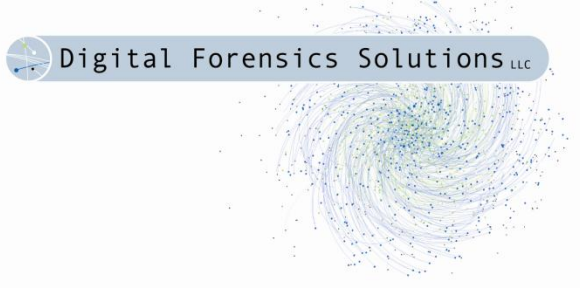


Android Mind Reading: Memory Acquisition and Analysis with LiME and Volatility

Joe Sylve

joe@digdeeply.com

@jtsylve

Digital Forensics Solutions, LLC

www.digitalforensicssolutions.com

About the Speaker

- Senior Security Researcher at Digital Forensics Solutions, LLC (New Orleans, La)
- GIAC Certified Forensic Analyst
- M.S. Computer Science
  - University of New Orleans

What We'll Cover

- Live Forensics
- Traditional Linux Memory Forensics Overview
- Problems with Android
- Acquisition Tools (LiME)
- Volatility
- Demo

What is Live Forensics?

- Traditional Forensics Deals with Non-Volatile Data
  - Hard Drives
  - Removable Media
  - Etc
- Live Forensics Deals with Volatile Data
  - RAM Mostly
  - Must be collected from a running machine
  - Not as much control over the environment

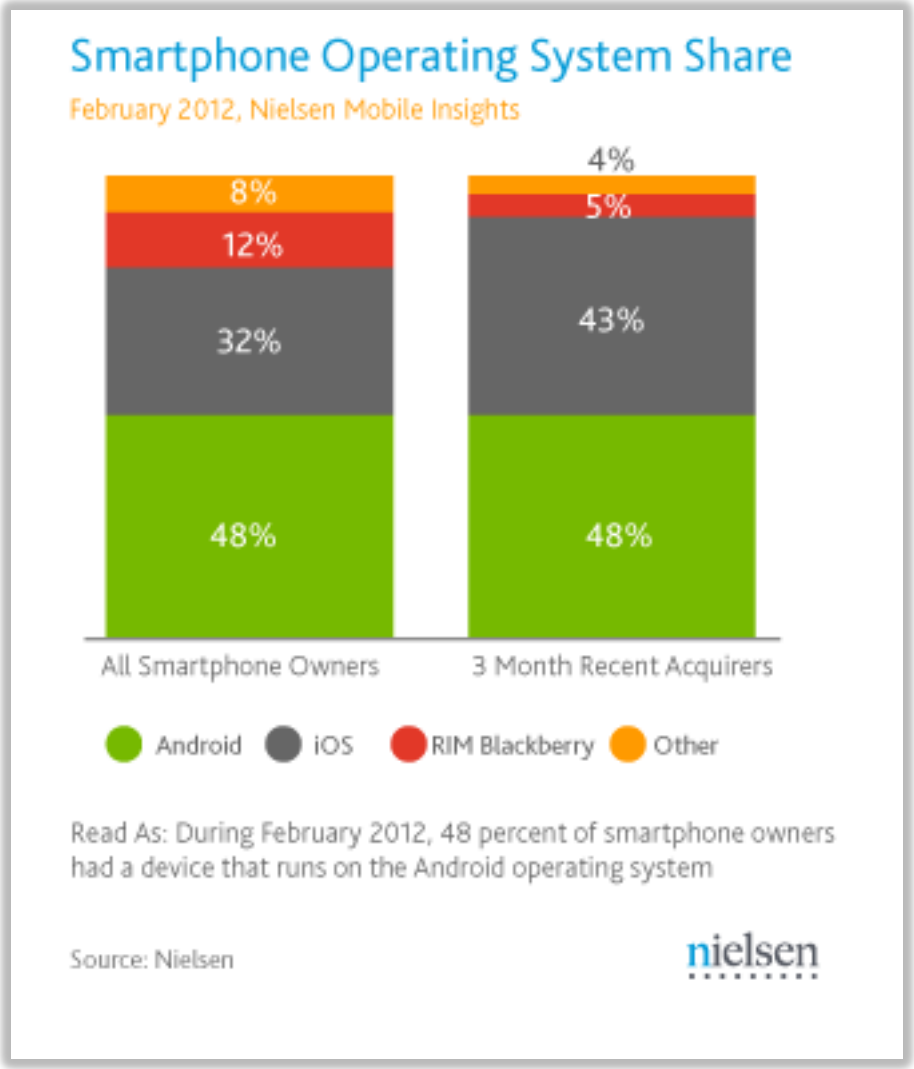
Why Live Forensics?

