E-KRAAL Innovation Hub

TOPIC: Android Internals

PRESENTERS: Christian Kisutsa & Ruby Nyambugi

Whoami - Christian

Speciality: Digital Forensics, Mobile Security and Network Security Monitoring

Projects: https://github.com/xtiankisutsa

Blog: www.shadowinfosec.io

Twitter: @xtian_kisutsa

Whoami - Francis

Speciality:

Twitter:

Blog: Coming soon!!

Whoami - Martin

Speciality:

Twitter:

Blog: Coming soon!!

Why are we here???

- Understand android Internals
- Reverse Engineer android apps
- Statically analyze android apps
- Dynamically analyze android apps
- Be able to perform android app assessments

Where to start??

e.KRAAL INNOVATION HUB

- Be INTERESTED or PASSIONATE
- Be able to GOOGLE
- READ books, blogs, articles, papers etc.
- WATCH videos, tutorials, walkthroughs etc.
- PRACTICE, PRACTICE, PRACTICE

Key Takeaways!

e.KRAAL INNOVATION HUB

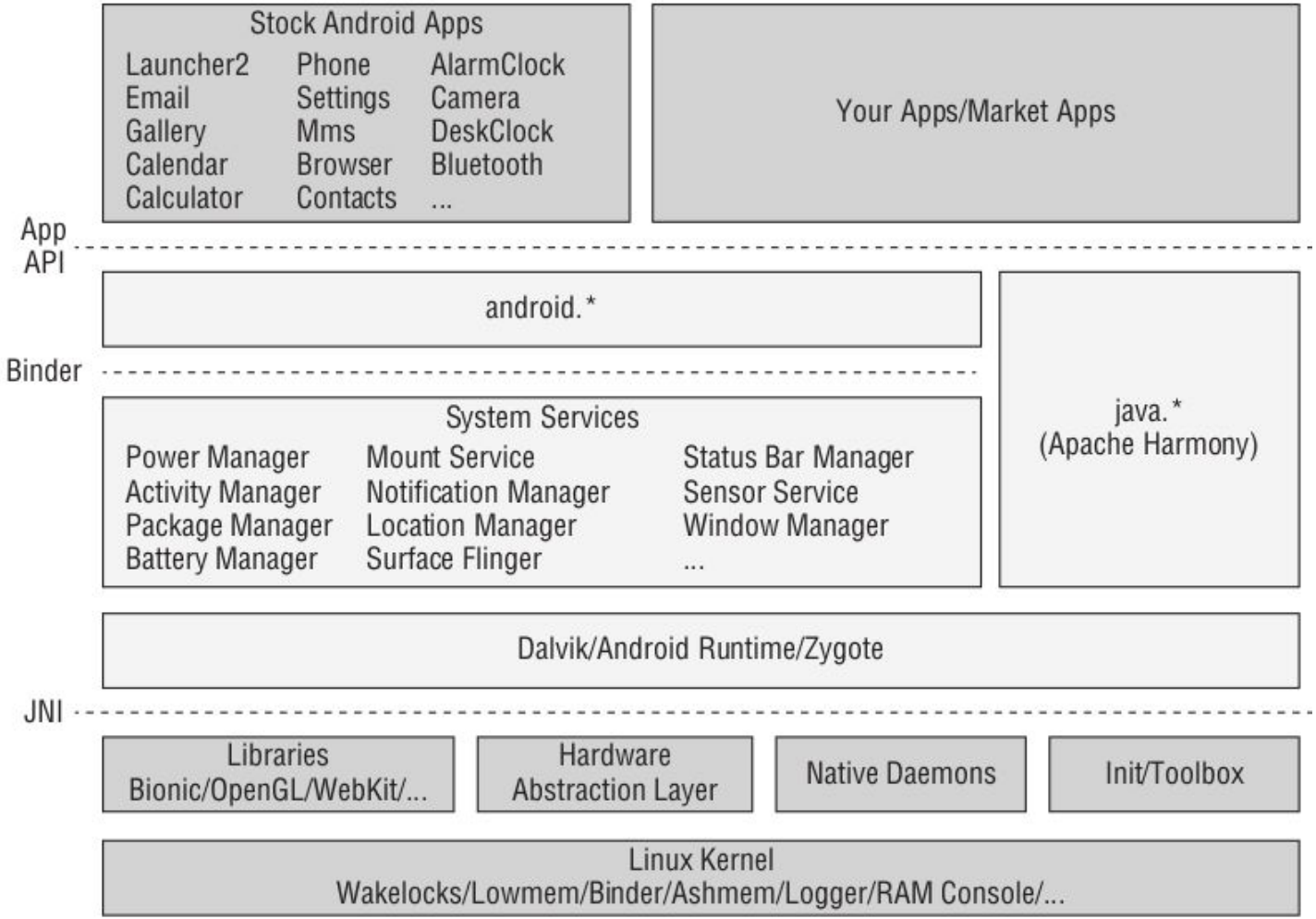
- Learn something new!!
- Gain some PRACTICAL skills and TECHNICAL knowledge
  - How to break android apps
  - How to identify potential vulns
- Understand the CONCEPTS on Android Security
- Inculcate a hacker MINDSET on how to accomplish tasks

e.KRAAL
INNOVATION HUB

Android Internals

Overview

- Kernel
  - First layer to interact with hardware
- Libraries
  - Exposed to Devs
  - Layer between kernel and application framework
  - Provides common service for apps
- Core Libraries
  - SSL, Sqlite, Surface Manager, WebKit, Font, Media, Display


Stock Android Apps
Launcher2 Phone AlarmClock
Email Settings Camera
Gallery Mms DeskClock
Calendar Browser Bluetooth
Calculator Contacts ...
Your Apps/Market Apps
App API
android.*
Binder
System Services
Power Manager Mount Service Status Bar Manager
Activity Manager Notification Manager Sensor Service
Package Manager Location Manager Window Manager
Battery Manager Surface Flinger ...
java.*
(Apache Harmony)
Dalvik/Android Runtime/Zygote
JNI
Libraries
Bionic/OpenGL/WebKit/...
Hardware
Abstraction Layer
Native Daemons
Init/Toolbox
Linux Kernel
Wakelocks/Lowmem/Binder/Ashmem/Logger/RAM Console/...

