Data Recovery on sold/stolen Android smartphones: a Security Analysis of Current Protections

Laurent Simon
lmrs2@cam.ac.uk
https://www.cl.cam.ac.uk/~lmrs2/
Intro

• Observation: a great number of papers focus on apps:
  • Malware
  • Potentially Harmful Apps (PHA)
  • benign but buggy apps that can be subverted

• Question: which attack vectors cannot be mitigated thru central app store?
Intro

• Resold devices

• Stolen devices

• Infiltrate supply chain of new or resold devices
Part 1: Android Factory Resets
Part 1: outline

- Background
- Methodology
- Results
- Practical recovery
- FR alternatives
- Mitigation
Background

- Second-hand phone market growth
  - 57M, 2014 (Gartner)
  - 2/3 second life, 2015 (Gartner)
  - 150-250M traded by 2018

- Data recovery success reported
  - Avast, BBC news, etc
Secure Deletion

- **Logical Sanitisation**: data cannot be recovered via standard hardware interfaces like standard eMMC commands
- **Digital Sanitisation**: data cannot be recovered via any digital means, including the bypass or compromise of the device’s controller or firmware, or via undocumented drive commands
- This talk: *logical sanitisation*
Data Storage Locations

- **Data partition** mounted on /data
  - Sensitive info, ext4 (eMMC), yaffs2 ("raw flash")

- **Internal (primary) "SD card"**: mounted on /sdcard
  - Music, pictures, FAT, emulated (FUSE)

- **External SD card**: removable
  - Same as internal one, FAT
  - Secondary SD card, or primary if no internal one
Data Storage Locations

- /data       /sdcard (primary)

- /data

- /data       /sdcard (primary)
Flash Memory - Overview

- Unlike HDDs, Solid State Storage (SSD) supports a limited number of erase cycles (100000)

=> memory management, wear-leveling algo

blocks used by file system

data 1

clean blocks
data’
data”
to-be-erased dirty blocks
Flash Memory - Overview

- Unlike HDD, Solid State Storage (SSD) support a limited number of erase cycles (10000)

=> memory management, wear-leveling algo
Flash Memory – File Systems

- Software: flash-aware file system yaffs2
- Hardware: eMMC (logical view for OS)
How to securely delete?

- Yaffs2:
  Exposed via ioctl(fd, MEMERASE, blk_num)

- eMMC: special commands to send to the chip
  Exposed via:
  - ioctl(fd, BLKDISCARD, blknum)
  - ioctl(fd, BLKSECURITYDISCARD, blknum)
EMMC deletion (BLKSECDISCARD)

- Case 1: digital sanitisation by purging physical blocks if chip/driver support it
- Case 2: logical sanitisation only by remapping blocks to a zero block

OS logical view

Flash memory
Part 1: outline

- Background
- Methodology
- Results
- Practical recovery
- FR alternatives
- Mitigation
Phone Acquisition

Proportion of devices (%)

June 2013
- Froyo (v2.2.x): 31M
- KK (v4.4): 0M
- GB (v2.3.x): 341M
- ICS (v4.0.x): 233M
- JB (v4.[1-3]): 379M

Our sample
- Froyo (v2.2.x): 31M
- KK (v4.4): 0M
- GB (v2.3.x): 341M
- ICS (v4.0.x): 233M
- JB (v4.[1-3]): 379M

March 2014
- Froyo (v2.2.x): 31M
- KK (v4.4): 0M
- GB (v2.3.x): 341M
- ICS (v4.0.x): 233M
- JB (v4.[1-3]): 379M

Other: 15M
Setup

- Overwrite "bit-by-bit" partitions (data, primary and secondary SD card) with identifying patterns
  - Bit-by-bit = lower level possible (dd-like)
  - Identifying patterns = unique ID
- Factory Reset
- Pattern recovery and identification
Setup (Cont'ed)

Recovery

![Android system recovery screen]

