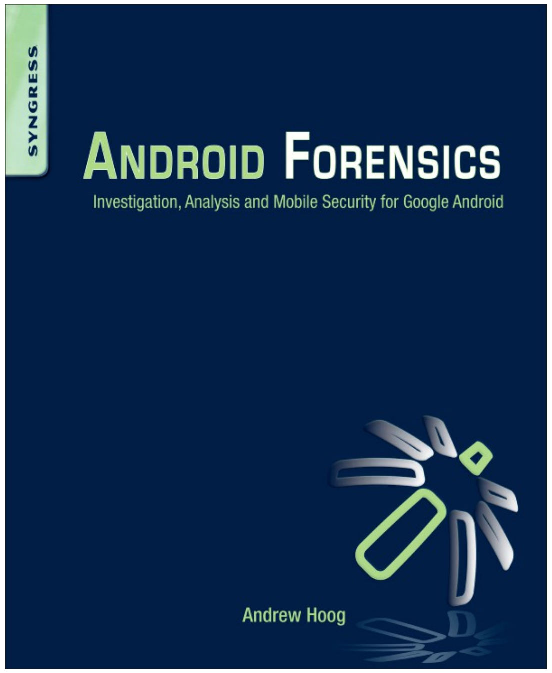
VIAFORENSICS

viaforensics.com

# Android forensics

boot process, security, system, rooting, dumping, analysis, etc.

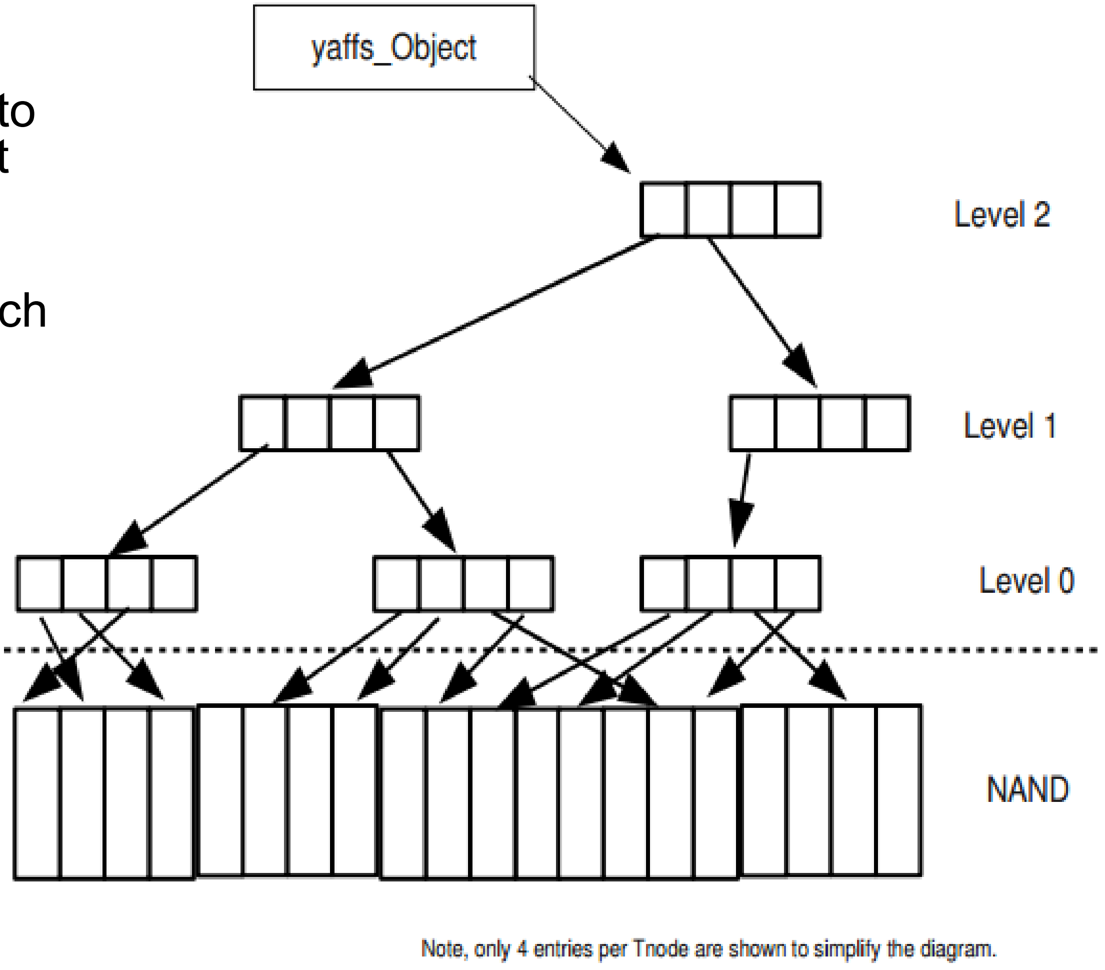

SYNGRESS
ANDROID FORENSICS
Investigation, Analysis and Mobile Security for Google Android
Andrew Hoog

## YAFFS2 (Yet Another Flash File System) 1

- Developed by Aleph One Ltd, (Charles Manning)
  - http://www.yaffs.net/how-yaffs-works-internals
- YAFFS2 was built specifically for the growing NAND flash devices and has a number of important features that address the stringent needs of this medium
- YAFFS2 is a
  - log-structured file system (which protects data even through unexpected power outages)
  - provides built in wear-leveling and error correction
  - capable of handling bad blocks
  - is fast and has a small footprint in RAM
- YAFFS2 address the memory in blocks (128 kB) through the MTD subsystem and each block contains a set number of pages (chunks in YAFFS doc with the size of 2 kB)
- All data structures stored in YAFFS2 are referred to as Objects and can be files, directories, symbolic links, and hard links
- Each chunk (page) either stores a yaffs_ObjectHeader (object metadata) or data (tnode tree, extents etc.) for the object

# YAFFS tree nodes

- Each file has a Tnode tree to provide the mapping from file position to actual NAND chunk address
- The Tnode tree is made up of Tnodes (tree nodes) arranged in levels, each of which holds either
  - At levels greater than 0, a Tnode has 8 pointers to other Tnodes in the next level down.
  - At level 0, a Tnode has 16 NAND chunk Ids which identify the chunk's location in RAM.
  - Tnodes typically make up the bulk of the YAFFS' RAM usage.


yaffs_Object
Level 2
Level 1
Level 0
NAND
Note, only 4 entries per Tnode are shown to simplify the diagram.

## YAFFS2 (Yet Another Flash File System) 2

- The yaffs_ObjectHeader tracks various information including the Object type, the parent object, a checksum of the name to speed up searching, the object name, permissions and ownership, MAC information, and the size of the object if it is a file
- In the 64-byte OOB/spare area, YAFFS2 not only stores critical information about the chunk but also shares the area with the MTD subsystem. The critical YAFFS2 tags are as follows:
  - 1 byte: block state (0xFF if block is good, any other value for a bad block)
  - 4 bytes: 32-bit chunk ID (0 indicates that chunk is storing a yaffs_ObjectHeader, else data)
  - 4 bytes: 32-bit Object ID (similar to traditional Unix inode)
  - 2 bytes: number of data blocks in this chunk (all but final chunk will be fully allocated)
  - 4 bytes: sequence number for this block
  - 3 bytes: ECC for tags (in Android, handled by MTD)
  - 12 bytes: ECC for data (in Android, handled by MTD)
- Find detailed information about the YAFFS2 file system by examining the /proc/yaffs file

# Process and RAM dumps

- Recently, solutions for examining Android memory have emerged
- Android provides a mechanism for dumping an application's memory to a file by sending the app a special signal (SIGUSR1)
  - To send the signal, you need an app's PID, which you can find with the ps command:

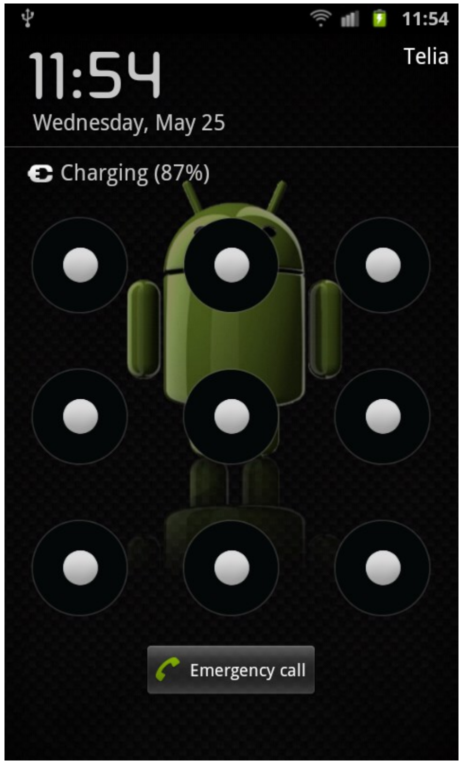
```
D:\>adb shell ps | less
USER      PID    PPID  VSIZE  RSS      WCHAN     PC          NAME
root       1      0     696    500   c02ae25a 0805a406 S /init
...
```

  - We must be root (su) and set # chmod 777 to /data/misc
  - Then we can dump the process memory with: # kill -10 1
  - If all went well we may find a file named something like: heap-dump-tm1296350817-pid1.hprof
  - When we have pulled the file to our computer we can run strings and perform other analysis methods on this file
- It is also possible to dump the whole RAM with a tool named LIME
- However it needs a kernel mode driver to be installed which immediately begins to dump the memory
  - # insmod /sdcard/lime.ko "path=/sdcard/dump.lime format=lime"
  - https://code.google.com/p/lime-forensics/

# Android forensic techniques 1

- Earlier discussed methods for handling mobile phones applies
- Powered off devices
  - Boot into recovery mode and check for ADB connectivity (device may have a custom recovery with root access)
  - This may give us data access without booting up the system fully
- Pass codes / screen locks
  - Face unlock, draw pattern, numeric PIN or alpha numeric password (voice code may also exist)
- The draw pattern is located in /data/system/gesture.key
  - Interpreted as a matrix (3x3, 6x6 and 9x9) with (0,0) in upper left corner of screen
  - The numbers (3 to ? digits) forms a SHA-1 hash
- Password or PIN have 4 – 16 characters
  - Stored as a salted hash in /data/system/password.key
- Projektarbete av Rickard Andersson
  - https://github.com/rickard2/Android-Lock-Screen

[server]\embedded_forensics\Android-tools\
Android Forensics Study of Password and Pattern Lock Protection



# Android forensic techniques 2

- Powered on devices that are locked
  - Try # adb devices to check if USB debugging is enabled, and if so with root access (# adb shell su)
  - Smudge attack – look for smudge patterns on the touch screen
  - Screen lock bypass app – install an app from Googple Play which send a special intent to the system which disable the lock screen
  - Use the Gmail user/pass (if you know it) when you have consumed all the lock screen attempts and reset the pass code
- Powered on devices which are open
  - Several methods are possible and care must be taken
  - Enable USB debugging and root the device?
- If locked out and USB connectivity attempts are un-successful consider flashing a modified recovery partition
  - Enter fastboot/download mode which is a NAND flash update protocol executed over USB while in bootloader mode
  - Most devices ship with bootloader protection enabled, which prohibits the use of this protocol
  - Security S-OFF or S-ON – only signed images can be flashed