FRAMEWORK SERVICE	DESCRIPTION
Activity Manager	Manages Intent resolution/destinations, app/activity launch, and so on
View System	Manages views (UI compositions that a user sees) in activities
Package Manager	Manages information and tasks about packages currently and previously queued to be installed on the system
Telephony Manager	Manages information and tasks related to telephony services, radio state(s), and network and subscriber information
Resource Manager	Provides access to non-code app resources such as graphics, UI layouts, string data, and so on
Location Manager	Provides an interface for setting and retrieving (GPS, cell, WiFi) location information, such as location fix/coordinates
Notification Manager	Manages various event notifications, such as playing sounds, vibrating, flashing LEDs, and displaying icons in the status bar

Android Runtimes

- Runtime - S/W instructions that execute when the program is running. Basically translated the s/w own code into code the phone can understand.

  Javac → bytecode (dex) → machine code

- Android uses a VM and its runtime environment to run apps (apk)
  - App runs in a contained environment from primary OS
  - Allows cross compatibility (app compiled on PC can still run on the VM)

Android Runtimes

e.KRAAL INNOVATION HUB

- DVM – Dalvik Virtual Machine (JIT)
  - DVM (32bit)
  - Apk → Dex → DexOpt → odex (optimal dalvic code)

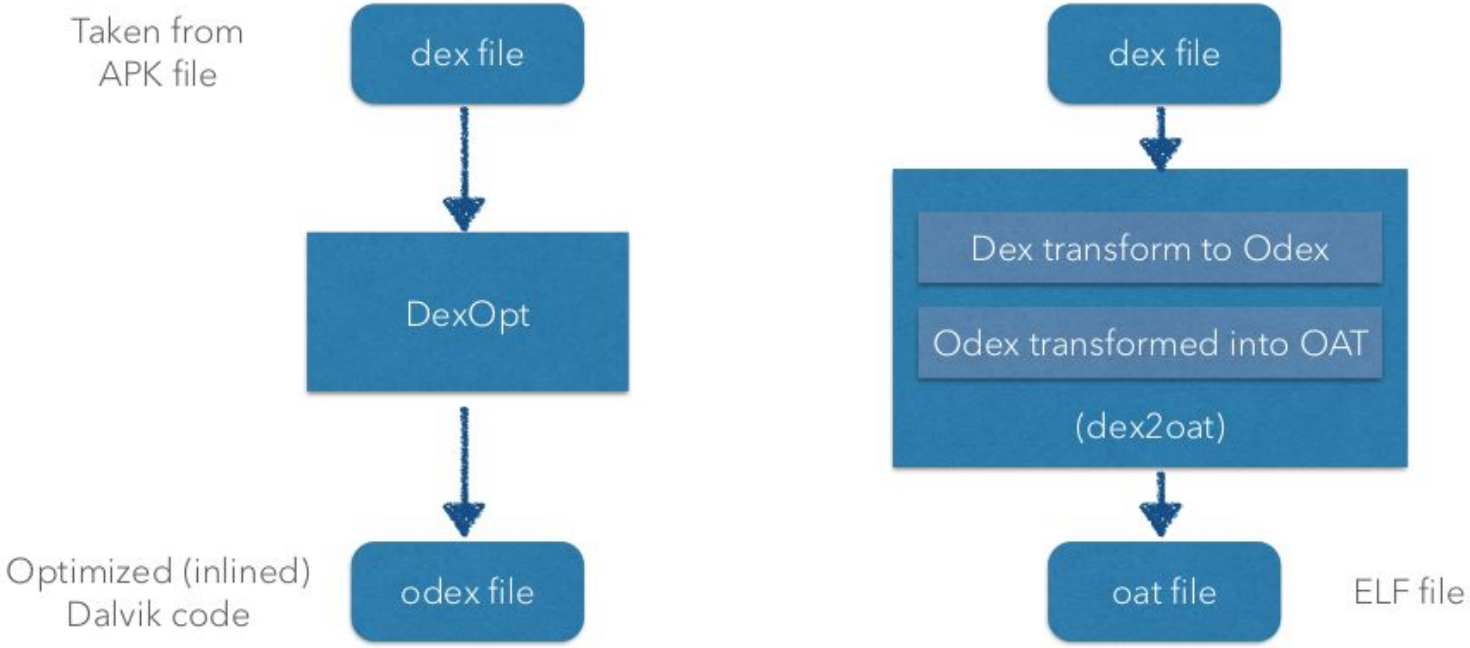
- ART – Android Runtime (Ahead of time) → introduced in android 4.4 (kitkat)
  - ART (32bit/64bit)
  - APK → dex → odex → oat (Executable and Linking Format (ELF))

