Android Forensics

Background, techniques and analysis tools

VIAFORENSICS

innovative digital forensics and security

About viaForensics

Forensics, Mobile Security, and Mobile Application Auditing

- Law enforcement
- Government
- Corporations
- Consumers

Based in Oak Park, IL

Training Outline

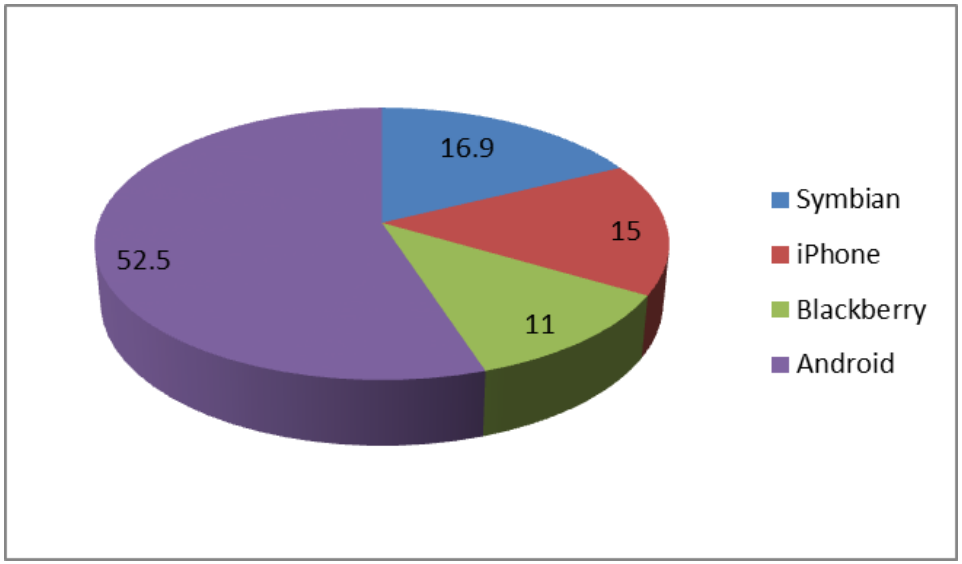
1. **Android Overview**
2. NAND Memory and Android File Systems
3. Forensic Techniques
   a. viaExtract acquisition
   b. Passcode demo
4. Conclusion

Android Devices

- 130 million devices 450,000 activations daily with 10m Gmail users
- 10+ phones run with Android installed after market, e.g. HTC Hero originally Windows but Android-ported
- Other devices
  - Garmin, Sony Ericsson (DVR), Acer, Huawei, Sharp (large, networked copiers), Sony (TV), medical devices, major car manufacturers
  - Home appliances: Android microwave, washer/dryer

Android Growth

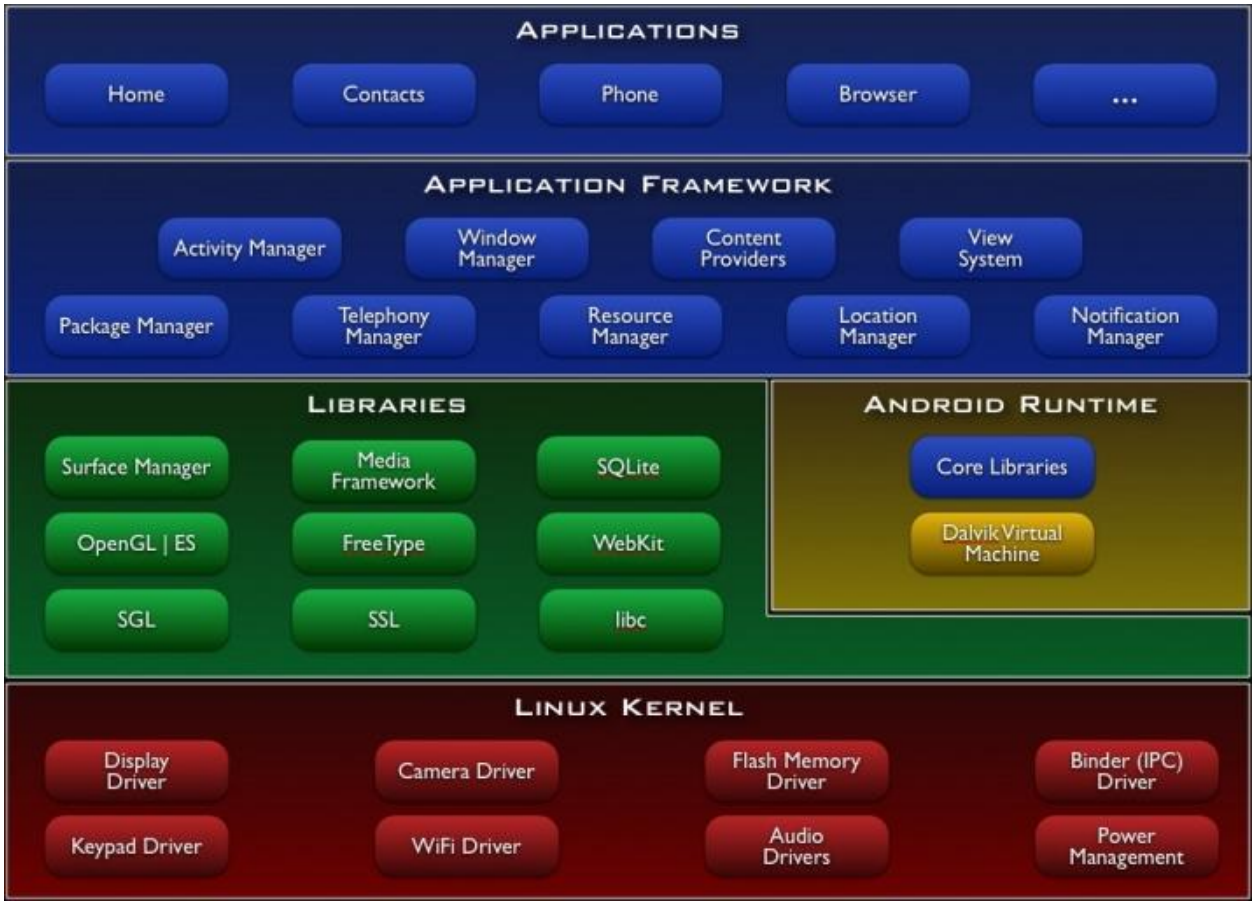
- Gartner reports that Android is now the largest mobile platform.
- Blackberry market share has fallen significantly since many corporations are now allowing Android/iPhones on the network



Android Technical Overview

- Based on Linux 2.6 kernel
- Porting to many processors, including Intel, ARM, MIPS, etc.
- Dalvik virtual machine
- SQLite for structured data storage
- Bionic C library (BSD-derived implementation)

Android Architecture



Android Application Architecture

- Beyond the base applications, all applications are installed by users
- All applications (.apk) must be signed via a certificate to be distributed in the Android Market
- Applications by same developer can share UID/data (e.g. trial and pro version)
- Content provider = interface for sharing data

Android Updates

- Responsibility of each carrier
- Can be applied OTA or manually
  - OTA is non-destructive
  - WARNING: Manual updates are sometimes destructive
- Anyone can branch Android code.
  - Carriers/manufacturers do
  - Can contribute back if registered, Google is gatekeeper
  - Very active enthusiast community which provides ports, updates, root and other updates

Android Update “Issues”

- Once user buys a phone and contract, carrier has them locked into contract with little economic incentive to upgrade OS.

