Android Security

New Threats, New Capabilities

Jonathan Levin, Technologeeks.com

http://technologeeks.com/

About this talk

- Provides tour of Android security features:
  - Linux inheritance (permissions, capabilities)
  - Dalvik level security (permissions, IFW)
  - SELinux and SEAndroid
  - Rooting and System Security

- Get the slides: http://www.newandroidbook.com/files/Andevcon-Sec.pdf
- Covered in “Android Internals: A Confectioner’s Cookbook”
  - http://www.NewAndroidBook.com/21-Security-L.pdf*

*** - Please wait till 11/24/14 before accessing link; previous version (32-Security.pdf) is avaiable now**

The Book

- "Android Internals: A Confectioner's Cookbook"
- Parallels "OS X and iOS Internals" (but for Android)
  - BTW OSXiI is getting a 2nd Edition (10.10/iOS 8) – March 2015!
- Book (volume I) is finally available for preorder!
  - preorder@newosxbook.com
  - Still looking for Amazon to publish Kindle edition (soon!)
  - Loads of L framework level changes require rewrite for Volume II
- Updated for L (5.0/API 21)
- http://newandroidbook.com/
  - FAQ, TOC and plenty of bonus materials
  - Check newandroidbook.com/rss.php
  - Check out technologeeks.com (@Technologeeks) for more

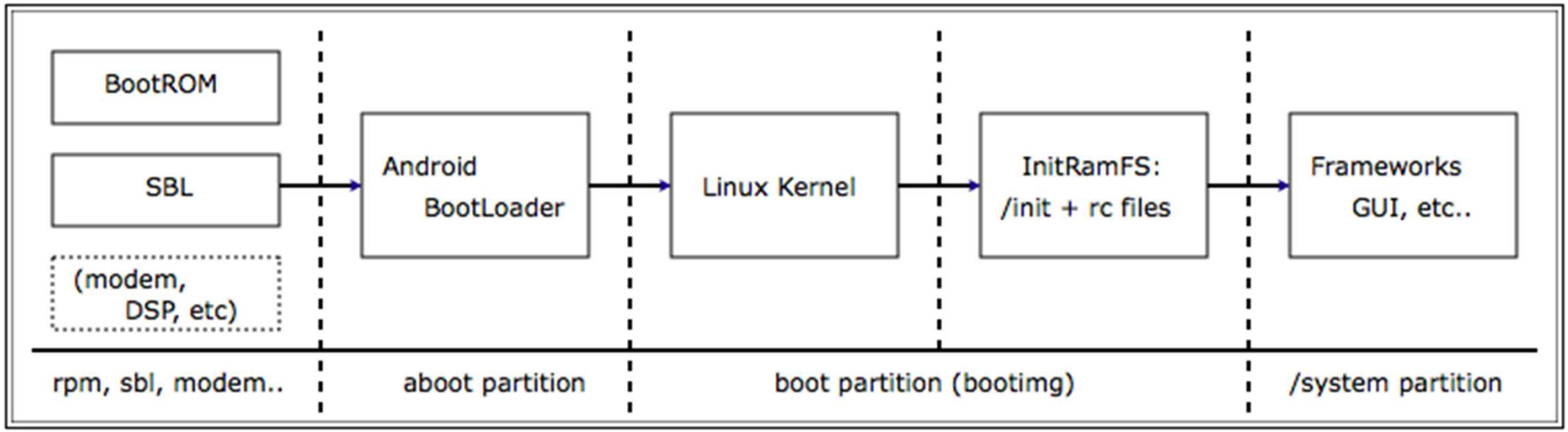
Attack Surface

- Threat models for mobiles consider three main vectors:
  - Rogue user (device theft, or unauthorized root)
    - Secure Boot Process
    - Encrypt User Data
    - Device lock
  - Rogue applications (malware)
    - Sandbox applications
    - Enforce Strong Permissions
    - Harden OS Component Security
  - Internet-borne attacks
    - Website drive-by, webkit/plugin code injection vectors

* We'll discount the internet-borne attack vector in this talk, since it isn't mobile specific

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

/Data Encryption

- Android offers data encryption as of Honeycomb
  - Default option as of L (for new install, not upgrade)
  - Encryption is only for /data, not SD-Card
  - Dependent on PIN (or, preferably, a passcode)