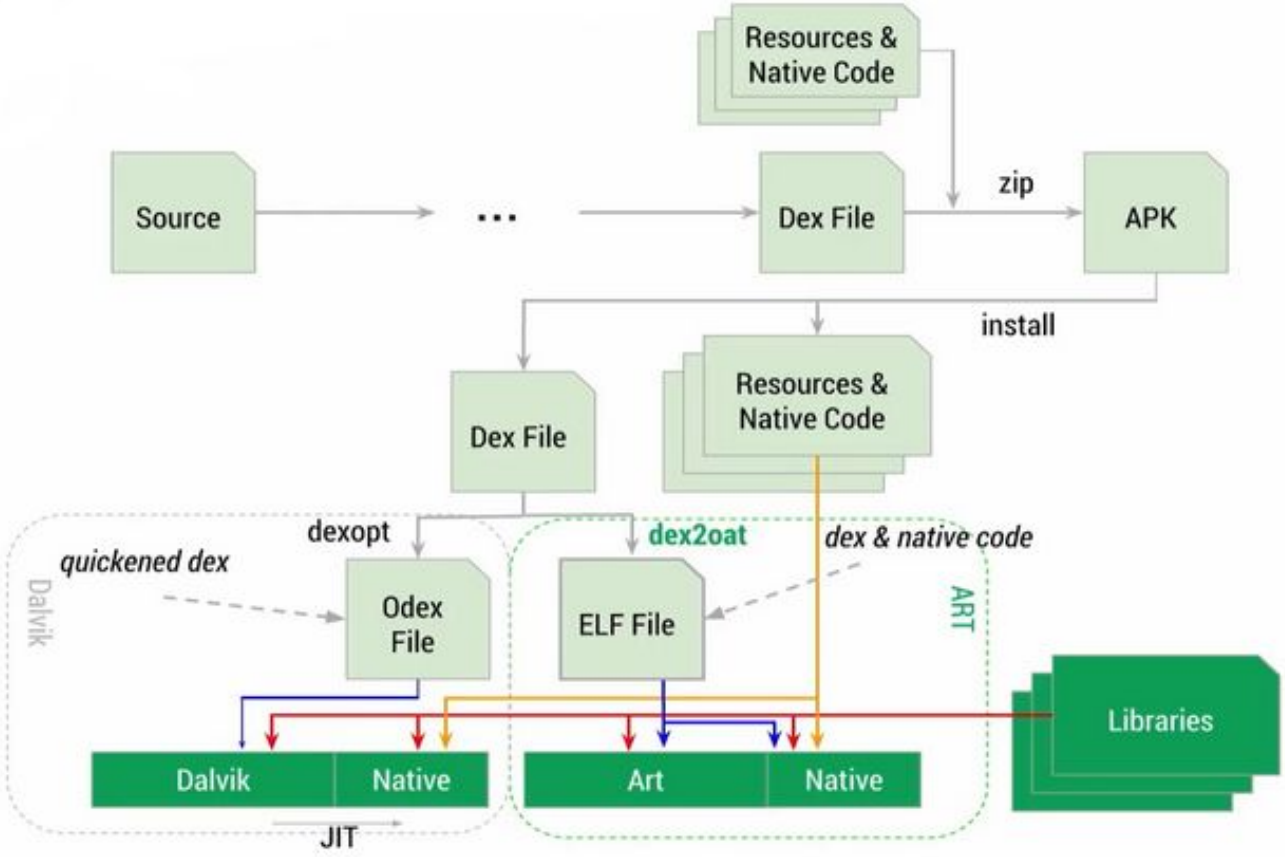
DVM and ART is the past, ART/JIT AOT is the present!!

DALVIK VM VS ART VM

32bit only
"Just In Time"

32bit and 64bit
"Ahead of Time"




Resources &
Native Code
Source
...
Dex File
zip
APK
install
Dex File
Resources &
Native Code
dexopt
dex2oat
dex & native code
quickened dex
Dalvik
Odex
File
ELF File
ART
Libraries
Dalvik
Native
Art
Native
JIT

install

ART vs DVM

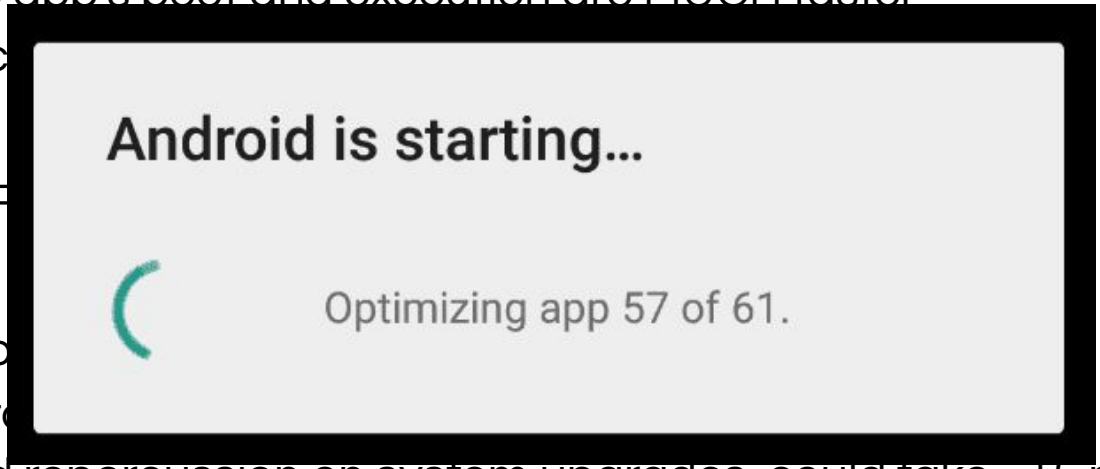
- Pro: The app's boot and execution are MUCH faster
  - Because everything is already compiled

- Cons: ART takes more space on RAM & disk

- Major cons:
  - Installation time takes MUCH longer
  - Bad repercussion on system upgrades, could take ~15 minutes

ART vs DVM

- Pro: The app's boot and execution are MUCH faster
  - Bec
- Cons: AF
- Major co
  - Inst
  - Bad repercussion on system upgrades, could take ~15 minutes



New Version of ART

- Profiled-guided JIT/AOT
  - Introduced in Android 7

- ART profiles an app and precompiles only the
  - "hot" methods, the ones most likely to be used

- Other parts of the app are left uncompiled

New Version of ART

- It is pretty smart...
  - It automatically precompiles methods that are "near to be used"
  - Precompilation only happens when the device is idle and charging

- Biggest Pro:
  - quick path to install / upgrade

DVM JIT vs ART AOT vs ART JIT/AOT

e.KRAAL INNOVATION HUB

	DVM JIT	ART AOT	ART JIT/AOT
App boot time	slowest	fastest	trade-off
App speed	slowest	fastest	trade-off
App install time	fastest	slowest	trade-off
System upgrade time	fastest	slowest	trade-off
RAM/disk usage	lowest	highest	trade-off

ODEX: Optimized DEX

- DEX → dexopt → ODEX
- It is optimized DEX: faster to boot and to run
- Most (all?) system apps that start at boot are ODEXed
- Note: ODEX is an additional file, next to an APK
- Cons
    - ODEX files take space
    - Device-dependent (note: it is still bytecode)

The analogous of ODEX for ART is tricky...