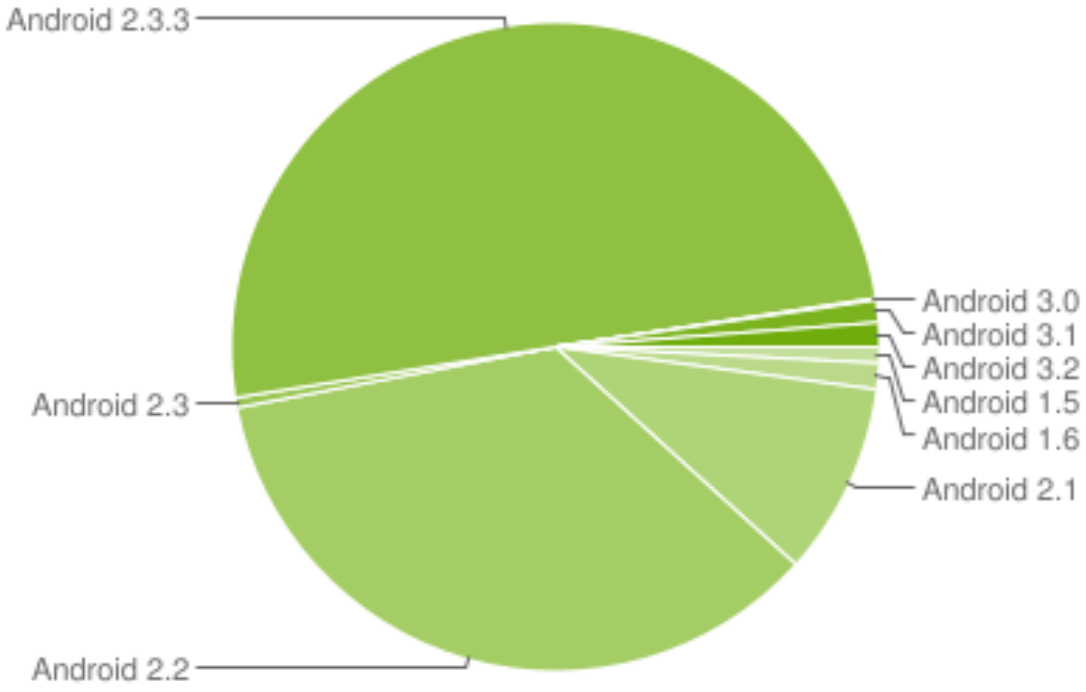


Image from developer.android.com current December, 2011

Data Storage on Android Devices

- Data can be stored on the device or the SD Card or EMMC (emulated SD)

- Developers have 4 mechanisms for data storage
  - **Preferences** allow a developer to store key-value pairs in a lightweight format
  - **Files** allow for more complicated data storage which are saved directly to the file system
  - **SQLite databases** are used for structure data store. Android uses SQLite3 databases and recovery of and from these files is a very key part of the forensic analysis.
  - **Cloud**: the final data storage is via the network, or to be trendy, the cloud. While certain investigations may require network analysis, most rely on data stored on the device.

Android Security Overview

- Each application is assigned a unique UID/GID and has its own VM, process and data storage
- Applications can choose what information they will share
  - Files
  - Databases
  - Preferences
- Shared data exposed in Content Providers
  - content://sms

Android Forensics Considerations

- Unlike traditional hard drives, cannot pull the plug, attach to write blocker and image
- Data from phones increasingly important
  - They go nearly everywhere with the owner
  - They are always online
  - They are powerful, fully featured and rarely delete data
- Any interaction with phone changes it. JTAG and other chip-off destructive, difficult and increasingly impossible
- Must apply most appropriate forensic technique, document any changes and provide reasons

Training Outline

1. Android Overview
2. **NAND Memory and Android File Systems**
3. Forensic Techniques
   a. viaExtract acquisition
   b. Passcode demo
4. Conclusion

Android MTD

- Android devices use raw flash device, need Flash Translation Layer (FTL)
- Memory Technology Device (MTD) subsystem for memory devices (esp. Flash), provides FTL
- Allows OS to interact with NAND as standard block device
- Using MTD approach, Google did not lock manufacturers into particular NAND providers

NAND/Flash memory characteristics

- NAND has high density and cost effective but limited by power consumption and endurance
- Ships with bad blocks
- Access is via a page (i.e. not random like NOR)
- Has limited write/erase lifespan: 10k – 100k erasures
- Page must be erased before writing, write all 1's
- Page must be written sequentially within a block
- Must only write a page once, cannot be updated without erase

NAND erase and write

- When flash memory is erased, the entire block is written with 0xFF (all 1's) and this is the only mechanism by which a 0 can be changed to a 1.

- As result, when flash is written to (not erased), it will only change the 1 bit to a 0.

- An example provided by the YAFFS2 developers:
  10110011 (byte contains)
  11011010 (write this byte)
  10010010 (result)

  (this is the "logical and" of the two values)

Android YAFFS2

- Yet Another Flash File System 2
- Open source
- Have to compile tools/kernel module yourself (some optional support in newer kernels)
- Provides
  - Log-structured file system (think versioning)
  - Wear leveling
  - Much faster and YAFFS and JFFS, uses less RAM
  - Supports many flash geometries
  - Built in error correction (important to use nandread/nandwrite tools!)

Log-structured File System (Wikipedia)

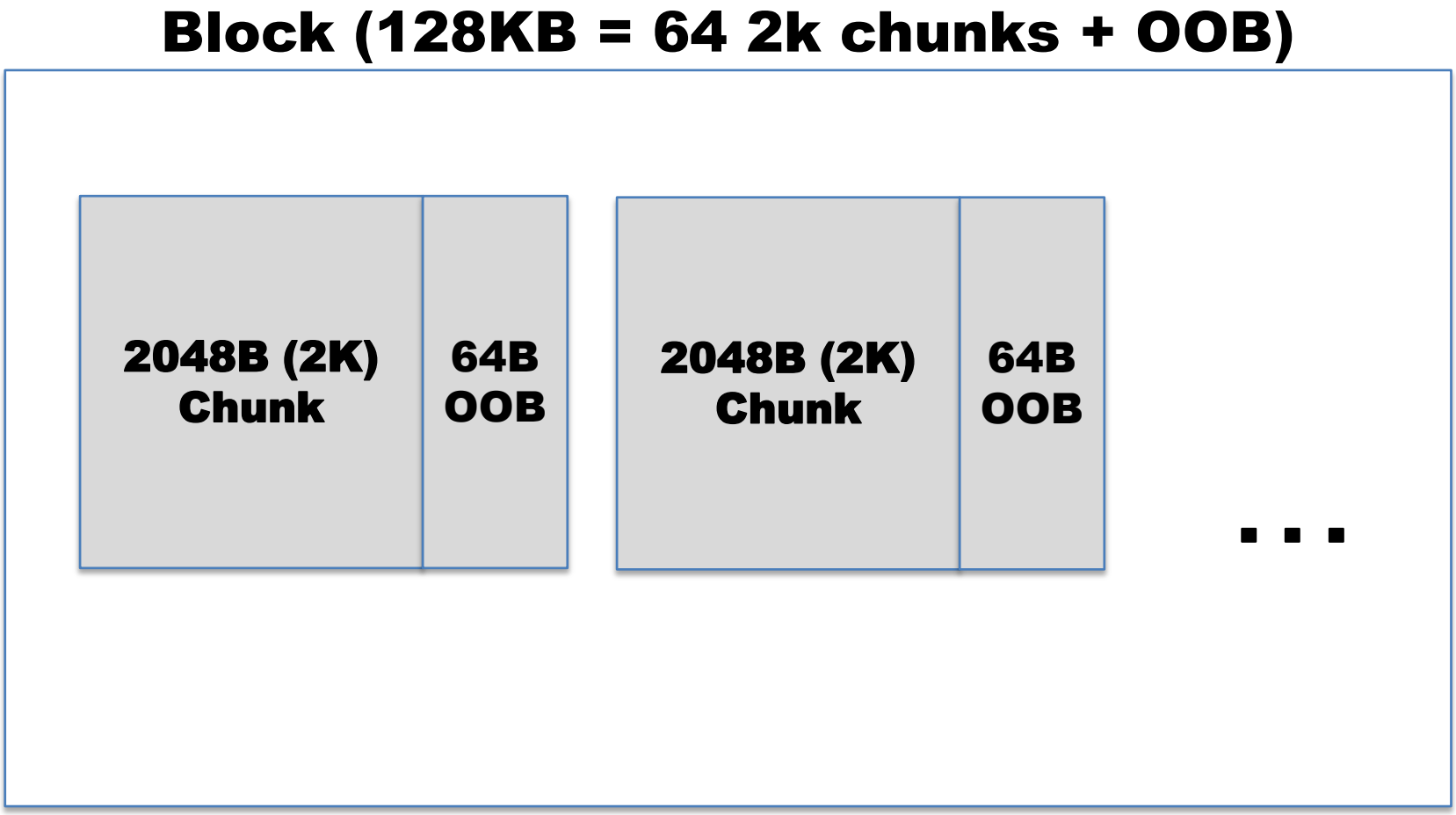
- All updates to data and metadata are written sequentially to a continuous stream, called a log

