**Kai**

Q1: What can be abused in user-space without root?

A1: Marius will explain you some of the elements bellow:

1. **Hidden API–s usage**

How to discover hidden api-s? 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.

1. **Unintentional app vulnerabilities** - For example (CVE-2011-1717) "Skype for Android stored sensitive user data without encryption in sqlite3 databases that have weak permissions, which allows local applications to read user IDs, contacts, phone numbers, date of birth, instant message logs, and other private information."
2. **Push-based installation** of apps from Market (based on the Google account)
3. If the **Google accoun**t associated with the device is **compromised**, malicious applications could be pushed directly to affected owner’s devices
4. **Android uses all-or-nothing access to networking** - Requested via the INTERNET permission, mapped to inet group, enforced via ANDROID\_PARANOID\_NETWORK kernel extension iptables is available, but not exposed to the user
5. Security issues with **device skins** (e.g. HTC Sense): "Security hole in HTC phones gives up e-mail addresses, location" - http://arstechnica.com/gadgets/news/2011/10/security-hole-in-htc-phones-gives-up-e-mail-addresses-location.ars
6. **Security of WebView** - calling addJavaScriptInterface() can lead to XSS, CSRF especially if used over un-encrypted HTTP
7. **UDEV exploit** (CVE-2009-1185)
8. **ADB setuid exhaustion attack** (CVE-2010-EASY)
9. **Zimperlich attack against Zygote**
10. **Ashmem memory protection attack** (CVE-2011-1149)
11. **Buffer Overrun on vold exploit (**CVE-2011-1823)
12. **Linux Local Privilege Escalation via SUID /proc/pid/mem Write** (CVE-2012-0056)
13. **Incorrect Memory Driver Permissions on Samsung Devices**
14. Etc etc

**Luhai**

Q1: Develop a kiosk app

A1: Marius will demo a Launcher from his portfolio.

Q2: Samsung Knox

A2: Samsung Knox is an app that creates a new ‘layer’ on your Samsung phone so that you can securely separate your personal and professional activities. This layer is essentially a second version of your phone that requires a password to be accessed and restricts the way in which the handset is used.

When in Knox mode, there are only certain apps that you can use. If Knox is installed by an administrator at your place of work, they will be able to specify which apps you can use. By default, the apps included are Camera, Gallery, Downloads, Email, S Planner, My files, Phone, Contacts and Internet (Samsung browser only, not 3rd party). You do not have access to the Play Store, but there are an additional 75 ‘Samsung Knox’ apps that are available for download.

Certain functions are disabled within Knox, such as capturing a screen shot. Device administrators can also specify which apps can be used as sharing intents from within Knox.

The KNOX 2.0 platform features major enhancements to the Application Container from the original KNOX platform. The most significant enhancement is the elimination of application wrapping. This is achieved by leveraging technology introduced by Google in Android 4.2 to support multiple users on tablet devices. This enables enterprises to easily deploy custom applications without requiring Samsung to wrap the applications. It also reduces the barrier to entry for independent software developers wishing to develop applications for the KNOX container.

Complete White paper can be found here: <http://www.samsung.com/ca/business-images/resource/white-paper/2014/03/Samsung_KNOX_tech_whitepaper_Final_140220-0.pdf>

Multiple user: (Complete Ref: http://developer.android.com/about/versions/android-4.2.html#MultipleUsers) Android now allows multiple user spaces on shareable devices such as tablets. Each user on a device has his or her own set of accounts, apps, system settings, files, and any other user-associated data.

As an app developer, there’s nothing different you need to do in order for your app to work properly with multiple users on a single device. Regardless of how many users may exist on a device, the data your app saves for a given user is kept separate from the data your app saves for other users. The system keeps track of which user data belongs to the user process in which your app is running and provides your app access to only that user’s data and does not allow access to other users’ data.