- The new Android Run-Time uses two formats
- The ART format (.art files)
    - It contains pre-initialized classes / objects
- The OAT files
    - Compiled bytecode to machine code, wrapped in an ELF file
    - It can contain one or more DEX files (the actual Dalvik bytecode)
    - Obtained with dex2oat (usually run at install time)
- The confusing part: you still have .odex files!
- Now .odex files are OAT-formatted files!

When are these two formats used?

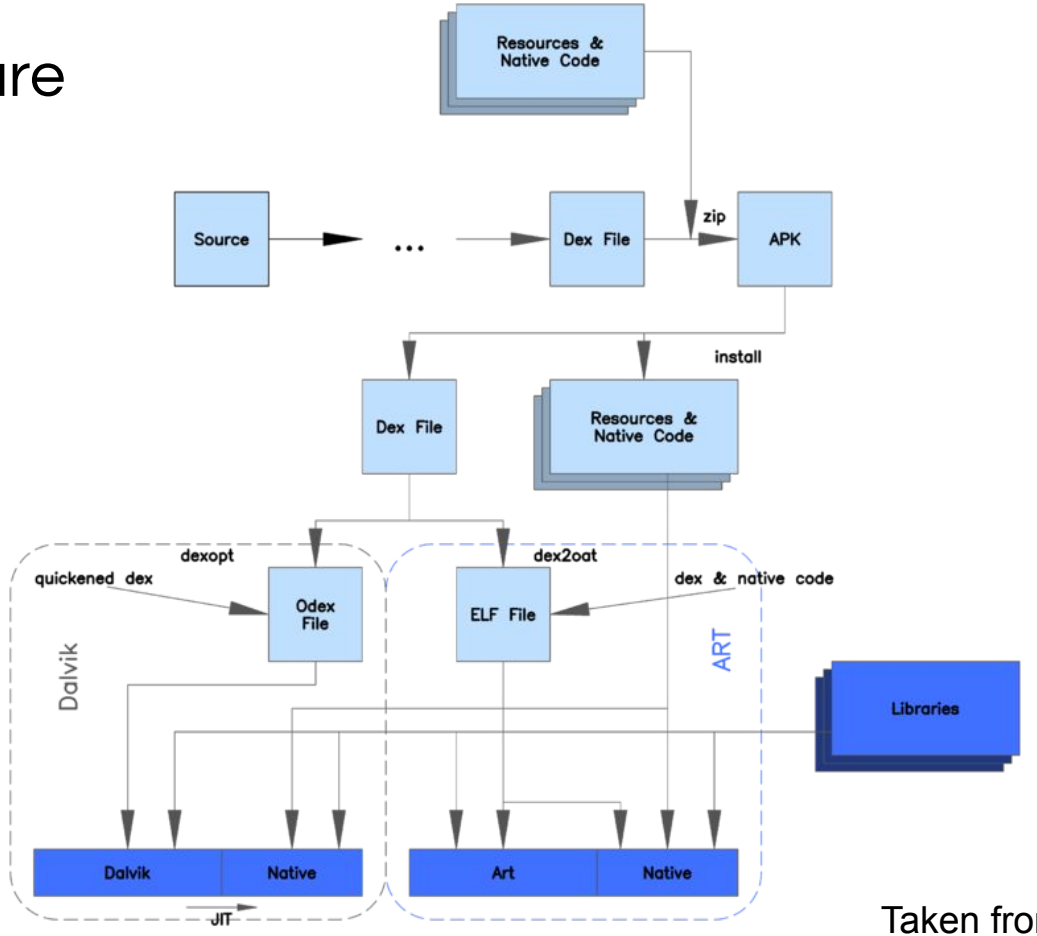
- ART format:
  - Only one file: boot.art
  - It contains the pre-initialized memory for most of the Android framework
    - Huge optimization trick

- OAT format:
  - One important file: boot.oat
    - It contains the pre-compiled most important Android framework libraries
  - All the "traditional" ODEX files are OAT files
  - You can inspect them with Android-provided oatdump

When a new app is starting

- All apps processes are created by forking Zygote
- Zygote can be seen as the "init" of Android
  - A "template" process for each app
- Optimization trick
  - boot.oat is already mapped in memory
  - No need to re-load the framework!

The Big Picture



Taken from [stackoverflow]

Android File System

Android File System

- /boot – This is the boot partition. Includes kernel and ramdisk
- /system – Contains most of the OS components, applications, binaries etc
- /recovery – This is for backup, can be considered an alternative boot partition
- /data – This is the user partition, where the use data is stored
- /cache – This is where frequently accessed data and app components are stored

Android File System

- /misc – This is where the device configurations are stored e.g. wifi

- /storage – This the user storage space to store whatever they wish.sdcard0 (internal) and sdcard1(external)

Shell user had limited access to the file system partitions

Root user has access to “everything”

Android Permission Model

- Low level
  - Linux Kernel does this via Users and Groups
  - Enforces access to the file system (Android Sandbox)

- High level
  - Application permissions defined by Android Runtime via the Dalvik VM and Android Framework
  - Focuses on limiting app permissions and its capabilities.

Android Users, Groups and Permissions

- Some permissions on the high level aspect, map to specific users, groups and capabilities of the OS. (INTERNET -> INET (GID 3003) -> create HTTPURL Connection (open AF_INET and AF_INET6 sockets)

- During installation the package manager extracts app permissions and installs them to /data/system/packages.xml. This file maps all the installed applications to their respective permissions and includes the app's User ID (UID).