- Side effects
  - Writes create multiple, chronologically-advancing versions of both file data and meta-data. This is very similar to a versioning/snapshotting file system.
  - Recovery from crashes is less complicated. Upon its next mount, the file system does not need to walk through all its data structures to fix any inconsistencies, but can reconstruct its state from the last consistent point in the log.
  - Free space must be constantly reclaimed from the tail of the log to prevent the file system from becoming full (garbage collection)

YAFFS2 – Block/Chunk/OOB

- Memory is addressed in blocks (thanks to MTD)
- Each block has 64 pages/chunks, 2048 bytes each (2K), total of 128K (often see 0x20000, erase block size)
- Each chunk has a 64 byte Out-Of-Band area (OOB)
  - YAFFS2 tags and other meta data
  - Bad block bits
  - ECC for tags and data
- Each block is assigned a sequence number when allocated, starting at 1 and incrementing

YAFFS2 – Block/Chunk/OOB diagram



YAFFS2 Data Structures

- Data stored in YAFFS2 are referred to as Objects
  - Files
  - Directories
  - Symbolic and hard links
- Chunk stores either an Object or an yaffs_ObjectHeader
  - Metadata about the Object
    - 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
  - All objects are identified by a unique objectId (i.e. inodes)

YAFFS2 OOB

- In the 64 byte OOB/spare area, YAFFS2 stores critical information about the chunk but also shares the area with the Memory Technology Devices (MTD) subsystem. The critical YAFFS2 tags are:
  - 1 byte: Block state (0xFF if block is good, any other value for a bad block)
  - 4 bytes: 32-bit chunk ID (0 indicates 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)

YAFFS2 – New Object

- yaffs_ObjectHeader is written, no data yet
- Data is entered into file. Upon save, data is saved to NAND.
- After data is saved to NAND, new yaffs_ObjectHeader written to NAND so it references new data blocks, size, etc.

YAFFS2 – Object Modified

- New yaffs_ObjectHeader is written
- Remember, NAND memory can only be written once before erasing.
- The old data and headers still exists but are ignored in the file structure by examining the values of the sequence number.

YAFFS2 – object deleted

- Moved to a special, hidden “unlinked” or deleted directory.
- The file remains in this directory until all of the chunks in the file are erased.
  - File system tracks the number of chunks in the system for the file
  - When it reaches 0, the remnants of the file no longer exist.
- At that point, it will no longer track the object in the “unlinked” directory.
  - Unless garbage collected, will remain on file system

File Systems

- Several flavors of file systems
  - Temp file systems
  - YAFFS2
  - EXT
  - RFS

Temp File Systems

- Many varieties of utility file systems supported by the Linux Kernel
  - `$ adb shell cat /proc/filesystems`

```
nodev sysfs
nodev rootfs
nodev proc
…
        ext3
        yaffs
        yaffs2
```

  - Nodev means that it is a virtual file system

Common temp file systems

- Rootfs – kernel mounts the root file system (top /)
- Devpts – simulated terminal sessions
  - Similar to connecting to a Linux server
- Sysfs – contains configuration and control files
- Proc – provides detailed information about kernel, processes, and configuration parameters (structured)
- Tmpfs – stores all of the files in virtual memory backed by RAM
- Cramfs – used for mounting /system on some devices

Android YAFFS2 File System (mount)

```
$ adb shell mount

rootfs on / type rootfs (ro)
tmpfs on /dev type tmpfs (rw,mode=755)
devpts on /dev/pts type devpts (rw,mode=600)
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
tmpfs on /sqlite_stmt_journals type tmpfs (rw,size=4096k)
/dev/block/mtdblock3 on /system type yaffs2 (ro)
/dev/block/loop0 on /system/modules type cramfs (ro)
/dev/block/loop1 on /system/xbin type cramfs (ro)
/dev/block/mtdblock5 on /data type yaffs2 (rw,nosuid,nodev)
/dev/block/mtdblock4 on /cache type yaffs2 (rw,nosuid,nodev)
/dev/block/mmcblk0p1 on /sdcard type vfat (rw, dirsync,
   nosuid, nodev, noexec, uid=1000, gid=1000, fmask=0711,
   dmask=0700, codepage=cp437, iocharset=iso8859-1, utf8)
```

YAFFS2 Partitions

```
# cat /proc/mtd
dev:   size     erasesize  name
mtd0: 00180000 00020000   "pds"
mtd1: 00060000 00020000   "misc"
mtd2: 00380000 00020000   "boot"
mtd3: 00480000 00020000   "recovery"
mtd4: 08c60000 00020000   "system"
mtd5: 05ca0000 00020000   "cache"
mtd6: 105c0000 00020000   "userdata"
mtd7: 00200000 00020000   "kpanic"
```

Android File Systems YAFFS (directories)

$ ls -l /

```
drwxrwx---    1 1000     2001          2048 Sep  3 18:36 cache
drwxrwx--x    1 1000     1000          2048 Oct 24 22:44 data
-rw-r--r--    1 0        0               93 Jan  1  1970 default.prop
drwxr-xr-x   11 0        0             2400 Feb 25 03:08 dev
lrwxrwxrwx    1 0        0               11 Feb 25 03:08 etc -> /system/etc
-rwxr-x---    1 0        0           102464 Jan  1  1970 init
-rwxr-x---    1 0        0             1567 Jan  1  1970 init.goldfish.rc
-rwxr-x---    1 0        0             8780 Jan  1  1970 init.rc
-rwxr-x---    1 0        0             1189 Jan  1  1970 init.trout.rc
dr-xr-xr-x   73 0        0                0 Jan  1  1970 proc
drwx------    2 0        0                0 Jan  1  1970 root
drwxr-x---    2 0        0                0 Jan  1  1970 sbin
d---rwxrwx    2 1000     1000          4096 Feb 25 12:35 sdcard
drwxrwxrwt    2 0        0               40 Feb 25 11:35 sqlite_stmt_journals
drwxr-xr-x   12 0        0                0 Jan  1  1970 sys
drwxr-xr-x    1 0        0             2048 Feb 24 22:07 system
```

Android RFS File System (mount)

```
af@ubuntu:~$ adb shell
$ mount
rootfs / rootfs ro 0 0
tmpfs /dev tmpfs rw,mode=755 0 0
devpts /dev/pts devpts rw,mode=600 0 0
proc /proc proc rw 0 0
sysfs /sys sysfs rw 0 0
/dev/block/stl6 /mnt/.lfs j4fs rw 0 0
tmpfs /sqlite_stmt_journals tmpfs rw,size=4096k 0 0
none /dev/cpuctl cgroup rw,cpu 0 0
/dev/block/stl9 /system rfs ro,vfat,log_off,check=no,gid/uid/rwx,iocharset=utf8 0 0
/dev/block/mmcblk0p2 /data rfs
    rw,nosuid,nodev,vfat,llw,check=no,gid/uid/rwx,iocharset=utf8 0 0
/dev/block/mmcblk0p3 /data_tmo rfs
    rw,nosuid,nodev,vfat,llw,check=no,gid/uid/rwx,iocharset=utf8 0 0
/dev/block/stl10 /dbdata rfs
    rw,nosuid,nodev,vfat,llw,check=no,gid/uid/rwx,iocharset=utf8 0 0
/dev/block/stl11 /cache rfs
    rw,nosuid,nodev,vfat,llw,check=no,gid/uid/rwx,iocharset=utf8 0 0
/dev/block/stl3 /efs rfs rw,nosuid,nodev,vfat,llw,check=no,gid/uid/rwx,iocharset=utf8 0
    0
/dev/block//vold/179:1 /sdcard vfat
    rw,dirsync,nosuid,nodev,noexec,uid=1000,gid=1015,fmask=0102,dmask=0002,allow_utime=
    0020,codepage=cp437,iocharset=iso8859-1,shortname=mixed,utf8,errors=remount-ro 0 0
/dev/block//vold/179:9 /sdcard/sd vfat
    rw,dirsync,nosuid,nodev,noexec,uid=1000,gid=1015,fmask=0000,dmask=0000,allow_utime=
    0022,codepage=cp437,iocharset=iso8859-1,shortname=mixed,utf8,errors=remount-ro 0 0
```