- Fairly well documented:
  - https://source.android.com/devices/tech/encryption/
  - http://nelenkov.blogspot.com/2014/10/revisiting-android-disk-encryption.html

/Data Encryption

- Encryption relies on Linux's dm-crypt mechanism
- Handled in user mode by vold (try `vdc cryptfs`)*

cryptfs	restart		Signal `init` to restart frameworks
	cryptocomplete		Query if filesystem is fully encrypted
	enablecrypto	inplace|wipe *password*	Encrypt filesystem, possibly erasing first
	changepw	*old_passwd new_passwd*	Change encryption password
	checkpw	*passwd*	Check if supplied password can mount encrypted fs
	verifypw	*passwd*	Used by `BackupManagerService`
	getfield	*name*	Get metadata field from cryptfs
	setfield	*name value*	Set metadata field in cryptfs

- Hardware backed (TZ, QSEE, etc) when possible

---

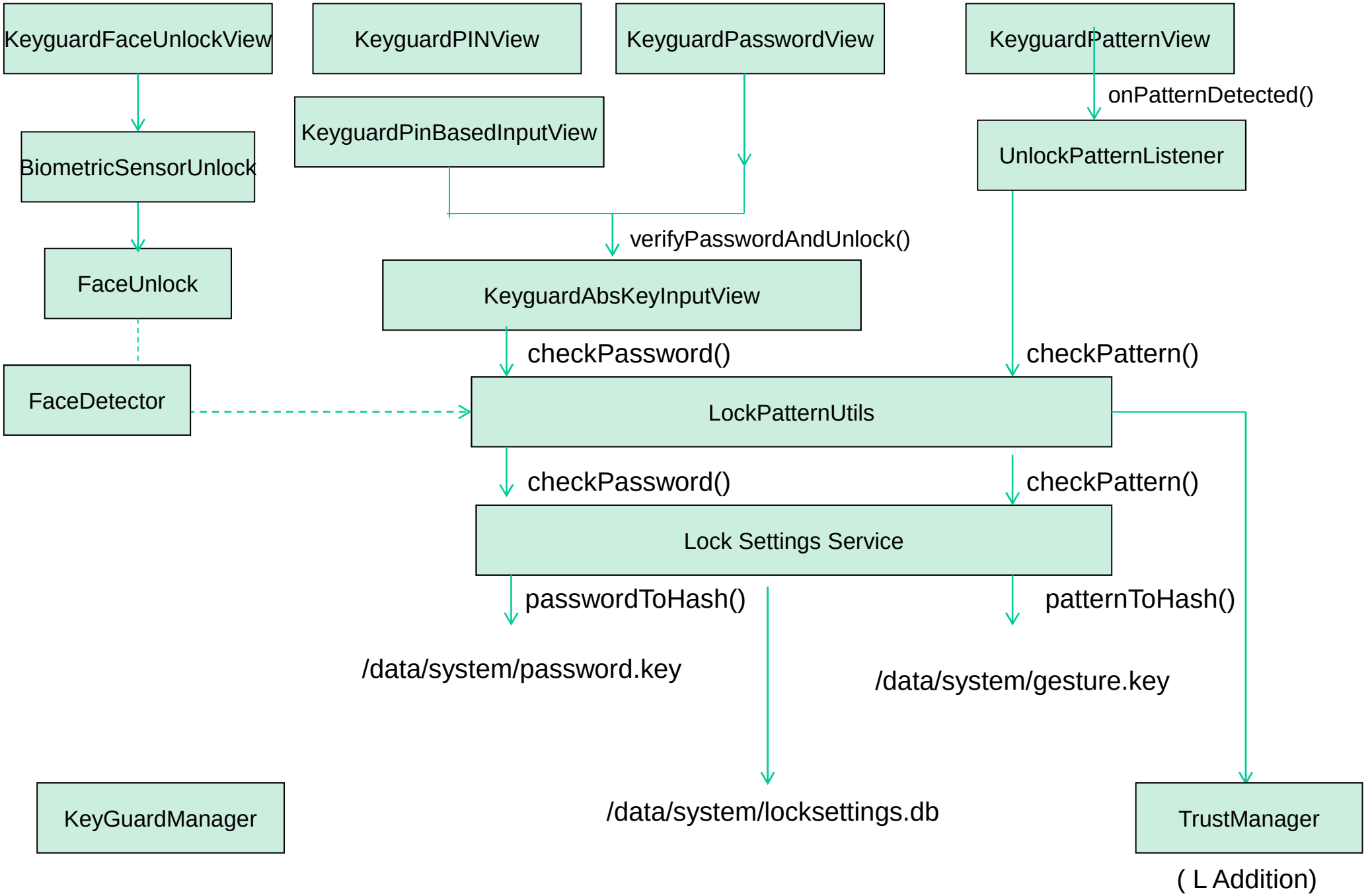
* Obviously, exercise discretion here, since you can render the encryption unusable

