Universidad de Buenos Aires
Facultades de Ciencias Económicas,
Ciencias Exactas y Naturales e Ingeniería

Maestría en Seguridad Informática

Tesis

Tema

Análisis forense de dispositivos móviles con sistema operativo Android

Título

Desarmando al Androide

Autor: Maximiliano Bendinelli
Director de Tesis: Hugo Pagola

Año 2013
Cohorte 2009

Tabla de Contenidos

Resumen

La penetración del sistema operativo Android en los usuarios de dispositivos móviles en los últimos años ha sido de un crecimiento exponencial, permitiendo una enorme interacción entre las personas mediante la utilización de distintas tecnologías y aplicaciones que se ofrecen hoy por internet o por redes de telefonía celular. Entre las más populares existen un gran número de mensajeros que han reemplazado a los SMS tradicionales por su bajo costo (o nulo), aquellas aplicaciones que permiten realizar llamadas sin costo como Skype, Line o Viber por mencionar sólo algunas y también la integración con redes sociales.

La creciente penetración en el mercado dispositivos móviles con sistema operativo Android y la falta de herramientas que sean de fácil acceso y que permitan realizar un análisis de estos dispositivos motivaron la realización de este trabajo.

Se comenzara en el primer capítulo conociendo su arquitectura , el kernel con el cual trabaja, su proceso de inicialización , se explicara que es el bootloader entre otras cuestiones técnicas que ayudaran a comprender el funcionamiento de Android por dentro y ayudaran a familiarizar al lector con distintos conceptos del funcionamiento de Android que serán necesarios para comprender el porqué del procedimiento adoptado para avanzar con la investigación que se llevó a cabo en este trabajo de tesis.

En siguiente capítulo se explicara cómo es posible realizar una extracción forense de un dispositivo con Android, se analizaran el uso de la herramientas y se demostrara cono es posible crear un firmware desde cero para poder realizar una copia bit a bit de la memoria del equipo en cuestión y poder extraer su contenido sin contaminar al evidencia allí contenida para un posterior análisis forense.

A lo largo de este trabajo también se demostrara cómo es posible realizar un análisis forense con la copia bit a bit de la memoria que fue extraída con la técnica desarrollada en el capítulo dos (2) y se podrá observar que de dicho análisis se puede recuperar información eliminada, historial de llamados, SMS, datos de geolocalización , historiales de chat y

fotos por nombrar son solo algunos de los muchos huellas que dejan las aplicaciones por su utilización.

Como conclusión se intentara realizar una virtualización de un dispositivo con la imagen forense extraída del dispositivo por medio de un emulador del procesador ARM7 incluido dentro de las herramientas de desarrollo de Andorid.

Palabras Clave

Evidencia Digital – Informática Forense – Dispositivos Móviles – Android

Introducción

Los avances tecnológicos en telefonía celular (o móvil) y el crecimiento de Internet revolucionaron las comunicaciones de manera tal que se han convertido en dispositivos imprescindibles para la vida de muchas personas. Los dispositivos móviles se han trasformado de simples teléfonos a dispositivos de comunicación con más capacidades como las de una computadora. Éstos son equipos que poseen las capacidades de un teléfono y una computadora en un mismo aparato.

Estas innovaciones llevaron al desarrollo de teléfonos “inteligentes” con sistemas operativos capaces de realizar diferentes tareas. Una de estas plataformas es el sistema operativo Android: una plataforma de código abierto (open source) para dispositivos móviles desarrollada por Open Handset Alliance [1]. La Open Handset Alliance (OHA) es una alianza comercial de 84 compañías que se dedica a desarrollar estándares abiertos para dispositivos móviles. Algunos de sus miembros son Google, HTC, Dell, Intel, Motorola, Qualcomm, Texas Instruments, Samsung, LG, T-Mobile, Nvidia y Wind River Systems. La OHA se fundó el 5 de noviembre de 2007, liderada por Google con otros 34 miembros entre los que se incluían fabricantes de dispositivos móviles, desarrolladores de aplicaciones, algunos operadores de comunicaciones y fabricantes de chips.