## Imaging UMS (USB Mass Storage)

- Every Android device to date has either an external Secure Digital (SD) card or an Embedded MultiMediaCard (eMMC)
- Both use NAND flash, are based on the MMC specification, and have embedded storage controllers supporting file systems that are not NAND flash aware
- The recommended approach for imaging UMS devices is via the UMS interface (USB cable)
  - Moving to eMMC storage meant that the mass storage was no longer removable
  - Apps can now run from the SD card and in this scenario, the .apk files are encrypted if SD card is removed
  - Newer devices are using RAM disks (tmpfs) more frequently to store user data that might be helpful in an investigation. Often, removing the SD card requires the device to be shut down and the battery removed, thus losing the ability to recover the temporal data
- If not possible use a cross compiled forensic version of dd and a tool as netcat to dump the image over the network or ADB port forwarding dumping it to local computer

# Logical techniques

- Logical techniques works in most scenarios if USB debugging is enabled and usually provide sufficient data for the case (much less effort than physical extraction)
- ADB pull with root privileges is simple and effective

```
D:\tmp>adb pull /data adbpull
pull: building file list.
... and then whole data folder is copied to local computer, may
fail in the middle so monitor the pull and make smaller data pulls
```

- Analyze data saved by users backup programs from Google play or built into custom recoverys as nandroid
  - If file system is YAFFS2 run unyaffs on them. If file system is ext4 nandroid store files in ordinary tar.gz archives
- Most forensic vendors have Android logical capture solutions
- Using the Content Providers which are a key feature of the Android platform
  - Apps can share their databases with other apps

# Physical techniques

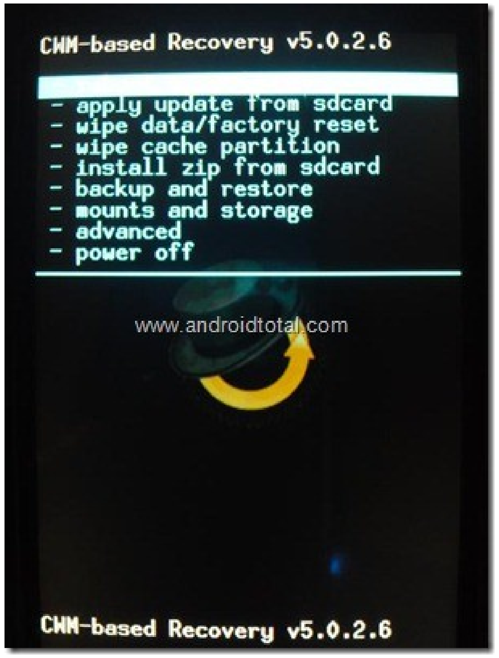
- Falls into two categories
  - Hardware based techniques as JTAG, chip-off etc.
  - Software based techniques (aside from being easier to execute)
    - Often provide direct access to file systems to allow a complete copy of all logical files (simplifies some analysis) or partitions (raw image)
    - Bootloader exploit, instead of doing a firmware update an image is created via an injected module (common vendor solution, no root needed)
    - Provide very little risk of damaging the device or data loss
- To execute the software-based physical techniques, you first must gain root privileges and then run the acquisition programs
  - **Temporary root** privileges attained by a root exploit, which does not survive a reboot
    - Typically the adb daemon is not running as root in this instance
  - **Full root access** attained through a custom ROM or **persistent root exploit**
    - Custom ROMs often run the adb daemon as root while most of the persistent root exploits do not
  - **Recovery mode root** attained by flashing a custom recovery partition or part of a custom ROM
    - Custom ROMs often run the adb daemon as root as do most of the modified recovery partitions

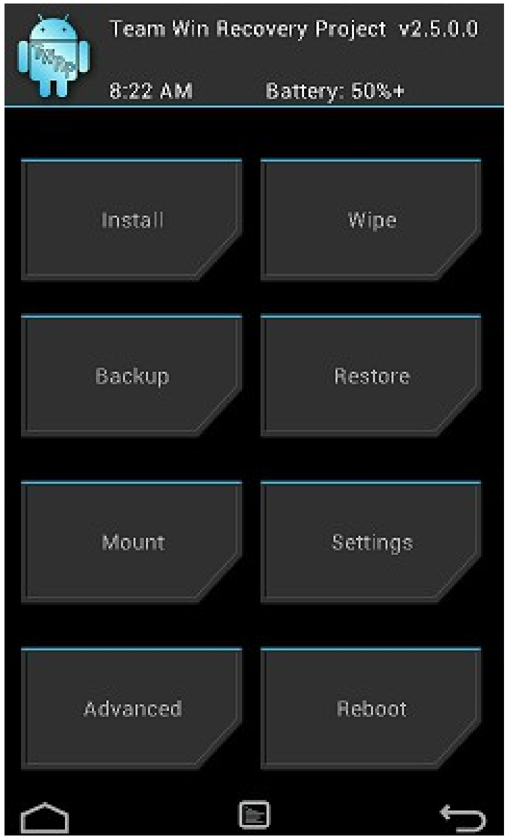
## Gaining root

- Gaining root privileges can be very difficult and frustrating, it also put changes into the device
- The techniques for root privileges differ not only for each manufacturer and device but for each version of Android and Linux kernel in use
  - The number of devices and versions result in many variations!
  - Newer Android versions has made the process harder
- You must perform and test the process on a similar device ensuring that no data is lost
- If the device does not already have root privileges, you can research possible techniques online as XDA-developers
  - Time consuming and may contain inaccurate information
- Recovery mode is an operating mode for Android that was designed to apply updates, format the device, and perform other maintenance on the devices
  - The stock recovery mode on most devices is very basic, only provides a number of limited functions

# Custom recovery

- Custom recovery partitions usually allow root privileges through the shell (**#** prompt = root, **$** prompt = no) and many other functions
- These new recovery partitions are typically installed by the user when the device is rooted and provide various functions that simplify the backup and update process needed from the custom ROMS
- The most popular are ClockworkMod and TWRP
  - http://www.clockworkmod.com/
  - http://teamw.in/project/twrp2
- Use caution when installing a custom recovery partition as the process may contain kernel and radio updates that could render the device unusable if there are incompatibilities between the device, kernel, and radio firmware
- Boot into fastboot/download mode (if possible) and restore the recovery

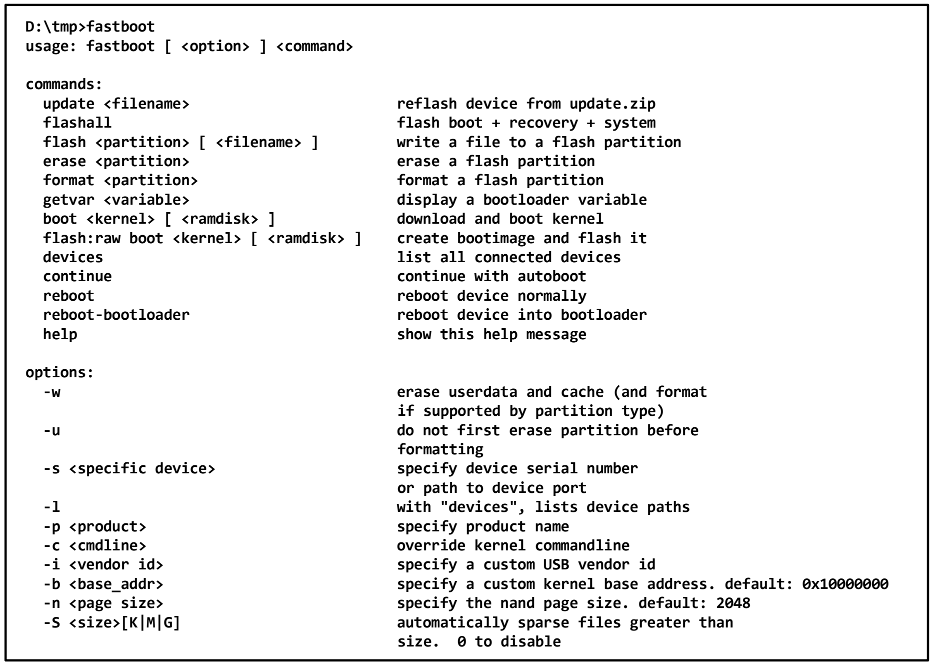

CWM-based Recovery v5.0.2.6
- apply update from sdcard
- wipe data/factory reset
- wipe cache partition
- install zip from sdcard
- backup and restore
- mounts and storage
- advanced
- power off
www.androidtotal.com
CWM-based Recovery v5.0.2.6



## Bootloader and fastboot/download mode

- The bootloader is executed early in the Android boot process and is responsible for selecting and loading a kernel
- For most devices special NAND flash (for writing) software exists, which typically is developed by the manufacturer
- This software can interact with the bootloader if support for fastboot/download mode is enabled
- However, the bootloaders of most devices are shipped from the factory in a locked state (S-ON), which prevents such updates

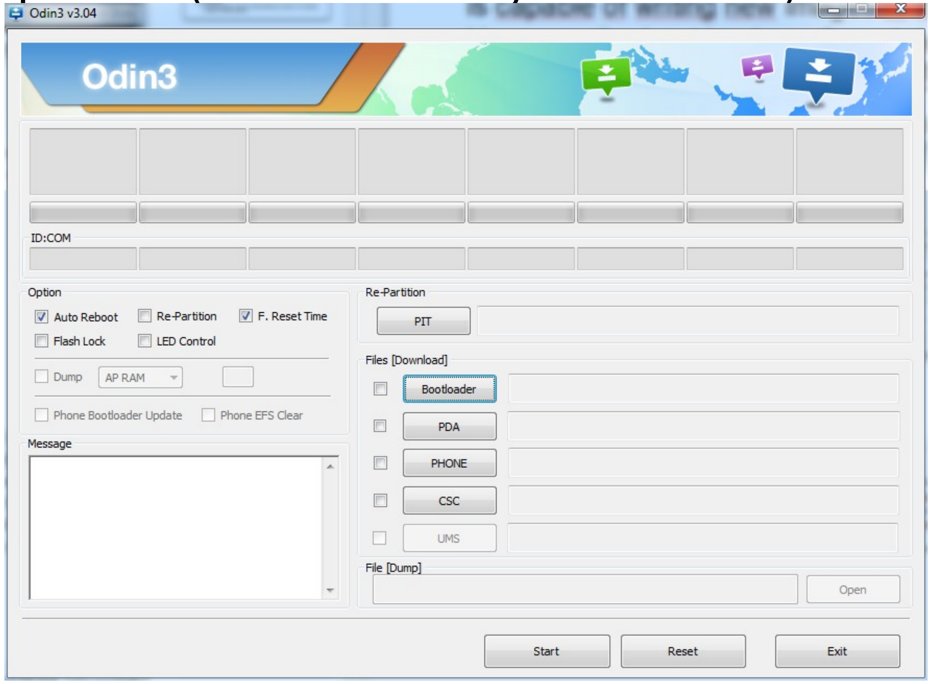
# fastboot oem unlock – will unlock Nexus phones (but also factory reset/erase)



```
D:\tmp>fastboot
usage: fastboot [ <option> ] <command>

commands:
  update <filename>                        reflash device from update.zip
  flashall                                 flash boot + recovery + system
  flash <partition> [ <filename> ]         write a file to a flash partition
  erase <partition>                        erase a flash partition
  format <partition>                       format a flash partition
  getvar <variable>                        display a bootloader variable
  boot <kernel> [ <ramdisk> ]              download and boot kernel
  flash:raw boot <kernel> [ <ramdisk> ]    create bootimage and flash it
  devices                                  list all connected devices
  continue                                 continue with autoboot
  reboot                                   reboot device normally
  reboot-bootloader                        reboot device into bootloader
  help                                     show this help message

options:
  -w                                       erase userdata and cache (and format
                                           if supported by partition type)
  -u                                       do not first erase partition before
                                           formatting
  -s <specific device>                     specify device serial number
                                           or path to device port
  -l                                       with "devices", lists device paths
  -p <product>                             specify product name
  -c <cmdline>                             override kernel commandline
  -i <vendor id>                           specify a custom USB vendor id
  -b <base_addr>                           specify a custom kernel base address. default: 0x10000000
  -n <page size>                           specify the nand page size. default: 2048
  -S <size>[K|M|G]                         automatically sparse files greater than
                                           size.  0 to disable
```



