**Din**

Q1: A service connecting on the Internet only if you are on a Tor network

A1: Not an easy task to detect/block on Tor network, but the logic will be based on:

* The signature of TOR traffic.
* Maintaining a list of TOR end-points.
* Block using the official list available <https://www.torproject.org/projects/tordnsel.html.en>

A stupid fast test can be the usage of a website like <https://check.torproject.org/> so we can check if a HTTP GET will return the correct value.

**Ramadan**

Q1: Automate to automate the interaction with the UI.

A1: Most of the operations are based on the Android testing UI tools combined with some injection tools.

* To send events check the tutorial from Radu M. - <http://www.pocketmagic.net/2012/04/injecting-events-programatically-on-android/#.UwW6OFt_tfU>
* Official UI testing <http://developer.android.com/tools/testing/testing_ui.html> with alternatives Robotium - <https://code.google.com/p/robotium/> and Testdroid/Testrecorder - <http://bitbar.com/>
* Introduction in monkeyrunner via <http://developer.android.com/tools/help/monkeyrunner_concepts.html>

Q2: How to defeat the forensics on application layer (solutions like Cellebrite, Encase, X-Ways Forensics)?

A2: Marius will discuss freely on this topic.

* Protect against ADB
* Protect against content providers an extra leaks
* Protect resources
* Protect the APK

Android 3.0 and later provides full filesystem encryption, so all user data can be encrypted in the kernel using the dmcrypt implementation of AES128 with CBC and ESSIV:SHA256. The encryption key is protected by AES128 using a key derived from the user password, preventing unauthorized access to stored data without the user device password.

To provide resistance against systematic password guessing attacks (e.g. “rainbow tables” or brute force), the password is combined with a random SALT and hashed repeatedly with SHA1 using the standard PBKDF2 algorithm prior to being used to decrypt the filesystem key. To provide resistance against dictionary password guessing attacks, Android provides password complexity rules that can be set by the device administrator and enforced by the operating system. Filesystem encryption requires the use of a user password; pattern-based screen lock is not supported.

More details on implementation of filesystem encryption are available at <https://source.android.com/devices/tech/encryption/android_crypto_implementation.html>

**Kamil**

Q1: Key logger on Android

A1: There is no out of the box solution to do this but you can use some of the bellow possibilities:

* Create your custom keyboard and detect the input keyboards from the input keyboard.Check how other apps are doing it: <http://kidlogger.net/kidlogger.apk>
* Modify the phone ROM and inject your own implementation of the keyboard or platform api for handling the key events.
* There is no possibility to have in user space a service in the background with a KeyListener activity to catch the key-events in all apps.

**Fathi**

Q1: Nexus 5,4 compiling

A1: For each different phone we have a unique mode of flashing, for most oft he Nexus devices we have the following steps:

* Copy the file update.zip in the root of the SD-CARD
* Reboot in recovery mode using the phone (power+vol up?!)
* Choose update.zip
* Flash the device
* Reboot the device

Once we compile the AOSP what we do?

*make updatepackage*

*fastboot -w update $ANDROID\_PRODUCT\_OUT/$TARGET\_PRODUCT-img-eng.$USER.zip*

* Official documentation for flashing is located at http://source.android.com/source/building-devices.html but a lot depends on the bootloader .
* How to build Android AOSP for Nexus 4 - <http://nosemaj.org/howto-build-android-nexus-4>
* How to build Android 4.3 for Nexus 4 - <http://nosemaj.org/build-android-4-3-nexus-4>
* Nexus 4 si Nexus 5 - <https://developers.google.com/android/nexus/drivers>
* Howto Build Android KitKat (4.4) for the Google Nexus 5 - <http://nosemaj.org/howto-build-android-kitkat-nexus-5>

Q2: The way the kernel boots the system.

A2: See the "Bootloading the Kernel" section on page 174.

Q3: More in Surface Flinger .

A3: **Surface**: A Surface in Android corresponds to an off screen buffer into which application renders its content. From application point of view, each application may correspond to one or more graphical interfaces. Each interface can be regarded as surface. Each surface has its position, size, content and other elements.

**Surfaceflinger** : It is a system-wide surface composer function which resides in android framework. It takes data( which is surface) from different application which could be 2D or 3D and finally combine it to obtain a main surface which will be fed to memory( which is framebuffer).

Surfaceflinger synthesize all the surface according to their position, size and other parameters, although this synthesis is done by OpenGL(which is invoked by Surfaceflinger), but we need Surfaceflinger to calculate the relevant parameters like overlapping function.

**HardwareComposer** : This is part of HAL, it is used by surfaceflinger to composite surfaces to the screen. The hardware composer abstracts things like overlays and 2D blitters and helps offload some things that would normally be done by OpenGL. This is done by using 3D GPU or a 2D graphics engine.

