

# Android Security

## New Threats, New Capabilities

Jonathan Levin, @Technogeeks

<http://technogeeks.com/>

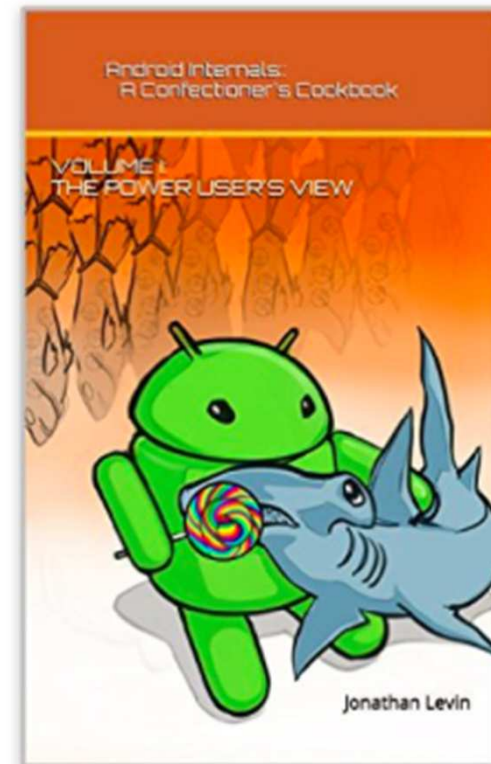


# About this talk

- Provides tour of Android security features:
  - Dālvik level security (permissions, IFW)
  - Linux inheritance (permissions, capabilities)
  - SELinux and SEAndroid
  - Rooting and System Security
- Covered in “Android Internals: A Confectioner’s Cookbook”
  - Chapter 8, to be exact

# The Book

- “Android Internals: A Confectioner’s Cookbook”
  - 深入解析Android 操作系统 - Coming in Chinese (by Dec 2015)
- Parallels “Mac OS X and iOS Internals” (but for Android)
  - BTW MOXiI is getting a 2<sup>nd</sup> Edition (10.10/iOS 8) – Nov 2015!
- [Volume I is available!](#) Updated for M PR1
  - M changes require rewrite for Volume II
- <http://newandroidbook.com/>
  - FAQ, TOC and plenty of bonus materials
  - Check out technologeeks.com courses!



# Attack Surface

- Threat models for mobiles consider three main vectors\*:
  - Rogue applications (malware)
    - Sandbox applications
    - Enforce Strong Permissions
    - Harden OS Component Security
  - Rogue user (device theft, or unauthorized root)
    - Lock Screen
    - Secure Boot Process
    - Encrypt User Data

App Security

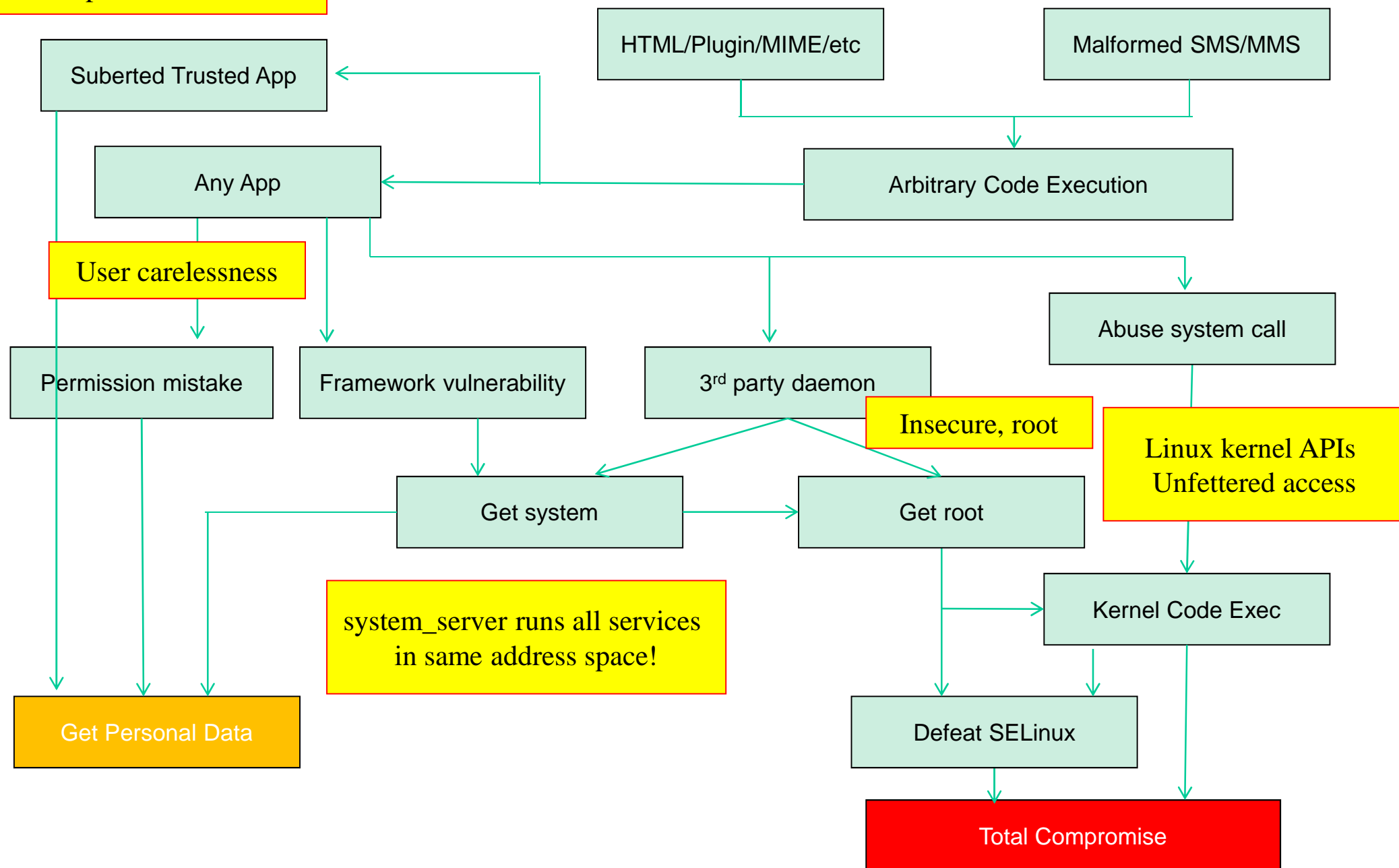
Device Security

---

\* We'll discount the remote attack vector in this talk, since it isn't mobile specific

