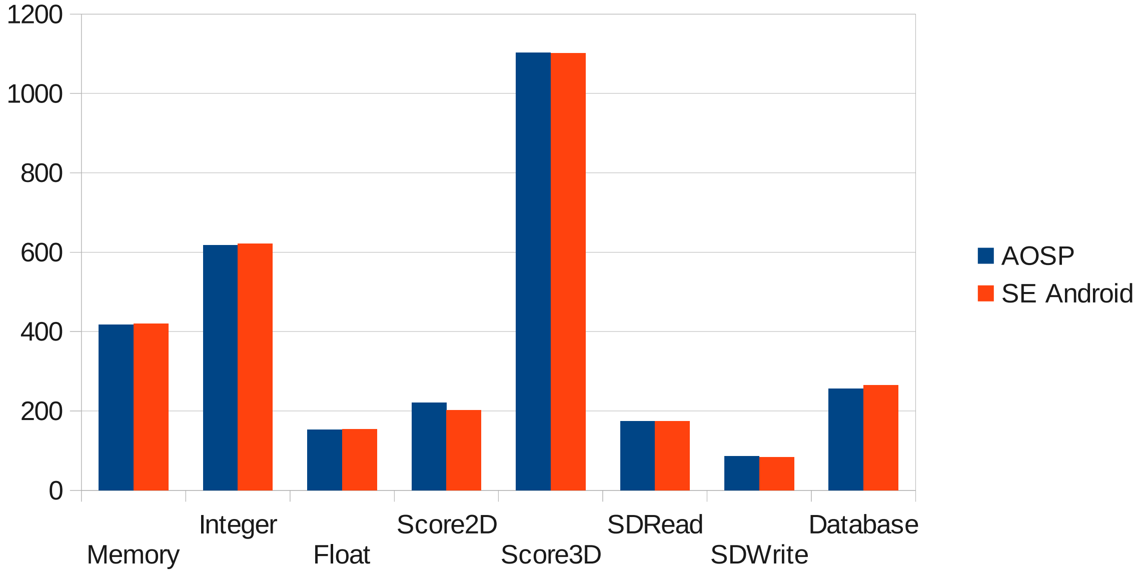
Size Comparison (crespo4g, 4.0.4)

	AOSP	**SE Android**	**Increase**
boot	3444K	3596K	+152K
system	161692K	161816K	+124K
recovery	3776K	3944K	+168K

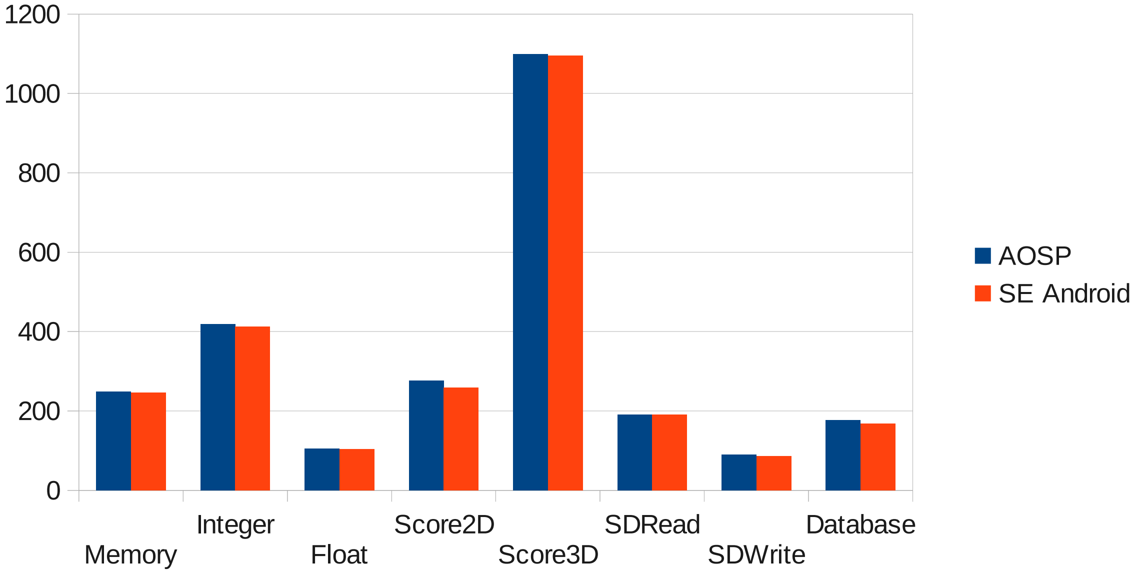
Size Comparison (crespo4g, 4.1.1)

	AOSP	SE Android	Increase
boot	3964K	4156K	+192K
system	178780K	178904K	+124K
recovery	4308K	4512K	+204K