More on: <https://source.android.com/devices/graphics.html>

Q4: How to handle the Android root

A4: Marius will demo with a sample app ID Compliance.

**IMA ( Integrity Measurement Architecture ) for Android 4.4**

* Check the code in platform: <https://android.googlesource.com/kernel/omap.git/+/android-omap-tuna-3.0-ics-mr1/security/integrity/ima/ima_api.c>
* Beyond Kernel-level Integrity Measurement: Enabling Remote Attestation for the Android Platform - <http://profsandhu.com/zhang/pub/trust10-android.pdf>
* Building Efficient IntegrityMeasurement and Attestation for Mobile Phone Platforms - <http://goo.gl/dn8FgU>

**Jun Ji**

Q1: Boot.

A1: Check the 117 in the course, bootloading the kernel.

The bootloader configures the device to an initial known state and has a means to select where to start executing the kernel. It can allow you to make this selection, which give you for example the opportunity to start an alternative Linux kernel, or Windows. Because the bootloader is an essential component of the boot process, it is stored in non-volatile memory, such as flash memory.

Bootloaders are written by hardware vendors and are specialized for the hardware they run on.

For Android devices, the bootloader typically starts either Android or Recovery. Android bootloaders often have a basic interactive mode that can be triggered by holding the "volume down" button while the bootloader is executing.

Locked/unlocked bootloaders

A locked bootloader is one that will only boot an OS that it "approves" of. This may mean that device's boot partition has an approved digital signature, or the carrier ID (CID) hard-coded into the OS matches a value hard-coded into the bootloader itself. See also Wikipedia:Hardware restrictions#Verified/trusted/secure boot and

For devices with a locked bootloader, booting an unsanctioned OS (e.g. CyanogenMod or Ubuntu) requires the device's owner to first unlock (or even replace[1]) the bootloader. Unlocking the bootloader sometimes voids the device's warranty. Procedures vary typically by manufacturer.

Bootloader unlocking should not be confused with Android rooting.

