Datalore: Android Memory Analysis

Joe Sylve
joe.sylve@gmail.com
@jtsylve
About the Speaker

- Practitioner
  - Digital Forensics Solutions, LLC (New Orleans, LA)

- Researcher / Co-Founder
  - 504ENSICS Labs (New Orleans, LA)

- GIAC Certified Forensics Analyst

- M.S. Computer Science
  - University of New Orleans
Acquisition
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
# cat /proc/iomem
02b00000-02effffff : msm_hdmi.o
03700000-039fffff : kgsl_phys_memory
03700000-039fffff : kgsl
03a00000-03a3fffff : ram_console
03b00000-03dffff : msm_panel.o
20000000-2e7fffff : System RAM
   20028000-20428fff : Kernel text
   2044a000-2058ca13 : Kernel data
30000000-3bffff : System RAM
a0000000-a001fffff : kgsl_reg_memory
a0000000-a001fffff : kgsl
a0200000-a0200fff : msm_serial_hs_bcm.o
a0300000-a0300fff : msm_sdcc.1
...

/proc/iomem
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.lime
LiME (SD Card)

$ adb push lime-evo.ko /sdcard/lime.ko

$ adb shell

$ su

# insmod /sdcard/lime.ko

"path=/sdcard/dump.lime format=lime"
LiME Forensics

- Free
- Open Source (GPL)
- http://code.google.com/p/lime-forensics/

- Soon
  - Video Card RAM
  - Registers
  - “Live” version
Analysis
ARM Address Space

- Official in Volatility 2.3
- Supports Fine and Course paging
  - 64K “large pages”
  - 4K “small pages”
  - 1K “tiny pages”
- No support for “Superpages”
  - Please let me know if any processors actually use this
- Windows 8 ARM???
  - TBD
Android Profiles

- Works the same way as Linux Profiles (Mostly)
- Download Kernel Source from Vendor
- Kernel Config
- Cross-Compile
- ZIP(Dwarfdump + System.map)

- Not quite as easy as Linux, because you can’t just type “make” and go...
ARM Compatible Plugins

- `linux_arp` Print the ARP table
- `linux_check_afinfo` Verifies the operation function pointers of network protocols
- `linux_dentry_cache` Gather files from the dentry cache
- `linux_dmesg` Gather dmesg buffer
- `linux_dump_map` Writes selected memory mappings to disk
- `linux_find_file` Recovers tmpfs filesystems from memory
- `linux_ifconfig` Gathers active interfaces
- `linux_iomem` Provides output similar to `/proc/iomem`
- `linux_lsmod` Gather loaded kernel modules
- `linux_lsof` Lists open files
- `linux_memmap` Dumps the memory map for linux tasks
- `linux_mount` Gather mounted fs/devices
ARM Compatible Plugins

- **linux_mount_cache**  
  Gather mounted fs/devices from kmem_cache
- **linux_pidhashtable**  
  Enumerates processes through the PID hash table
- **linux_pkt_queues**  
  Writes per-process packet queues out to disk
- **linux_proc_maps**  
  Gathers process maps for linux
- **linux_psaux**  
  Gathers processes along with full command line and start time
- **linux_pslist**  
  Gather active tasks by walking the task_struct->task list
- **linux_pslist_cache**  
  Gather tasks from the kmem_cache
- **linux_pstree**  
  Shows the parent/child relationship between processes
- **linux_psxview**  
  Find hidden processes with various process listings
- **linux_route_cache**  
  Recovers the routing cache from memory
- **linux_sk_buff_cache**  
  Recovers packets from the sk_buff kmem_cache
- **linux_slabinfo**  
  Mimics /proc/slabinfo on a running machine
- **linux_tmpfs**  
  Recovers tmpfs filesystems from memory
- **linux_vma_cache**  
  Gather VMAs from the vm_area_struct cache
Example 1

linux_tmpfs
linux_tmpfs

- Parse mount table
  - mount_hashtable
- Look for tmpfs superblocks
- Walks the root dentry structs
- Dumps file and directory contents to disk
Some Android phones have a tmpfs mount called `/app-cache`

The stock Android browser uses this for it’s Webview Cache

- `/app-cache/com.android.browser/cache/webviewCache`
- Never written to disk
$ python vol.py --profile=LinuxEvo4GARM -f ../Evo4G3.lime
linux_tmpfs -L
Volatile Systems Volatility Framework 2.3_alpha
1 -> /mnt/sdcard/.android_secure
2 -> /dev
3 -> /app-cache
4 -> /mnt/obb
5 -> /mnt/asec
mkdir tmpfs-out
python vol.py --profile=LinuxEvo4Gx86 -f ../Evo4G3.lime
linux_tmpfs -S 3 -D tmpfs-out
linux_tmpfs Example
Example 2