[✓] Auto Reboot  [ ] Re-Partition  [✓] F. Reset Time
[ ] Flash Lock  [ ] LED Control

# Bootloader, fastboot and recovery?

- SPL (Secondary Program Loader)
  - A piece of bootcode that initiates the startup of the phone, displaying the initial splashscreen for the device, and loading the initial files from the ROM
  - It checks to see if a button combination is pressed on bootup (such as that to enter recovery mode or the bootloader), and loads the relevant system software
  - If no special instruction is given by holding keys, the bootloader loads the normal system software by initialising the boot process from the boot partition
  - Flashing your SPL is risky, as the process failing will probably result in a broken, or bricked phone, since the SPL is executed very early on in the boot process, and any error here will prevent access to the recovery or bootloader features
- The bootloader consists of SPL and IPL (Initial Program Loader)
  - The IPL or Boot ROM is a firmware that runs on every startup
  - Usuallly press pwr+vol-down to enter bootloader
- Fastboot or download mode
  - A state from where you can reflash the device via the fastboot command (or other manufacturer specific utility) in the Android-SDK etc.
- Recovery mode
  - The recovery partition is a boot mode for your phone that allows you to wipe or update (re-program) partitions on your phone, usuallly pwr+vol-up

# SPL, RUU and OTA?

- A crafted SPL in conjunction with the IPL can comprise a device's bootloader
- Aside from bootstrapping (a self-sustaining process that proceeds without external help) Android, the bootloader also fulfills various diagnostic functions. One of these functions is the manipulation of data in the device's internal flash and RAM
- Depending on the SPL installed - as custom executable code
  - The user can apply signed or unsigned flash NAND images and do many other tasks. Note that the SPL is installed and operates independently of the Android build that runs atop of it
  - Replace the stock Android recovery image with a custom one, which adds many features including “nandroid” backups, the ability to use custom ROMs, and a greater amount of flexibility and customization of your Android phone
- A RUU (ROM Upgrade Utility) or factory image is a program or zipped archive which upgrades/reset your phone from your computer via fastboot/download mode
- OTA (Over-the-air programming) is an incremental factory image which do not reset the phone (user data is preserved)

# root/recovery - practical LG G2

- root
  - A special g2_security (LG dev?) file is pushed to the SD card
  - When connecting with ADB the LG phone looks this file up and enable root via ADB only
  - If we want to have root in the ROM we must push the su binary to /system/xbin/ and chmod it with 06755 (setuid + setgid)
  - Users executing it will gain the privileges of the user who owns it
  - Then we install superuser.apk - we can grant apps su if requested
- Recovery

```
adb wait-for-device
adb push g2_security /sdcard/g2_security
adb push recovery.img /sdcard/recovery.img
adb root
adb wait-for-device
adb shell dd if=/sdcard/recovery.img of=/dev/block/platform/msm_sdcc.1/by-name/recovery
echo If show msg like "12374016 bytes transferred in ... sec ...", you have succeeded...
echo Press any key reboot to recovery
pause > nul
adb shell sync
adb shell sync
adb reboot recovery
```

# Loki by Dan Rosenberg (@djrbliss)

- Exploiting Samsung Galaxy S4 (and others) Secure Boot
- Qualcomm use software ”QFuses” with signatures which are validated at each stage of the boot chain (pbl -> sbl1 -> sbl2 -> sbl3 -> aboot) to implement a trusted boot sequence
- Secure Boot with aboot enforce signature checks on the boot and recovery (kernel and ramdisk) partitions
- Aboot is a open source project named lk ("Little Kernel") with a major flaw
  - It first loads the recovery into memory and calculate a SHA1 signature
  - Then aboot decrypt its own signature hash for the recovery partition with a public key matching it against the SHA1 hash signature – the check_sig() function from step above
- It is possible to make aboot read a specially crafted boot image that specifies a ramdisk load address equal to the address of the check_sig() function in aboot physical memory
- When aboot reads the supposed ramdisk from eMMC flash, it actually overwrites the check_sig() function with shellcode, and then invokes it
- The shellcode patches everything together and return zero – success!
  - http://blog.azimuthsecurity.com/2013/05/exploiting-samsung-galaxy-s4-secure-boot.html

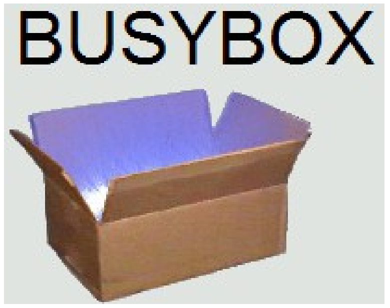
## ViaForensics AFPhysical method with nanddump

- The overall process for AFPhysical in free chapter 6 of Android Forensics and Mobile Security, page 278 →
- Acquire root privileges on the target Android device
  - Replace existing recovery partition
- Identify NAND flash partitions which need to be imaged
- Upload forensic binaries to the target Android device
  - We may need to cross-compile nanddump from the MTD-utils package since we need ARM architecture binaries
  - Adb push needed binaries to /dev/local or other tmpfs
- Acquire physical image of NAND flash partitions
  - Use **adb port forward to create a network between the pc and Android device over USB**. We need to do this since we are running from recovery (WiFi does not work)
  - Or place a SD card into the device, mount, and save locally
- Remove forensic binaries if any were stored on nonvolatile storage

## ViaForensics AFPhysical method - further notes

- Remote mirroring using netcat and dd over Wi-Fi
  - D:\tmp> netcat -l -p 12345 > mtd6.ext4.dd
  - root@android:/ # dd if=/dev/block/mtdblock6 | netcat ip-number 12345
- It is possible to dump with cat as well
  - cat /dev/block/mmcblk0p2 > /sdcard/data.dd
- There is no point of using hashes before we have written the dump to computers disk - we deal with NAND!
- Nanddump grabs everything including OOB/spare area
  - We need too ”clean” the image from OOB if we want to carve
  - We do not ”clean” the image if the image is to be mounted (in Linux)
- If the images are NOT in extX format we do an adb pull as well
  - We need to mount the partitions in read only mode if we are running from a custom recovery
- Netcat cross-compile for Android with NDK tool chain
  - http://codeseekah.com/2012/08/07/port-forwarding-an-android-local-port/

# BusyBox



- BusyBox is a single multicall binary that packages the functionality of most widely used standard Unix tools
  - http://www.busybox.net/
- Persistent or temporary install?
- Partitions may need to be remounted read-write
- Put the binary in /data/local for example
  - # chmod 755 busybox
  - # mount -o remount, rw /system
  - and install to some path with
  - # ./busybox – install -s /[path] where PATH=/sbin:/system/sbin:/system/bin:/system/xbin
- Or copy the file to tmpfs and create symbolic links for the few commands you actually need
- Cross-compiled Android versions of BusyBox exists

**BusyBox binary highlights**

**chown**, chgrp: change permission ownership.
**awk**, sed: languages to both process & transform text
**grep**: a text search utility
**du**: shows disk usage
**vi**: a shell based text editor
**pidof**: return the pid of a running process
**less**: text reader with back and forward nav
**tail**: trail the end of a file for activity
**gunzip**, gzip, tar, bzip2: archival compression software
**clear**: clear screen
**crontab**, crond: task scheduling
**diff**: compare files
**httpd**: a light webserver
**telnet**: basic TCP remote login
**xargs**: use the output of a command as the args for another
**su**: masquerade as another system user
**wget**: retrieves content from a web server
**which**: identifies the location of an executable
**nc**: netcat
**dd**: disk dumper

# Android application and forensic analysis

- Most of the techniques used in traditional forensic investigations are applicable in Android forensics analysis
  - If image is a YAFFS2 image the work can be very complicated
- Timline analysis is slow unless special software is used
  - log2timeline
- Also consider that different file systems metadata have different time resolution and on top of that can delay updates to the file system

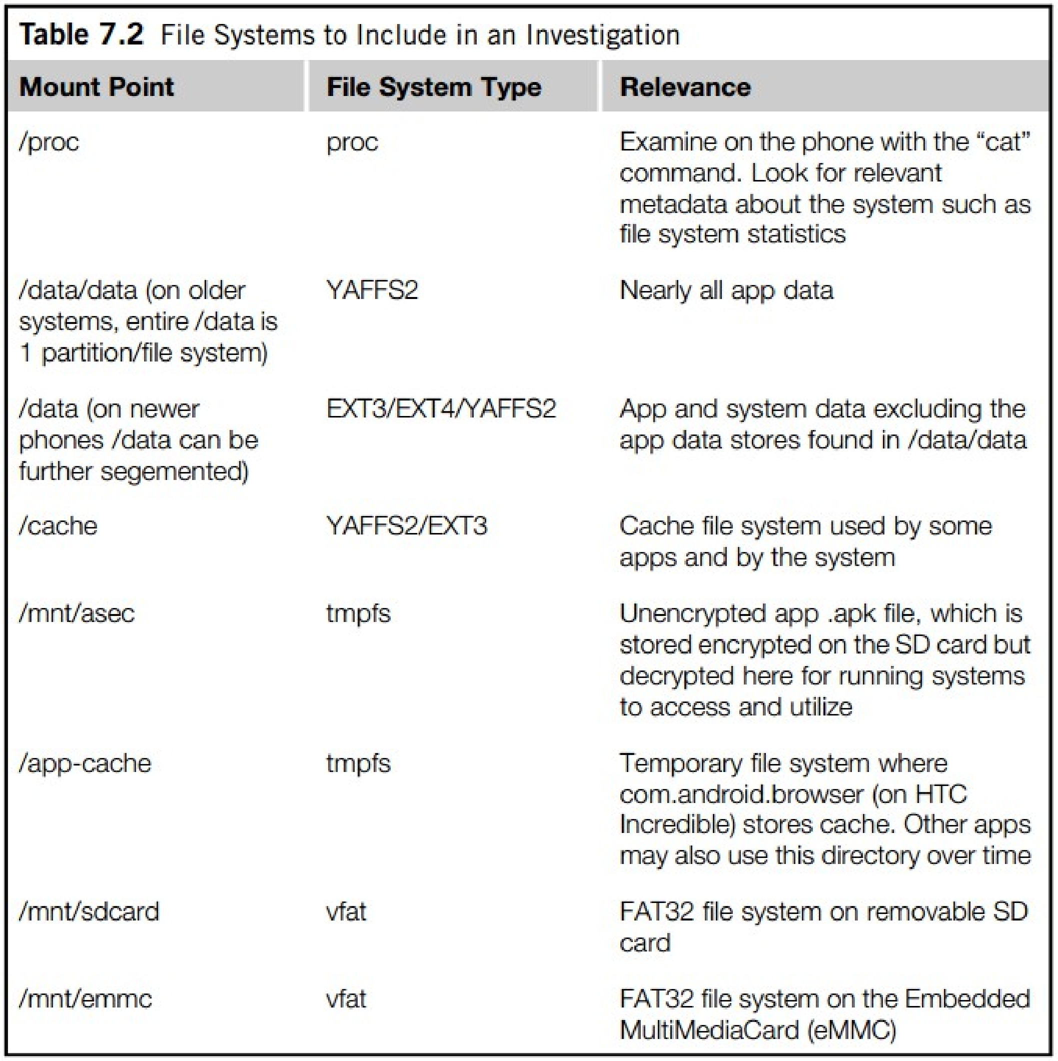
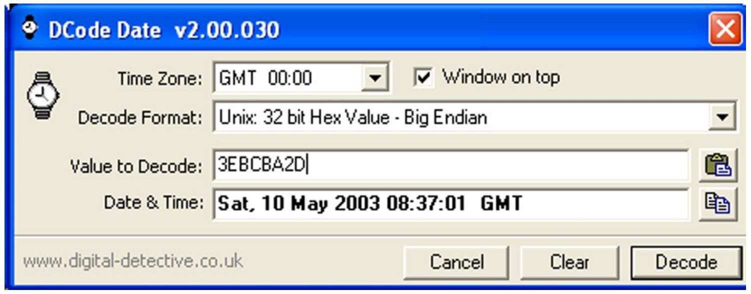