Screen Lock

- 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


KeyguardFaceUnlockView
KeyguardPINView
KeyguardPasswordView
KeyguardPatternView
onPatternDetected()
BiometricSensorUnlock
KeyguardPinBasedInputView
UnlockPatternListener
verifyPasswordAndUnlock()
FaceUnlock
KeyguardAbsKeyInputView
checkPassword()
checkPattern()
FaceDetector
LockPatternUtils
checkPassword()
checkPattern()
Lock Settings Service
passwordToHash()
patternToHash()
/data/system/password.key
/data/system/gesture.key
KeyGuardManager
/data/system/locksettings.db
TrustManager
(L Addition)

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 Kill Switch

- As a last resort, remote wipe the phone
- Kill Switch functionality actually required by law (.ca.us)
- Does require device to be online to activate
- Likely not too usable on rooted devices
    - Or those with open/vulnerable bootloaders

Android Application Security Model

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

Android Application 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
    - L augments by SE-enabling init and servicemanager

Case Study: system_server

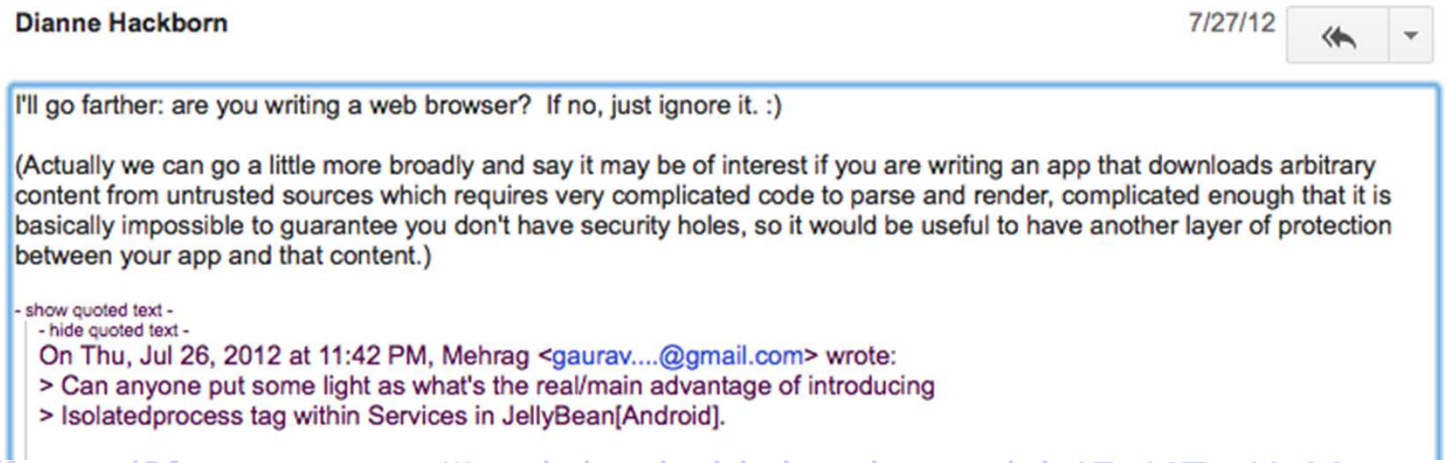
Table s2-ssp:: Group memberships of `system_server`

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

- **L adds 1032 as well (AID_PACKAGE_INFO)**

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].