Android File Systems RFS (partitions)

- $ cat /proc/partitions

```
major minor  #blocks  name

 139        0     513024 tfsr0/c
 139        1        256 tfsr1
 139        2        256 tfsr2
 139        3      10240 tfsr3
 139        4       1280 tfsr4
 139        5       1280 tfsr5
 139        6       5120 tfsr6
 139        7       7680 tfsr7
 139        8       7680 tfsr8
 139        9     293376 tfsr9
 139       10     137216 tfsr10
 139       11      35840 tfsr11
 139       12      12800 tfsr12
 179        0   15630336 mmcblk0
 179        1   13402104 mmcblk0p1
 179        2    1966080 mmcblk0p2
 179        3     262144 mmcblk0p3
 179        8    1985024 mmcblk1
 179        9    1983619 mmcblk1p1
 137        0     513024 bml0/c
 137        1        256 bml1
 137        2        256 bml2
 137        3      10240 bml3
 137        4       1280 bml4
 137        5       1280 bml5
 137        6       5120 bml6
 137        7       7680 bml7
 137        8       7680 bml8
 137        9     293376 bml9
 137       10     137216 bml10
 137       11      35840 bml11
 137       12      12800 bml12
 138        3       6400 stl3
 138        6       1280 stl6
 138        9     284672 stl9
 138       10     131584 stl10
 138       11      32000 stl11
```

Android File Systems RFS (directories)

$ ls -l /

```
drwxrwx--x system   system            2011-06-30 13:24 dbdata
drwxrwx--x system   system            2011-06-30 13:24 data_tmo
drwxrwx--x system   system            2011-06-30 13:24 userdata
drwxrwxrwt root     root              2011-06-30 13:33 sqlite_stmt_journals
dr-x------ root     root              2011-06-30 13:24 config
drwxrwx--- system   cache             2011-06-30 13:24 cache
drwxrwx--x radio    radio             2011-06-30 13:24 efs
drwxrwx--x system   system            2011-06-30 13:24 data
drwxrwxr-x system   sdcard_rw          2011-06-30 13:26 sdcard
drwxr-xr-x root     root              2011-01-12 22:18 mnt
-rwxr-xr-x root     root          335 2011-01-12 21:07 init.smdkc110.sh
drwxr-xr-x root     root              2011-01-12 22:18 lib
lrwxrwxrwx root     root              2011-01-12 22:18 etc -> system/etc
-rwxrwxrwx root     root        51428 2011-01-12 22:15 vold
drwxr-xr-x root     root              2011-06-30 13:24 dev
drwxr-xr-x root     root              2011-01-12 22:18 tmp
drwxr-xr-x root     root              2011-01-12 22:18 sbin
dr-xr-xr-x root     root              1969-12-31 18:00 proc
-rwxr-xr-x root     root           93 2011-01-12 22:15 default.prop
lrwxrwxrwx root     root              2011-01-12 22:18 init -> sbin/init
-rwxr-xr-x root     root        24717 2011-01-12 22:15 init.rc
-rwxr-xr-x root     root          727 2011-01-12 21:07 lpm.rc
drwxr-xr-x root     root              2011-06-30 13:24 system
-rwxr-xr-x root     root          154 2011-01-12 21:07 system.prop
-rwxr-xr-x root     root          444 2011-01-12 21:07 init.smdkc110.rc
drwxr-xr-x root     root              2011-01-12 22:18 res
-rwxr-xr-x root     root         1244 2011-01-12 21:07 recovery.rc
-rwxr-xr-x root     root         2948 2011-01-12 21:07 fota.rc
drwxr-xr-x root     root              1969-12-31 18:00 sys
```

/data partition

- /data/
  - dalvik-cache: .dex files that were run
  - app: .apk files (install bundle for applications)
  - data: subdirectories per application with sqlite databases
  - misc: dhcp, wifi, etc. files
  - system:
    - packages.xml (installed applications)
    - accounts.db
    - etc.

/cache partition

- The /cache partition stores some important data from a forensics standpoint
  - Gmail attachment previews
  - Browser DRM
  - Downloads
  - Market downloads
  - OTA updates

Session Summary

- There can be many different file systems running on an Android device.

- FTL manages the nand sub processes including wear leveling, garbage collection, and address mapping. Some chip manufacturers are now building the FTL into the chip, making the OOB data irrecoverable.

- Data contained on a nand cannot be modified. It must be erased completely then re-written (log-structured). This happens at the block level so every page within that block must be marked for deletion before the block will be erased.

Training Outline

1. Android Overview
2. NAND Memory and Android File Systems
3. **Forensic Techniques**
   a. viaExtract acquisition
   b. Passcode demo
4. Conclusion

Android Forensic Acquisition Techniques

- SD Card analysis (physical and simulated)
- Examining Backups
- Android Debug Bridge
- Commercial tools (i.e viaExtract application)
- Other methods: Flasher Box, JTAG, Chip-off
- AFPhysical Method (viaForensics)

SD Card Analysis

- Most Android phones come with an SD Card for external data storage
- This card should be removed immediately upon receiving an Android phone.
- The SD Card should be forensically acquired using a USB write blocker
- Traditional tools and techniques can be used since this is FAT32
- EMMC
- viaExtract will image the SD card

SD Card – Partitions

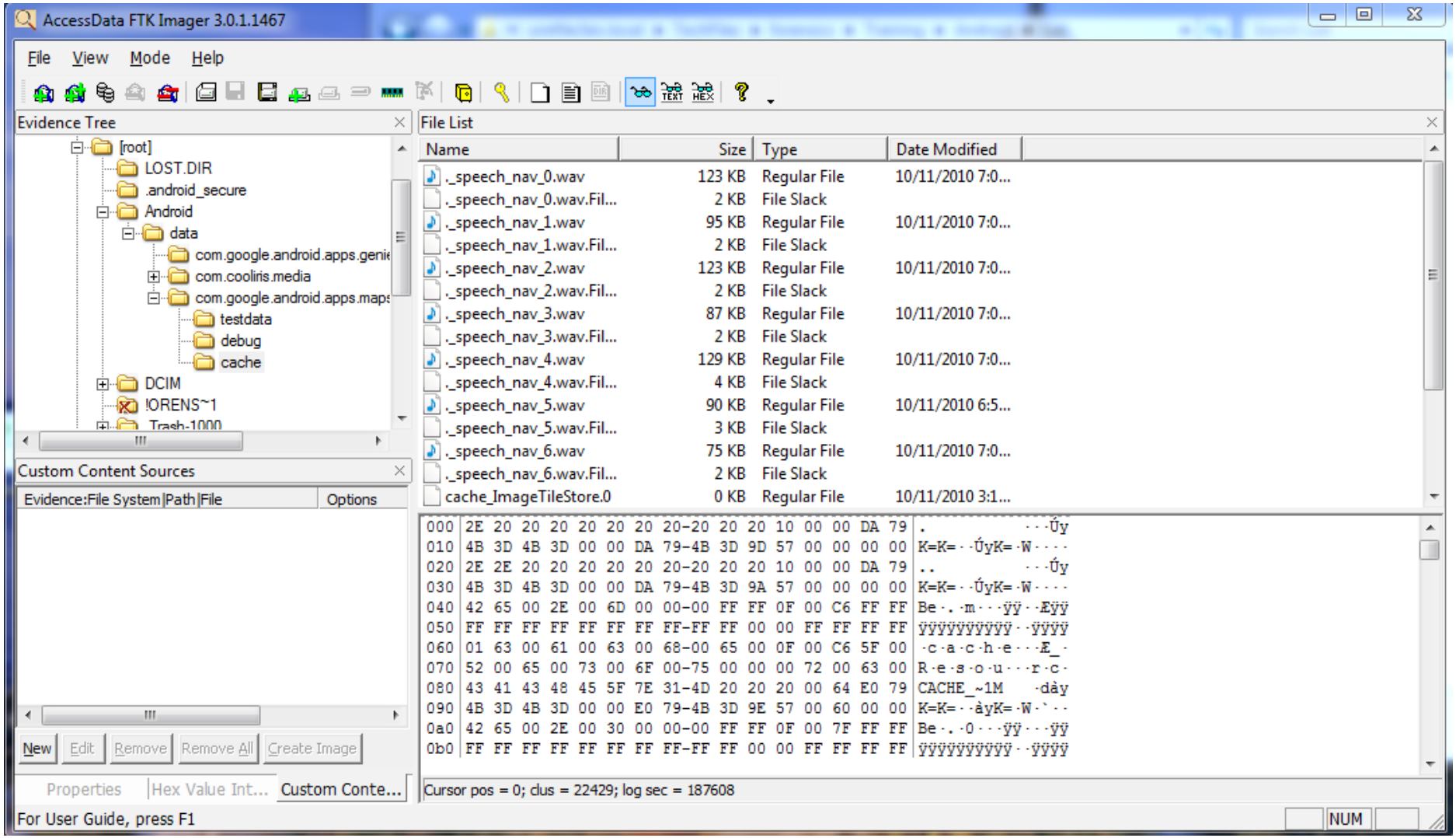
- The SD Card is formatted with FAT32

```
$ mmls ~/droid/sd/viaforensics/SDcardImage.dc3dd

DOS Partition Table
Offset Sector: 0
Units are in 512-byte sectors

     Slot    Start        End          Length       Description
00:  Meta    0000000000   0000000000   0000000001   Primary Table (#0)
01:  -----   0000000000   0000008191   0000008192   Unallocated
02:  00:00   0000008192   0031326207   0031318016   Win95 FAT32 (0x0C)
```

