Post-Mortem Memory Analysis of Cold-Booted Android Devices

Christian Hilgers   Holger Macht
Tilo Müller   Michael Spreitzenbarth

FAU Erlangen-Nuremberg
Chair of Computer Science 1
Prof. Felix Freiling

IMF 2014

8th International Conference on
IT Security Incident Management & IT Forensics
May 12th - 14th, 2014
Münster, Germany

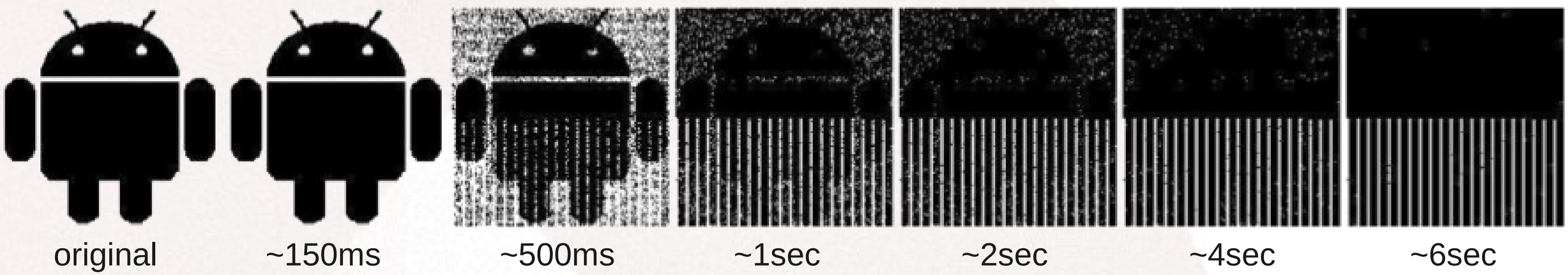
Introduction:
Cold-Boot Attacks against Android

FROST

- FROST: *Forensic Recovery of Scrambled Telephones*
- Cold-boot based recovery tool for encrypted Android smartphones.
- Scenario:
  - Criminal leaves phone behind at the scene, or the phone gets confiscated.
  - The suspect is not able or willing to tell the PIN.
  - Phone is *switched-on* when police accesses it, but its user partition is *encrypted*.
  - Although all data on disk are encrypted, RAM contents are never encrypted!

Remanence Effect

- RAM is not lost immediately after power is cut but fades away gradually over time.