Application Permissions
Andrew explained this
Android Security Model

- Each application digitally signed by author
- Applications assigned uid at install time
  - Usually unique per application
  - Author can request same uid to be assigned for any of his/her applications
- Linux User Access Model keeps data private by default
- Concept of “permissions”
  - Access sensitive APIs
  - Share data and functionality
  - Can be user-created
<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;normal&quot;</td>
<td>The default value. A lower-risk permission that gives requesting applications access to isolated application-level features, with minimal risk to other applications, the system, or the user. The system automatically grants this type of permission to a requesting application at installation, without asking for the user's explicit approval (though the user always has the option to review these permissions before installing).</td>
</tr>
<tr>
<td>&quot;dangerous&quot;</td>
<td>A higher-risk permission that would give a requesting application access to private user data or control over the device that can negatively impact the user. Because this type of permission introduces potential risk, the system may not automatically grant it to the requesting application. For example, any dangerous permissions requested by an application may be displayed to the user and require confirmation before proceeding, or some other approach may be taken to avoid the user automatically allowing the use of such facilities.</td>
</tr>
<tr>
<td>&quot;signature&quot;</td>
<td>A permission that the system grants only if the requesting application is signed with the same certificate as the application that declared the permission. If the certificates match, the system automatically grants the permission without notifying the user or asking for the user's explicit approval.</td>
</tr>
<tr>
<td>&quot;signatureOrSystem&quot;</td>
<td>A permission that the system grants only to applications that are in the Android system image or that are signed with the same certificates as those in the system image. Please avoid using this option, as the signature protection level should be sufficient for most needs and works regardless of exactly where applications are installed. The signatureOrSystem permission is used for certain special situations where multiple vendors have applications built into a system image and need to share specific features explicitly because they are being built together.</td>
</tr>
</tbody>
</table>
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
<packages>

<permissions>

<item name="android.permission.RECEIVE_SMS" package="android" protection="1"/>
<item name="android.permission.CALL_PHONE" package="android" protection="1"/>
<item name="android.permission.BACKUP" package="android" protection="3"/>
<item name="android.permission.READ_CALENDAR" package="android" protection="1"/>
<item name="android.permission.RECEIVE_BOOT_COMPLETED" package="android"/>
<item name="android.permission.SET_TIME" package="android" protection="3"/>
<item name="android.permission.ACCESS_UPLOAD_DATA" package="com.htc.providers.uploads" protection="2"/>

</permissions>
</packages>
<package name= com.weather.Weather codePath= /data/app/com.weather.Weather-2.apk userId= 10058 />
<sigs count= 1 >
<cert index= 1 key= ...
</cert>
</sigs>
<perms>
<item name= android.permission.SET_WALLPAPER />
<item name= android.permission.SEND_SMS />
<item name= android.permission.WRITE_EXTERNAL_STORAGE />
<item name= android.permission.ACCESS_WIFI_STATE />
<item name= android.permission.ACCESS_COARSE_LOCATION />
<item name= android.permission.CALL_PHONE />
<item name= android.permission.WRITE_CALENDAR />
<item name= android.permission.READ_CALENDAR />
<item name= android.permission.CAMERA />
<item name= android.permission.INTERNET />
<item name= android.permission.ACCESS_FINE_LOCATION />
<item name= android.permission.VIBRATE />
<item name= android.permission.ACCESS_NETWORK_STATE />
<item name= android.permission.RECORD_AUDIO />
</perms>
</package>
linux_find_file Example

```
linux_find_file -F /data/system/packages.xml

Inode Number  Inode
-------------------
1019        0xd3aad948
```

```
linux_find_file -i 0xd3aad948 -O packages.xml
```
Example 3

Screen Lock Password
Screen Lock
Screen Lock
Screen Lock

- Takes sequence as a string
  - “6305741238”
- Hash It (SHA-1)
  - SHA-1(“6305741238”)
  - No Salt...
- Compare to hash in file
  - /system/gesture.key
android_screenlock

- Uses `linux_find_file` to pull hash from `/system/gesture.key`
- Looks hash -> pattern mapping in database

```
python vol.py ... android_screenlock

[8, 4, 0, 1, 2, 6]
```
Screen Lock (PIN or Key)

- SHA1 hash stored in /system/password.key
- Random salt stored in database
  - /data/data/com.android.providers.settings/databases/settings.db
- SHA1(SHA1(PIN + SALT) + MD5(PIN))
  - ... for some reason
- Salted, so no pre-computed tables
- Still easy enough to brute-force
Example 4

yaffs2
YAFFS/YAFFS2 enabled devices export a symbol called `yaffs_dev_list`

**yaffs_DeviceStruct**
- Tons of information about yaffs devices
- Block Sizes
- Group Info
- Object Lists (Allocated and Free)
- Stats
android_yaffs_info

- python vol.py --profile=LinuxEvo4Gx86 -f ..;/Evo4G2.lime android_yaffs_info
  Device 1 "userdata"
  startBlock.........0
  endBlock..........3420
  totalBytesPerChunk.2048
  nDataBytesPerChunk.2048
  chunkGroupBits....0
  chunkGroupSize.....1
  nErasedBlocks......2583
  nReservedBlocks....5
  blocksInCheckpoint.0
  nTnodesCreated.....2100
  n_FreeTnodes.........6
  n_ObjectsCreated....1000
  n_FreeObjects......59
android_yaffs_info

nFreeChunks........ 193765
nPageWrites........ 60896
nPageReads......... 21507
nBlockErasures..... 159
nGCCopies.......... 0
garbageCollections. 0
passiveGCs......... 0
nRetriedWrites..... 0
nShortOpCaches..... 10
nRetireBlocks...... 0
android_yaffs_info

eccFixed............ 0
eccUnfixed......... 0
tagsEccFixed....... 0
tagsEccUnfixed..... 0
cacheHits.......... 51881
nDeletedFiles...... 0
nUnlinkedFiles..... 3459
nBackgroundDeletions 0
useNANDECC........ 1
isYaffs2.......... 1
inbandTags........ 0
Questions?

- Joe Sylve
- 504ENSICS Labs
- Digital Forensics Solutions, LLC
- joe.sylve@gmail.com
- @jtsylve