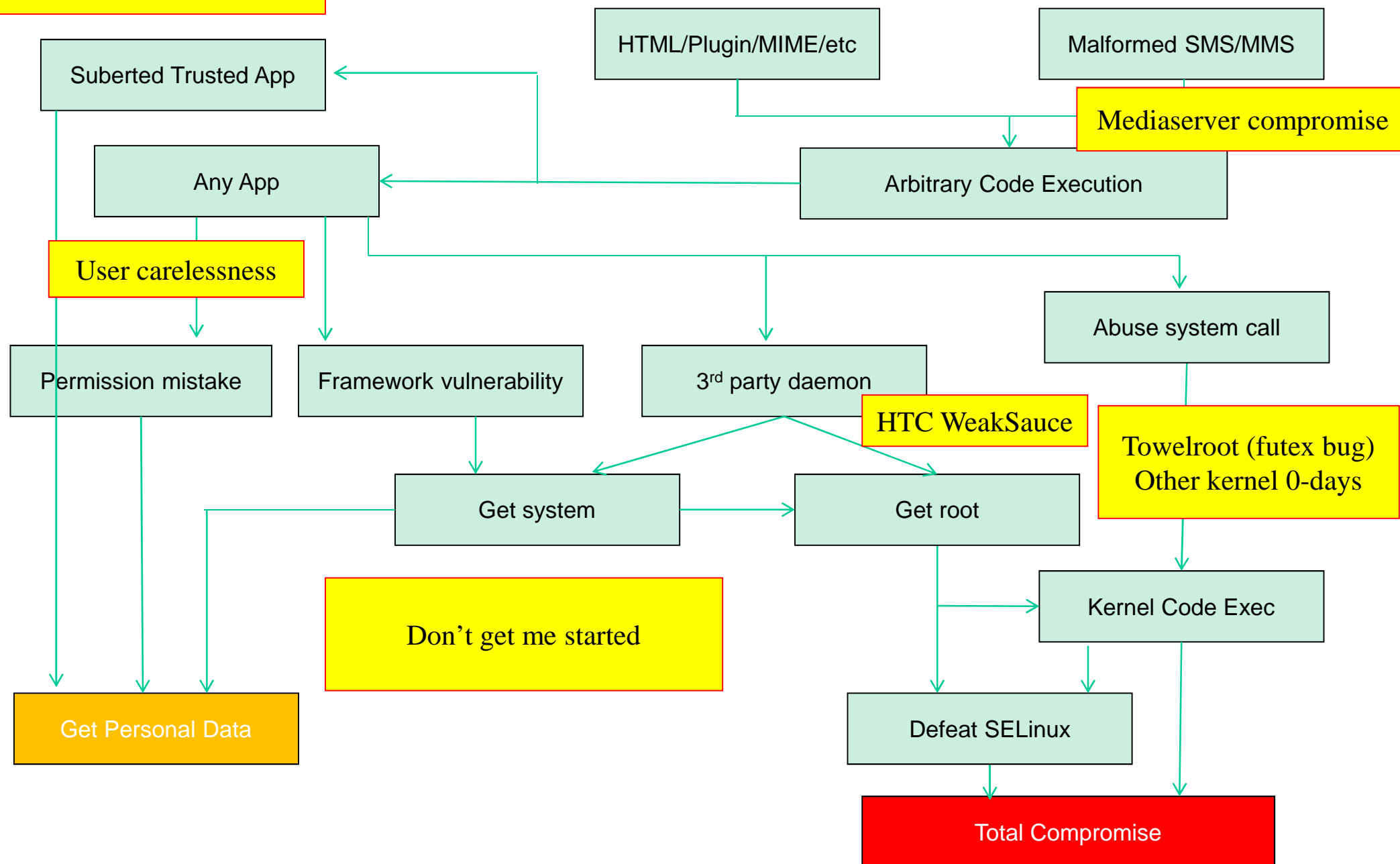
# Attack Vectors (simplified)