**What you need to actually use Knox**

To use Knox, your device must support its virtualization technology at the hardware level, which restricts Knox to these Samsung devices: the Galaxy Note 3 "phablet," the Galaxy S III smartphone, the Galaxy S 4 smartphone, and the 2014 model of the Galaxy Note 10.1 tablet. Today, the Note 3 and S4 can run Knox, but only on some carriers' models: Sprint and Verizon for the S 4; AT&T and Verizon for the Note 3, if you install their Premier Suite updates. The Wi-Fi-only Note 10.1 also runs Knox.

Samsung says it will deliver updates to make Knox work on the S III and on other carriers' S 4 and Note 3 versions, but it also notes that each carrier decides when and if Knox compatibility is made available for the devices on its network. Not only do few devices support Knox, the carrier you use determines when or if those devices will actually be able to work with Knox. (Welcome to the fractured mess that is Android!)

You also need the Knox application and its included set of client apps, such as for email. That's only recently been made available in the Google Play store for download.

You need a Knox-compatible mobile management server, for which you pay a monthly fee per user to manage Android and iOS devices; the fee depends on the management features you select. You cannot manage Knox with Microsoft's Exchange ActiveSync (EAS) protocol, which supports a base set of MDM protocols used by Apple and Google and is thus the "free" approach to MDM.

Q3: Device management and antivirus sample

A3: Marius will demo 2 apps: **00\_AntiMalware** and **00\_DevicePolicyDemo**.

Q4: Keystore, signing, credentials management

A4: Marius will lecture based on a separate presentation (page 4).

**Fangjian**

Q1: PowerManagement and how can we reduce the battery usage

A1: Check the tech paper from Google <https://source.android.com/devices/tech/power.html>

Check the official platform implementation of Battery tab in Settings app, the project app is located at <https://github.com/android/platform_packages_apps_settings> and the class doing all the magic is <https://github.com/android/platform_packages_apps_settings/blob/master/src/com/android/settings/BatteryInfo.java>

Check the Intel PowerTop open source project at <https://01.org/powertop/> and sources at: <https://github.com/fenrus75/powertop>

The *BatteryInfo.java* has an included *BroadcastReceiver* for *Intent.ACTION\_BATTERY\_CHANGED* and is taking from the passed Intent most of

the informations need it like temperature, etc.

Check the Power Tutor application website at : <http://ziyang.eecs.umich.edu/projects/powertutor/> and the source code is available at: <https://github.com/msg555/PowerTutor>

Check also the SysPower app website <https://code.google.com/p/syspower/> and the source code is available at: <https://code.google.com/p/syspower/>

Also command line alternatives + java code alternatives:

*adb shell dumpsys batteryinfo*

*public static void dumpBatteryInfo() {*

*try {*

*String cmd = "dumpsys battery";*

*Process script = Runtime.getRuntime().exec(cmd);*

*BufferedReader in = new BufferedReader(*

*new InputStreamReader(script.getInputStream()));*

*String line = null;*

*while ((line = in.readLine()) != null) {*

*Log.i ("BATTERY","Battery stats: " + line);*

*}*

*} catch (Exception ex) {*

*}*

*}*

*<uses-permission android:name="android.permission.DUMP" />*

**Wei**

Q1: AIDL example

A1: Marius will demo the **TelephonyManager** via AIDL from DS.

Q2: Android Internals – Binder services.

A2: Marius will present using the Android Internals material. The rest of the material over email wei.a.deng@intel.com

**Xiaofei**

Q1: SE Linux and IMA in Android

A1:

**SE Android (SE-Linux on Android)**

* Discretionary Access Control (DAC) vs. Mandatory Access Control (MAC)
* DAC on Android
  1. Default form of access control (also true for most Linux systems)
  2. Access to data is controlled by the app developers ( Except for root )
  3. Based on user/group identity associated with each app and its data
  4. Coarse-grained decentralized control ( No easy way to establish a system-wide policy )
* MAC with SE-Linux
  1. System-wide security policy applies to all processes, data, and system operations
  2. Based on security labels
  3. Confines flawed/malicious apps as well as system processes ( including those that run as root!, Prevent privilege escalation )
  4. Centralized/manageable device-wide policy
* See SEAndroid website on how to build it - <http://selinuxproject.org/page/SEAndroid>
* From Stephen Smalley’s "The Case for Security Enhanced (SE) Android": <https://events.linuxfoundation.org/images/stories/pdf/lf_abs12_smalley.pdf>
  1. SE Android brings MAC to Android and has the following goals:
  2. Prevent privilege escalation by apps
  3. Prevent data leakage by apps
  4. Prevent bypass of security features
  5. Enforce legal restrictions on data
  6. Protect integrity of apps and data
  7. Targeted at consumers, businesses, and government
* What SE Android can help with:

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

* What SE Android can not help with:

1. Kernel vulnerabilities (in general)
2. May block exploitation of specific vulnerabilities
3. Anything allowed by security policy
4. Good policy is important
5. Architecture of system applications matters
6. Decomposition, least privilege

**Challenges**

1. Kernel
   * No support for per-file security labeling (yaffs2)
   * Unique kernel subsystems lack SELinux support
2. Userspace

* No existing SELinux support
* Sharing through framework services

1. Policy

* Existing policies unsuited to Android

**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>

**Hao**

Q1: WindowManager - examples of priority, overlay app

A1: How to show on the top of the application a visible layer with information? Hard to find a good solution but here you go <http://stackoverflow.com/questions/4481226/creating-a-system-overlay-always-on-top-button-in-android> and working example at <http://goo.gl/2qX50G>

Check also the settings->GPU equivalent source code <https://github.com/android/platform_packages_apps_settings/tree/master/src/com/android/settings>

**Jing**

Q1: Monitoring the power usage as service in background

A1: See the response for Fangjian.

**Biyao**

Q1: Attack/secure entry points in Android applications

A1: Marius will discuss free on this elements based on some example. See first the response for Kai**.**

Also check my presentation on SlideShare - <http://de.slideshare.net/fastlink2/droidcon-eastern-europe-2013-how-secure-is-an-androidapp> . A good ideea is to have a look also at the project dexguard - <http://www.saikoa.com/dexguard>

**Guilan**

Q1: Android Bouncer , alternatives.

A1: Google’s App Verification Service information bellow:

* Introduced in Android 4.2
* Apps installed via Google Play are already verified on the "server side"
* Meant to protect against malicious apps installed via alternative channels:
  + Side-loaded from SDCARD, remote HTTP server, message attachment, etc.
  + Alternative app markets (such as Amazon app-store)
  + ADB installed (including through and IDE such as Eclipse)
* Enabled via Setting → Security → Verify apps
* The actual implementation is a combination of Google Play client-side app, and Google Play cloud infrastructure
* When an app is about to be installed:
  1. Google Play client collects the information about the app
  + Package/app name
  + App version
  + SHA1 digest of the app contents
  + The location (e.g. URL) from where the app is being installed
  + Information about the device (ID, IP, etc)
  1. Google Play cloud uses this information to verify the app based on some internal heuristics and responds to the client
  2. When it receives the response, the client:
* Blocks the installation if the app is found to be "dangerous"
* Gives user a warning if the app is found to be "potentially harmful or dangerous", at which point the user can cancel the installation or proceed at their own risk
* Resumes the installation without user’s intervention if the app appears to be safe

**Observations:**

* As of November 30, 2012, the Google’s App Verification Service was found to have a detection rate of only 15-20% - on a sample of 1260 apps belonging to 49 families of known Android malware (compared to 50-100% for established AV vendors)
* Most malware is distributed such that it is mutated on every download, so digest-only-based detection offers weak protection
* For more info, see An Evaluation of the Application ("App") Verification Service in Android 4.2 from NC State University - <http://www.cs.ncsu.edu/faculty/jiang/appverify/>

**Junli**

Q1: UI/Unit test in Android

A1: Marius will demo the Android testing UI 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>

**Haoren**

Q1: Debug the code using an x86 emulator (cross C++/Java)

A1: Marius will demo a debug application testing.