**Table 7.2** File Systems to Include in an Investigation

| Mount Point | File System Type | Relevance |
|---|---|---|
| /proc | proc | Examine on the phone with the "cat" command. Look for relevant metadata about the system such as file system statistics |
| /data/data (on older systems, entire /data is 1 partition/file system) | YAFFS2 | Nearly all app data |
| /data (on newer phones /data can be further segemented) | EXT3/EXT4/YAFFS2 | App and system data excluding the app data stores found in /data/data |
| /cache | YAFFS2/EXT3 | Cache file system used by some apps and by the system |
| /mnt/asec | tmpfs | Unencrypted app .apk file, which is stored encrypted on the SD card but decrypted here for running systems to access and utilize |
| /app-cache | tmpfs | Temporary file system where com.android.browser (on HTC Incredible) stores cache. Other apps may also use this directory over time |
| /mnt/sdcard | vfat | FAT32 file system on removable SD card |
| /mnt/emmc | vfat | FAT32 file system on the Embedded MultiMediaCard (eMMC) |

# Android application and forensic analysis

- File carving
  - Filesystem is fragmented so special carvers are needed as smart carver from Digital Assembly: http://digital-assembly.com/
- Strings
  - Remember to scan for Unicode strings (Android is Unicode)
  - Check SQLite databases for deleted records (special tools)
- SQLite databases
  - View in different SQLite db viewers or with sqlite tool
  - SQL As Understood By SQLite: http://www.sqlite.org/lang.html
- Hex editor
  - Deeper analysis
- Time conversions
  - Unix Epoch time – with script or ...
- Android directory structure is important to understand!


DCode Date v2.00.030

Time Zone: GMT 00:00
Window on top: ✓
Decode Format: Unix: 32 bit Hex Value - Big Endian
Value to Decode: 3EBCBA2D
Date & Time: Sat, 10 May 2003 08:37:01 GMT

www.digital-detective.co.uk

http://www.digital-detective.co.uk/

# Android directory structure

```
/
├── app-cache
│   └── com.android.browser
│       └── cache
│           └── webviewCache
├── cache
│   ├── lost+found
│   └── recovery
├── data
│   ├── anr
│   ├── app
│   ├── app-private
│   ├── backup
│   ├── btips
│   ├── dalvik-cache
│   ├── data
│   │   ├── com.facebook.katana
│   │   │   ├── cache
│   │   │   │   └── webviewCache
│   │   │   ├── databases
│   │   │   ├── files
│   │   │   ├── lib
│   │   │   └── shared_prefs
│   ├── dontpanic
│   ├── local
│   ├── lost+found
│   ├── misc
│   │   ├── bluetooth
│   │   ├── bluetoothd
│   │   ├── dhcp
│   │   ├── keystore
│   │   ├── lockscreen
│   │   ├── systemkeys
│   │   ├── vpn
│   │   └── wifi
│   ├── property
│   ├── system
│   │   ├── registered_services
│   │   ├── shared_prefs
│   │   ├── sync
│   │   ├── throttle
│   │   └── usagestats
│   └── tombstones
├── mnt
│   ├── asec
│   ├── emmc
│   │   ├── Android
│   │   │   └── data
│   │   │       └── com.android.providers.media
│   │   │           └── albumthumbs
│   │   ├── DCIM
│   │   │   └── 100MEDIA
│   │   ├── LOST.DIR
│   │   ├── MP3
│   │   │   ├── People Under the Stairs
│   ├── sdcard
│   │   ├── Android
│   │   │   └── data
│   │   │       ├── com.google.android.apps.maps
│   │   │       │   ├── cache
│   │   │       │   ├── debug
│   │   │       │   └── testdata
│   │   │       └── com.yelp.android
│   │   │           └── cache
│   │   │               └── images
│   │   ├── dcim
│   │   ├── download
│   │   ├── Downloads
│   │   ├── LOST.DIR
│   │   └── tmp
│   └── secure
│       ├── asec
│       └── staging
└── system
    ├── app
    ├── bin
    ├── customize
    │   ├── CID
    │   ├── MNS
    │   └── resource
    ├── etc
    │   ├── bluetooth
    │   ├── clockwidget
    │   ├── dhcpcd
    │   │   └── dhcpcd-hooks
    │   ├── firmware
    │   ├── iproute2
    │   ├── permissions
    │   ├── ppp
    │   ├── security
    │   ├── updatecmds
    │   ├── wifi
    │   └── wimax
    ├── fonts
    ├── framework
    ├── lib
    │   ├── bluez-plugin
    │   ├── egl
    │   ├── hw
    │   └── modules
    ├── lost+found
    ├── media
    │   └── audio
    │       ├── alarms
    │       ├── notifications
    │       ├── ringtones
    │       └── ui
    ├── tts
    │   └── lang_pico
    ├── usr
    │   ├── keychars
    │   ├── keylayout
    │   ├── share
    │   │   ├── bmd
    │   │   └── zoneinfo
    │   └── srec
    └── xbin
```

There are far more directories and files to explore but the this overview provides a good starting point

With 4.4 there is also a /system/priv-app for Googles privileged apps - Some system apps are more system than others!

# Android directory structure 1

**Line 1**: At the top is the root directory, which creates the structure and mount points for the other file systems explored previously.
**Line 2**: As previously discussed, the HTC Incredible created an “/app-cache” directory of type tmpfs. You can see the browser cache structure. Presumably,over time, other apps may leverage this directory.
**Lines 6-8**: Android devices from the start had a dedicated “/cache” directory that originally appeared to be unused. However, this is certainly not the case and the “/cache” partition should be imaged for full analysis. Files including Gmail attachment previews, Browser DRM, some downloads (Market and other), as well as Over The Air (OTA) updates from the wireless carriers can be found here.
**Line 9**: The root level “/data” directory has a number of important subdirectories covered next. Note that some phones (such as the HTC Incredible) have a dedicated partition for the “/data/data” subdirectory.
**Line 10**: The “/data/anr” directory contains stack traces (debugging) from the system and is generally not accessible to the shell user. However, some of the adb debug commands appear to read this data.
**Line 11**: The “/data/app” directory contains the .apk files from the Android Market.
**Line 12**: The “/data/app-private” directory stores protected apps from the Android Market.
**Line 13**: More recent versions of Android have a secure cloud backup API that developers can integrate into their apps. The “/data/backup” directory is used to queue and manage these backups. However, thus far meaningful data has not been recovered from directory.
**Line 14**: The “/data/btips” (Texas Instrument’s Bluetooth Protocol Stack) directory stores the log files if the associated app (com.ti.btips) crashes.
**Line 15**: The “/data/davlik-cache” directory contains the Davlik VM’s cached dex files used to run apps.
**Line 16**: The “/data/data directory” contains the application specific data, easily the most important area to focus on in an investigation.
**Lines 17-23**: One app was kept in the directory hierarchy for demonstration purposes. The directory is named according to the package name and often clearly identifies the developer (Facebook in this case).

# Android directory structure 2

**Line 24**: For HHGTTG fans (famous advice to intergalactic travelers from the classic novelThe Hitchhiker's Guide to the Galaxy:DON'T PANIC), there's a great directory named "/data/dontpanic," which is simply a place to store some error log files from the system. Again, a great benefit of an open system is the ability to examine code. Short of that, we would have simply had to guess the purpose or perform significant testing.
**Line 25**: The "/data/local" directory is important as it allows shell (the user account nonrooted phones run adbd as) read/write access. When an app is installed, it is first copied to "/data/local." Also, some forensic techniques rely on this directory to upload important files, typically binaries.
**Line 26**: The "/data/lost+found" directory shows up in several places in YAFFS2 file systems. Again, a quick search (try "grep -R lost+found *.c" from the YAFFS2 source directory we downloaded) will explain that any files or directories found that do not have a path to the root directory will be placed in this
Folder.
**Lines 27-35**: The "/data/misc" directory contains files related to Bluetooth, dhcp, vpn, Wi-Fi, and more. One important file to point out is "/data/misc/wifi/wpa_supplicant.conf" that contains a list of Wi-Fi.com networks to which thedevice got connected. If the wireless access point required a password, it is stored in plain text in the file (have fun pen testers).
**Line 36**: The "/data/property" directory contains various system properties such as time zone, country, and language.
**Line 37**: Beyond the subdirectories you can see /data/system contains several key files. First, the accounts.db contains a list of accounts that require authentication and provides the name, type, password (encrypted), and authentication tokens (among other data). There are also two very important files related to the pass code or PIN for the device. The files are gesture.key and password.key and
contain an encoded/encrypted hex value for the pass code.
**Line 43**: When a process crashes, a special tombstone file can be created. The file is ASCII and thus readable. More information can be found online such as one informative post on Crazydaks.com (Debugging in Android, n.d.).
**Line 44**: The "/mnt" directory is where the system mounts various file systems, including the SD card, the eMMC, and others.