Trusted app may be compromised as well

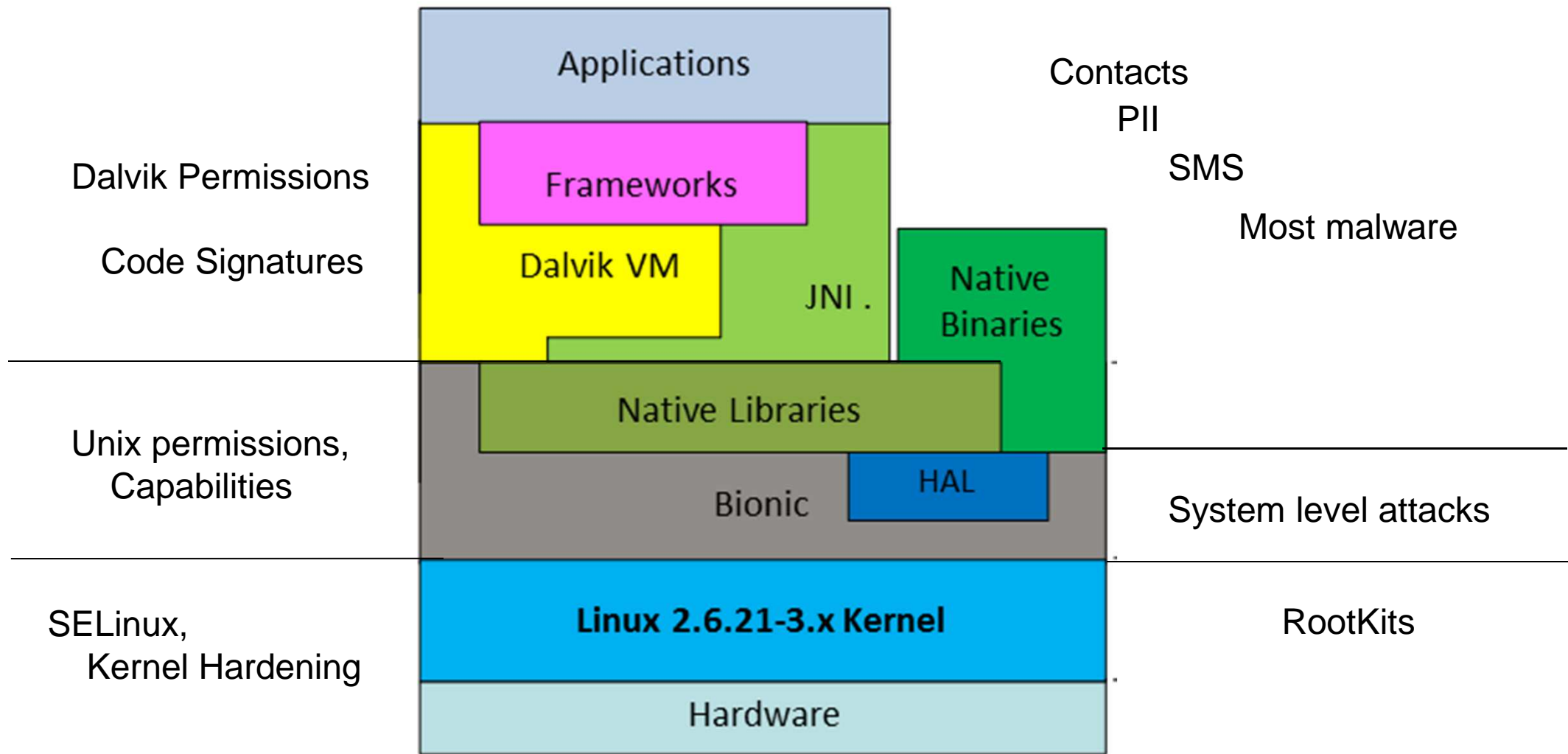


# Attack Vectors (exploited)

Mediaserver compromise



# Android Architecture



# Android Application Security Model

- Android's security is derived from that of Linux and Java
- Linux inheritance:
  - Applications run as separate UIDs
  - Kernel supports capabilities
  - Network access filtered in kernel by UserID
- Java Inheritance:
  - Dalvik VM provides sandbox for applications
  - Declarative security model for operations



# Application Security Model: Dalvik

- Permissions can be declared in the Application Manifest

<http://developer.android.com/reference/android/Manifest.permission.html>

- Permissions groups in permission sets:

Permission Set	For ..
Normal	Every day, security insensitive operations
Dangerous	Potentially hazardous operations e.g. SMS sending or dialing
Signature	Signed code only
SignatureOfSystem	Signed code + hardware access

- Applications can further define own custom permissions
- Package Manager interface (pm) can list/manipulate

# Top 20 Malware Permissions

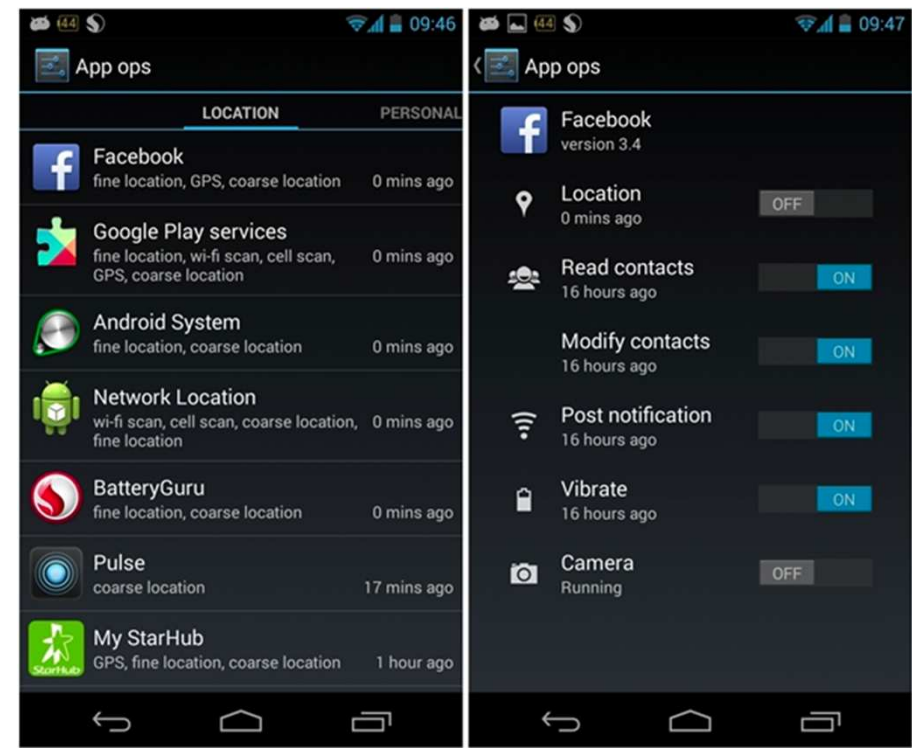
- From [Wei, Gomez et al \(2012\)](#) :

Permission	% of apps using it
INTERNET	97.8%
READ_PHONE_STATE	93.6%
ACCESS_NETWORK_STATE	81.2%
WRITE_EXTERNAL_STORAGE	67.2%
ACCESS_WIFI_STATE	63.8%
READ_SMS	62.7%
RECEIVE_BOOT_COMPLETED	54.6%
WRITE_SMS	52.2%
SEND_SMS	43.9%
VIBRATE	38.3%
ACCESS_COARSE_LOCATION	38.1%
READ_CONTACTS	36.3%
ACCESS_FINE_LOCATION	34.3%
WAKE_LOCK	33.7%
CALL_PHONE	33.7%
CHANGE_WIFI_STATE	31.6%
WRITE_CONTACTS	29.7%
WRITE_APN_SETTINGS	27.7%
RESTART_PACKAGES	26.4%

- All boils down to a security popup dialog (q.v. iBanking)
- Exploits: Social Engineering + User ignorance

# The rise and fall of AppOps

- Introduced in JB (API 18)
- Allows fine grained permission manipulation
  - Similar to iOS's tccd (coincidentally, of course)
- Interface removed by KK 4.4.2 “security update”
- Functionality still there
  - Can restore on modded phone
  - Can also use pm grant/revoke



# Permissions rise again

- Major improvement: revokable permissions
  - Permissions now **checked on use, not on install**
  - Closely follows the model of iOS (TCCd, anyone?)
- Challenge for developers: security exceptions

# Android “Code Signing”

- APK files must be signed.. But.. By whom?
  - Poor model, since self-signed certificates are allowed
  - System APKs are signed with a CA (and also read-only)
- Google warns on non Android-Market App sources
  - .. But malware gets into Android Market all too often.
  - Better to beg forgiveness than ask permission...
- RiskIQ (02/14):
  - Malicious app growth: 388% from 2011 to 2013
  - Google malware removal rate: 60% (2011) → 23% (2013)

# The Intent Firewall

- Little known (and unused) feature of 4.3
  - `base/services/java/com/android/server/firewall/IntentFirewall.java`
- Rulebase built from XML files in `/data/system/ifw`
  - Directory still left empty on most devices
  - IFW registers a `FileObserver()` to watch for rule changes
- `ActivityManager` calls out to `IntentFirewall`'s `checkXXX`:
  - `checkStartActivity`, `checkService` and `checkBroadcast`.