https://groups.google.com/forum/?fromgroups=#!topic/android-developers/pk45eUFmKcM

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/capabilty.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 mlock(2)/mlockall(2)/shmctl(2)
CAP_IPC_OWNER	Bypass permission checks on IPC objects
CAP_KILL	Bypass permission operations on signals
CAP_LEASE	Allow file leases (e.g. fcntl(2))
CAP_LINUX_IMMUTABLE	Allow chattr +i (immutable ext2 file attributes)
CAP_MKNOD	Create device files (using mknod(2))
CAP_NET_ADMIN	Ifconfig/routing operations
CAP_NET_BIND	Bind privileged (i.e. <1024) ports
CAP_NET_RAW	Permit PF_RAW and PF_PACKET sockets

Capabilities

Capability	Application
CAP_SETUID/CAP_SETGID	Enable set[ug]id, GID creds over domain sockets
CAP_SETPCAP	Modify own or other process capabilties
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)

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	

- L also uses `CAP_MAC_OVERRIDE` (0000001007813c20)

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

The Intent Firewall

- Little known (and unused) feature of 4.3 **(expanded in 5.0**)

  - base/services/core/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 check*XXX*:

  - `checkStartActivity`, `checkService` and `checkBroadcast`.

The Intent Firewall

- XML rulebase format:

```
<rules>
  <activity block="true/false" log="true/false" >
    <intent-filter>
      <path literal="literal" prefix="prefix" sglob="sglob" />
      <auth host="[host]" port="[port]" />
      <ssp literal="[literal]" prefix="prefix" sglob="sglob" />
      <scheme name="[name]" />
      <type name="[name]" />
      <cat name="NameOfCategory" />
      <action name="nameOfIntent" />
    </intent-filter>
    <component-filter name="nameOfActivity" />
  </activity>
</rules>
```

Great reference: http://www.cis.syr.edu/~wedu/android/IntentFirewall/

(Also covered along with practical exercises and examples in Book)

Android Permissions

- The “pm” shell command manages permissions:

```
usage: pm [list|path|install|uninstall]
       pm list packages [-f] [-d] [-e] [-u] [FILTER]
       pm list permission-groups
       pm list permissions [-g] [-f] [-d] [-u] [GROUP]
       pm list instrumentation [-f] [TARGET-PACKAGE]
       pm list features
       pm list libraries
       pm path PACKAGE
       pm install [-l] [-r] [-t] [-i INSTALLER_PACKAGE_NAME] [-s] [-f]
PATH
       pm uninstall [-k] PACKAGE
       pm clear PACKAGE
       pm enable PACKAGE_OR_COMPONENT
       pm disable PACKAGE_OR_COMPONENT
       pm setInstallLocation [0/auto] [1/internal] [2/external]
```

- Really a wrapper over com.Android.commands.pm.PM

Android Permissions (AppOps)

- AppOps Service (introduced in 4.2) further refines model:
  - Per-Application permissions may be assigned and revoked
    - Revoked permissions will trigger security exception
  - GUI for service mysteriously disappeared in KK
    - GUI could have been used to kill ads and enhance privacy..
  - Service, however, is still very much alive and well

AppOps

The ActivityManager also contains the hidden AppOps service. This service was added in Jellybean, with the aim of providing fine grained permission control for various installed packages. Initially, it had its own GUI, but the GUI was removed in KitKat 4.4.1. The service, however, is not going away, and has been further extended in Android L.

AppOps	
Name:	appops
Interface:	com.android.internal.app.IAppOpsService
File:	/data/system/appops.xml
Started by:	ActivityManagerService