- RAM dump provides both structured and unstructured information
- Strings: application data, fragments of communications, encryption keys, etc.
- Kernel and application structures
- Processes, open files, network structures, etc.

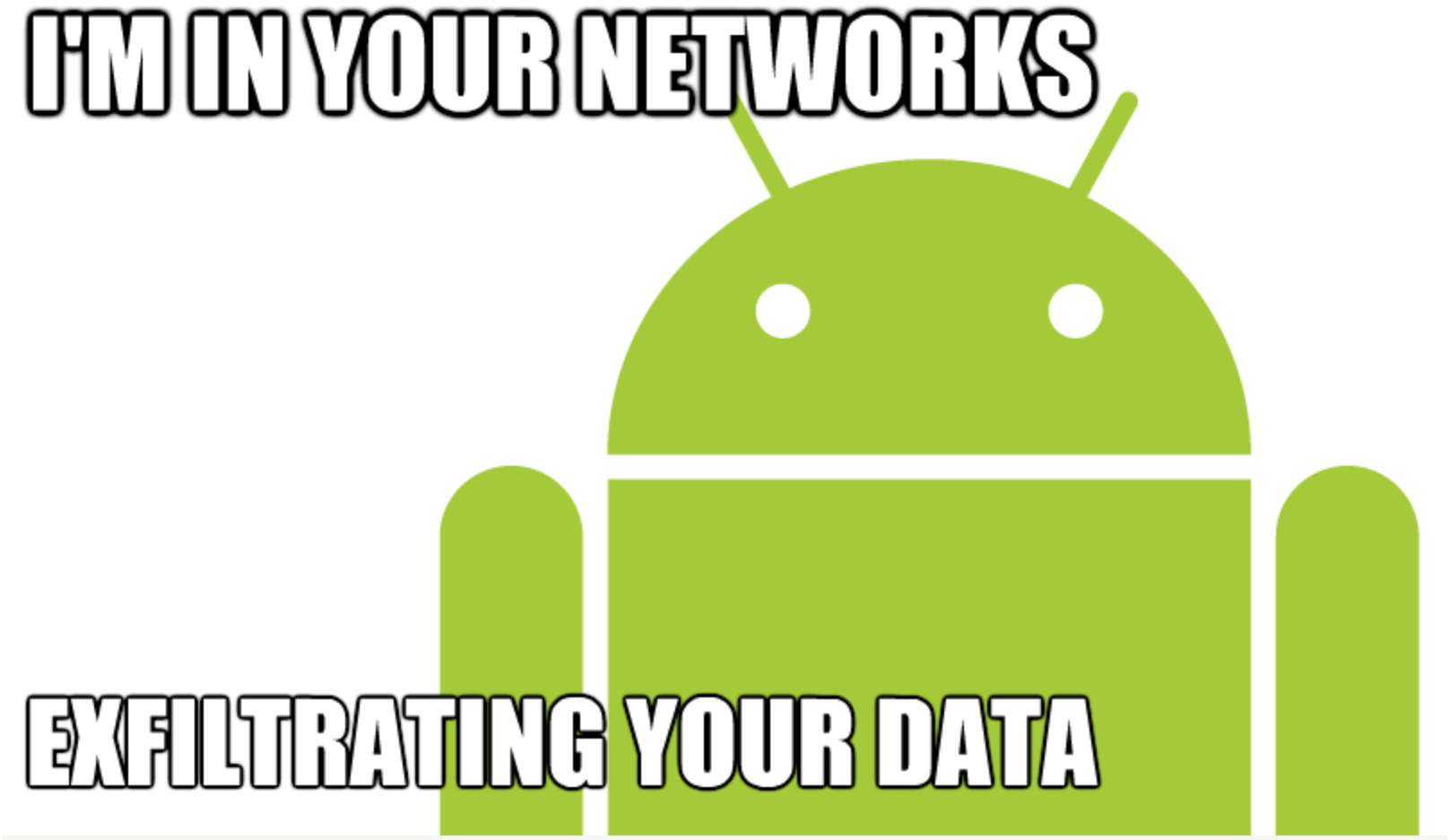
Why Live Forensics?