Helping info can be found bellow:

1. In the Internals course at the page 226-243.
2. Services debugging in Android - <http://stackoverflow.com/questions/9226451/how-to-debug-android-framework-services> and <http://android.opensourceror.org/2010/01/18/android-source/>
3. Step by step debuging NDK - <http://mhandroid.wordpress.com/2011/01/23/using-eclipse-for-android-cc-debugging/>
4. Check also the following article - <http://www.eweek.com/c/a/Linux-and-Open-Source/How-to-Set-Up-Android-Platform-Development-and-Debugging/>
5. A cool alternative article linked tombstone, ndk-stack and addr2line - <http://bytesthink.com/blog/?p=133>

**Yuan**

Q1: Use the NDK to compile the c/c++ code and use it using JNI in java and the other way around.

Please refer to the sample files in /android/ndk/samples and also to our example FibonacciNative. Also a good tutorial is to be found on <https://thenewcircle.com/s/post/49/using_ndk_to_call_c_code_from_android_apps>

Marius will demo also the complete FibonacciNative sample.

**ShunDa**

Q1: Security exception on Samsung mobiles.

A1: Bellow is an example on article I wrote <http://www.androider.ro/cum-deblochez-samsung-galaxy-y-s5360s5363-blocat-in-retea-7180> (language is romanian).

The sample bellow is a netlock detect/removal for Samsung Galaxy Y S5360/S5363

The important information is stored in bml15.bin ( block level representation for /dev/stl5

$ su

# cat dev/block/bml15 > /data/local/bml15.bin

# exit

$ exit

adb pull data/local/

... modify the bml15.bin in a new file named bml15\_unlocked.bin

adb push bml15\_unlocked.bin /data/local/bml15\_unlocked

$ su

$ dd if=/data/local/bml15\_unlocked of=/dev/bml15

# exit

$ exit

restart ☺

**Zhao Hui**

Q1: Port or use the C/C++ libraries in Android.

A1: See the response for Yuan.

**Jingke**

Q1: App sample of Wifi P2P.

A1: A step by step instruction on how to use the Android Wifi P2P is gathered in a nice tutorial on: <http://developer.android.com/guide/topics/connectivity/wifip2p.html> . Android provides in the Samples folder a sample app named: **WiFiDirectDemo**

A fork of this app can be found in Github - <https://github.com/ahmontero/wifi-direct-demo> and allows you to send images via Wifi P2P.

A2: App sample with NFC.

Q2: Official documentation can be found on <http://nfc.android.com/> . A modified sample of the Google NFCDemo can be found on <https://code.google.com/p/ndef-tools-for-android/source/browse/#git/ndeftools-boilerplate>

A great step by step tutorial with sample code also on <http://www.framentos.com/en/android-tutorial/2012/07/31/write-hello-world-into-a-nfc-tag-with-a/>

A released Google Play app with great NFC capabilities is located on <https://github.com/nadam/nfc-reader>

**Shantong**

Q1: What is the best solution to communicate with a server?

A1: On Android the best mobile optimized solution is to use the HTTP Restfull API-s using JSON. If the server supports also caching and is transporting the information gziped, will be of minimal impact for he mobile experience.

We finished together yesterday a side project named **WeatherApp**,

My strongly suggestion is to use for networking communication the **Volley** library to enable the usage of cached headers from the server and also the usage of local caching (discussion about the media caching).

**Yalei**

Q1: Cartoon like animations.

A1: See the example animation a drop down from top to bottom:

**res/anim/fall\_down\_top.xml**

*<?xml version="1.0" encoding="utf-8"?>*

*<set xmlns:android="http://schemas.android.com/apk/res/android" >*

*<translate*

*android:duration="400"*

*android:fromYDelta="-200%"*

*android:interpolator="@android:anim/linear\_interpolator"*

*android:toYDelta="0" />*

*</set>*

….

**The java class doing the animantion:**

*public void animateButtons() {*

*int duration = 400;*

*for (int i = 0; i < buttons.size(); i++) {*

*View child = buttons.get(i);*

*Animation anim = AnimationUtils.loadAnimation(getContext(),*

***R.anim.fall\_down\_top****);*

*anim.setDuration(duration);*

*child.startAnimation(anim);*

*duration += 100;*

*}*

*}*