# Linux Security Model

- Linux serves as the first (and last) tier for security:
  - Each application gets unique runtime ID
  - No apps (except system) run as root
  - Groups for Bluetooth, network access

GID	Is authorized to..
AID_NET_BT_ADMIN (3001)	Manage BlueTooth sockets
AID_NET_BT (3002)	Create a BlueTooth socket
AID_INET (3003)	Create an AF_INET or AF_INET6 socket
AID_NET_RAW (3004)	Create raw sockets (for ICMP, or non TCP/UDP)
AID_NET_ADMIN (3005)	Can bring down interfaces, change IPs, etc.
AID_NET_BW_STATS (3006)	Read network bandwidth statistics
AID_NET_BW_ACCT (3007)	Modify network bandwidth statistics

## android\_filesystem\_config.h

- Android's source tree hard-codes “well known” AIDs
  - Reserved for system or native use only
  - Ownership of device and conf files set appropriately
    - /init double checks when started, from /init.rc
- Some system property namespaces keyed to AIDs
- ServiceManager whitelists IDs for some services



# Case Study: system\_server

**Table s2-ssp::** Group memberships of system\_server

gid	#define	Permits
1001	AID_RADIO	/dev/socket/rild, on the other side of which is the Radio Interface Layer Daemon
1002	AID_BLUETOOTH	Bluetooth configuration files
1003	AID_GRAPHICS	/dev/graphics/fb0, the framebuffer
1004	AID_INPUT	/dev/input/*, the device nodes for input devices.
1005	AID_AUDIO	/dev/eac, or other audio device nodes.
1006	AID_CAMERA	Access to camera sockets
1007	AID_LOG	/dev/log/*
1008	AID_COMPASS	Compass and location services
1009	AID_MOUNT	/dev/socket/vold, on the other side of which is the VOLUME Daemon
1010	AID_WIFI	WiFi Configuration files
1018	AID_USB	USB Devices
3001	AID_BT_ADMIN	Creation of AF_BLUETOOTH sockets
3002	AID_NET_BT	Creation of sco, rfcomm, or l2cap sockets
3003	AID_NET_INET	/dev/socket/dnsproxyd, and creation of AF_INET[6] (IPv4, IPv6) sockets
3006	AID_NET_BW_STATS	Reading bandwidth statistics accounting
3007	AID_NET_BW_ACCT	Modifying bandwidth statistics accounting

# Android Application Security Model

- API 16 (JB4.1) adds isolated services:
  - Add `android:isolatedProcess="true"` to service tag
  - System allocates a uid between `AID_ISOLATED_[START|END]`
  - UID is effectively powerless (can't access other services)
  - (Somewhat) similar to iOS's XPC

Dianne Hackborn

7/27/12



I'll go farther: are you writing a web browser? If no, just ignore it. :)

(Actually we can go a little more broadly and say it may be of interest if you are writing an app that downloads arbitrary content from untrusted sources which requires very complicated code to parse and render, complicated enough that it is basically impossible to guarantee you don't have security holes, so it would be useful to have another layer of protection between your app and that content.)

- show quoted text -

- hide quoted text -

On Thu, Jul 26, 2012 at 11:42 PM, Mehrag <gaurav....@gmail.com> wrote:

> Can anyone put some light as what's the real/main advantage of introducing  
> Isolatedprocess tag within Services in JellyBean[Android].

# Linux Capabilities

- Originally introduced as part of POSIX 1.e
- A “Divide and Conquer” approach, restricting operations
- Rather than look at EUID, capability mask is considered
- Some 25+ capabilities, supported by Kernel
- Not enabled by default on Linux, but used in Android

# Capabilities

Defined in `<linux/capability.h>` (see `capabilities(7)`)