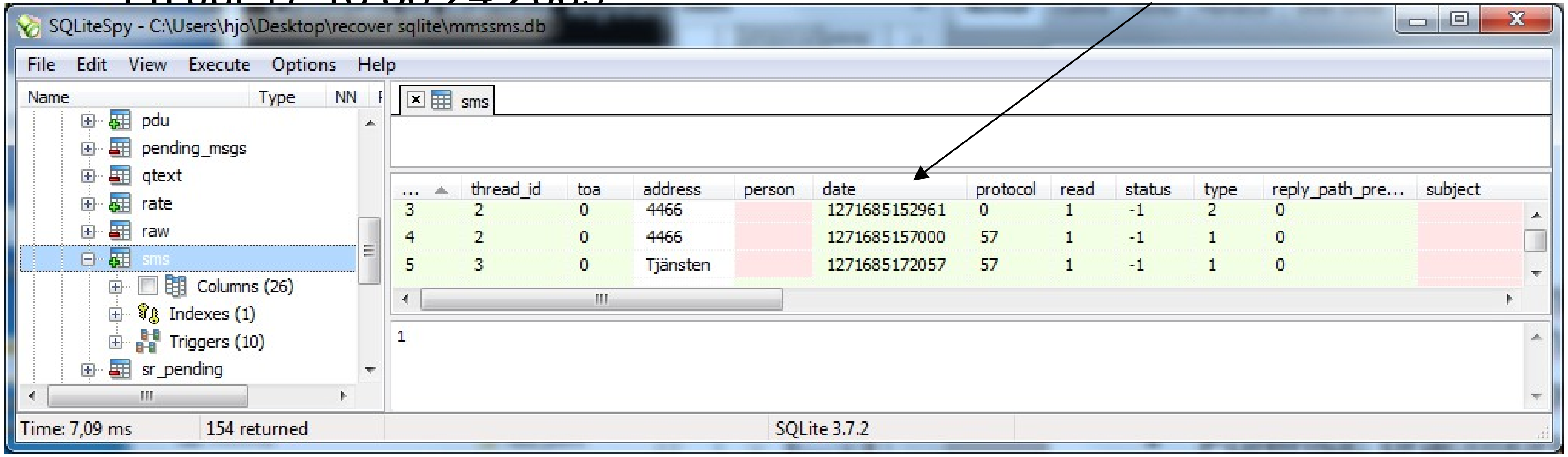
# Android directory structure 3

**Line 45**: The “/mnt/asec” directory contains the unencrypted apps that are stored on the SD card. When Android introduced the ability to store apps on the SD card, they encrypted the contents for security reasons. However, when the system is up and running and unencrypted access to the files is necessary, they are decrypted and mounted in “/mnt/asec.”
**Line 46**: The “/mnt/emmc” contains the FAT32 file system that resides on the NAND flash for some devices. Lines 47 through 55 are several examples of eMMC subdirectories.
**Line 51**: The “/mnt/emmc/DCIM directory,” album thumbnails are stored here.
**Line 52**: The “/mnt/emmc/DCIM/100MEDIA” directory contains any pictures or videos taken by the HTC Incredible.
**Line 53**: The “/mnt/emmc/LOST.DIR” directories are found on FAT32 partitions and may contain files or fragments that the file system lost track of (similar to YAFFS2 lost+found directory). This directory should be examined.
**Line 56**: If a physical SD card is present, it is mounted at “/mnt/sdcard.”
**Line 66**: As with the eMMC, the “/mnt/sdcard/dcim” directory would store pictures and videos from the device. On the HTC Incredible, they are stored in “/mnt/emmc/DCIM,” so they are not present on the physical SD card.
**Lines 67-68**: The “/mnt/sdcard/download” and “/mnt/sdcard/Downloads” directories contain files downloaded by the browser, e-mail clients, and others.
**Line 72**: As mentioned previously, the “/mnt/sdcard/secure/asec” directory is encrypted and is where apps that reside on the SD card (instead of the NAND flash) store data.
**Line 75**: The “/system/app” directory contains .apk app files for apps that are provided with the system. This includes apps bundled by Google/Android, the manufacturer (HTC in this case), and the wireless carrier (Verizon in this case). In the case of the HTC Incredible, the directory contains a significant 152 .apk files. It’s important to know this location in case app analysis is required for a case(which means you need access to the apk file).
**Lines 76 and 117**: The “/system/bin” and “/system/xbin” directories contain the Android binary files used on the system. Forensic analysts and security engineers (and most definitely Android researchers) can find many useful and undocumented commands by experimenting with files in these directories.
**Lines 77-80**: The “/system/customize” directories contain carrier-specific customizations for the phone, notably UI.
**Line 81**: The “/system/etc” directory is where Android stores the typical Linux/Unix configuration (/etc) directory. It contains numerous configuration files worthy of examination - too many to discuss in this book - but can vary from device to device.

# SQLite - .db files

- An increasing number of programs are employing SQLite to store data that can be of relevance in an investigation
  - Android, Apple products, Tizen (Maemo/MeeGo), Symbian, etc.
- Forensic practitioners who become familiar with SQLite and learn how to interpret these files will be in a better position to obtain the most usable information from available digital evidence
  - SQLite databases can be examined using a command line tool like sqlite3.exe (http://www.sqlite.org/) or with a GUI tools like SQLite Database Browser, SQLite Spy etc.
  - Dates are in Unix string format and can be converted using Perl etc.
  - $ perl -e "print scalar(gmtime(1247848584))"
  - Fri Jul 17 16:36:24 2009

Seconds since 1970



SQLiteSpy - C:\Users\hjo\Desktop\recover sqlite\mmssms.db

| ... | thread_id | toa | address | person | date | protocol | read | status | type | reply_path_pre... | subject |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 3 | 2 | 0 | 4466 | | 1271685152961 | 0 | 1 | -1 | 2 | 0 | |
| 4 | 2 | 0 | 4466 | | 1271685157000 | 57 | 1 | -1 | 1 | 0 | |
| 5 | 3 | 0 | Tjänsten | | 1271685172057 | 57 | 1 | -1 | 1 | 0 | |

1

Time: 7,09 ms | 154 returned | SQLite 3.7.2

# SQLite and deleted records

- SQLite database do not delete data until overwritten or the database is Vacuumed
  - Open .db in a hex editor and verify
- Epilog - a tool that can do undelete on records in a .db file
  - http://www.ccl-forensics.com
- Gary Kessler SQLite Parser
  - http://www.garykessler.net/software/
- Ff3hr can recover deleted history records from Firefox 3
  - Described in Murilo Tito Pereira's article "Forensic analysis of the Firefox 3 internet history and recovery of deleted SQLite records"
  - http://sourceforge.net/projects/ff3hr/
- Finding and Reverse Engineering Deleted SMS Messages
  - http://az4n6.blogspot.se/2013/02/finding-and-reverse-engineering-deleted_1865.html
- [server]\embedded_forensics\docs\common_documents\SQLite
  - Examensrapport av Sebastian Zankl - python script

# Location dump (exempel)

- Android versioner före 4.0 lagrar de 50 senaste mobilmasterna (cell-id) och 200 senaste Wi-Fi access punkterna i två binära filer
  - cache.cell och cache.wifi filerna finns i "/data/data/com.google.android.location/files/" mappen
- En parser i Python för filerna finns här
  - https://github.com/packetlss/android-locdump
- Båda har ett liknande format där key endera är cell-id (MCC, MNC, LAC, sektor) för en mobilmast eller en AP MAC-adress

**key accuracy conf. latitude longitude time**

- Output cache.cell

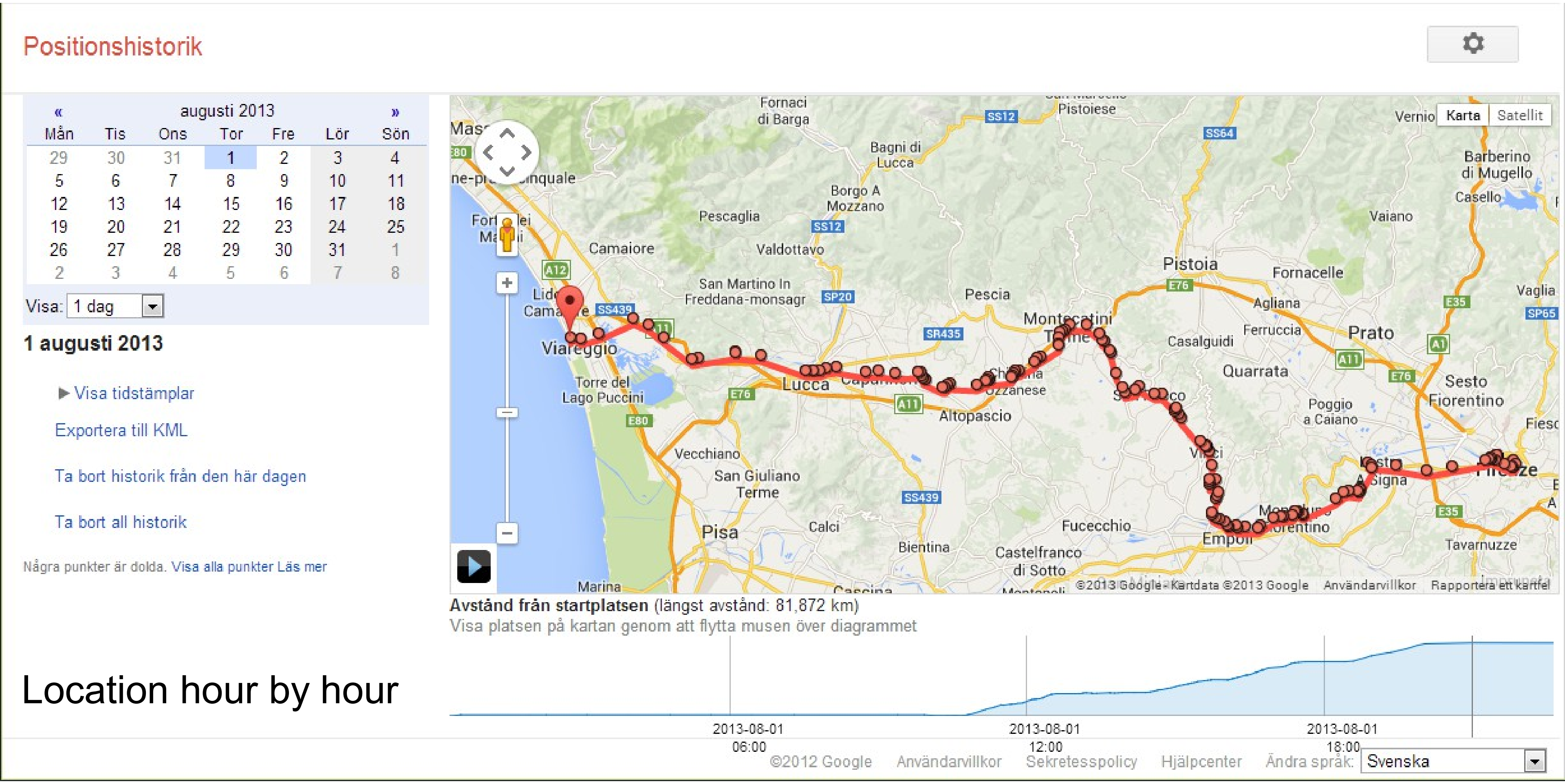
```
240:5:23:1510154    1677    75   60.492045   15.407134  01/31/11
  09:05:22 W. Europe Standard Time
```

- Output cache.wifi

```
00:27:0d:0a:34:02      71    92   60.487707   15.408977  02/08/11
  10:41:06 W. Europe Standard Time
```

# Locationhistory via Gmail account

- If activated the phone collects location continuously
- https://maps.google.se/locationhistory



Positionshistorik

| « | | | augusti 2013 | | | » |
|---|---|---|---|---|---|---|
| Mån | Tis | Ons | Tor | Fre | Lör | Sön |
| 29 | 30 | 31 | 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 | 23 | 24 | 25 |
| 26 | 27 | 28 | 29 | 30 | 31 | 1 |
| 2 | 3 | 4 | 5 | 6 | 7 | 8 |

Visa: 1 dag

**1 augusti 2013**

▶ Visa tidstämplar

Exportera till KML

Ta bort historik från den här dagen

Ta bort all historik

Några punkter är dolda. Visa alla punkter Läs mer

**Avstånd från startplatsen** (längst avstånd: 81,872 km)

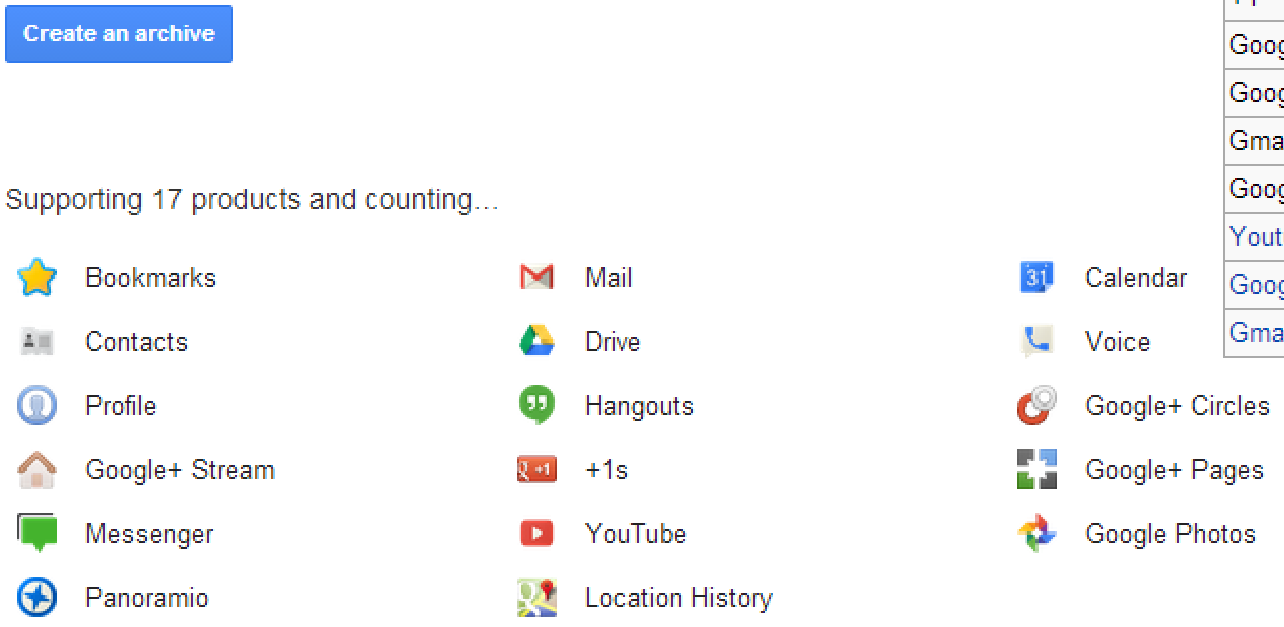
Visa platsen på kartan genom att flytta musen över diagrammet