SD Card DD Analysis



SD Card - Files

- Files on my Droid
  - Pictures and videos
  - Music
  - Google Maps Navigation - com.google.android.apps.maps
  - Voice Recorder - com.tokasiki.android.voicerecorder
  - Downloads from the browser or email
  - LOST.DIR which can contain one or more JFIF files.
  - Cache for NewRob, a popular RSS reader
  - Rerware's My Backup Pro has an option to store backups to the SD Card. If present, will contain important user data
  - Twidroid - com.twidroid image cache from a popular Twitter application
  - Much more application / media files

Typical Backup Methods

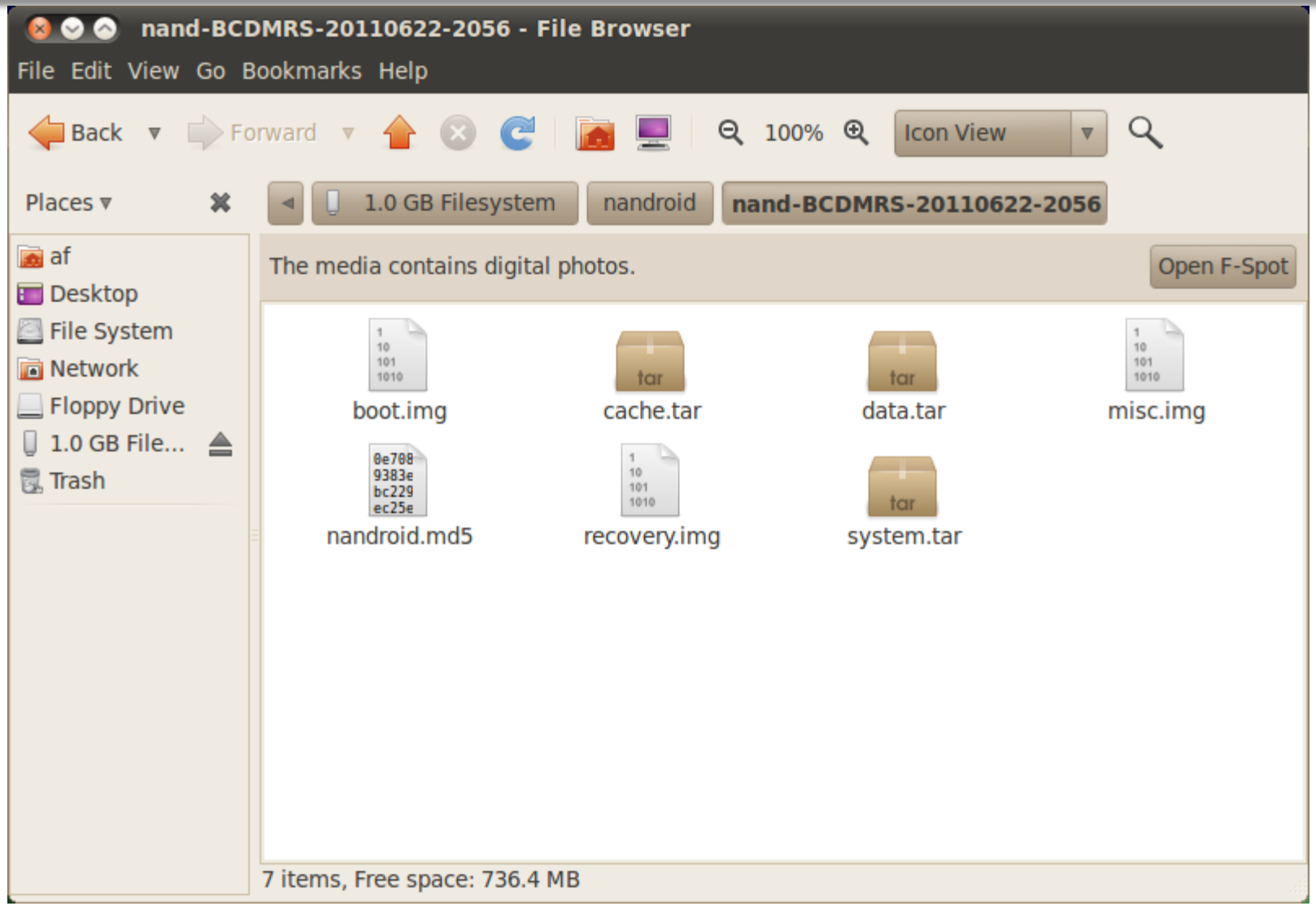
- Commercial backup options
  - MyBackup Pro
    - Seems to be most popular
    - Backups up much of the standard data and then applications
    - Can backup to SD Card or Cloud
    - Creates 1 SQLite file on SD Card and then key files (i.e. MMS)
  - Others include Titanium Backup, Lookout Mobile Security, etc.

- Built in backup options
  - Export to SD
  - Google cloud backup

Nandroid backup

- Users can install a custom recovery partition which may have tools to acquire data (tar)
  - Creates a backup of the selected nand partition on SD card (logical only)
  - Comes with mounting options for USB connectivity
  - Byproduct is shell root

Nandroid Contents



adb Method

- adb can recursively pull data from Android devices (need to turn on USB debugging)
- The daemon on the device (adbd) generally runs as shell user so device must be rooted.

```
root@wintermute:/home/ahoog/adb-pul# adb pull /data adbpull/
pull: building file list...
<snip>
pull: /data/miscrild_nitz_long_name_31026 ->
data/misc/rild_nitz_long_name_31026
pull: /data/misc/akmd_set.txt -> data/misc/akmd_set.txt

712 files pulled. 0 files skipped.
963 KB/s (208943249 bytes in 211.671s)
```

pulled 1,255 files (19MB) in about 90 seconds.