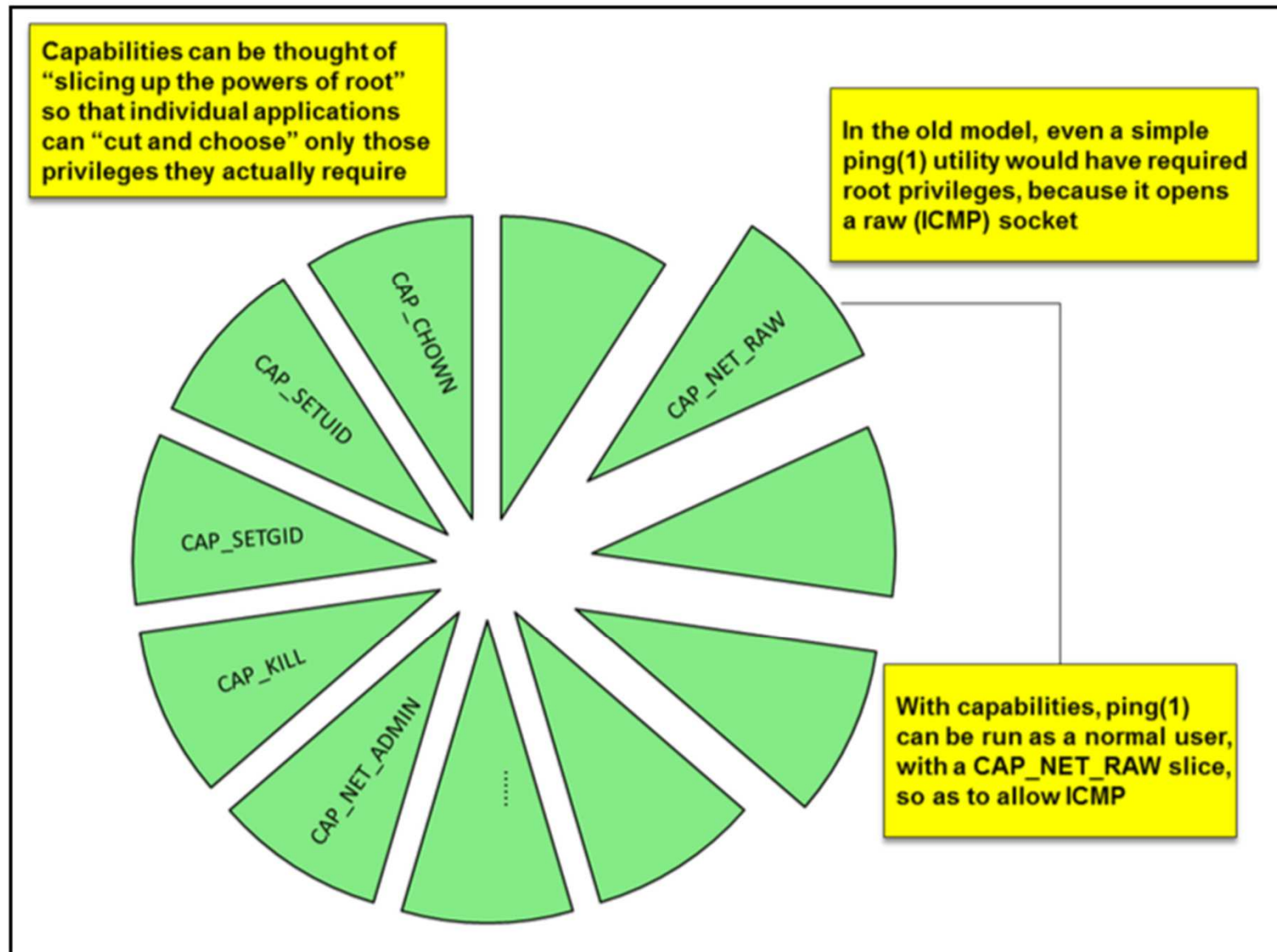
Capability	Application
CAP_CHOWN	Allow arbitrary changes to file UIDs and GIDs
CAP_DAC_OVERRIDE	Bypass Discretionary Access Controls
CAP_DAC_READ_SEARCH	Limited form of CAP_DAC_OVERRIDE
CAP_FOWNER	Ignore sticky bit, or owner-only operations
CAP_FSETID	Don't clear SetUID/SetGID bits on files
CAP_IPC_LOCK	Permit <code>mlock(2)</code> / <code>mlockall(2)</code> / <code>shmctl(2)</code>
CAP_IPC_OWNER	Bypass permission checks on IPC objects
CAP_KILL	Bypass permission operations on signals
CAP_LEASE	Allow file leases (e.g. <code>fcntl(2)</code> )
CAP_LINUX_IMMUTABLE	Allow <code>chattr +i</code> (immutable ext2 file attributes)
CAP_MKNOD	Create device files (using <code>mknod(2)</code> )
CAP_NET_ADMIN	Ifconfig/routing operations
CAP_NET_BIND	Bind privileged (i.e. <code>&lt;1024</code> ) ports
CAP_NET_RAW	Permit <code>PF_RAW</code> and <code>PF_PACKET</code> sockets

# Capabilities

Capability	Application
CAP_SETUID/CAP_SETGID	Enable set[ug]id, GID creds over domain sockets
CAP_SETPCAP	Modify own or other process capabilities
CAP_SYS_ADMIN	Catch-all: quotactl(2), mount(2), swapon(2), sethost/domainname(2), IPC_SET/IPC_RMID, UID creds over domain sockets
CAP_SYS_BOOT	Permit reboot(2)
CAP_SYS_CHROOT	Permit chroot(2)
CAP_SYS_MODULE	Enable create_module(2) and such
CAP_SYS_NICE	For nice(2), setpriority(2) and sched functions
CAP_SYS_PACCT	Permit calls to pacct(2)
CAP_SYS_PTRACE	Enable ptrace(2)
CAP_SYS_RAWIO	Permit iopl(2) and ioperm(2)
CAP_SYS_RESOURCE	Use of reserved FS space, setrlimit(2), etc.
CAP_SYS_TIME	Change system time (settimeofday(2), adjtimex(2)).
CAP_SYS_TTY_CONFIG	Permit vhangup(2)

# Capabilities

**Figure 8-1:** A logical representation of capabilities





# Case Study: system\_server

- system\_server once more provides a great example:

**Table s2-ssc::** Capabilities used by system\_server

capability	#define	Permits
0x20	CAP_KILL	Kill processes not belonging to the same uid
0x400	CAP_NET_BIND_SERVICE	Bind local ports at under 1024
0x800	CAP_NET_BROADCAST	Broadcasting/Multicasting
0x1000	CAP_NET_ADMIN	Interface configuration, Routing Tables, etc.
0x2000	CAP_NET_RAW	Raw sockets
0x10000	CAP_SYS_MODULE	Insert/remove module into kernel
0x800000	CAP_SYS_NICE	Set process priority and affinity
0x1000000	CAP_SYS_RESOURCE	Set resource limits for processes
0x2000000	CAP_SYS_TIME	Set real-time clock
0x4000000	CAP_SYS_TTY_CONFIG	Configure/Hangup tty devices
0x7813C20	Resulting BitMask	

# Other Linux security features

- Removed `/proc/kcore` (and definitely `/dev/kmem`)
- Restrict `dmesg` and kernel pointers (via `sysctl`)
- Code injection counter measures:
  - ASLR (ELF PIE + `randomize_va_space`)
  - DEP
  - Stack Canaries and compiler level protections.



# SE-Linux on Android

- Probably the most important security feature in Android
  - JellyBean introduced in permissive mode
  - KitKat is the first version to enforce
    - Enforcement still minimal (zygote, netd, vold, and installd)
- SE-Linux protects file, property and application contexts
  - Init runs in root:system context (still omnipotent)
  - Can set SE context (using sestatus), enable/disable

# SEAndroid

- The policy is comprised of type enforcement (.te) files
- Files provide labels to define *types* and *domains*
  - types are files and resources (policy objects)
  - domains are for processes (policy subjects)
- Policy can then allow or disallow access by labels

# SEAndroid

- AOSP provides base policy in `external/sepolicy`
- Vendors encouraged to add files in device directory
  - e.g. `device/lge/hammerhead/sepolicy`
- BoardConfig.mk defines:
  - **BOARD\_SEPOLICY\_DIRS**: directory containing TE files
  - **BOARD\_SEPOLICY\_UNION**: name of files to include
- Policy files are copied to device:

File	Usage
file_contexts	Restricts access to files
property_contexts	Restricts access to properties
seapp_contexts	Application (user contexts)
sepolicy	Compiled policy