- The permissions to group mappings are stored in /etc/permissions/platform.xml

Android Users, Groups and Permissions

- Every application (apk) gets a unique linux user ID and group ID (sandbox)
- Apps run with their unique user ID
- Each running app gets its own dedicated process and Dalvik VM
- Each app has its own storage location in /data/data/<app_name>

Android Users, Groups and Permissions

- All app data is stored in /data/data/<app_name> including their SQLite db
- Each app's folder location is only accessible by the app's unique user ID and group ID
- Cannot copy data from privileged areas directly to the computer (we pivot via the sdcard)

What about security?

- Can an app always do all these things? Nope.

- It has a private folder... that's it?
  - It can start other apps (the main activity is always "exported")
  - It can show things on the screen (when the app is in foreground)

- It can't
  - Open internet connection
  - Get current location
  - Write on the external storage

Android Permission System (overview, ref)

- Android framework defines a long list of permissions

- Each of these "protects" security-sensitive capabilities
  - The ability to "do" something sensitive
    - Open Internet connection, send SMS
  - The ability to "access" sensitive information
    - Location, user contacts, ...

Examples of Permissions

- INTERNET (string: "android.permission.INTERNET")
- ACCESS_NETWORK_STATE, ACCESS_WIFI_STATE, CHANGE_NETWORK_STATE, READ_PHONE_STATE
- ACCESS_COARSE_LOCATION, ACCESS_FINE_LOCATION
- READ_SMS, RECEIVE_SMS, SEND_SMS
- ANSWER_PHONE_CALLS, CALL_PHONE, READ_CALL_LOG, WRITE_CALL_LOG
- READ_CONTACTS, WRITE_CONTACTS
- READ_CALENDAR, WRITE_CALENDAR
- READ_EXTERNAL_STORAGE, WRITE_EXTERNAL_STORAGE
- RECORD_AUDIO, CAMERA
- BLUETOOTH, NFC
- RECEIVE_BOOT_COMPLETED
- SYSTEM_ALERT_WINDOW
- SET_WALLPAPER

{READ,WRITE}_EXTERNAL_STORAGE

- Each app has access to a private directory
  - No other apps can access this directory *

- The device offers an "external storage"
  - In the past: physical "removable" SD Card
  - Currently: part of the file system that apps can use to share files
  - "/sdcard”

/sdcard

- Where your photos & 'downloaded' files are stored
  generic_x86:/sdcard $ ls -l
  - drwxrwx--x 2 root sdcard_rw 4096 2018-09-28 20:59 Alarms
  - drwxrwx--x 3 root sdcard_rw 4096 2018-09-28 20:59 Android
  - drwxrwx--x 2 root sdcard_rw 4096 2018-09-28 20:59 DCIM
  - drwxrwx--x 2 root sdcard_rw 4096 2018-09-28 20:59 Download
  - drwxrwx--x 2 root sdcard_rw 4096 2018-09-28 20:59 Movies
  - drwxrwx--x 2 root sdcard_rw 4096 2018-09-28 20:59 Music
  - drwxrwx--x 2 root sdcard_rw 4096 2018-09-28 20:59 Notifications
  - drwxrwx--x 2 root sdcard_rw 4096 2018-09-28 20:59 Pictures
  - drwxrwx--x 2 root sdcard_rw 4096 2018-09-28 20:59 Podcasts
  - drwxrwx--x 2 root sdcard_rw 4096 2018-09-28 20:59 Ringtones

RECEIVE_BOOT_COMPLETED

- When the system boots, it broadcasts an Intent with the "ACTION_BOOT_COMPLETED" action

- An app can declare an intent filter for this intent so that it can automatically start at boot!

- Useful to gain persistence / survive reboots
  - And that's why the Android folks added a permission requirement

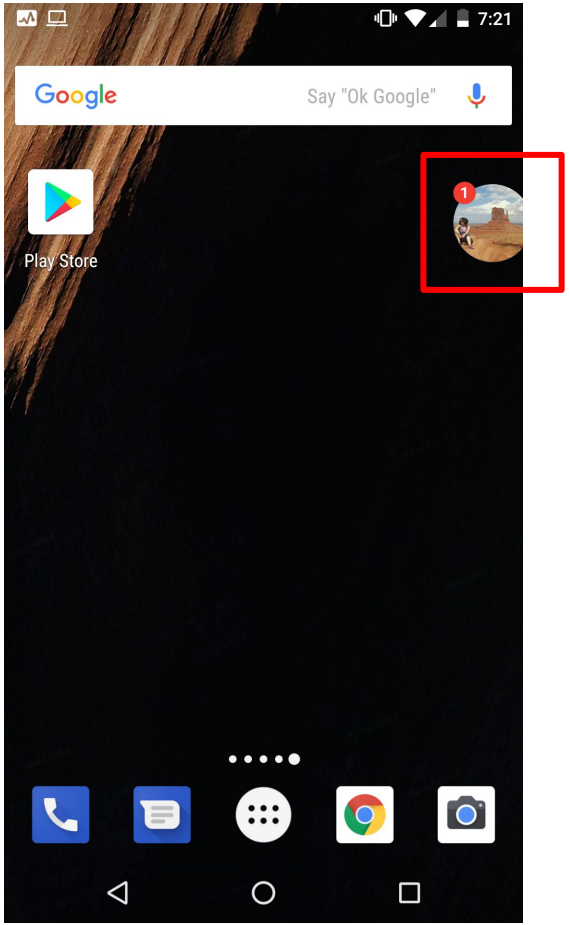
- Note: the app needs to be manually started at least once to receive it