©2012 Google Användarvillkor Sekretesspolicy Hjälpcenter Ändra språk: Svenska

Location hour by hour

# Google Data Liberation Front

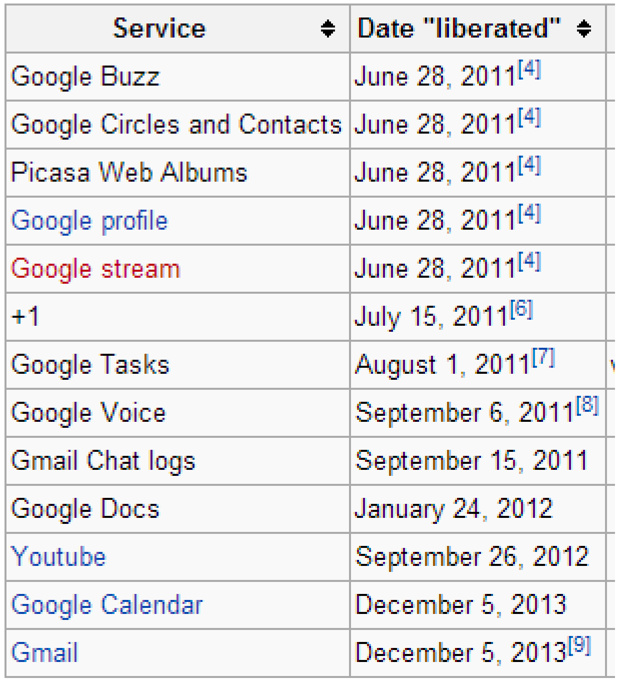
- Google Takeout
  - Since device may be encrypted
  - Export data from various supported services
  - https://www.google.com/settings/takeout



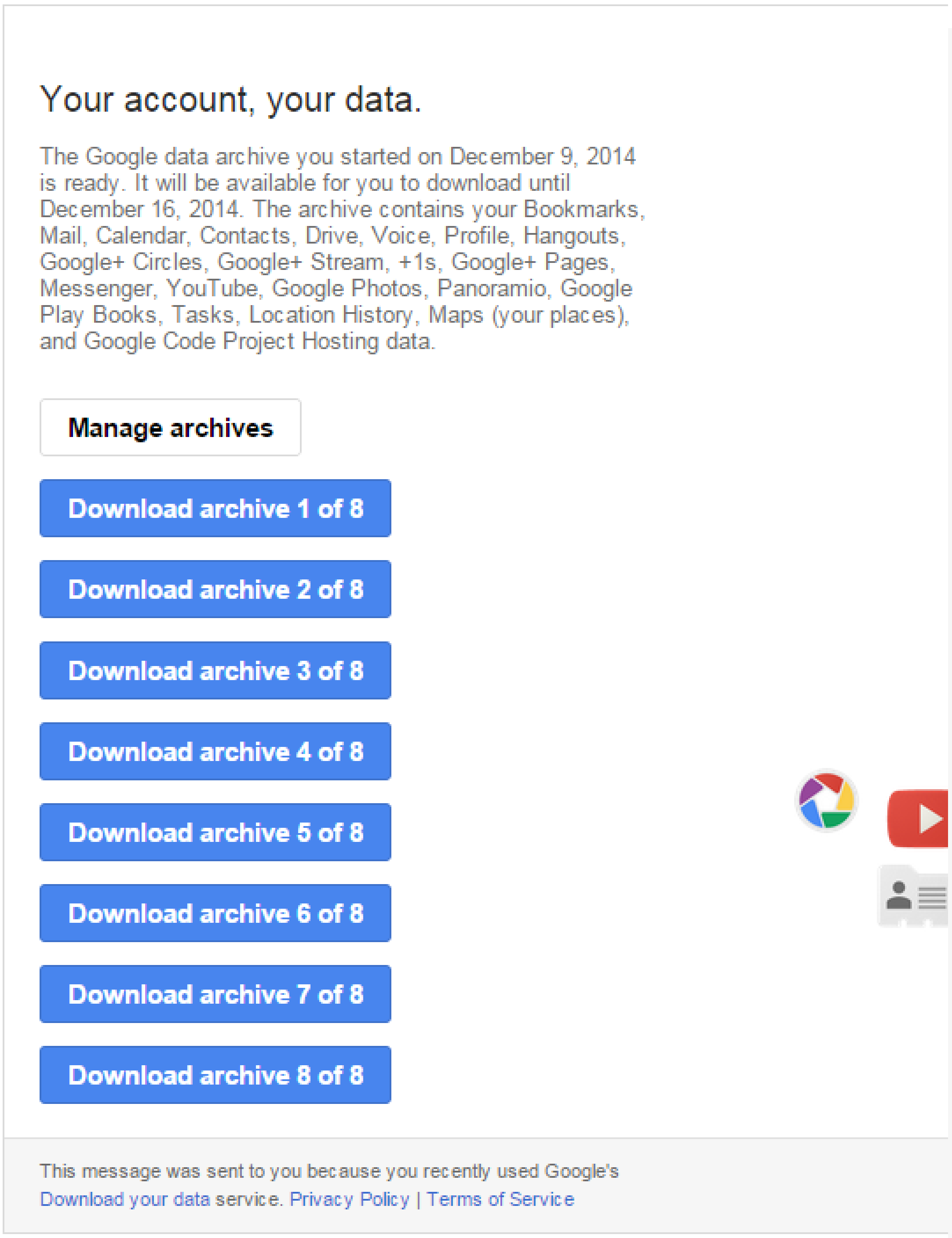
Create an archive

Supporting 17 products and counting...

| | | |
|---|---|---|
| Bookmarks | Mail | Calendar |
| Contacts | Drive | Voice |
| Profile | Hangouts | Google+ Circles |
| Google+ Stream | +1s | Google+ Pages |
| Messenger | YouTube | Google Photos |
| Panoramio | Location History | |



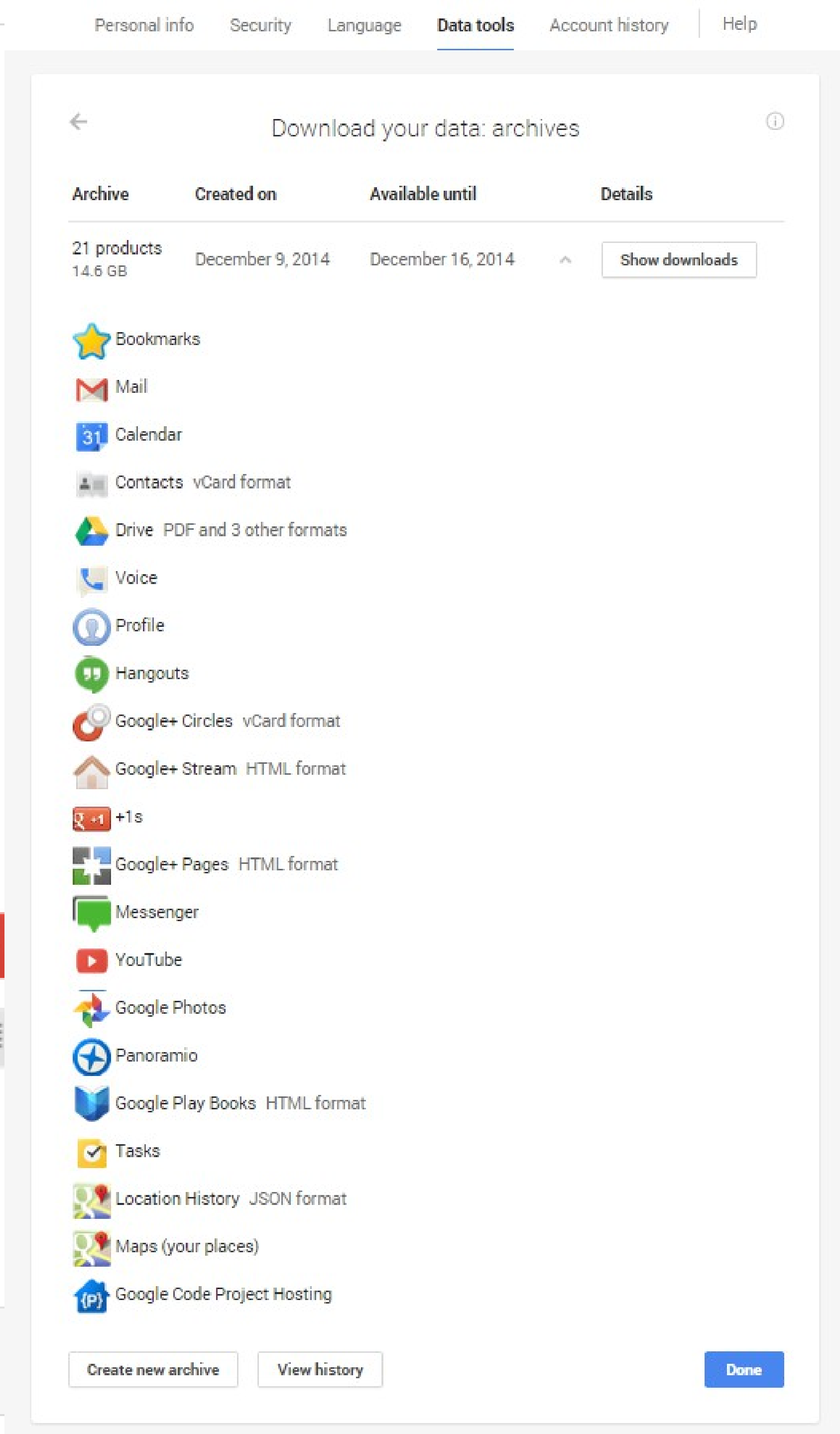
| Service | Date "liberated" |
|---|---|
| Google Buzz | June 28, 2011[4] |
| Google Circles and Contacts | June 28, 2011[4] |
| Picasa Web Albums | June 28, 2011[4] |
| Google profile | June 28, 2011[4] |
| Google stream | June 28, 2011[4] |
| +1 | July 15, 2011[6] |
| Google Tasks | August 1, 2011[7] |
| Google Voice | September 6, 2011[8] |
| Gmail Chat logs | September 15, 2011 |
| Google Docs | January 24, 2012 |
| Youtube | September 26, 2012 |
| Google Calendar | December 5, 2013 |
| Gmail | December 5, 2013[9] |



Your account, your data.