# SEAndroid

```
# Data files
/adb_keys                u:object_r:rootfs:s0
/default.prop            u:object_r:rootfs:s0..
/fstab\..*               u:object_r:rootfs:s0
..

/sys/class/rfkill/rfkill[0-9]*/state --
u:object_r:sysfs_bluetooth_writable:s0
/sys/class/rfkill/rfkill[0-9]*/type --
u:object_r:sysfs_bluetooth_writable:s0
#####
# asec containers
/mnt/asec(/.*)?          u:object_r:asec_apk_file:s0
/data/app-asec(/.*)?     u:object_r:asec_image_file:s0
```

File	Usage
<b>file_contexts</b>	<b>Restricts access to files</b>
property_contexts	Restricts access to properties
seapp_contexts	Application (user contexts)
sepolicy	Compiled policy

# SEAndroid

```

net.rmnet0      u:object_r:radio_prop:s0
net.gprs        u:object_r:radio_prop:s0
net.ppp         u:object_r:radio_prop:s0
net.qmi         u:object_r:radio_prop:s0
net.lte         u:object_r:radio_prop:s0
net.cdma        u:object_r:radio_prop:s0
gsm.           u:object_r:radio_prop:s0
persist.radio   u:object_r:radio_prop:s0
net.dns         u:object_r:radio_prop:s0
sys.usb.config  u:object_r:radio_prop:s0

ril.           u:object_r:ril_d_prop:s0

...

```

File	Usage
file_contexts	Restricts access to files
<b>property_contexts</b>	<b>Restricts access to properties</b>
seapp_contexts	Application (user contexts)
sepolicy	Compiled policy

# SEAndroid

```
isSystemServer=true domain=system
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=_app domain=untrusted_app type=app_data_file levelFrom=none
user=_app seinfo=platform domain=platform_app
type=platform_app_data_file
user=_app seinfo=shared domain=shared_app type=platform_app_data_file
user=_app seinfo=media domain=media_app type=platform_app_data_file
user=_app seinfo=release domain=release_app
type=platform_app_data_file
user=_isolated domain=isolated_app
user=shell domain=shell type=shell_data_file
```

File	Usage
file_contexts	Restricts access to files
property_contexts	Restricts access to properties
<b>seapp_contexts</b>	<b>Application (user contexts)</b>
sepolicy	Compiled policy

# SEAndroid

- The /sepolicy is produced by compiling the .te files
- Loaded policy can be found in /sys/fs/selinux/policy
- Can be decompiled with sedispol (from checkpolicy)

File	Usage
file_contexts	Restricts access to files
property_contexts	Restricts access to properties
seapp_contexts	Application (user contexts)
<b>sepolicy</b>	<b>Compiled policy</b>

# Back to the Attack Surface

- So far we focused on the rogue app/injected code cases
- That was only part of the attack landscape. Remember:
  - Rogue user (device theft, or unauthorized root)
  - Lock Screen
  - Secure Boot Process
  - Encrypt User Data



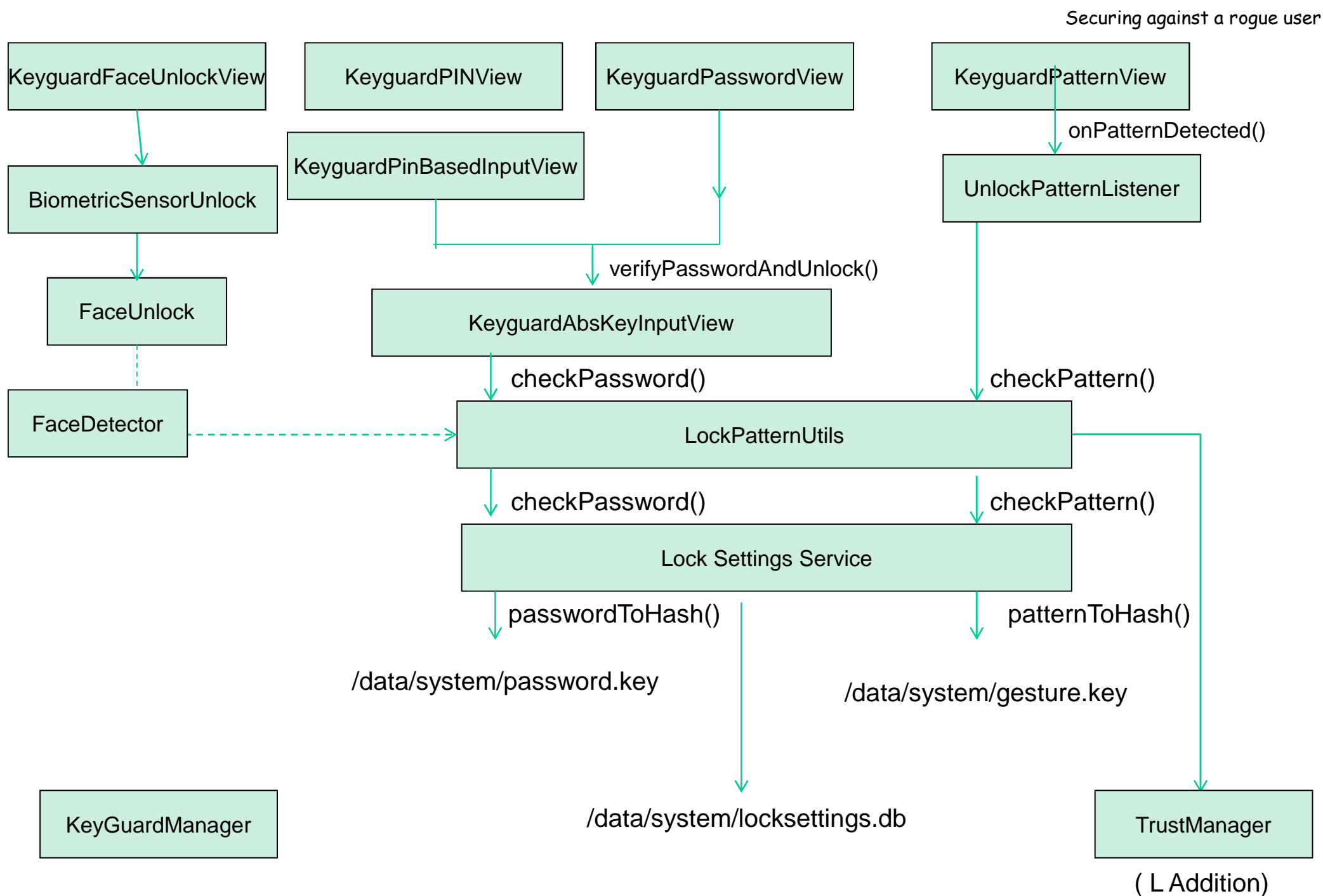
# Lock Screen

- Complementary to device encryption
  - Encryption vs. cold attacks, locking vs. hot attacks
  - Pluggable mechanism:

Mechanism	Notes
Face	Gimmicky, fails miserably with a photo
Gesture	Essentially a PIN, but weaker
PIN	Classic PIN combination
Passcode	Superset of PIN, allows full unicode
Fingerprint (L*)	Varies greatly with vendor supports
Trusted Devices (L)	Unlock via device pairing over NDEF push (“Android Beam”)

---

\* L is the first to “officially” support with FingerPrint service, though Samsung had this in KK



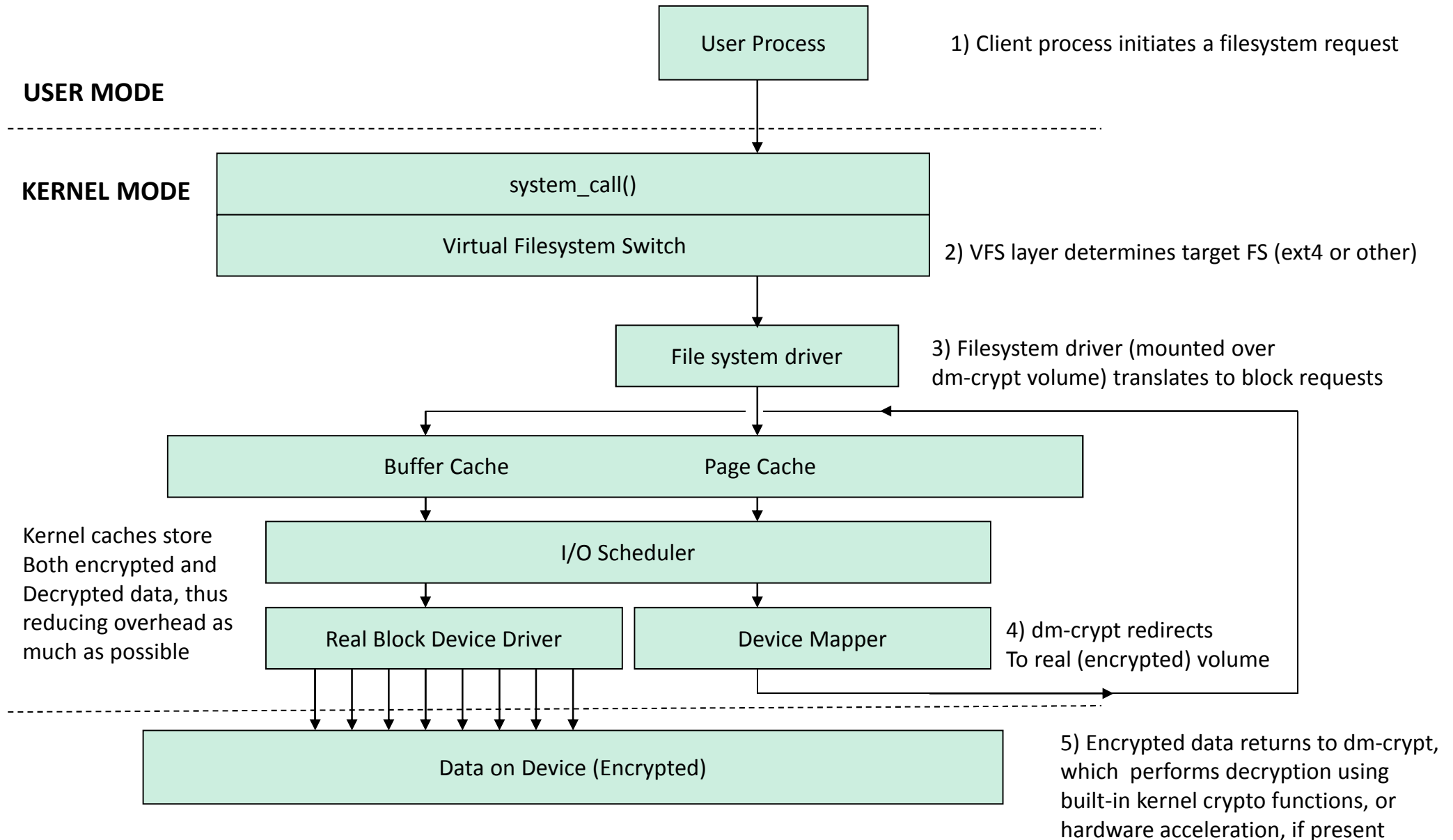
# Viewing lock settings in action

```
root@htc_m8wl:/data # sqlite3 /data/system/locksettings.db
SQLite version 3.7.11 2012-03-20 11:35:50
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> .dump
PRAGMA foreign_keys=OFF;
BEGIN TRANSACTION;
CREATE TABLE android_metadata (locale TEXT);
INSERT INTO "android_metadata" VALUES('en_US');
CREATE TABLE locksettings (_id INTEGER PRIMARY KEY AUTOINCREMENT,
                           name TEXT,user INTEGER,value TEXT);
INSERT INTO locksettings VALUES(2,'lockscreen.options',0,'enable_facelock');
INSERT INTO locksettings VALUES(3,'migrated',0,'true');
INSERT INTO locksettings VALUES(4,'lock_screen_owner_info_enabled',0,'0');
INSERT INTO locksettings VALUES(5,'migrated_user_specific',0,'true');
INSERT INTO locksettings VALUES(9,'lockscreen.patterneverchosen',0,'1');
INSERT INTO locksettings VALUES(11,'lock_pattern_visible_pattern',0,'1');
INSERT INTO locksettings VALUES(12,'lockscreen.password_salt',0,'-3846188034160474427');
INSERT INTO locksettings VALUES(81,'lockscreen.disabled',0,'1'); # No Lock
INSERT INTO locksettings VALUES(82,'lock_fingerprint_autolock',0,'0');
INSERT INTO locksettings VALUES(83,'lockscreen.alternate_method',0,'0');
INSERT INTO locksettings VALUES(84,'lock_pattern_autolock',0,'0');
INSERT INTO locksettings VALUES(86,'lockscreen.password_type_alternate',0,'0');
INSERT INTO locksettings VALUES(87,'lockscreen.password_type',0,'131072'); # PIN
INSERT INTO locksettings VALUES(88,'lockscreen.passwordhistory',0,'');
DELETE FROM sqlite_sequence;
INSERT INTO "sqlite_sequence" VALUES('locksettings',88);
COMMIT;
sqlite>
```