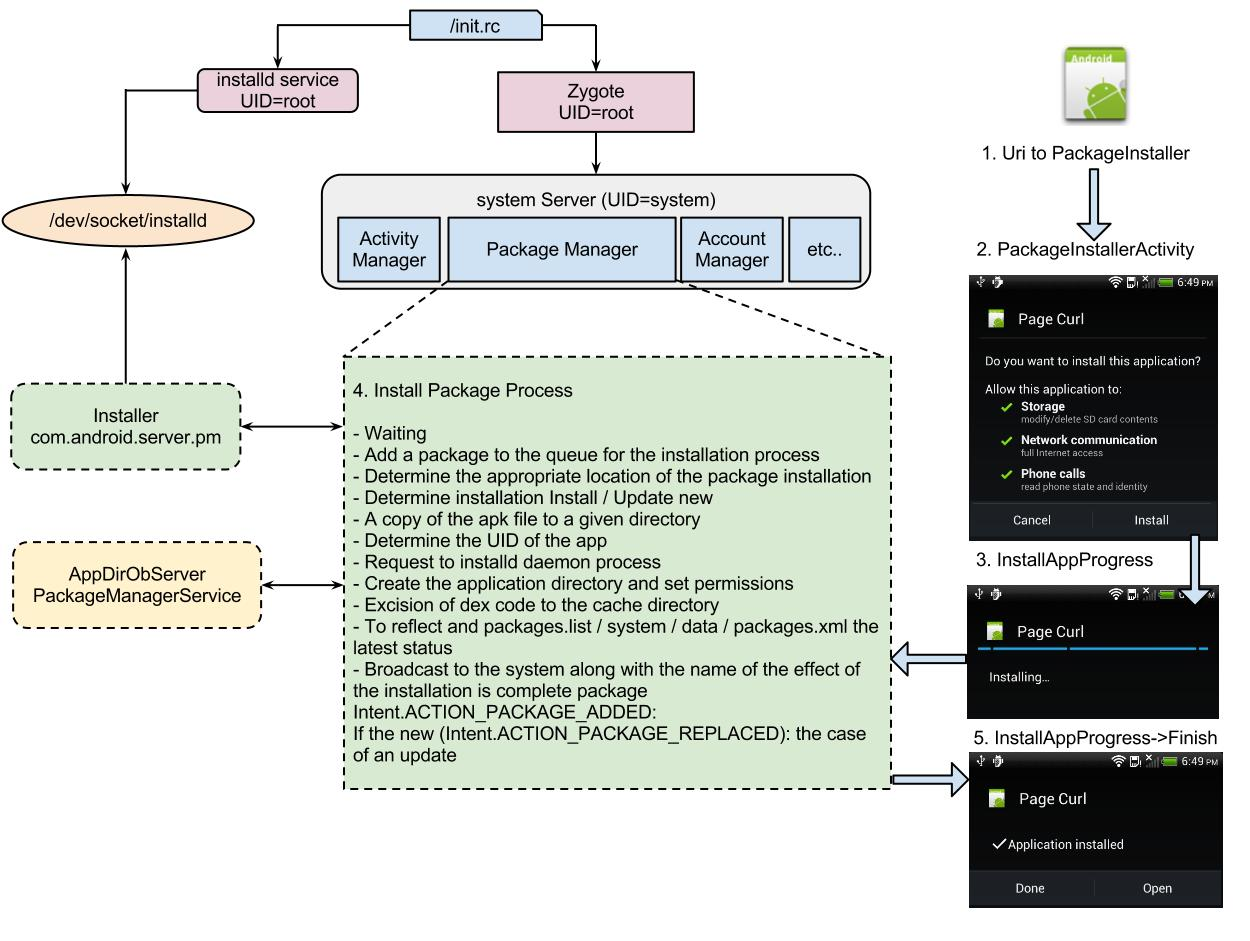
Examples

* HBOOT, an Android bootloader from HTC
* rrload, an Android bootloader from Motorola
* AndroidRoot, a replacement bootloader patched by Androidroot.mobi members for the ASUS Eee Pad Transformer Prime TF201, TF300, and TF700
* Das U-Boot, an open-source bootloader for embedded devices; used in Chromium OS; can replace the bootloader in some Android devices
* IPL+SPL, a bootloader from HTC
* GNU GRUB, the bootloader used by most desktop Linux distributions and also x86 - <http://www.android-x86.org/documents/installhowto>
* WallabyBootloader
* bootldr, a Linux bootloader for PocketPC. A copy can be found in the HandHelds CVS: http://cvs.handhelds.org/cgi-bin/viewcvs.cgi/bootldr/
* HaRET, a bootloader for booting Linux kernels on Windows CE devices
* LinExec

Q2: Who is responsible for APK parsing, permissions etc

A2: PackageManager

* Along with installd responsible for installation of .apk-s on the Android system
* Maintains internal data structures representing installed packages as well as their individual components
* Used by Activity Manager when handling intents (i.e. intent resolution is handled here)
* Provides this info on demand to other services and apps
* Very central to the platform’s security

****

**Hafiz**

Q1: How to monitor the API calls and system calls

A1: Efforts in these directions like FireDroid try to get an service injected in *init.rc* and launch a watcher system in the same time with the boot of the system. Related work on <http://web.cs.ucdavis.edu/~hchen/paper/mobisys2013rs.pdf>

Q2: Detect all BR registered to an event, including the one registered on runtime

A2: *„PackageManager.queryBroadcastReceivers() (*[*http://developer.android.com/reference/android/content/pm/PackageManager.html#queryBroadcastReceivers*](http://developer.android.com/reference/android/content/pm/PackageManager.html#queryBroadcastReceivers) *)returns all receivers declared in application manifests matching a given Intent. Note however that this will not include receivers registered with Context.registerReceiver(); there is currently no way to get information about those. “*

Dianne Hackborn, Android framework engineer

So a partial solution, I will go with a framework modification to get also the one registered in code!

**Rasyid**

Q1: Trace all installed artifacts from an Android app

A1: You can use either the platform implementation for the <https://github.com/android/platform_frameworks_base/blob/master/core/java/android/os/FileObserver.java> ( documentation on: <http://developer.android.com/reference/android/os/FileObserver.html> ) or a recursive implementation via <https://github.com/owncloud/android/blob/master/src/com/owncloud/android/utils/RecursiveFileObserver.java>

**Ina**

Q1: Android architecture

A1: Hopefully the course responded on this question. The New Circle will allow you 1-year usage of the updated course material, having also in a short time the Android 5.0 changes.

**Wira**

Q1: Faces detection in Android

A1: I will demo the *FaceDetection* sample via <http://www.androider.ro/tutorial-android-si-face-detection-7281> and a sample is located in Samples.

Q2: Own locker app.

A2: You start by <uses-permission android:name="android.permission.DISABLE\_KEYGUARD"></uses-permission> and follow most of the code on <https://github.com/ankitdaf/LockApp>

The first step is to understand first the Android platform implementation:

The keyguard is moved in the new versions in the *com.android.keyguard*, see the implementation on: <https://github.com/android/platform_frameworks_base/tree/6b8a3a52acf1c2722551f1ea1ce47831f87939cd/packages/Keyguard/src/com/android/keyguard>

Here we have the FaceUnlock and all the other methods as you find them in an AOSP firmware.

How to do something similar?

Option 1: Do changes in platform and influence the com.android.keyguard package.