Settings

Privacy settings

- **Backup and restore**
  - Back up my data
    - Back up application data, Wi-Fi passwords, and other settings to Google servers
  - **Automatic restore**
    - If I reinstall an application, restore backed up settings or other data
- **Personal data**
  - **Factory data reset**
    - Erases all data on phone

Factory data reset

- This will erase all data from your phone's **internal storage**, including:
  - Your Google account
  - System and application data and settings
  - Downloaded applications
- To also clear music, pictures, and other user data, the **SD card** needs to be erased.
- **Erase SD card**
  - Erase all the data on the phone's **SD card**

![Android settings screen]
Part 1: outline

- Background
- Methodology
- Results
- Practical recovery
- FR alternatives
- Mitigation
Results: Data partition

- Froyo (v2.2.x)
  - ioctl(MEMERASE)
  - 0M

- GB (v2.3.x)
  - ioctl(BLKDISCARD)
  - 295M

- ICS (v4.0.x)
  - ioctl(BLK SEC DISCARD)
  - 138M

- JB (v4.[1-3])
  - ioctl(BLK SEC DISCARD)
  - 53M

- Devices with insecure deletion (%)

22/05/15
Laurent Simon - Samsung - USA
Results: Data partition (Cont'ed)

• Upgrade from GB (2.3.x) to ICS (4.0.x)
  • ioctl(BLKSECDISCARD) return errno 95 EOPNOTSUPP

• 2007 eMMC standard has compulsory support for logical sanitisation

• HTC Sensation XE correctly wipes data partition in Bootloader mode but not for Android Factory Reset
Results: Data partition

- Froyo (v2.2.x)
  - ioctl(MEMERASE)
  - Devices with insecure deletion (%): 0M

- GB (v2.3.x)
  - ioctl(BLKDISCARD)
  - Devices with insecure deletion (%): 295M

- ICS (v4.0.x)
  - ioctl(BLK SEC DISCARD)
  - Devices with insecure deletion (%): 138M

- JB (v4.[1-3])
  - ioctl(BLK SEC DISCARD)
  - Devices with insecure deletion (%): 53M
Results: Primary SD card

<table>
<thead>
<tr>
<th>Devices with insecure deletion (%)</th>
<th>0</th>
<th>20</th>
<th>40</th>
<th>60</th>
<th>80</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Froyo (v2.2.x)</td>
<td>30M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GB (v2.3.x)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>138M</td>
</tr>
<tr>
<td>ICS (v4.0.x)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JB (v4.[1-3])</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>159M</td>
</tr>
</tbody>
</table>

- `format()`
- `ioctl(BLKDISCARD)`
Results: Secondary SD card

Not supported in AOSP code
Results: vendor customisations

- Recall: FR in settings wipes primary SD card, Recovery mode does not
  - HTC One series do the opposite!
- Nexus 4: 20% of the time, last 16KB not logically sanitised
  - EMMC/driver bug?
- Asserting FR characteristics from android version is error prone and should not be relied upon
Part 1: outline

- Background
- Methodology
- Results
- Practical recovery
- FR alternatives
- Mitigation
Practical Recovery

- Contact (Facebook, Phonebook, WhatsApp, etc)
- Conversation (emails, SMSs, Facebook & WhatsApp chats, etc)
- Browsing history
- Credentials (Facebook cookies, etc)
- Multimedia
Practical Recovery (Cont'ed)

- Android (master) auth token(s)
- Master token can be used to get other tokens from Google
- Tokens recovered 100% of the time, master one 80%

username@gmail.com.com.googleAFcb4KRs88NZlzN-r6qHrSHGF1Twyh...TKw==
c1DQAAAJ4AAABQPfQhNXLTDDYDLgHoIFDdDIEojBokYr_6ad0WeSr2kVpK4...B-0pd
androidmarketDQAAAJ8AAD1NNQaeO_yxfgNMtSvnQVangE3DAatlKtTo...INkZV
Part 1: outline