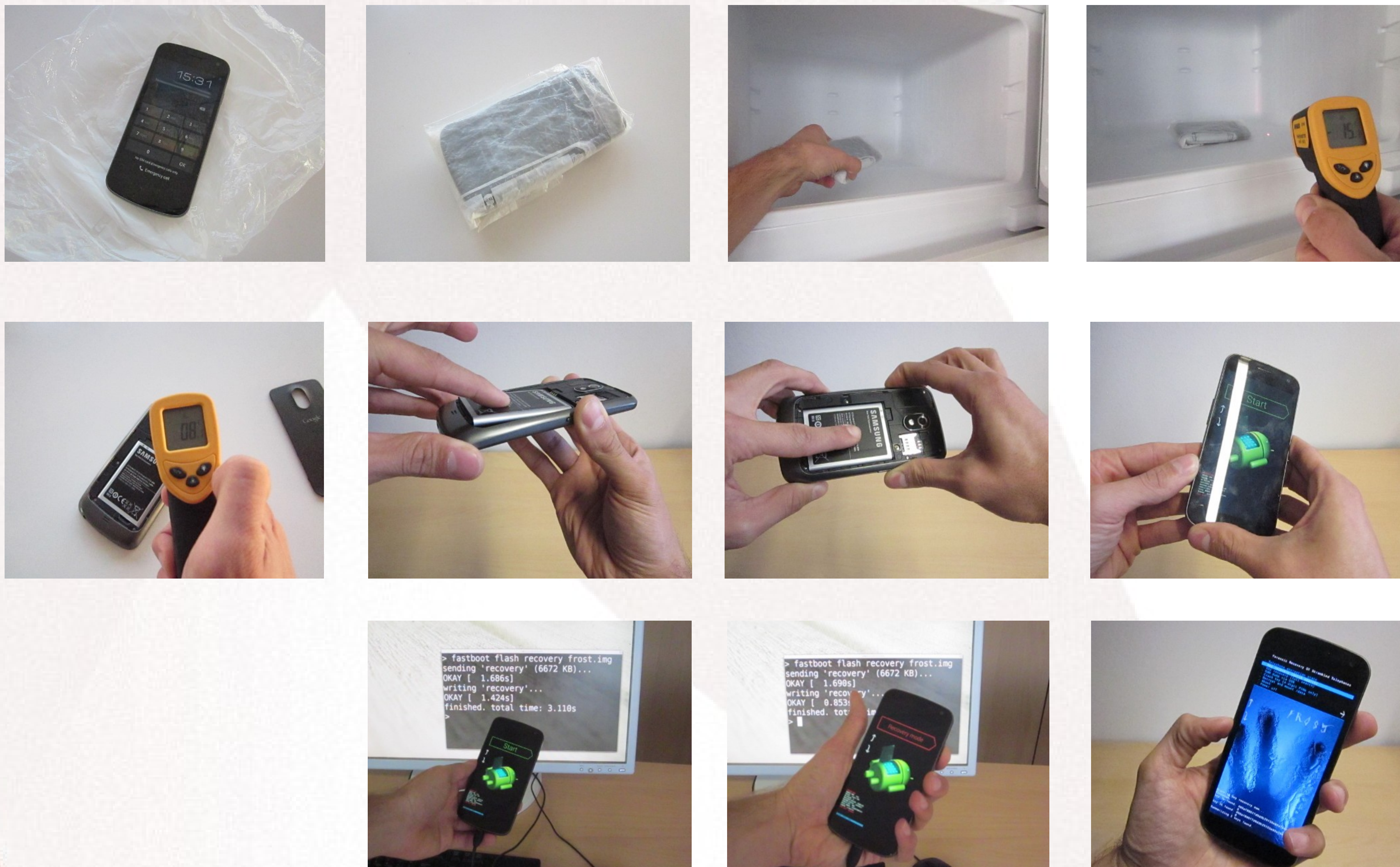
- Cooling down RAM chips slows down the fading process (e.g, on PCs up to 40 sec).
- Question: How to acquire RAM dumps from cold-booted Android phones?