The Google data archive you started on December 9, 2014 is ready. It will be available for you to download until December 16, 2014. The archive contains your Bookmarks, Mail, Calendar, Contacts, Drive, Voice, Profile, Hangouts, Google+ Circles, Google+ Stream, +1s, Google+ Pages, Messenger, YouTube, Google Photos, Panoramio, Google Play Books, Tasks, Location History, Maps (your places), and Google Code Project Hosting data.

This message was sent to you because you recently used Google's Download your data service. Privacy Policy | Terms of Service

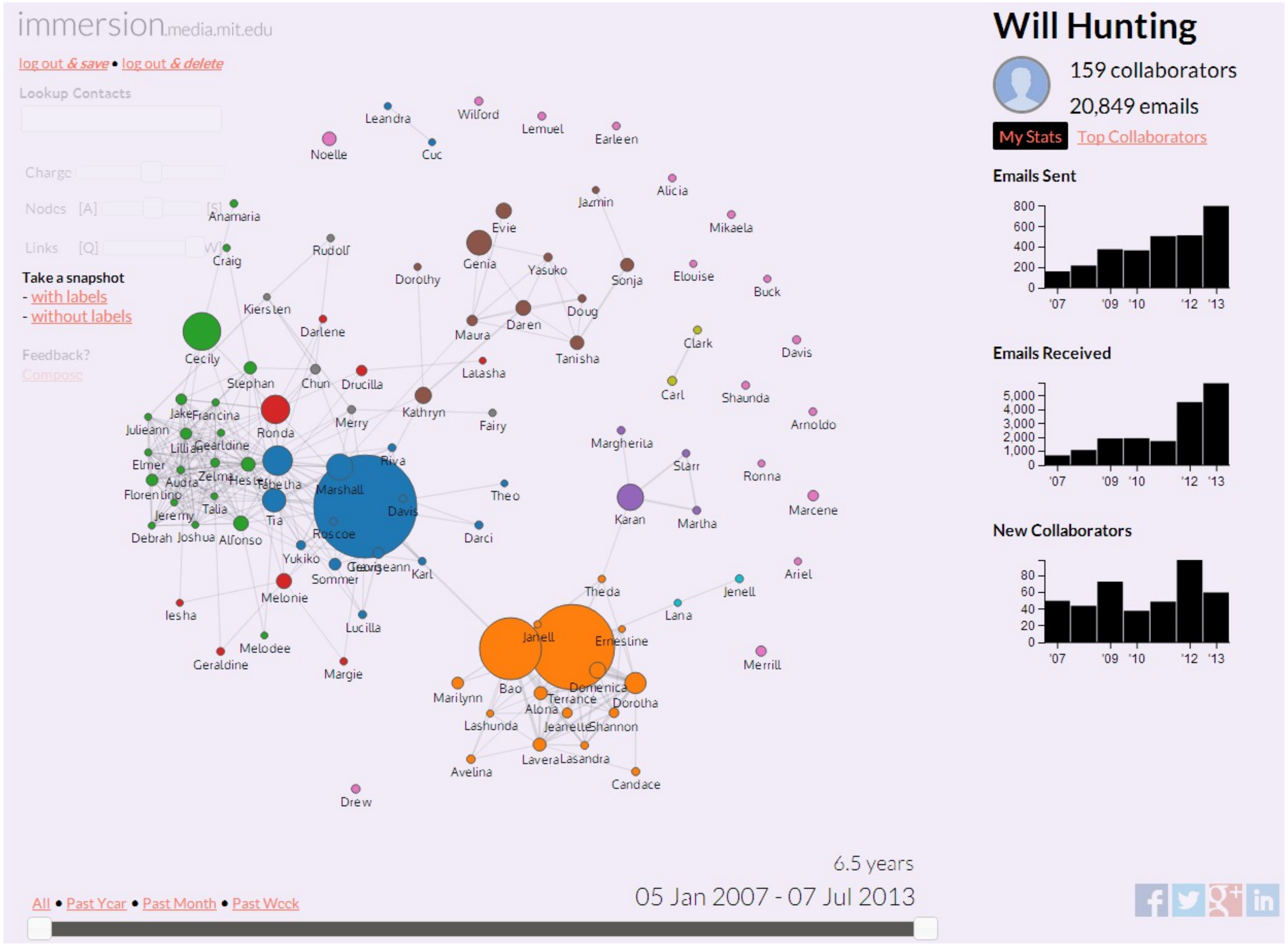


Download your data: archives

| Archive | Created on | Available until | Details |
|---|---|---|---|
| 21 products 14.6 GB | December 9, 2014 | December 16, 2014 | Show downloads |

Bookmarks
Mail
Calendar
Contacts vCard format
Drive PDF and 3 other formats
Voice
Profile
Hangouts
Google+ Circles vCard format
Google+ Stream HTML format
+1s
Google+ Pages HTML format
Messenger
YouTube
Google Photos
Panoramio
Google Play Books HTML format
Tasks
Location History JSON format
Maps (your places)
Google Code Project Hosting

# Gmail history metadata



## Deodex vs. Odex and protected apps

- ODEx stands for Optimized Dalvik Executable
  - In any stock Android ROM, you will find some .odex files in addition to .apk files, for example Phone.apk as well as Phone.odex, ie. the app is divided in two files (.apk and .odex)
  - The odex file will contain some parts of the app in compressed form that will be preloaded
  - The purpose of the .odex file is to save space and also to speed up the boot process
- What is Deodex?
  - It's the process to take all the packages out from .odex file and reassemble them all together in a classes.dex file
  - A deodexed app is open to customizations as theming etc.
- Inspect the protected and encrypted apps in /mnt/asec/<package name>/
  - Just copy the pkg.apk file out of the directory
  - Run it where you like (unencrypted next time?)

## From Android >= 2.3

- The default filesystem for Gingerbread is ext4, it is also used in Googles datacenter. Thats what I would call scaleable!!
- Beginning with FTK Imager 3.0 support is included for VxFS, exFAT, and Ext4 file systems
- Beginning with FTK 3.2 FTK enhanced its file system support to include DMG / Ext4 / ExFAT / VxFS /MSVHD / AFF (Advanced Forensic Format) / Blackberry IPD
- The problem with YAFFS is that it is single-threaded and would likely have been a bottleneck on dual-core systems
  - Concurrency is important on Android devices that use multi-core processors
- Applications that use the platform’s high-level storage abstractions will generally not have to worry about the transition
- Developers who are accessing the filesystem directly will have to be mindful about Ext4's buffering behavior and make sure that the data is actually reaching persistent storage in a timely manner so that it won’t be lost in the event of a system failure.

## Ext4 and Btrfs

T'so says that there isn't much need for concern. Google and the handset makers will catch platform-level filesystem reliability issues, ensuring that the high-level storage APIs are safe. He also believes that the significant amount of product QA conducted by the vendors will reduce the risk of random crashes. Short of users yanking out the battery, he says, it's unlikely that Android devices will routinely run into the kind of system failure that causes data loss for applications that don't properly use fsync.

T'so also addressed the question of why Google would pass on Oracle's Btrfs, which is expected to eventually displace Ext4 in the future. Btrfs simply isn't mature enough yet for use in production. Canonical considered using Btrfs as the default filesystem in Ubuntu 10.10, but postponed adoption after similarly deciding that it needed more time in the oven. Nokia and Intel have adopted Btrfs for MeeGo, though it's unclear if they will stick with that decision when MeeGo ships on actual consumer devices. It's clear that Ext4 still has an important role to play while the issues in Btrfs are being ironed out.

# Android dev. blog

If you just use SharedPreferences or SQLite, you can relax, because we've made sure they Do The Right Thing about buffering.

To start with, for raw data consider using one of the synchronous modes of java.io.RandomAccessFile, which take care of calling fsync() for you in the appropriate way. If you can't, you'll want Java code that looks something like this.

```
public static boolean sync(FileOutputStream stream) {
    try {
        if (stream != null) {
            stream.getFD().sync();
        }
        return true;
    } catch (IOException e) {
    }
    return false;
```

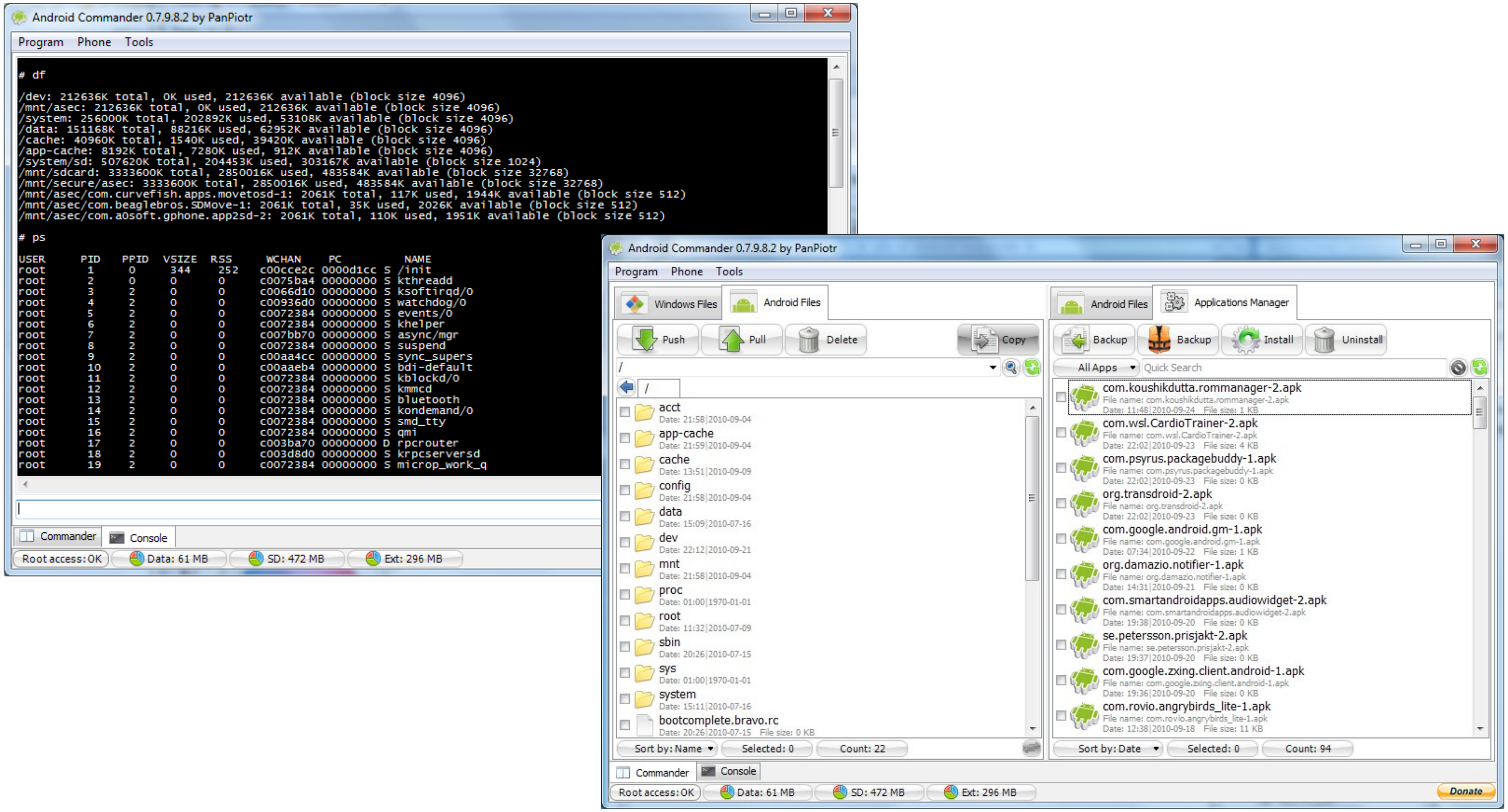
In some applications, you might even want to check the return status of the close() call.