- Advanced Malware
- Encrypted or Temp File Systems
- Analysis
  - FatKit
  - Memparser
  - Volatility

Android


Smartphone Operating System Share
February 2012, Nielsen Mobile Insights
8%
12%
32%
48%
4%
5%
43%
48%
All Smartphone Owners
3 Month Recent Acquirers
Android
iOS
RIM Blackberry
Other
Read As: During February 2012, 48 percent of smartphone owners had a device that runs on the Android operating system
Source: Nielsen
nielsen

Not Just Phones


I'M IN YOUR NETWORKS
EXFILTRATING YOUR DATA

Acquisition

Traditional Memory Acquisition

- Hardware
  - JTAG
  - Firewire
  - Thunderbolt
  - Can of Compressed Air
- Software
  - Full Physical Memory
    - /dev/(k)mem
    - Fmem
    - Crash
  - Process Specific
    - Ptrace
    - Core dumps

Traditional Memory Acquisition (Android Edition)

- Hardware
  - JTAG (unlikely)
  - ~~Firewire~~
  - ~~Thunderbolt~~
  - ~~Can of Compressed Air~~
- Software
  - Full Physical Memory
    - ~~/dev/(k)mem~~
    - ~~Fmem~~
    - ~~Crash~~
  - Process Specific
    - Ptrace
    - Core dumps

Fmem Internals

1. Obtaining the starting offset specified by the read operation.
2. Checking that the page corresponding to this offset is physical RAM and not part of a hardware device's address space.
3. Obtaining a pointer to the physical page associated with the offset.
4. Writing the contents of the acquired page to the userland output buffer.

Fmem Internals

1. **Obtaining the starting offset specified by the read operation.**
2. Checking that the page corresponding to this offset is physical RAM and not part of a hardware device's address space.
3. Obtaining a pointer to the physical page associated with the offset.
4. Writing the contents of the acquired page to the userland output buffer.

/proc/iomem

 # cat /proc/iomem

02b00000-02efffff : msm_hdmi.0
03700000-039fffff : kgsl_phys_memory
03700000-039fffff : kgsl
03a00000-03a3ffff : ram_console
03b00000-03dfffff : msm_panel.0
20000000-2e7fffff : System RAM 
20028000-20428fff : Kernel text
2044a000-2058ca13 : Kernel data
30000000-3bffffff : System RAM 
a0000000-a001ffff : kgsl_reg_memory
a0000000-a001ffff : kgsl
a0200000-a0200fff : msm_serial_hs_bcm.0
a0300000-a0300fff : msm_sdcc.1
…

Problem 1: dd

- dd if=/dev/fmem of=ram.dd count=yyyy **skip=xxxx**
- lseek(unsigned int fd, **off_t offset**, unsigned int origin)
- vfs_llseek(struct file *file, **loff_t offset**, int origin)
- Original Offset: 0x80000000
- Signed Extension: 0xFFFFFFFF80000000

Problem 1:dd

- Not really Fmem's fault
- Problem is in implementation of Android's dd
- However, it would still be suboptimal if dd worked
  - dd performs a read operation for every block
  - Context Switches

Fmem Internals

1. Obtaining the starting offset specified by the read operation.
2. **Checking that the page corresponding to this offset is physical RAM and not part of a hardware device's address space.**
3. Obtaining a pointer to the physical page associated with the offset.
4. Writing the contents of the acquired page to the userland output buffer.