Example: Samsung Galaxy Nexus

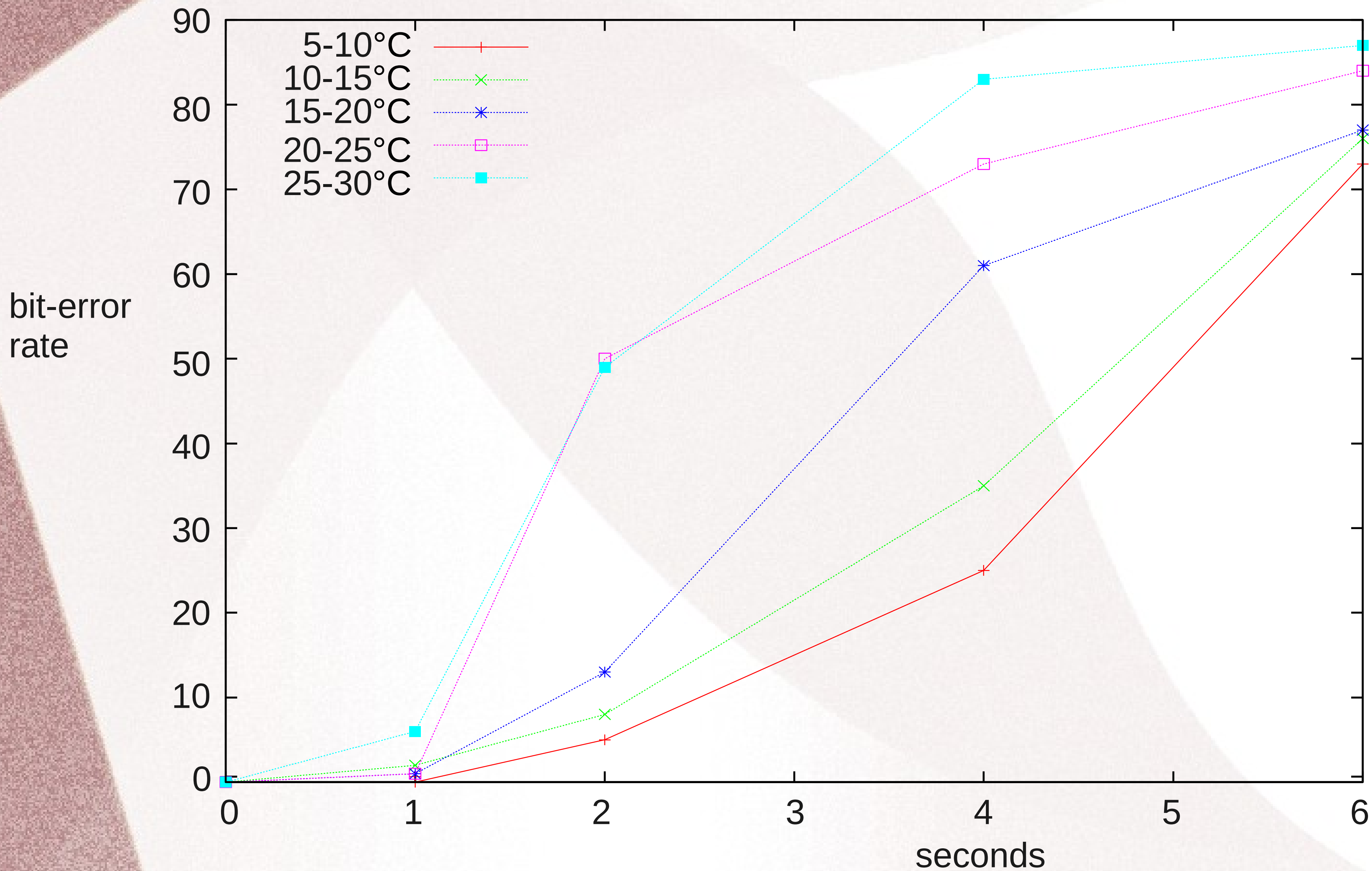
Android phones have open bootloaders that enable us to run our own system code:

- Bootloaders are locked by default
- Bootloaders can be unlocked with physical access via USB
- Unlocking wipes the user partition...
- …but *RAM gets not wiped!*