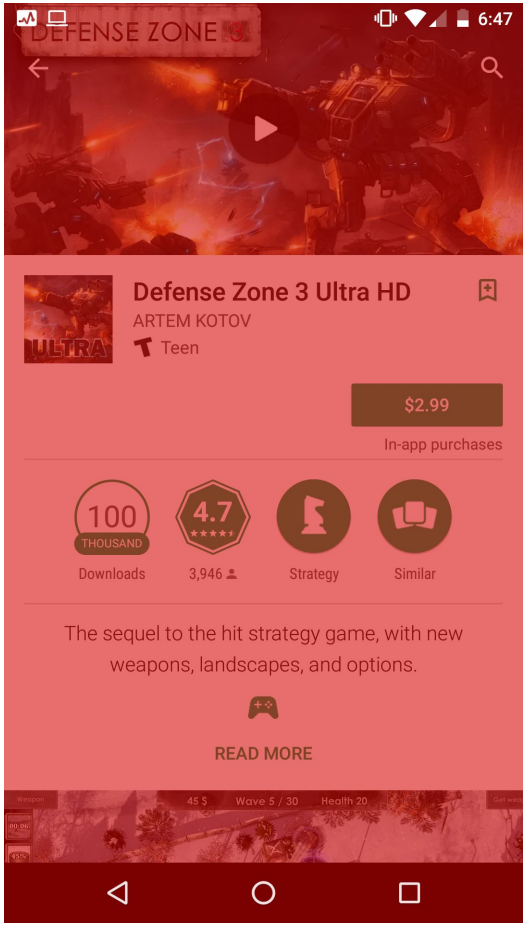
SYSTEM_ALERT_WINDOW

- When the systemDraw arbitrary windows/overlays on top of other apps
  - Can be completely custom: position, shape, content, transparency
  - Can be clickable ⊕ passthrough

- It leads to many UI attacks
  - UI confusion, clickjacking, phishing
  - Teaser: Cloak & Dagger

- My favorite permission!


7:21
Google
Say "Ok Google"
Play Store


6:47
DEFENSE ZONE 3
Defense Zone 3 Ultra HD
ARTEM KOTOV
Teen
$2.99
In-app purchases
100
THOUSAND
Downloads
4.7
3,946
Strategy
Similar
The sequel to the hit strategy game, with new weapons, landscapes, and options.
READ MORE

Permissions Protection Levels (doc)

- Normal
  - The system automatically grants the app that permission at install time. The system doesn't prompt the user to grant normal permissions, and users cannot revoke these permissions.

- Dangerous
  - To use a dangerous permission, your app must prompt the user to grant permission at runtime.

Permission Granting

- Normal permissions
  - no explicit granting necessary

- Dangerous permissions
  - The user needs to be asked

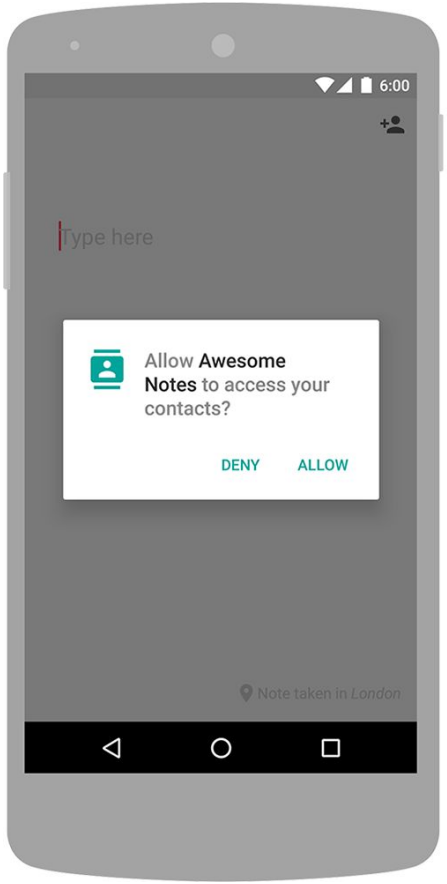
- Signature permissions
  - It depends
    - Granted at install time when app is signed by same certificate of defining app
    - Otherwise, the user is asked
  - Not all of these are available to third-party apps

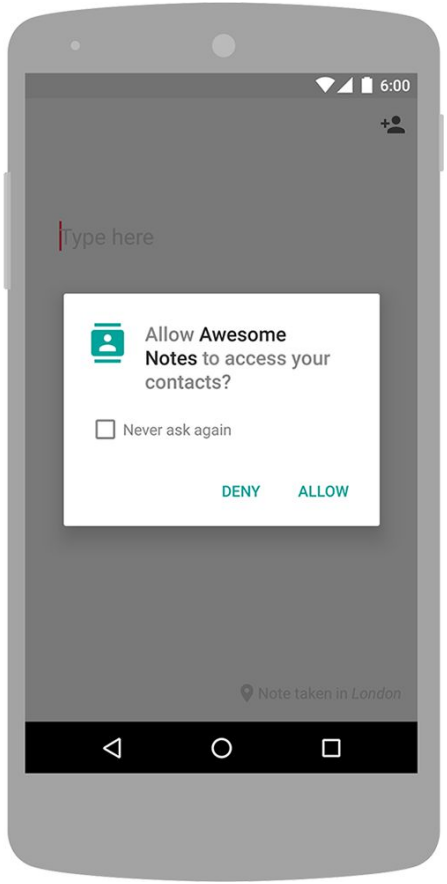
Granting Dangerous Permissions (doc)

- Runtime requests
  - If device's API level >=23 (Android 6) AND app's targetSdkVersion >= 23

- Facts
  - The user is not notified at install time
  - The app initially doesn't have the permission, but it can be run OK
  - App needs to ask at runtime ("runtime prompt")

- Users have the option to disable them


6:00
Type here
Allow Awesome Notes to access your contacts?
DENY
ALLOW
Note taken in London