Exploits

- Various root exploits allow rooting without overwriting user data
  - Recovery partition
- Rage.bin
- PSNueter
- Gingerbreak
- Motorola 12m
- One click root solutions

JTAG

- Some Android phones and NAND devices still have JTAG pads on PCB
- With proper equipment, a dump of the NAND memory chip can be achieved
- EMMC data is generally not obtainable through JTAG.
- Recovery of gesture.key

Commercial Tools

- Vendors who support Android
  - viaForensics viaExtract & AFLogical
  - Cellebrite
  - Microsystemation XRY
  - Oxygen Forensic Suite
  - EnCase Smartphone Examiner

- Like any situation, forensic analysis should test the tools, understand how they work and be able to explain if needed.

viaForensics AFLogical App

- Android has enforced security at the application level very well
- Framework provides for applications sharing data
  - i.e. Twitter applications need access to SMS data. Default install of applications (contacts, call logs, SMS, etc.) allow information sharing, if the user approves
- viaForensics developed AFLogical application
  - Free to law enforcement/government
  - Read data from applications
  - Write to CSV on SD Card
  - http://viaforensics.com/products/aflogical

AFLogical – Supported Content Providers

- Browser history
- Call Logs
- Contact Methods
- External Image Media (meta data)
- External Image Thumbnail Media (meta data)
- External Media, Audio and Misc (meta data)
- External Videos (meta data)
- MMS
- MMSParts (includes full images sent via MMS)
- Organizations
- People
- SMS
- List of all applications installed and version
- Contacts Extensions
- Contacts Groups

See up to date feature list at
http://viaforensics.com/products/aflogical

AFLogical – Supported Content Providers

- Contacts Phones
- Contacts Settings
- Browser Searches
- Maps-Friends
- Maps-Friends extra
- Maps-Friends contacts
- Maps Search History
- Notes
- People Deleted (phone specific)
- IM Providers
- IM Account
- IM Accounts
- IM Contacts
- IM Messages
- IM Invitations

See up to date feature list at
http://www.viaforensics.com/products/aflogical

viaForensics viaExtract Application

- viaExtract is the latest Android forensic solution from viaForensics
- Cross platform support – Windows, MAC, Linux
- Ubuntu virtual machine, one stop for mobile forensics

viaExtract Features

- Enhanced AFLogical
- Crowd sourcing allows for continual additions to supported content provider list
- Gesture key decoding
- SD card imaging
- Excellent reporting capability
- Load previous AFLogical cases

viaExtract 2012 Initiatives

- Optimized queries
- SQLite parsing
- Physical module

Taking any of our training courses entitles you to a 10% discount on viaExtract:

- iOS Forensics
- Android Forensics
- Advanced Mobile Forensics
- Advanced File Carving

Training Outline

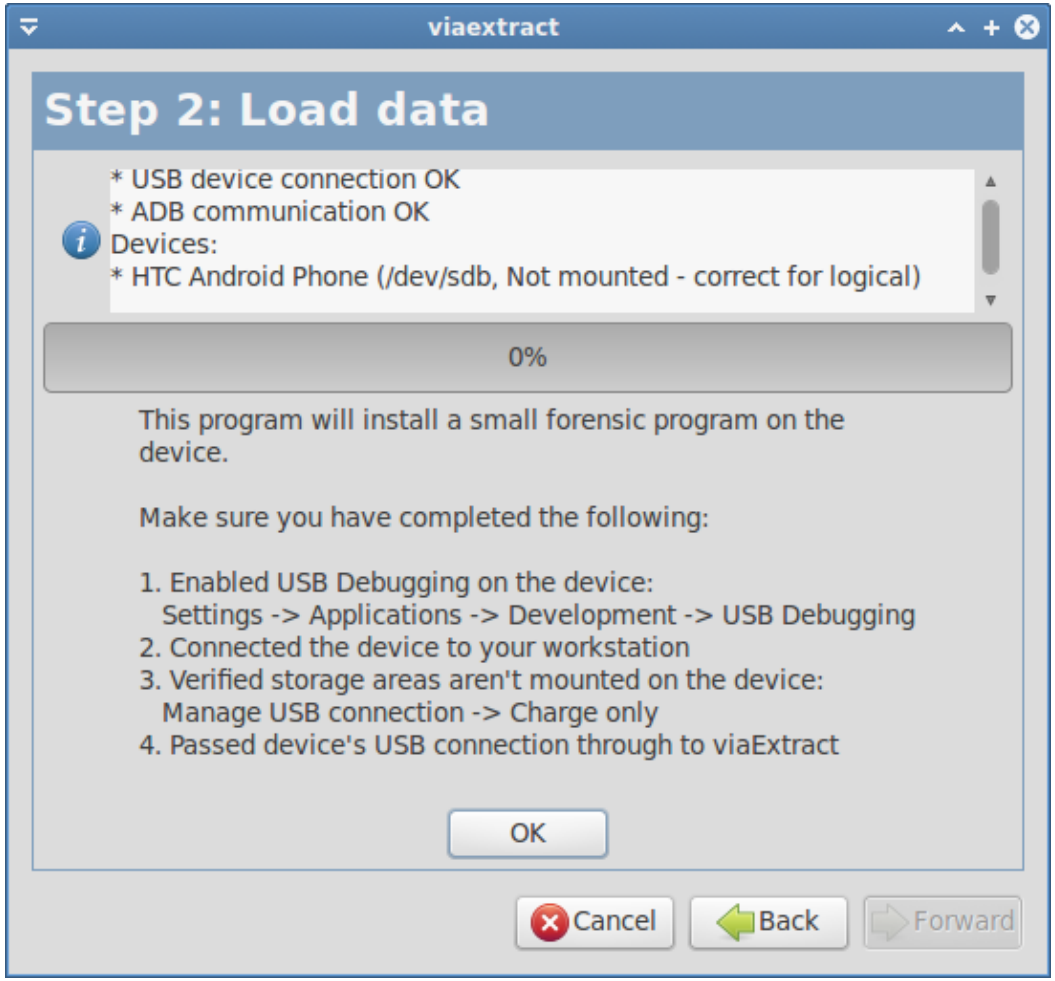
1. Android Overview
2. NAND Memory and Android File Systems
3. Forensic Techniques
   a. **viaExtract acquisition**
   b. Passcode demo
4. Conclusion