/proc/iomem

 # cat /proc/iomem

02b00000-02efffff : msm_hdmi.0
03700000-039fffff : kgsl_phys_memory
03700000-039fffff : kgsl
03a00000-03a3ffff : ram_console
03b00000-03dfffff : msm_panel.0
20000000-2e7fffff : System RAM 
20028000-20428fff : Kernel text
2044a000-2058ca13 : Kernel data
30000000-3bffffff : System RAM 
a0000000-a001ffff : kgsl_reg_memory
a0000000-a001ffff : kgsl
a0200000-a0200fff : msm_serial_hs_bcm.0
a0300000-a0300fff : msm_sdcc.1
…

Problem 2: page_is_ram

- http://lxr.linux.no/#linux+v3.0.4/kernel/resource.c#L363
- Missing in Linux kernel on ARM (Android)
- Essentially walks *iomem_resource* in the kernel to find pages in the physical address space that are RAM
- Not cool to walk across pages that aren't RAM (likely mapped to I/O devices, etc.)
- Can get the basic idea by looking at */proc/iomem*

LiME Forensics

- Linux Memory Extractor
  - Formerly DMD
- Loadable Kernel Module
- Dump Memory directly to the SD card or over the network
  - Network dump over adb (Android Debug Bridge)
- Minimizes interaction between userland and kernelland

Linux Memory Extractor (LiME)

1. Parsing the kernel's *iomem_resource* structure to learn the physical memory address ranges of system RAM.
2. Performing physical to virtual address translation for each page of memory.
3. Reading all pages in each range and writing them to either a file (typically on the device's SD card) or a TCP socket.

LiME 1.1 Arguments

- path
  - Either a filename to write on the local system (SD Card) or tcp:<port>
- format
  - raw
    - Simply concatenates all System RAM ranges
  - padded
    - Starting from physical address 0, pads all non-System RAM ranges with 0s
  - lime
    - Each range is prepended with a fixed-sized header which contains address space information
    - Volatility address space developed to support this format
- dio (optional)
  - 1 to enable Direct IO attempt (default), 0 to disable

LiME (TCP)

```
$ adb push lime-evo.ko /sdcard/lime.ko
$ adb forward tcp:4444 tcp:4444
$ adb shell
$ su
# insmod /sdcard/lime.ko
"path=tcp:4444 format=lime"
```

Then on host:

```
$ nc localhost 4444 > evo.dump
```

LiME (SD Card)

```
$ adb push lime-evo.ko /sdcard/lime.ko
$ adb shell
$ su
# insmod /sdcard/lime.ko
“path=/sdcard/dump.lime format=lime”
```

Forensics Note

- Writing to SD card requires “violating” a common forensic rule of thumb:
- **Order of Volatility**
  - **RAM → on-the-spot live forensics → non-volatile memory (hard drives, flash, etc.) → CDs, etc.**
- Acquire and preserve most volatile evidence first
- On Android, the only non-volatile removable storage that we can use to store memory dump is the SD card
- Commonly underneath the battery
- Removable of battery == power failure for device!
- Solution: Tether Android phone, USB mode, image SD, then dump memory to SD

DEMO

Please do what you must to appease the Live Demo Gods…

Testing for Soundness

1. Use emulator to get RAM snapshot
2. Use LiME to acquire RAM image
3. Compare (1) and (2) for identical pages

Method	*Total Number of Pages*	*Number of Identical Pages*	*Percentage of Identical Pages*
dmd (TCP)	131072	130365	99.46%
dmd (SD Card)	131072	129953	99.15%
fmem (SD Card)	131072	105080	80.17%

Not Just Android…

- LiME works on Linux too!