Now of course, there are two sides to this story. When you call fsync() and force the data onto storage, that can be slow; worse, unpredictably slow. So be sure to call it when you need it, but be careful not to call it carelessly.

http://android-developers.blogspot.com/2010/12/saving-data-safely.html

# Other tools riding on ADB

- Android Commander, QtADB, Droid Explorer etc.



```
# df

/dev: 212636K total, 0K used, 212636K available (block size 4096)
/mnt/asec: 212636K total, 0K used, 212636K available (block size 4096)
/system: 256000K total, 202892K used, 53108K available (block size 4096)
/data: 151168K total, 88216K used, 62952K available (block size 4096)
/cache: 40960K total, 1540K used, 39420K available (block size 4096)
/app-cache: 8192K total, 7280K used, 912K available (block size 4096)
/system/sd: 507620K total, 204453K used, 303167K available (block size 1024)
/mnt/sdcard: 3333600K total, 2850016K used, 483584K available (block size 32768)
/mnt/secure/asec: 3333600K total, 2850016K used, 483584K available (block size 32768)
/mnt/asec/com.curvefish.apps.movetosd-1: 2061K total, 117K used, 1944K available (block size 512)
/mnt/asec/com.beaglebros.SDMove-1: 2061K total, 35K used, 2026K available (block size 512)
/mnt/asec/com.a0soft.gphone.app2sd-2: 2061K total, 110K used, 1951K available (block size 512)

# ps

USER     PID   PPID  VSIZE  RSS     WCHAN    PC         NAME
root      1     0     344    252   c00cce2c 0000d1cc S /init
root      2     0     0      0     c0075ba4 00000000 S kthreadd
root      3     2     0      0     c0066d10 00000000 S ksoftirqd/0
root      4     2     0      0     c00936d0 00000000 S watchdog/0
root      5     2     0      0     c0072384 00000000 S events/0
root      6     2     0      0     c0072384 00000000 S khelper
root      7     2     0      0     c007bb70 00000000 S async/mgr
root      8     2     0      0     c0072384 00000000 S suspend
root      9     2     0      0     c00aa4cc 00000000 S sync_supers
root      10    2     0      0     c00aaeb4 00000000 S bdi-default
root      11    2     0      0     c0072384 00000000 S kblockd/0
root      12    2     0      0     c0072384 00000000 S kmmcd
root      13    2     0      0     c0072384 00000000 S bluetooth
root      14    2     0      0     c0072384 00000000 S kondemand/0
root      15    2     0      0     c0072384 00000000 S smd_tty
root      16    2     0      0     c0072384 00000000 S qmi
root      17    2     0      0     c003ba70 00000000 D rpcrouter
root      18    2     0      0     c003d8d0 00000000 S krpcserversd
root      19    2     0      0     c0072384 00000000 S microp_work_q
```

acct
Date: 21:58|2010-09-04
app-cache
Date: 21:59|2010-09-04
cache
Date: 13:51|2010-09-09
config
Date: 21:58|2010-09-04
data
Date: 15:09|2010-07-16
dev
Date: 22:12|2010-09-21
mnt
Date: 21:58|2010-09-04
proc
Date: 01:00|1970-01-01
root
Date: 11:32|2010-07-09
sbin
Date: 20:26|2010-07-15
sys
Date: 01:00|1970-01-01
system
Date: 15:11|2010-07-16
bootcomplete.bravo.rc
Date: 20:26|2010-07-15 File size: 0 KB

Sort by: Name | Selected: 0 | Count: 22

com.koushikdutta.rommanager-2.apk
File name: com.koushikdutta.rommanager-2.apk
Date: 11:48|2010-09-24 File size: 1 KB
com.wsl.CardioTrainer-2.apk
File name: com.wsl.CardioTrainer-2.apk
Date: 22:02|2010-09-23 File size: 4 KB
com.psyrus.packagebuddy-1.apk
File name: com.psyrus.packagebuddy-1.apk
Date: 22:02|2010-09-23 File size: 0 KB
org.transdroid-2.apk
File name: org.transdroid-2.apk
Date: 22:02|2010-09-23 File size: 0 KB
com.google.android.gm-1.apk
File name: com.google.android.gm-1.apk
Date: 07:34|2010-09-22 File size: 1 KB
org.damazio.notifier-1.apk
File name: org.damazio.notifier-1.apk
Date: 14:31|2010-09-21 File size: 0 KB
com.smartandroidapps.audiowidget-2.apk
File name: com.smartandroidapps.audiowidget-2.apk
Date: 19:38|2010-09-20 File size: 0 KB
se.petersson.prisjakt-2.apk
File name: se.petersson.prisjakt-2.apk
Date: 19:37|2010-09-20 File size: 0 KB
com.google.zxing.client.android-1.apk
File name: com.google.zxing.client.android-1.apk
Date: 19:36|2010-09-20 File size: 0 KB
com.rovio.angrybirds_lite-1.apk
File name: com.rovio.angrybirds_lite-1.apk
Date: 12:38|2010-09-18 File size: 11 KB

Sort by: Date | Selected: 0 | Count: 94

# Partitions SGT 1

p1wifi_20110128_r10_00.pit (4 KB) (see PIT-info dumped below)

GT-P1010-CSC-SERKB3/
    cache.rfs (10.9 MB) (see content listing below)
    movinand.mst (51MB) (can be extracted with MoviTool, based on Volker1's method)

P1010XWKB5-REV03-ALL-low-CL913814/
    boot.bin (256 KB)
    cache.rfs (672 KB)
    normalboot.img (4.3 MB)
    param.lfs (612 KB)
    recovery.img (4.3 MB)
    Sbl.bin (1.2 MB)
    system.rfs (331 MB)
    userdata.rfs (1.2 MB)

```
----------------------------------------------------------------------

Contents of PIT file: p1wifi_20110128_r10_00.pit
----------------------------------------------------------------------
file magic = 0x12349876  (expected value)
Unknown data: 0 0 0 0 0
Number of partitions = 13  (usual value)

Partition #1
  Usual content: boot.bin, the primary boot loader (low-level hardware initialization)
  partition entry type: 0 0  (normal partition)
  ID = 0;  flags = 0;          unknown: 0
  size = 1 blocks of 256 * 512 bytes  = 131072 B = 128 kB = 0 MB
  unknown string: [........]
  partition name = [IBL+PBL........................]
  file name = [boot.bin........................................................]

Partition #2
  Usual content: partition information table (PIT)
  partition entry type: 0 0  (normal partition)
  ID = 0x1;          flags = 0;          unknown: 0
  size = 1 blocks of 256 * 512 bytes  = 131072 B = 128 kB = 0 MB
  unknown string: [........]
  partition name = [PIT............................]
  file name = [p1wifi.pit.....................................................]

Partition #3
  Usual content: efs.rfs
  partition entry type: 0 0  (normal partition)
  ID = 0x14;         flags = 0x2  (rfs file system);      unknown: 0
  size = 40 blocks of 256 * 512 bytes  = 5242880 B = 5120 kB = 5 MB
  unknown string: [........]
  partition name = [EFS............................]
  file name = [efs.rfs........................................................]

Partition #4
  Usual content: Sbl.bin, the secondary boot loader (loads linux kernel)
  partition entry type: 0 0  (normal partition)
  ID = 0x3;          flags = 0;          unknown: 0
  size = 5 blocks of 256 * 512 bytes  = 655360 B = 640 kB = 0 MB
  unknown string: [........]
  partition name = [SBL............................]
  file name = [sbl.bin........................................................]

Partition #5
  Usual content: backup of secondary boot loader
  partition entry type: 0 0  (normal partition)
  ID = 0x4;          flags = 0;          unknown: 0
  size = 5 blocks of 256 * 512 bytes  = 655360 B = 640 kB = 0 MB
  unknown string: [........]
  partition name = [SBL2...........................]
  file name = [sbl.bin........................................................]
```

# Partitions SGT 2

```
Partition #6
  Usual content: param.lfs /mnt/.lfs j4fs
  partition entry type: 0 0  (normal partition)
  ID = 0x15;         flags = 0x2  (rfs file system);      unknown: 0
  size = 20 blocks of 256 * 512 bytes  = 2621440 B = 2560 kB = 2 MB
  unknown string: [........]
  partition name = [PARAM..........................]
  file name = [param.lfs.....................................................]

Partition #7
  Usual content: zImage, the linux kernel
  partition entry type: 0 0  (normal partition)
  ID = 0x5;          flags = 0;          unknown: 0
  size = 30 blocks of 256 * 512 bytes  = 3932160 B = 3840 kB = 3 MB
  unknown string: [........]
  partition name = [NORMALBOOT.....................]
  file name = [normalboot.img.................................................]

Partition #8
  Usual content: recovery.bin, the backup copy of zImage/initramfs
  partition entry type: 0 0  (normal partition)
  ID = 0x8;          flags = 0;          unknown: 0
  size = 30 blocks of 256 * 512 bytes  = 3932160 B = 3840 kB = 3 MB
  unknown string: [........]
  partition name = [RECOVERY.......................]
  file name = [recovery.img...................................................]

Partition #9
  Usual content: factoryfs.rfs
  partition entry type: 0 0  (normal partition)
  ID = 0x16;         flags = 0x2  (rfs file system);      unknown: 0
  size = 1430 blocks of 256 * 512 bytes  = 187432960 B = 183040 kB = 178 MB
  unknown string: [........]
  partition name = [SYSTEM.........................]
  file name = [system.rfs....................................................]

Partition #10
  Usual content: dbdata.rfs
  partition entry type: 0 0  (normal partition)
  ID = 0x17;         flags = 0x2  (rfs file system);      unknown: 0
  size = 302 blocks of 256 * 512 bytes  = 39583744 B = 38656 kB = 37 MB
  unknown string: [........]
  partition name = [USERDATA.......................]
  file name = [userdata.rfs..................................................]

Partition #11
  Usual content: cache.rfs
  partition entry type: 0 0  (normal partition)
  ID = 0x18;         flags = 0x2  (rfs file system);      unknown: 0
  size = 140 blocks of 256 * 512 bytes  = 18350080 B = 17920 kB = 17 MB
  unknown string: [........]
  partition name = [CACHE..........................]
  file name = [cache.rfs.....................................................]

Partition #12
  Usual content: modem.bin
  partition entry type: 0 2  (unknown value)
  ID = 0x3;          flags = 0x1;       unknown: 0
  size = 0 blocks of 0 * 512 bytes  = 0 B = 0 kB = 0 MB
  unknown string: [........]
  partition name = [HIDDEN.D.......................]
  file name = [hidden.rfs.t..................................................]

Partition #13
  Usual content: Unknown
  partition entry type: 0 2  (unknown value)
  ID = 0;  flags = 0x1;       unknown: 0
  size = 0 blocks of 0 * 512 bytes  = 0 B = 0 kB = 0 MB
  unknown string: [........]
  partition name = [MOVINAND.......................]
  file name = [movinand.mst..................................................]
```

# Resources

- TWRP custom recovery for the Android emulator
  - http://teamw.in/project/twrp2/169
- viaForensics
  - https://viaforensics.com/
- Android Forensics
  - http://simson.net/ref/2011/2011-07-12%20Android%20Forensics.pdf
- opensecuritytraining.info
  - http://opensecuritytraining.info/
- The Android boot process from power on
  - http://www.androidenea.com/2009/06/android-boot-process-from-power-on.html
- Android Forensic Capability andEvaluation of Extraction Tools
  - http://www.academia.edu/1632597/Android_Forensic_Capability_and_Evaluation_of_Extraction_Tools