- Background
- Methodology
- Results
- Practical recovery
- FR alternatives
- Mitigation
Alternatives to built-in FR

- Overwrite bit-by-bit: one pass enough to provide logical sanitisation
- Filling unallocated space (create files) to overwrite: discarded because:
  - Extra level of indirection
  - File systems vary (ext4, FAT, FUSE, Samsung's proprietary RFS)
Alternatives to built-in FR (Cont'ed)

- Full Disk Encryption (FDE), >= ICS only (v4.0.x)
  => not possible on GB (2.3.x) vulnerable devices
- Only support for data partition
- Encryption key stored encrypted using user's PIN in so called "crypto footer"
  - Crypto footer not sanitised with flawed FR
  - Crypto footer allows PIN brute-force
- Android lollipop (5.x): default encryption has hardcoded password "default_password"
Part 1: outline

- Background
- Methodology
- Results
- Practical recovery
- FR alternatives
- Mitigation
Mitigation

- Use flash that support *digital sanitisation*
- Use emulated SD card partition
- Implement sanitisation of *all* partitions in one place
- Have option to perform sanitisation validation
- Test units. In Compliance Suite Test (CST)?
- Store crypto footer as a header
Part 1: Conclusion

- Android FR in messy state

- Android code, vendors' customisations and lack of proper testing

- Mostly available on the second-hand market NOW
Part 2:
Anti Theft Solutions
by
Mobile Anti Virus (MAV) apps
Background

- Phone theft is a growing problem
  - 2013:
    - 3.1M devices stolen in the USA
    - 120,000 in London
  - 50% of users don't lock their phone
Anti-Theft Solutions

- Wide offering – enterprise and consumer-grade

  => This talk: *consumer grade* only

- Top 10 Mobile Anti Virus apps (MAV), downloaded from Google Play hundreds of millions of times (top 2 between 100M and 500M)

- Anti-theft enable *remote wipe* and *remote lock* with an app on phone + remote trigger via
  - web page
  - SMS
Part 2: outline

● Background
● Mobile Anti Virus (MAV) sample
● Wipe
● Lock
Admin API

- Provides admin features, i.e. sensitive functions
- Access to various "policies": e.g. `force-lock`, `wipe-data`, `reset-password`
- Like traditional Android permissions, each policy declared in Android manifest file
- Like traditional Android permissions, policies not accepted at installation but manually enabled/disabled in the phone Settings
Admin API (Cont'ed)

Device administrators

- avast! Anti-Theft
- McAfee Security
- Sample Device Admin

Activate device administrator?

McAfee Security

Activating this administrator will allow the application McAfee Security to perform the following operations:

- **Erase all data**
  Erase the phone's data without warning, by performing a factory data reset

- **Change the screen-unlock password**
  Change the screen-unlock password

- **Set password rules**
  Control the length and the characters allowed in screen-unlock passwords

- **Monitor screen-unlock attempts**

[Activate] [Cancel]
Admin API (Cont'ed)

- If user does not grant admin access, app can still run ... without admin privileges

- To uninstall/remove admin app, admin privileges must be disabled first

- Restrictions imposed: cannot read other apps' data or read/write chip at block level
Admin API (Cont'ed)

• Focus of this talk: force-lock and wipe-data policies

• `wipeData(int flag)`:
  • Triggers the built-in Factory Reset
  • Flag indicates:
    – Wipe only data partition
    – Wipe data partition AND primary SD card

• `LockNow()`: lock the screen with default Android PIN

• No admin granted: ad-hoc solutions
Modes

- Normal mode: Android
- Safe mode
- Recovery/Bootloader mode
Part 2: outline

- Background
- Mobile Anti Virus (MAV) sample
- Wipe
- Lock
Apps studied

- 10 most downloaded Mobile Anti Virus (MAV) apps on Google Play
  - AVG, Lookout, Avast, Dr.web, Norton, McAFe, Kaspersky, TrustGo, TrendMicro, Avira
- Top 2 downloaded 100M-500M
- Following top 4 10M-50M
Part 2: outline