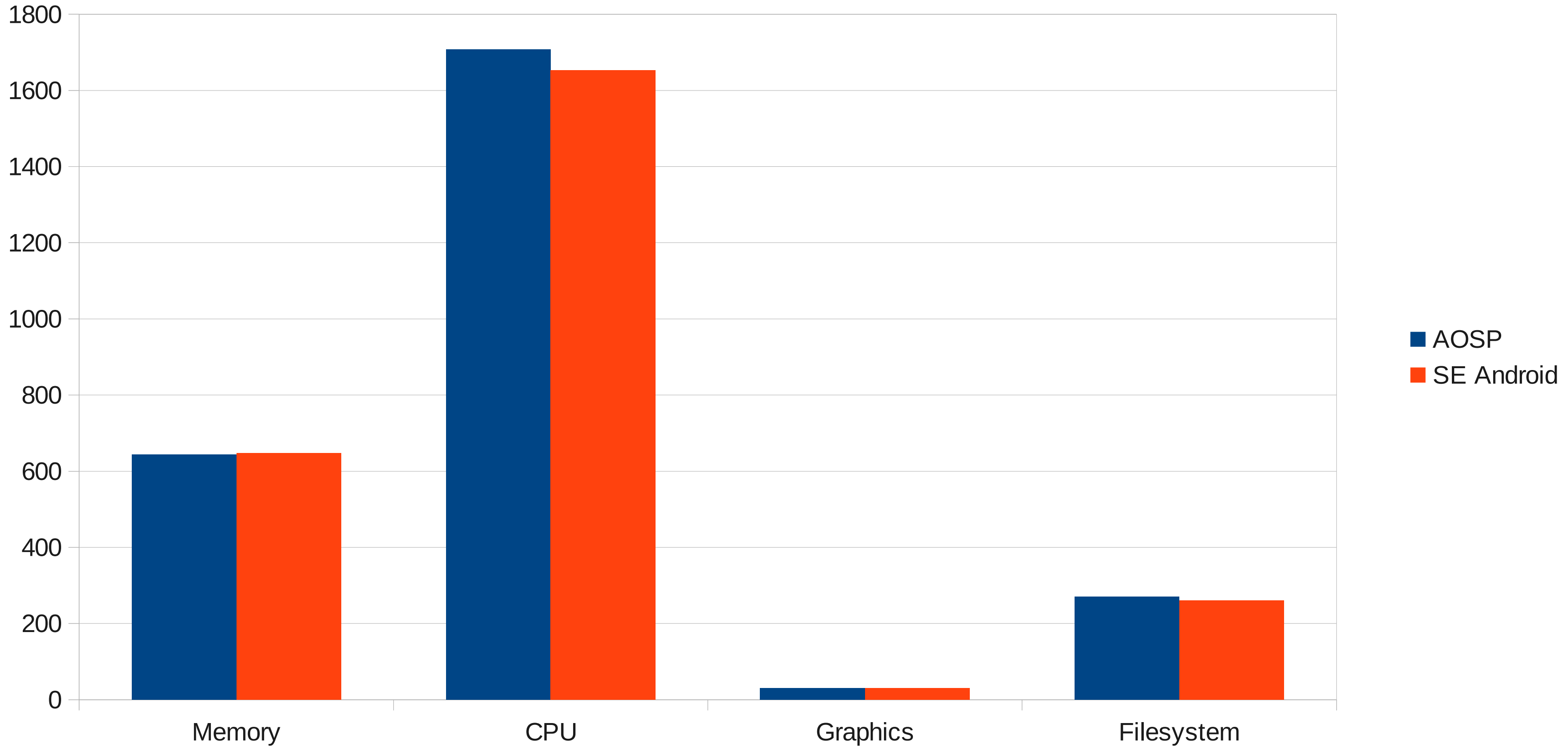
AnTuTu (crespo4g, 4.0.4)