Lab 1 – viaExtract

- Start new Case

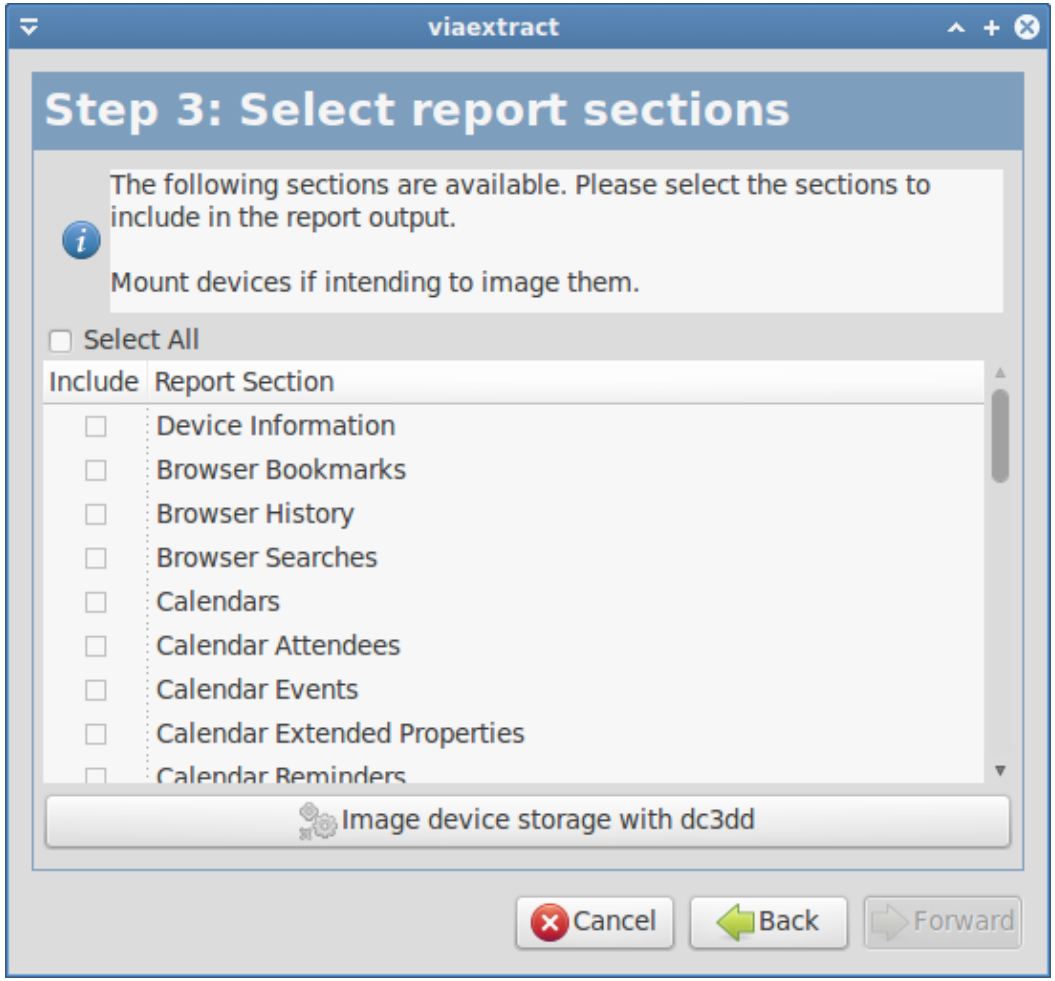
Lab 1 – viaExtract

- Load data



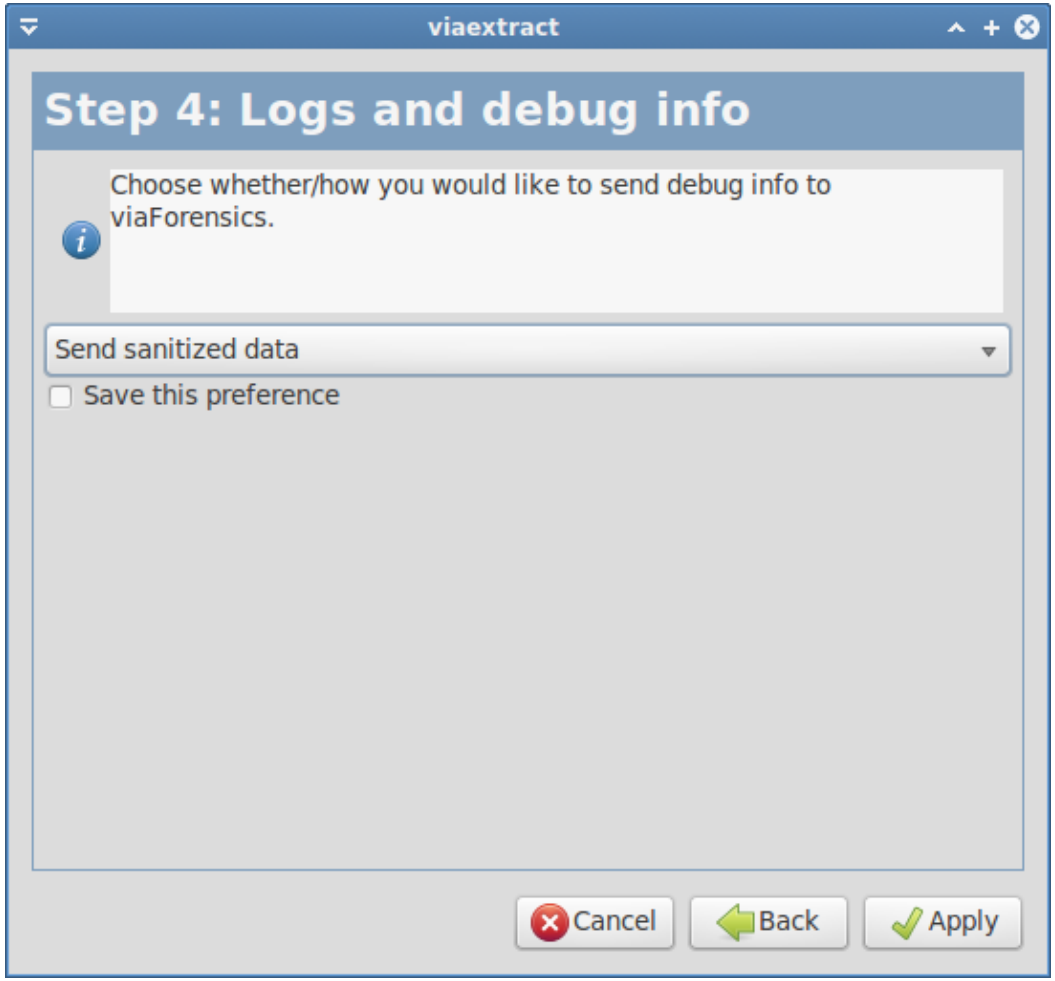
Lab 1 – viaExtract

- Report selections



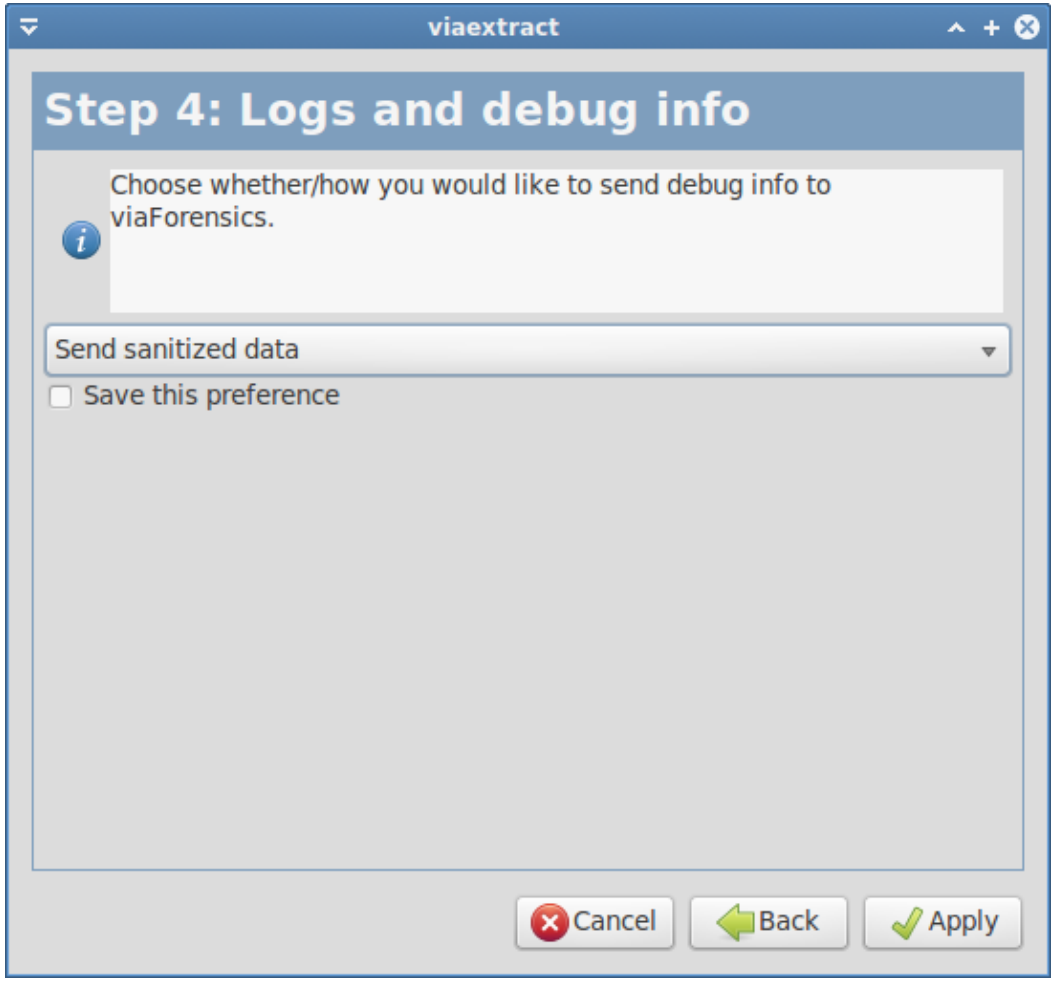
Lab 1 – viaExtract

- Logs and debug info



Lab 1 – viaExtract

- Logs and debug info



Lab 1 – viaExtract

- Image storage device

Training Outline

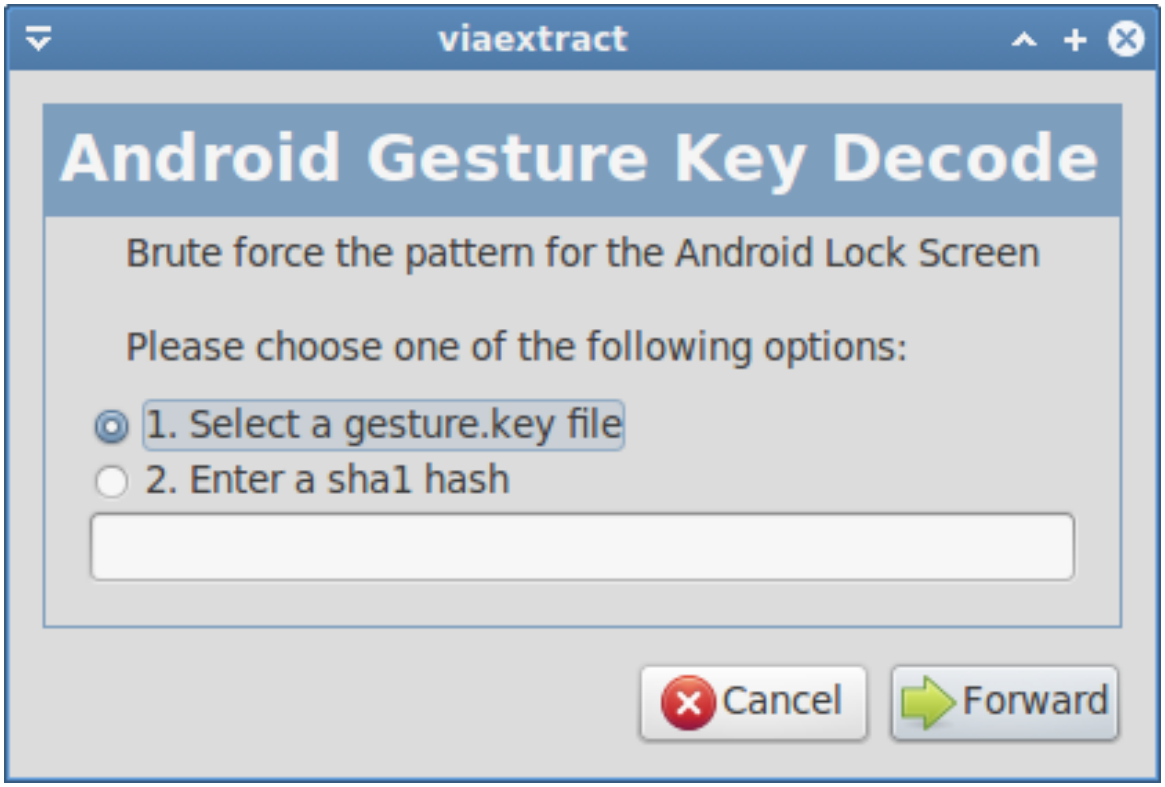
1. Android Overview
2. NAND Memory and Android File Systems
3. Forensic Techniques
   a. viaExtract acquisition
   b. Passcode demo
4. Conclusion

Lab 2 – Gesture Key Decoding

- Phone must be rooted in order to obtain a copy of the gesture.key
- Desirable feature if agency requires project-a-phone or other screenshot tool for evidence collection
- Gesture.key located in /data/system

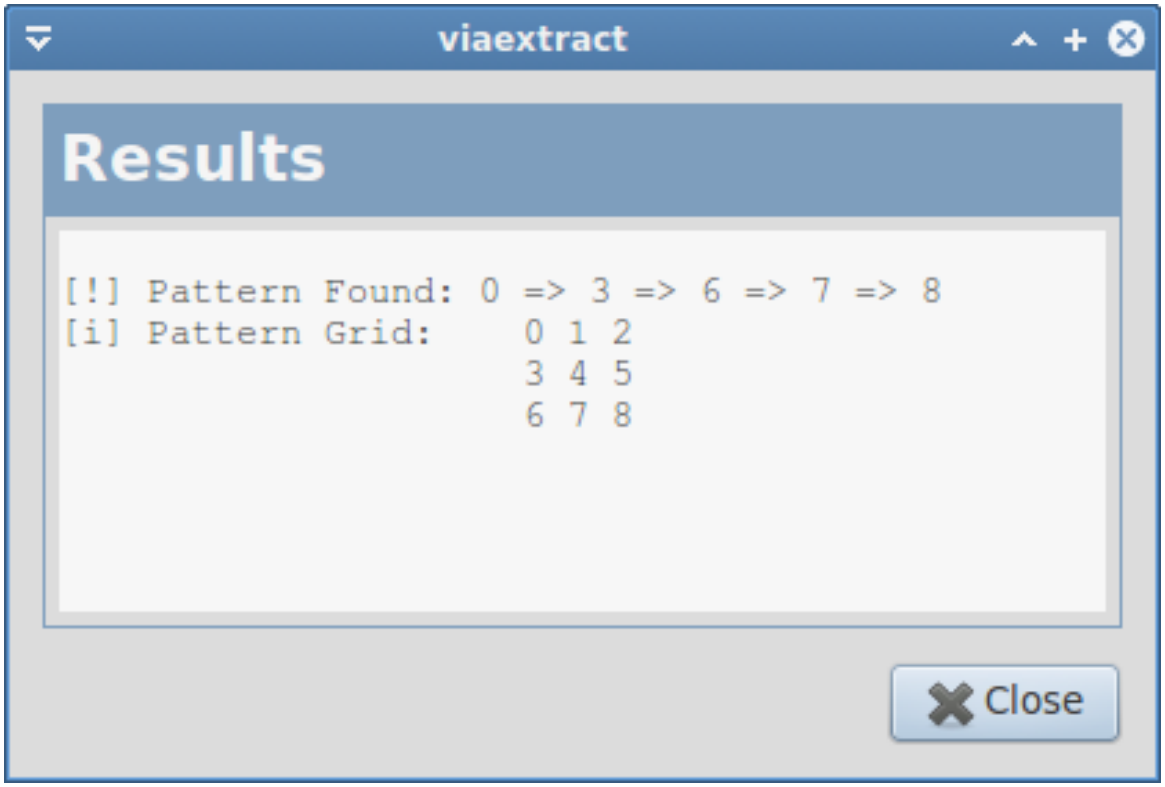
Lab 2 – Gesture Key Decoding

- Gesture decode input



Lab 2 – Gesture Key Decoding

- Gesture decode output



Training Outline

1. Android Overview
2. NAND Memory and Android File Systems
3. Forensic Techniques
   a. viaExtract acquisition
   b. Passcode demo
4. **Conclusion**

Upcoming Training

- We will be offering a full five day training session in Chicago May 21-25!
  - Monday May 21; Intro to Linux in Forensics
  - Tuesday May 22; iOS Forensics
  - Wednesday May 23; Android Forensics
  - Thursday May 24; Advanced Mobile Forensics
  - Friday May 25; Advanced File Carving

  Full series is $6295 and discounted to $4295 for LE. Contact Chris Triplett if interested in hosting the event!

  ctriplett@viaforensics.com – 414-331-5030

Training Contact Information

VIAFORENSICS
innovative digital forensics and security

Christopher Triplett

Sr. Forensic Engineer

T +1 312-878-1100
F +1 312-268-7281
E ctriplett@viaforensics.com

1100 Lake Street
Suite 203
Oak Park, IL 60301-1131

viaExtract Contact Information

Joshua D. Laborde

Director of Product Development

T +1 312-878-1100

F +1 312-268-7281

E jlaborde@viaforensics.com

1100 Lake Street

Suite 203

Oak Park, IL 60301-1131