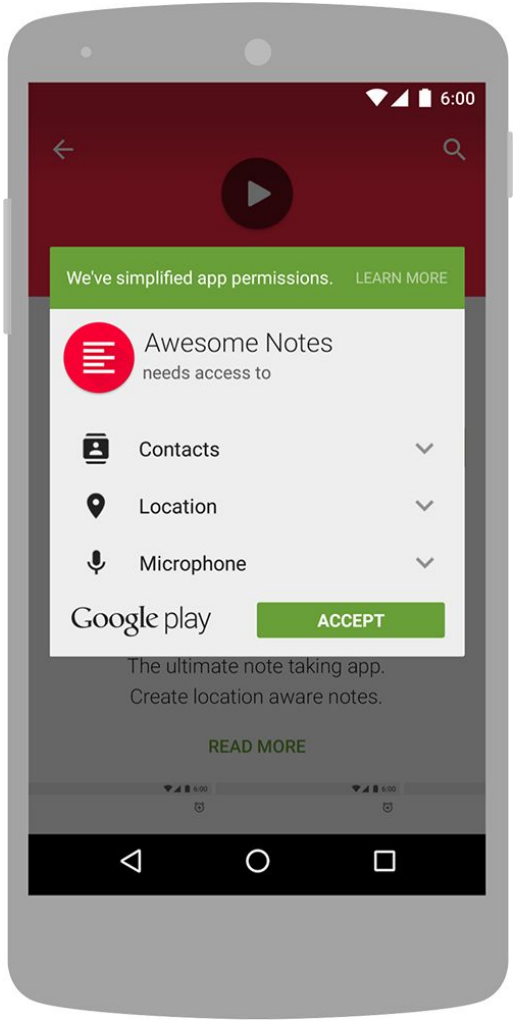

6:00
Type here
Allow Awesome Notes to access your contacts?
Never ask again
DENY
ALLOW
Note taken in London

Granting Dangerous Permissions (doc)

- Install-time requests
  - If device's API level <23 OR app's targetSdkVersion < 23

- Facts
  - The user is asked about all permissions at installation time
  - If user accepts: all permissions are granted
  - If user does not accept: app installation is aborted

- Users do not have the option to disable them*
  - *Starting from Android 10, the user can disable these as well


6:00
We've simplified app permissions.
LEARN MORE
Awesome Notes
needs access to
Contacts
Location
Microphone
Google play
ACCEPT
The ultimate note taking app.
Create location aware notes.
READ MORE

Runtime vs Install-time Prompts

- Runtime
  - Pros: Users can install apps without giving all permissions
  - Pros: Users have contextual information to decide accept/reject
  - Pros: Permissions can be selectively enabled/disabled
  - Cons: Multiple prompts can be annoying

- Install time
  - Pros: no annoying prompts
  - Cons: "all-or-nothing", grant all permissions or app can't be installed
  - Cons: No contextual info to take informed decisions

Permissions Groups

- Permissions are organized in groups

- Permissions requests are handled at a group level
  - User grants X ~> all permissions in X's group are granted as well
  - User grants X => all other permissions in X's group can be requested WITHOUT triggering an explicit prompt to the user

- Security implications!

Permissions Groups: An Example

- PerSMS permission group
  - RECEIVE_SMS, READ_SMS, **SEND_SMS**

- PHONE permission group
  - READ_PHONE_STATE, READ_PHONE_NUMBERS, **CALL_PHONE, ANSWER_PHONE_CALLS**

- Group/permission mappings: link

Permissions from an app's perspective

Permission Request

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="com.example.awesomeapp">

    <uses-permission android:name="android.permission.SEND_SMS"/>
    <application ...>
        ...
    </application>
</manifest>
```

Custom Permissions (doc)

- Apps can define "custom" permissions!

```
<permission
        android:name="com.example.myapp.permission.DEADLY_STUFF"
        android:label="@string/permlab_deadlyStuff"
        android:description="@string/permdesc_deadlyStuff"
        android:permissionGroup="android.permission-group.DEADLY"
        android:protectionLevel="signature" />
```

- The "system" permissions are defined in the same way
  - AndroidManifest.xml

Components Permission Enforcement

- Apps' components can specify which permissions are required to use them