The FROST Attack



Evaluation: Bit-Error Rate



Post-Mortem Memory Analysis

Android Memory Contents

Simple Memory Analysis

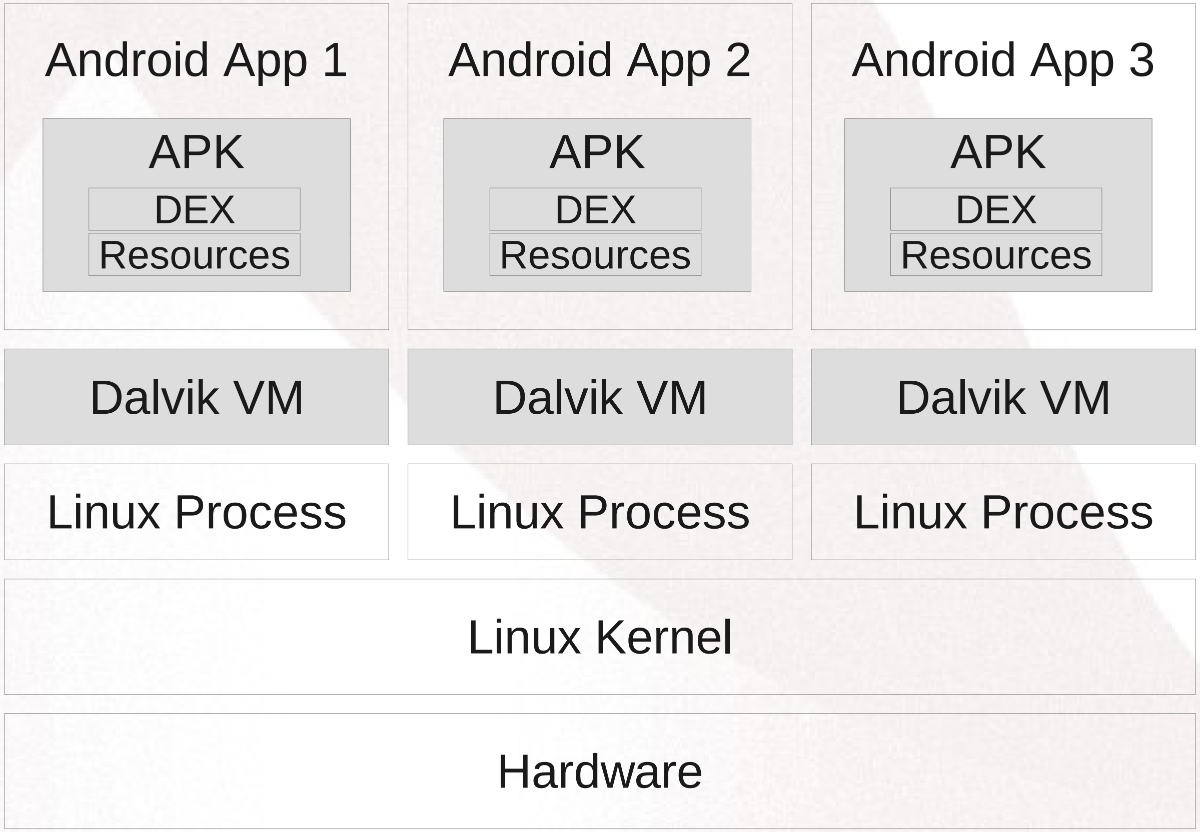
- Tools like *PhotoRec* and *Strings* can recover plenty of sensitive data from Android images:

	fully recovered	partly recovered
Address book contacts	✓	
Calendar entries		✓
Emails and messaging		✓
Thumbnail pictures	✓	
Web browsing history		✓
WhatsApp history	✓	
WiFi credentials	✓	

- However, forensically more accurate analyses of Android memory structures are needed:
  - **Which data belongs to which process / App?**
  - **Can recovery be automated by *Volatility* plugins?**

Background: Dalvik VM

- Dalvik VM = Java Runtime Environment
- one DVM instance per Android App
- to be replaced by ART in future (Android 4.4)



Volatility Plugins for Linux

- Android is based on the Linux kernel
- each DVM instance is a Linux process
- hence, existing Volatility plugins for Linux memory images can be used:
  - `linux_ifconfig`
  - `linux_route_cache`
  - ...
  - `linux_pslist`
  - `linux_proc_maps`

    (acquires memory mappings of individual processes, i.e. for DVM instances / Apps)

Locate DVM Instances

- With existing Linux plugins, we can identify memory regions per process: `linux_proc_maps`
- Entry point to each DVM instance: `DvmGlobals`
- To analyze a specific App, it is essential to locate the offset to DvmGlobals in the process memory.
- Therefore, we provide a Volatility plugin: `dalvik_find_gdvm_offset`

dalvik_find_gdvm_offset

- Volatility plugin to locate DvmGlobals:

```
class dalvik_find_gdvm_offset(linux_common.AbstractLinuxCommand):
 def calculate(self):
   offset = 0x0
   mytask = None

   for task, vma in dalvik.get_data_section_libdvm(self._config):
     if not self._config.PID:
       if task.comm}%""% != %"%zygote%"%:
         continue
     mytask = task
     break

   proc_as = mytask.get_process_address_space()

   gDvm = None
   offset = vma.vm_start
   while offset < vma.vm_end:
     offset }= 1
     gDvm = obj.Object(%'%DvmGlobals%'%, vm = proc_as, offset = offset)
     if dalvik.isDvmGlobals(gDvm):
       yield (offset - vma.vm_start)
```