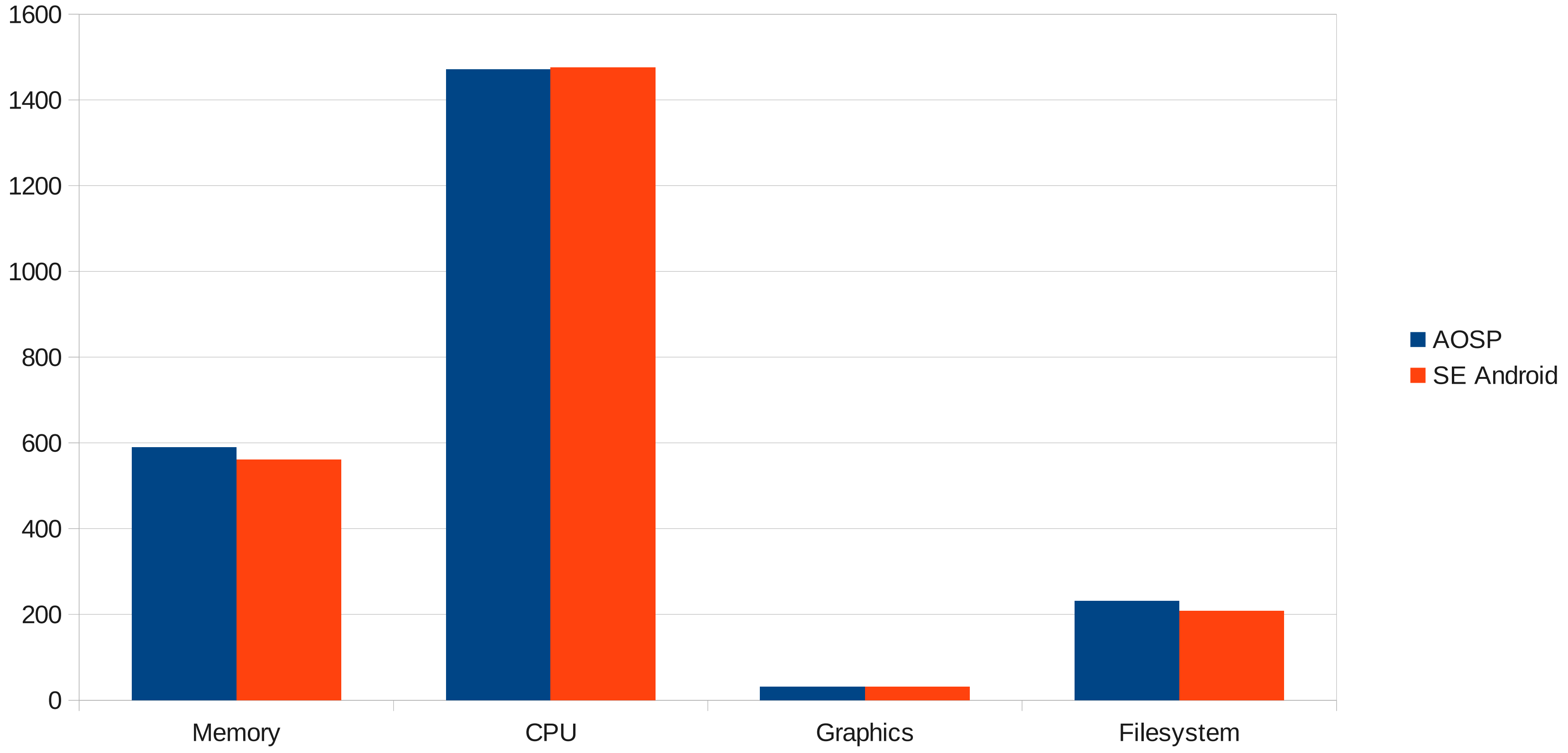
AnTuTu (crespo4g, 4.1.1)