```
<receiver
  android:name="com.example.myapp.DeadlyReceiver"
  android:permission="com.example.myapp.permission.DEADLY_STUFF">
  <intent-filter>
    <action android:name="com.example.myapp.action.SHOOT"/>
  </intent-filter>
</receiver>
```

Custom Permission Use Cases

- protectionLevel = "signature"
  - Only apps signed by the same developer / company can get it
  - Example: big company with many apps
    - Facebook wants all its apps to have access to users' posts
    - Facebook does not want any other app to have access to this information

- protectionLevel = "dangerous"
  - App controls security-related things / information (which are not strictly related to Android)
  - App wants to provide this capability to other apps, but it wants to warn the user first

Permission Enforcement Implementation

- Two technical ways: Linux groups vs. explicit checks

- Linux groups
  - INTERNET permission ~> app's user is added to "inet" Linux group
  - BLUETOOTH permission ~> app's user is added to "bt_net" Linux group
  - Declaration in AOSP: code

- Explicit check

e.KRAAL
INNOVATION HUB

Android SDK

Android SDK

- This is a set of tools used to develop apps for android. It includes required libraries, debugger, android application interfaces (APIs) etc

- Android Virtual Device (AVD)
  - These are basically the android virtual devices that emulate a real android device.

- Android Debug Bridge (ADB)
  - Tool that facilitates interaction with connected devices and emulators.

Android AVD

- AVD/emulator are useful for learning app execution and behaviour on a device and validating your findings

- Useful for testing forensic or reverse engineering tools, android devices or apps

- The emulator data is stored in:
  - Linux - /home/user/.android
  - Mac OS X - /Users/user/.android
  - Windows – C:\Users\username\.android

Android AVD

- Key files of interest:
  - cache.img – disk image of /cache partition
  - sdcard.img – disk image of SD card (if created during setup)
  - userdata-qemu.img – disk image of /data partition

- These files can very useful when testing applications and conducting application forensics

Android ADB

- Definition
  - This is a toolkit that allows interactive debugging and inspecting device state.
- Keeps track of all devices/emulators connected to a host
- Offers various services to clients

Android ADB - Caveats

- USB debugging has to be manually enabled via the developer options
- Developer options is hidden since android 4.2+ (look for build number under settings, tap 7 times to enable)
- On android 4.2.2+ it requires authentication
- Enabling developer options on a locked phone is difficult

Android ADB - Components

- ADB server – Runs on host machine as a background process
- ADB daemon (adbd) – Runs on android device/emulator and provides the actual service
- ADB client (cli based) – Allows you to send commands to the particular device/emulator

Android ADB - Functionality

- ADB Functionality
  - Copy files from a device
  - Debug apps running on the device
  - Execute shell commands on device
  - Get system app and logs
  - Install and remove apps

- ADB Commands
  - adb restart-server
  - adb devices
  - adb shell

More adb commands

- $ adb devices
- $ adb install app.apk
- $ adb uninstall com.mobisec.testapp # package name
- $ adb logcat
- $ adb push file.txt /sdcard/file.txt # push to device
- $ adb pull /sdcard/file.txt file.txt # pull from device

More adb commands

- $ adb shell
  - Get a shell on the device

- $ adb shell ls
  - Execute "ls" on the device

- $ adb shell am start -n <pkgname>/<component>
- $ adb shell pm grant <pkgname> <permission>
- $ adb shell dumpsys

apt

- It comes with Android SDK
  - <sdk>/build-tools/26.0.2/aapt
- It takes an APK as input
- It can dump tons of useful info
  - Package name, components, main activity, permissions
  - strings, resources, ...
- $ aapt dump badging <apk_path>

Android ADB - Scenarios

- Scenario 1: Traditional adb

Computer **Phone**

adb server

adb client -------> adb daemon

Android ADB - Scenarios

- Scenario 2: peer to peer adb

Phone **Phone**

adb server

adb client -------> adb daemon

e.KRAAL
INNOVATION HUB

Android Rooting

Getting Root!

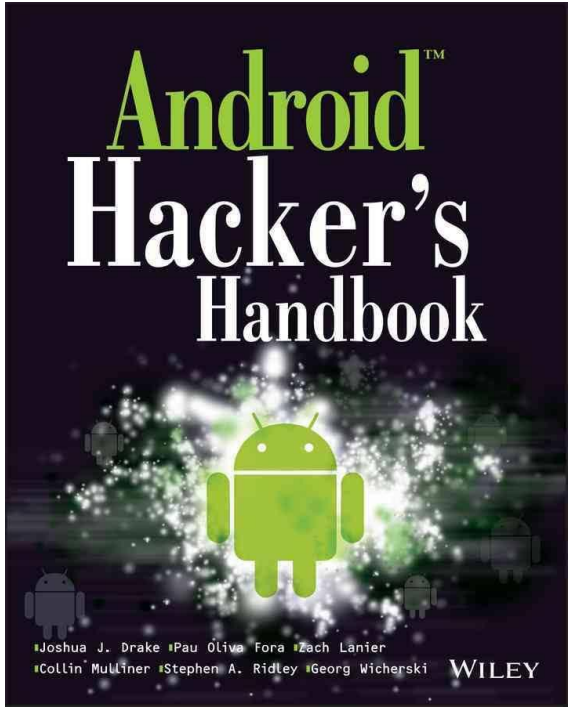
- This is needed for getting full access to the device in order to acquire data
- Not enabled by default and quite difficult to gain on some devices
- It isn't always the best choice in forensics because
  - It will change data on the device hence altering the evidence
  - It's time consuming to get due to device customizations by vendors
- It makes the device vulnerable to many exploits
- Rooting apps are available e.g. KingRoot, KingoRoot,One click root,

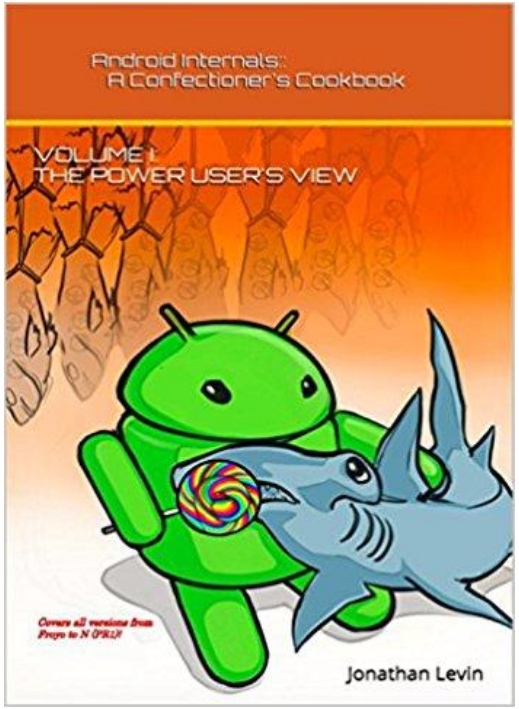
Getting Root!

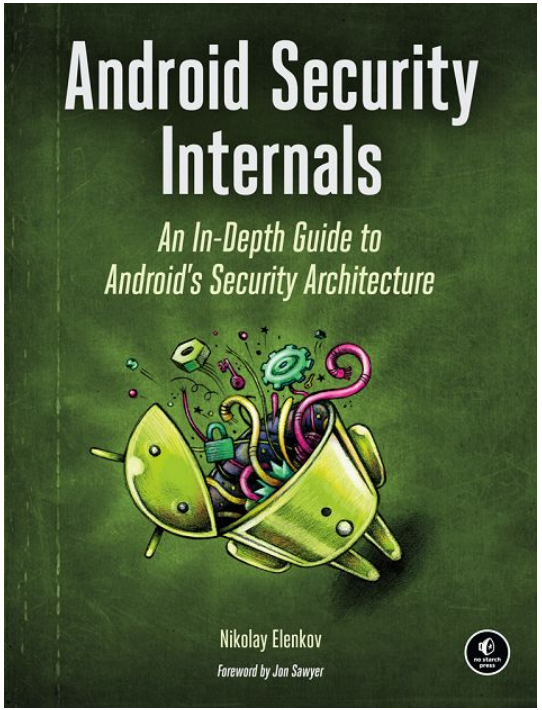
- Soft root
  - Roots the device only until it is rebooted, then root is disabled. This is what we want.
- Hard root
  - Root persists after reboot, and can be permanent.
- Recovery mode root
  - Flashing/installing a custom recovery partition, allowing root access only in recovery mode.

Lab Walkthrough

Recommended reading







References

- https://mobilesecuritywiki.com
- https://github.com/sindresorhus/awesome
- https://github.com/rednaga