Analysis

- We've got the RAM dumps so now what?
- Volatility
  - https://www.volatilesystems.com/default/volatility
- Andrew Case (@attrc)
  - Worked on Linux port of Volatility
  - Worked on ARM port ☺
  - Wrote LiME address space into Volatility

Volatility

- The goal is to recreate the set of commands that would be run on a Linux system to investigate activity and possible compromise

Recovered Process Information

- Process listing (ps aux)
  - Command line arguments are retrieved from userland*
- Memory Maps (/proc/<pid>/maps)
  - Can also recover (to disk) specific address ranges*
- Open Files (/proc/<pid>/fd)

Networking Information

- Network interface information (ifconfig)
- Open and listening sockets (netstat)
- ARP tables (arp –a)
- Routing table (route –n)
- Routing cache (route –C)
- Queued Packets
- Netfilter NAT table (/proc/net/nf_conntrack)
  - Src/Dst IP, # of packets sent, and total bytes for each NAT'd connection

Misc. Information

- Kernel debug buffer (dmesg)
- Loaded kernel modules (lsmod)
- Mounted filesystems (mount, /proc/mounts)

Historical Information

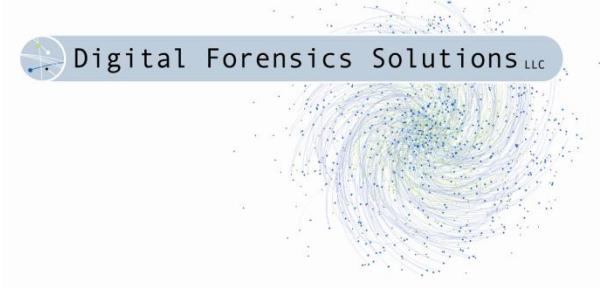
- kmem_cache
  - Provides a consistent and fast interface to allocate objects (C structures) of the same size
  - Keep freelists of previously allocated objects for fast allocation
- Walking the freelists provides an orderly method to recover previous structures

Historical Information

- Can recover a number of useful structures:
  - Processes
  - Memory Maps
  - Networking Information
- Two limitations:
  - The aggressiveness of the allocator (SLAB / SLUB) when removing freelists
  - Needed references being set to NULL or freed on deallocation

Other Cool Stuff

- See: Linux Memory Analysis with Volatility
  - 2011 Open Memory Forensics Workshop
  - Andrew Case
  - http://bit.ly/xVnwyP
- Rootkit detection
- Live CD Analysis
- Dalvik Analysis (coming)



DEMO 2

If the first demo didn’t work this is going to be a really short one…

Digital Forensics Solutions, LLC

- Registry Decoder
  - digitalforensicssolutions.com/registrydecoder/
- Scalpel
  - digitalforensicssolutions.com/Scalpel/
- LiME
  - digitalforensicssolutions.com/lime/

Digital Forensics Solutions, LLC

- DARPA Cyber Fast Track Awards (In Progress)
  - "Forensic Capabilities for Embedded File Systems"
  - "Automatically Generated Regular Expression-based Signatures for File Carving"
- Registry Decoder Enhancements (In Progress)
- Registry Decoder Training Workshop
  - August 16th
  - September 20th

Questions?

- Joe Sylve
  - joe@digdeeply.com
  - @jtsylve
- “Acquisition and analysis of volatile memory from android devices”
  - Digital Investigation (2012)
  - http://bit.ly/xFEPoj
- Digital Forensics Solutions, LLC
  - www.digitalforensicssolutions.com
  - dfsforensics.blogspot.com
  - @dfsforensics