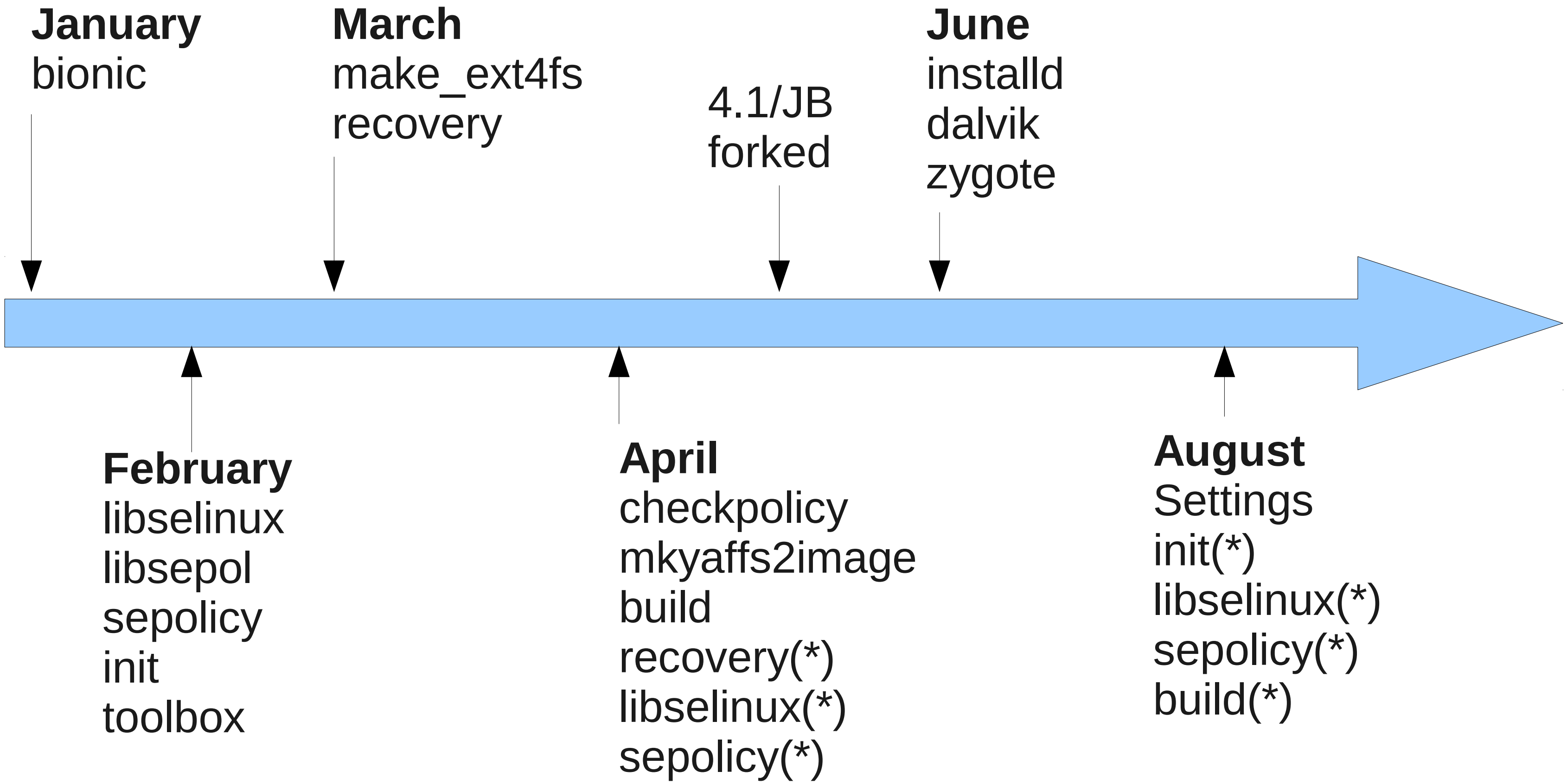
Softweg (crespo4g, 4.0.4)



Softweg (crespo4g, 4.1.1)



AOSP merging



January
bionic

February
libselinux
libsepol
sepolicy
init
toolbox

March
make_ext4fs
recovery

April
checkpolicy
mkyaffs2image
build
recovery(*)
libselinux(*)
sepolicy(*)

4.1/JB
forked

June
installd
dalvik
zygote

August
Settings
init(*)
libselinux(*)
sepolicy(*)
build(*)

AOSP merge status

- Before 4.1 freeze: 12 changes merged.
- Since 4.1 freeze: 16 changes merged.
- Spanning 10 different projects.
- 3 open changes pending.
- Not yet submitted: install-time MAC, kernel/*, device/*.

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 vs SE Android

- 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 SE Android #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 SE Android #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 SE Android #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 SE Android #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 SE Android #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 SE Android #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

- SE Android would have stopped the exploit six different ways.
- SE Android would have forced the exploit writer to tailor the exploit to the target.
- SE Android made the underlying vulnerability completely unreachable.
  - And all vulnerabilities of the same type.
  - e.g. Exploid exploit against ueventd.

Case Study: /proc/pid/mem

- /proc/pid/mem
  - Kernel interface for accessing process memory.
  - Write access enabled in Linux 2.6.39+.
- CVE-2012-0056
  - Incorrect permission checking.
  - Induce setuid program into writing own memory.
- Demonstrated by mempodroid exploit.

Mempodroid: Overview

- Some complexity omitted.
- Exploit invokes setuid root program with open fd to /proc/pid/mem as stderr and shellcode as argument.
- Setuid root program overwrites self with shellcode when writing error message.
- Shell code sets uid/gid to 0 and execs shell or command.

Mempodroid vs SE Android

- Write to /proc/pid/mem will still succeed.
- But setuid root program runs in caller's security context (or policy-defined one).
  - Still restricted by SELinux policy.
  - No privilege escalation.

Other Root Exploits

- uevendt / Exploid, vold / zergRush
  - similar to vold / GingerBreak
- adbd / RageAgainstTheCage, zygote / Zimperlich
  - RLIMIT_NPROC setuid() failure
- 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.

SE Android 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
  - LOOK-11-001
- Opera Mobile
  - Cache Poisoning XAS
- Android SQLite Journal
  - CVE-2011-3901

Case Studies: Conclusion

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

What's Next?

- Middleware MAC (MMAC).
- Device admin support for policy.
- Analyze other Android-specific drivers.
- Optimize SELinux for Android.
- Trusted input & display.

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