Generic Volatility Plugins

Altogether, we provide five Volatility plugins that can generically be applied to Android Apps:

- `dalvik_find_gdvm_offset`
  find the DVM instance of a process
- `dalvik_vms`
  find all DVM instances in memory
- `dalvik_loaded_classes`
  list all classes of a DVM instance
- `dalvik_class_information`
  list information of a specific class
- `dalvik_find_class_instance`
  find a specific class instance

Example Outputs

- find DVM instances:

```
$ ./vol.py [...] dalvik_vms -o HEX
PID   name               heapStartingSize heapMaximumSize
----- ---------------- ---------------- ---------------
 2508 zygote                     5242880       134217728
 2612 system_server              5242880       134217728
 2717 ndroid.systemui            5242880       134217728
 stackSize  tableSize  numDeadEntries  numEntries
 ---------- ---------- --------------- ---------------
      16384       4096               0            2507
      16384       8192               0            4123
      16384       8192               0            2787
```

- find loaded classes:

```
$ ./vol.py [...] dalvik_vloaded_classes -o HEX -p 4614
PID   Offset     Descriptor                         sourceFile
---- ---------- -------------------------------- ----------------
4614 0x40c378b8 Ljava/lang/Long;                 Long.java
4614 0x40deb6d0 Ljava/io/Writer;                 Writer.java
4614 0x414e2f60 Lde/homac/Mirrored/ArticlesList; ArticlesList.jav
```

- ...

Specific Volatility Plugins

- The generic plugins are designed to support data recovery from any Android App.
- Additionally, we provide four examples how to use these plugins in forensically interesting use cases:
  - `dalvik_app_calllog`
  - `dalvik_app_lastInput`
  - `dalvik_app_password`
  - `dalvik_app_pictures`

Case A) Call Log Recovery

- Goal: recover list of incoming/outgoing phone calls from confiscated phones
- Target process:

  `com.android.contacts`

- Target class:

  `PhoneClassDetails.java`

  One instance of this class is in memory per call log entry. Class members:

  - type (incoming, outgoing, missed)
  - duration, date and time
  - telephone number, contact name, photo

Case B) Last User Input Recovery

- Goal: retrieve the last given user input from a confiscated phone
- Target process:

  `com.android.inputmethod.latin`
- Target class:

  `RhichInputConnection`
- Target field:

  `mCommittedTextBeforeComposingText`

  (this field is like a keyboard buffer)

Case C)
User PIN Recovery

- Goal: recover the user PIN (if entered at least once before phone is confiscated)
- Target process:

  `keystore`

  (Note: this process is an Android system process and not running a DVM instance)
- Target location:
  - relative address inside `keystore`
  - +/- 200 kBytes at maximum

Case D)
Photo Metadata Recovery

- Goal: recover metadata like date, time and GPS coordinates from photo gallery
- Target process:

  `com.android.gallery3d`

- Target class:

  ```
  LocalAlbum
   └ LocalImage
  ```

  Class members:

  - name, size, date and time
  - GPS coordinates (if activated)