# The Linux Device Mapper

- Android relies on the Linux Device Mapper heavily
- Filesystem/Container encryption (“FDE”, “ASEC”)
- Filesystem verification (DM-Verity)

# The Linux Device Mapper



# Dm-crypt

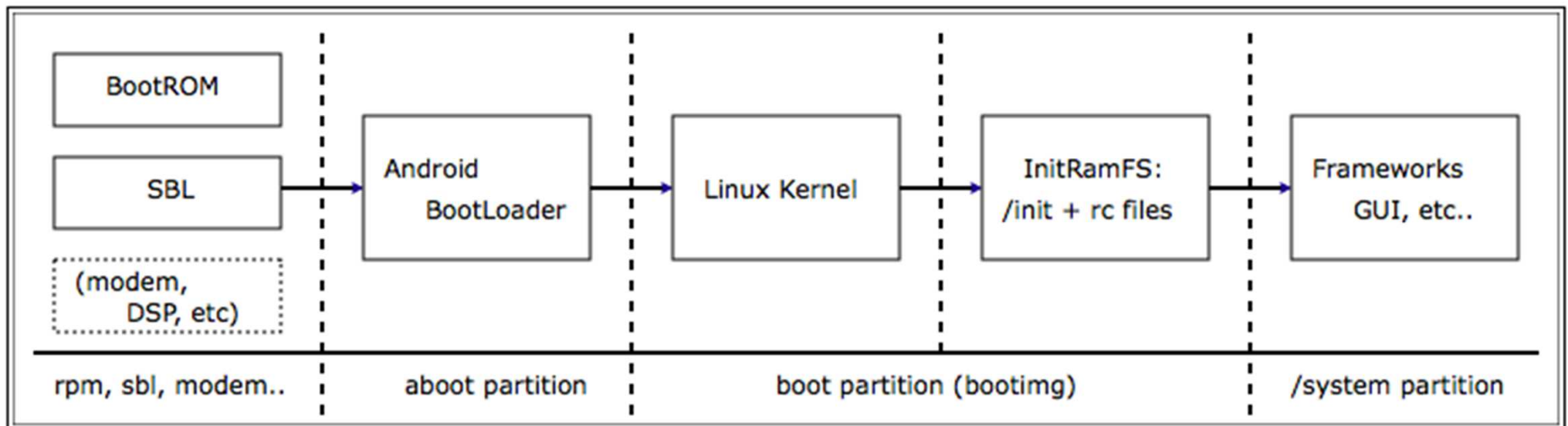
- Originated with OBB
- “Matured” with ASEC
- Fully fledged with /data encryption (3.0, effectively 5.0)
- Enhanced for hardware acceleration with L 5.1
- NOT AS SECURE AS YOU WOULD THINK
  - Transparent, so apps/injected code/adb unaffected

# Dm-verity

- New feature in KitKat – Used in Nexi as of late L
- Prevents booting into a modified filesystem (/system)
- Documentation: <http://source.android.com/devices/tech/security/dm-verity.html>
- Discussion: <http://nelenkov.blogspot.com/2014/05/using-kitkat-verified-boot.html>
- Will mitigate boot-to-root, but not runtime exploits

# The Android Boot Process

- Recall Android Generalized Boot:



---

Chain of Trust extends to kernel + initRAM (root filesystem)

---

DM-Verity (in KitKat) extends the chain of trust onto the /system partition as well



# Personal Data Protection

- Android M (finally) advances hardware-backed encryption
  - Filesystem encryption now accelerated
  - keystore service fully integrated with TrustZone
  - New GateKeeper service further hides user secrets (passwords)
    - Allows temporary tokens for passwords (auto-expiration)

got root?



# Rooting

- Goal: Obtain UID 0 (root) on device
  - Note shell access/app-install is given anyway with USB dev
  - Impact: inspect app data, peruse and “mod” system files  
can also mod kernel (cyanogen, etc)
- Corollary: Entire security model of Android shatters
  - No more ASEC, OBB, encryption, or trust
- May require boot-to-root or be a “1 click”
  - Via Fastboot: Reboot device, “update” from alternate ramdisk
    - Run modified /init as root, drop “su” in /system/[x]bin.
  - “1 click”: Exploit Linux kernel/Android vulnerability

# Boot-To-Root

- Android devices (for the most part) allow unlocking
  - Notable Exception: Amazon Kindle
- Can make your own “update.zip” or use ones from Web
  - Requires unlocking bootloader (“fastboot oem unlock”, if available)
  - Unlocking will wipe /data
  - Also permanently marks boot-loader (to void warranty)
- Far better to create your own
  - Internet-borne rooting tools can potentially contain malware

# “1-Click”

- Android is not really supposed to allow “1-Click”
- “1 click” a lot more convenient – but DANGEROUS
  - Can occur without user’s permission, or knowledge(!)
  - Q.v. Jay Freeman (Saurik) and Google Glass
  - Not just code injection! (q.v. [HTC One and “WeakSauce”](#))
- similar in logic/complexity to iOS “untethered” JB

# Attack Surface: Linux =< Android

- Remember: Android is based on Linux
  - Any Linux kernel vulnerability is automatically inherited
  - Towelroot, anyone?
- Even more potential vulnerabilities in external packages
  - Racoon, mdnsd, etc..
- Additionally, Android contains idiosyncratic bugs
  - Two days ago: Zimperium, StageFright
- And we don't know of any 0-days.. Until they're out.
  - Well, almost – upcoming 0-day kernel (local root) next week!\*

# Rooting will bury content protection

- Android's content protections disintegrate in face of root
  - Any application's data directory (or code) can be read
  - OBBs can be mounted and read
  - ASEC containers can be mounted, their keys can be read
  - DRM can be bypassed, one way or another.
- Coupled with DEX decompilation, this is a big problem
  - Your app can be decompiled, modded and repackaged
- No real way to detect a rooted device from a running app

# So, overall..



- Sad Truth: Android “spitballs” Linux features together
- Sometimes it holds. Others.. It doesn't.