- Background
- Mobile Anti Virus (MAV) sample
- Wipe
- Lock
Wipe implementations

- Data partition: 10/10 use admin API to wipe it
  - If no admin privileges, just use phone APIs (contact, SMS, etc)
- Primary SD: 5/10 MAVs use admin API to wipe it
  - Other MAVs unlink and/or overwrite files and/or format partition
- Secondary SD: 10/10 MAVs use ad-hoc solutions (unlink, overwrite files, format partition). *Android has no API to wipe it.*
Lookout implementation

- Overwrites files and unlinks them
- Dev assume file update occurs "in-place"
- On Galaxy S Plus, FAT-formatted primary SD: >90% data recoverable
Avast implementation

- "Thorough wipe" option:
  - Unlinks all files from external storage
  - Creates a 1MB file and overwrites it 1000 times with zeros
• Dev assume file update does NOT occurs "in-place", so 1GB (1000x1MB) unallocated space is overwritten
• Partitions formatted with ext4 update "in-place", 99% of data is recoverable
TOWTOS

- Time-Of-Wipe-Time-Of-Success (TOWTOS)
- Change API with a url + cookie?
Part 2: outline

- Background
- Mobile Anti Virus (MAV) sample
- Wipe
- Lock
Removal of MAVs & API Misuse

- **Scenario: admin + non-locked:**
- **7/10** MAVs do not prevent disabling admin privileges
- McAfee and Avast prompt user with PIN when trying to disable admin
Removal of MAVs & API Misuse

public class McAfeeReceiver extends DeviceAdminReceiver {

    public void onDisabled(Context paramContext, Intent paramIntent) {
        [...] // removed
        displayLockScreen();
    }
}

• Android doc: "called prior to the administrator being disabled"

• BUT called after on Gingerbread (GB, v2.3.x)

• OnDisabledRequested() called prior on GB, ICS, JB
Other API Misuses

• **Scenario:** admin + locked: proper lock implementation requires:
  - Force-lock policy declared in manifest file by MAV
  - Manual granting of admin by users
  - Proper use of API by MAV, e.g. lockNow()

• **4/10** MAVs do not use `lockNow()` even when granted admin privileges
  - Bypass thru Safe mode
Rate Limiting

- Scenario: admin + locked + use lockNow()

- Overlay of custom lock screen on top of default Android PIN screen
Rate Limiting

- **Scenario:** admin + locked + use \textit{lockNow()} + rate limiting
- Some devices have no rate limiting (e.g. Samsung Galaxy S Plus)
- Reboot into Safe mode where user-installed apps do not run automatically
- Counter storing glitches: e.g. for Lookout, removing battery resets the state
Rate Limiting

- **5/10** MAVs do not enforce rate limiting in their screen => brute-force PIN feasible

- For a 4-digit PIN and 5sec/PIN attempt, about 7hrs on average for randomly selected PINs

- <5mn for 60 most common PINs ~ 30%
- <40mn for 400 most common PINs ~ 50%
Network-level attacks: GSM

• Avast (100M-500M download) sends temp PIN in clear
• Similar issue for Dr.Web with commands sent via SMS
Network-level attacks: TLS

- Impersonate as cloud server to send an unlock command
- One app did not validate the CN of certs
Vendor customisations

- Charging mode gives shell: e.g. LG L7 running JB (v4.1.2)

- Unprotected Recovery/Booolder: flash arbitrary binaries to access data regardless of Android lock. Most Samsung/LG phones in our sample.
Part 2: Conclusion

- Lock implementations can be circumvented because of misuse of APIs, vendor customisations, restrictions imposed by Android

- Wipe implementations are not better than the built-in (possibly flawed) Factory Reset

- Vendor solutions only have the potential to increase reliability
Laurent Simon

lmrs2@cam.ac.uk

https://www.cl.cam.ac.uk/~lmrs2/