Option 2: Get clever behind the platform and do 2 things:

* Detect the power off of the screen and react with a BroadcastReceiver on ACTION\_SCREEN\_ON, ACTION\_SCREEN\_OFF
* Handle the home key ( like here: <https://github.com/shaobin0604/Android-HomeKey-Locker> ) or make it in the launcher way - <https://github.com/Joisar/LockScreenApp>

**Fadhil**

No questions, Marius will demo something cool.

**Q1:** Move notification to the bottom, and slide it from bottom-top

**A1:** Change in PhoneStatusBar.java the getStatusBarGravity() method:

*protected int getStatusBarGravity() {*

*return Gravity.TOP | Gravity.FILL\_HORIZONTAL;*

*}*

into

*protected int getStatusBarGravity() {*

*return Gravity.BOTTOM | Gravity.FILL\_HORIZONTAL;*

*}*

maybe also:

*mNotificationPanelGravity = Gravity.START | Gravity.BOTTOM;*

also some info on <http://stackoverflow.com/questions/5971585/move-the-android-status-bar>

**Q2:** 180 degree rotation of the screen

**A2:** Add to build.prop:

*windowsmgr.support\_rotation\_180=true;*

*windowsmgr.support\_rotation\_270=true;*

It may not work on your launcher, but try for example rotate other apps.

Q3: How to discover hidden api-s

A3: Android has two types of APIs that are not accessible via SDK.

* Package *com.android.internal* - internal API
* Collection of classes and functions marked with @hide javadoc attribute

Steps to enable them, check on <http://devmaze.wordpress.com/2011/01/18/using-com-android-internal-part-1-introduction/> Compiling the platform by removing the @hide will help you get visibility.

Q4: Use reflection for some method not existing in an older API.

Q4:

* Check the bellow example for the transitions between activities, the transitions are available since API level 5 only.

***public class GeneralActivity extends Activity {***

*static Method mDebug\_overridePendingTransition;*

***static {***

***initCompatibility();***

***};***

***private static void initCompatibility() {***

*try {*

*mDebug\_overridePendingTransition = Debug.class.getMethod(*

*"overridePendingTransition", new Class[] { String.class } );*

*// success, this is a newer device*

*} catch (NoSuchMethodException nsme) {*

*// failure, must be older device*

*}*

*}*

***public void overridePendingTransition(int enterAnim, int exitAnim){***

*try {*

*mDebug\_overridePendingTransition.invoke(null, enterAnim, exitAnim);*

*} catch (Exception ite) {*

*}*

*}*

......

*//return on main activity*

*Intent intent = new Intent();*

*intent.setClass( getApplicationContext(), ApiAccountsMainActivity.class );*

***overridePendingTransition(R.anim.fade, R.anim.hold);***

*startActivity(intent);*

*finish();*

As alternative you can also use sometimes code similar with the one bellow:

private int isNetworkNotificationDisabled() {

int status= 0;

int notification;

**if(Build.VERSION.SDK\_INT >= 4){**

notification = Settings.Global.getInt(context.getContentResolver(),

Settings.Global.WIFI\_NETWORKS\_AVAILABLE\_NOTIFICATION\_ON, 0);

} else {

notification = Settings.System.getInt(context.getContentResolver(),

Settings.System.WIFI\_NETWORKS\_AVAILABLE\_NOTIFICATION\_ON, 0);

}

if (notification == 0)

status = 1;

update(status);

return status;

}

Q5: Please develop a piano app.

A5: A hard one. I attached in *Samples/ChildPlay* a piano sample app derived from <https://github.com/piusvelte/childsplay>

Q6: Settings do disable only some rights for an app

A6: My comments goes toward 2 solutions: AOSP App Ops Manager and the Cyangenmod Privacy Guard.

* The App Ops Manager code is introduced in the AOSP in the Android 4.3 version and slowly propagates also on the new versions - <https://github.com/android/platform_frameworks_base/blob/master/core/java/android/app/AppOpsManager.java>
* Some screenshots with the screen of the App Ops Manager can be found on <http://goo.gl/9TT50k>
* An interesting history of App Ops Manager can be seen on M. Murphy blog post - <http://goo.gl/JlxVcm>
* As alternative there is also the Cyangenmod Privacy Guard app offering the same functionality on the on an Cayngenmod

**Q7:** How simcard is interacting with the phone, they way the phone is „loading“ the SIM apps; it is possibile to „stop“ these apps without breaking calling functionality

**A7:** All the magic is done via Universal Integrated Circuit Card – UICC in the simcard, the implementation in Android is located in :<https://android.googlesource.com/platform/frameworks/opt/telephony/+/3522c54/src/java/com/android/internal/telephony/uicc/UiccCardApplication.java> and <https://android.googlesource.com/platform/frameworks/opt/telephony/+/master/src/java/com/android/internal/telephony/uicc/IccCardProxy.java> .

Some extra info via Tom: <http://nelenkov.blogspot.co.uk/2012/08/accessing-embedded-secure-element-in.html>