Listing s2-aos: The `AppOpsService` methods defined in `IAppOpsService.aidl`

```
interface IAppOpsService {
    // These first methods are also called by native code, so must
    // be kept in sync with frameworks/native/include/binder/IAppOpsService.h
 /*  1 */ int checkOperation(int code, int uid, String packageName);
 /*  2 */ int noteOperation(int code, int uid, String packageName);
 /*  3 */ int startOperation(IBinder token, int code, int uid, String packageName);
 /*  4 */ void finishOperation(IBinder token, int code, int uid, String packageName);
 /*  5 */ void startWatchingMode(int op, String packageName, IAppOpsCallback callback);
 /*  6 */ void stopWatchingMode(IAppOpsCallback callback);
 /*  7 */ IBinder getToken(IBinder clientToken);
         // Remaining methods are only used in Java.
 /*  8 */ int checkPackage(int uid, String packageName);
 /*  9 */ List getPackagesForOps(in int[] ops);
 /* 10 */ List getOpsForPackage(int uid, String packageName, in int[] ops);
 /* 11 */ void setMode(int code, int uid, String packageName, int mode);
 /* 12 */ void resetAllModes();
   /* The following are new in L */
 /* 13 */ int checkAudioOperation(int code, int usage, int uid, String packageName);
 /* 14 */ void setAudioRestriction(int code, int usage, int uid, int mode,
                                   in String[] exceptionPackages);

 /* 15 */ void setUserRestrictions(in Bundle restrictions, int userHandle);
 /* 16 */ void removeUser(int userHandle);
}
```

The services register with the ServiceManager when the ActivityManagerService's setSystemProcess() is called, right before a call to initialize the Entropy manager. Additionally, the BatteryStats, UsageStats and AppOpsService require explicit calls to their publish() method, which is done in ActivityManagerService's main().

The Android Security Model

- 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)

Android “Master Key” vulnerability

- Doesn’t really involve any master keys, but equally bad
- Duplicate APK entries handled incorrectly:
  - Signature validation uses Java library – validates 1st instance
  - Extraction uses Dalvik native library – extracts 2nd instance
- Outcome: Malware can impersonate any valid package

http://securityaffairs.co/wordpress/19400/hacking/android44-master-key-vulnerability.html

Android “Fake ID” vulnerability

- Allows faking identity of trusted apps via self signed certs
- Android didn’t verify the certificate chain correctly
  - Application could bundle a fake cert along with a real one
  - Real cert does not actually link to fake one, but OS doesn’t care
- Outcome: Malware can impersonate any valid package
  - Favorite target: Adobe WebView plugin (flash)

(finally patched in L)

* - L actually allows WebView to auto-update independently of other components

SE-Linux on Android

- Probably the most important security feature in Android
  - JellyBean introduced in permissive mode
  - KitKat was the first version to enforce
    - Enfrocement still minimal (zygote, netd, vold, and installd)
  - L enforces all throughout the system
- SE-Linux protects file, property and application contexts
  - Init runs in root:system context (still omnipotent)
  - Can set SE context (using sesetcon), 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, as part of the initramfs*

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

* - Question: What's the benefit of putting the policy files into the initramfs?

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
file_contexts	**Restricts access to files**
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:rild_prop:s0

...
```

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

SEAndroid: Experiment

- Compile the following program

Listing 21-5: A simple implementation of **su**, for non SE-Linux enforced devices

```
#include <stdio.h>
void main(int argc, char **argv)
{
        setuid(0);
        setgid(0);
        system("/system/bin/sh");

}
```

- chmod 4775, and drop into /system/bin
  - You'll need to mount –o remount,rw /system first
  - Won't work on /data, because /data is mounted nosuid
- Run it, and channel the power of root!
  - Or, well. Maybe not. Pre-KitKat? Yep. Post KitKat: Not really.
  - Use `ps -Z` and `ls -Z` to find out why

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”)
- May result from vendor vulnerability
  - q.v. HTC (“WeakSauce”, “FireWater”), and QSEECOM
- similar in logic/complexity to iOS “untethered” JB

TowelRoot

- Released just after Andevcon Boston
- Perfect example of a 1-click
- Uses a well known Linux kernel bug
  - CVE-2014-3153 – The FUTEX bug
- Exploitable with no permissions, even w/SELinux

Dm-verity

- New feature in KitKat – still optional
- 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

Attack Surface: Linux =< Android

- Remember: Android is based on Linux
  - Any Linux kernel vulnerability is automatically inherited
  - October 2011: Researchers demonstrate 2.6.35 priv esc.
- Additionally, Android may contain idiosyncratic bugs
  - October 2011: Researchers bypass security prompts.
- And we don’t know of any 0-days.. Until they’re out.

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, modd’ed and repackaged
- No real way to detect a rooted device from a running app

So, overall..

2014 : 7+ major security bugs for Android.

Oh well. Maybe next year?