Volatility Plugins Availability

- GNU General Public License 2.0
- Link: https://www1.cs.fau.de/filepool/projects/android_volatility_plugins.zip

```
volatility/
volatility/plugins/
volatility/plugins/overlays/
volatility/plugins/overlays/linux/
volatility/plugins/overlays/linux/dalvik_vtypes.py
volatility/plugins/linux/
volatility/plugins/linux/dalvik_app_calllog.py
volatility/plugins/linux/dalvik_find_class_instance.py
volatility/plugins/linux/dalvik_app_password.py
volatility/plugins/linux/dalvik.py
volatility/plugins/linux/flags.py
volatility/plugins/linux/dalvik_app_pictures.py
volatility/plugins/linux/dalvik_loaded_classes.py
volatility/plugins/linux/dalvik_class_information.py
volatility/plugins/linux/dalvik_vms.py
volatility/plugins/linux/dalvik_app_lastInput.py
```

Anti-Forensics
Thwarting the Cold-Boot Attack

Anti-Forensics by Manufacturers

- Smartphone manufacturers could change their bootloader policy, such that:
  - *bootloaders cannot be unlocked* (like in iPhones and Windows Phones)
  - or *RAM is wiped* (not only disks) when bootloaders get unlocked

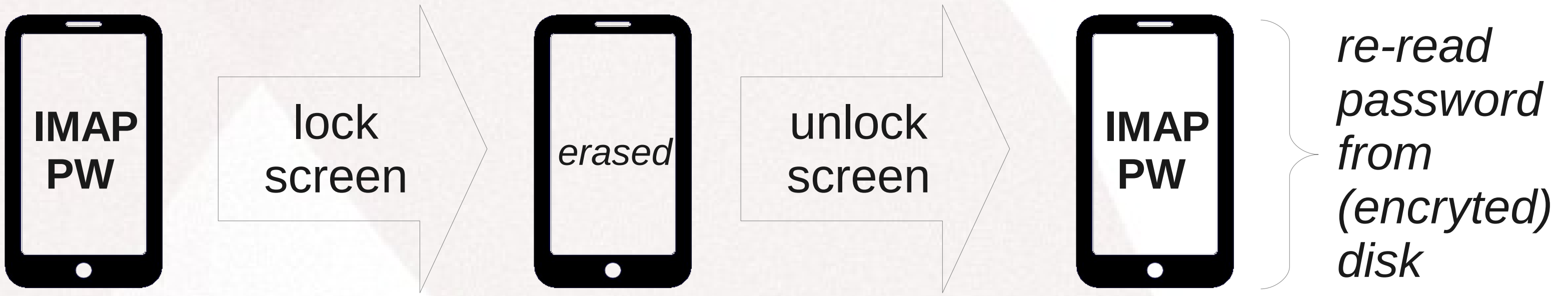
- However, this only raises the bar for forensic memory acquisition. The root problem, i.e., sensitive data in RAM, is not solved.

Anti-Forensics through Full Memory Encryption

- Obviously, full disk encryption (FDE) does not counteract cold-boot attacks on Android RAM.
- In analogy to FDE, *main memory must be encrypted*.
- However, due to performance and hardware constraints, only academic solutions exist:
  - *M. Henson and S. Taylor, “Beyond Full Disk Encryption: Protection on Security-Enhanced Commodity Processors,” Jun. 2013.*
  - *A. Wurstlein, “Design and Implementation of a Transparent Memory Encryption and Transformation System,” Aug. 2012.*

Anti-Forensics through Secure Deallocation

- Idea: Erase highly sensitive data from RAM on screen lock events (e.g., PINs and passwords).



- Problem: Dalvik VM does *not* enable the application level programmer to reliably erase data from RAM.
- Future Work: Patch the DVM to allow secure deallocation.

Conlusions

Conclusions

- ***Screen locks (e.g., PINs) and disk encryption are insufficient to protect sensitive data on smartphones today***

- “Smartphone Security Survey” by Ponemon / AVG (2011)
  - 89% use their smartphone for email
  - 66% keep sensitive business data on it
  - 34% use their smartphone for e-payment

- “Smartphones are “perfect targets” for cold boot attacks:
  - smartphones contain sensitive data
  - smartphones are more often lost than laptops
  - smartphones are usually switched on (but locked)

Thank You!

Questions?