En los últimos tiempos se ha generado una interacción de las personas con los dispositivos móviles a niveles impensados, logrando que los usuarios tomen fotografías, filmen videos, realicen intercambio de mensajes (o chat), hagan llamadas telefónicas, utilicen un GPS y puedan geolocalizarse y/o compartir su ubicación o llegar a algún destino por medio de aplicaciones para este fin.. Por esto es que los dispositivos móviles se han transformado en equipos casi indispensables para la vida cotidiana de quienes son usuarios y los ha llevados a estar “hiperconectados”.

A partir de este acceso y conexión continua, el análisis de estos dispositivos comienza a ser clave en algunas situaciones como resolver

litigios o investigar un acto criminal. Por esta razón, el trabajo se concentra en analizar e investigar procedimientos que respeten las mejores prácticas para la preservación de la evidencia contenida en estos dispositivos, sin contaminarla.

Recientes estudios realizados por Lockout [2] muestran que al menos el 42% de los dispositivos con Android tenían corriendo alguna clase de malware que le realizaba gastos en la facturación de los usuarios de los dispositivos por medio del envío de SMS de alto costo, generación de *"clicks"* en banners publicitarios, fraude por llamados, son solo algunos de los propósitos de los malwares/virus que infectan a los equipo.

De esta manera, el análisis forense de dispositivos con Android como parte de la informática toma cada vez más importancia y ofrece, al mismo tiempo, muchos desafíos.

En este contexto, el objetivo del siguiente trabajo es realizar una extracción y análisis de información digital con bajo costo y sin alterar la evidencia. La motivación se encuentra debido a las escasas herramientas que le permiten a un investigador poder realizar una copia bit a bit del contenido de la memoria de estos dispositivos para su posterior análisis respetando las mejores prácticas para la preservación de la evidencia.

El crecimiento de dispositivos móviles con Android, su uso para diferentes actividades o el desconocimiento de estos dispositivos por parte de los delincuentes presenta un reto en relación a cómo hacer efectiva la extracción y análisis de datos con propósitos forenses. Para poder realizarla, es importante un buen entendimiento de la plataforma y de las herramientas

A continuación se expondrá un recorrido posible.

Capítulo 1: Android

Android es un sistema operativo basado en Linux y de código abierto , para dispositivos móviles, que inicialmente se desarrolló para arquitectura ARM pero que con el tiempo fue portado a x86. Comenzó a desarrollarse por la *Open Handset Alliance (OHA)* [1] en el año 2005 hasta que fue adquirido por Google en el año 2007. Es esta empresa la que creó el primer dispositivo con Android que recibió el nombre de G1 y al ser de código abierto fue adoptado por una gran cantidad de fabricantes que hoy lideran el mercado. Debido a que cualquier fabricante puede adaptar Android para sus dispositivos, la penetración que tiene en el mercado es enorme y continúa creciendo convirtiéndose en el sistema operativo para dispositivos móviles más utilizado con más de 900 millones de aparatos activos

Arquitectura

Una de las mejores características de Android, es que está diseñado para funcionar en una gran variedad de hardware. Esto se debe a su kernel de Linux el cual ofrece compatibilidad con una amplia variedad de hardware. Por esta razón, muchos fabricantes lo adoptan porque pueden adaptarlo con facilidad al hardware que cada uno desarrolla. Este beneficio del sistema operativo representa un problema para el análisis forense porque la gran diversidad de fabricantes con su respectivo hardware y sus constantes cambios generan modificaciones continuas sobre las distintas plataformas y versiones de Android, aunque existen características que se mantienen La arquitectura está formada por cuatro (4) capas en donde la primera está basada en un Kernel de Linux como se muestra en la figura que está a continuación:

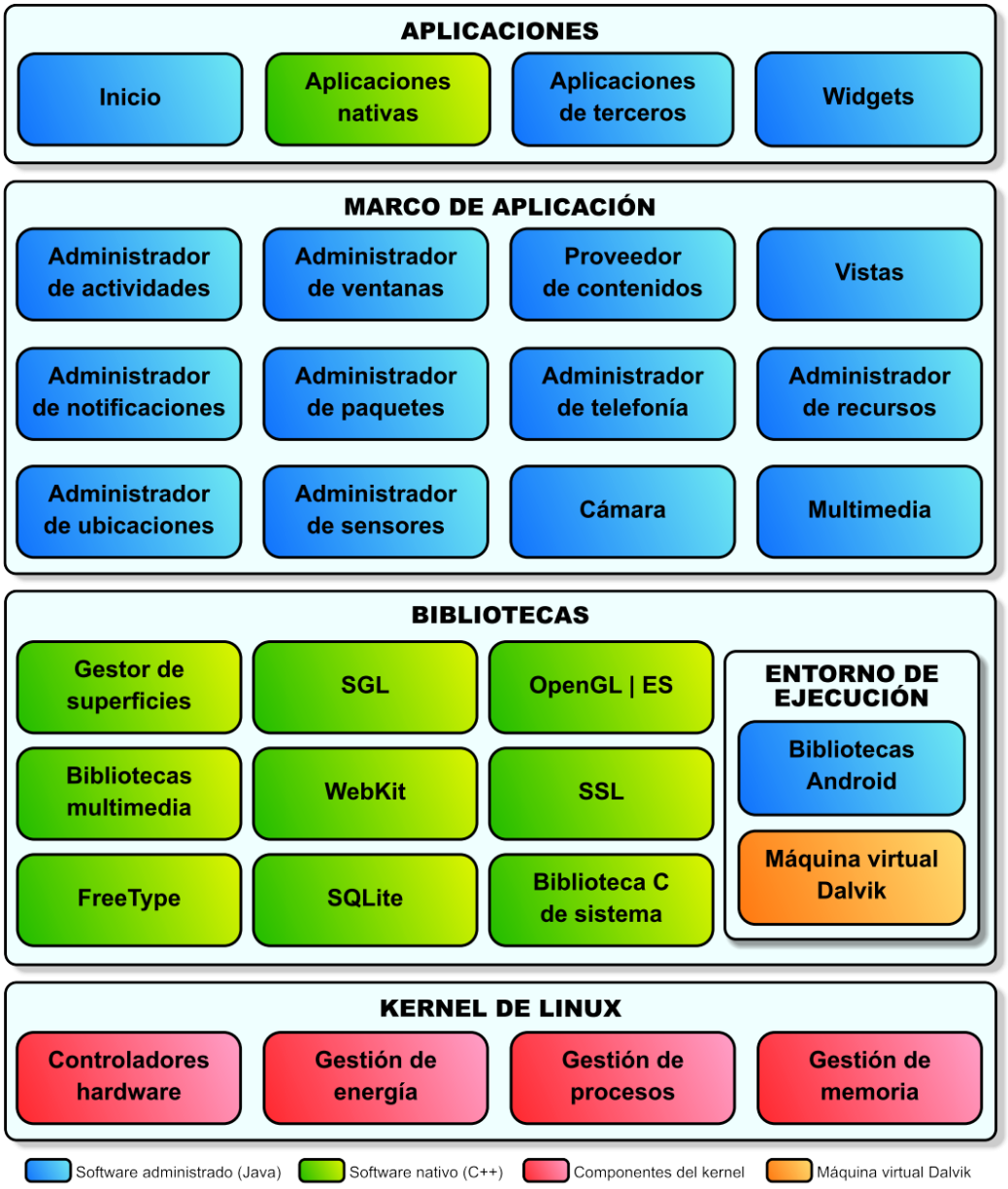


Figura 1.1 Arquitectura de Android

Kernel Linux

El núcleo de Android está formado por el sistema operativo Linux con un Kernel versión 2.6, similar al que puede incluir cualquier distribución de Linux- como Ubuntu- solo que adaptado a las características del hardware en el que se ejecutará Android, es decir, para dispositivos móviles. Esta capa proporciona servicios como la seguridad, el manejo de la memoria, el multiproceso, la pila de protocolos y el soporte de drivers para dispositivos.

Además, actúa como capa de abstracción entre el hardware y el resto de la pila. Por lo tanto, es la única que es dependiente del hardware.

Librerías

La siguiente capa que se sitúa justo sobre el kernel la componen las bibliotecas nativas de Android, también llamadas librería. Incluye un conjunto de librerías en C/C++ usadas en varios componentes de Android, están compiladas en código nativo del procesador y muchas utilizan proyectos de código abierto. Normalmente están hechas por los fabricantes, quienes también se encarga de instalarlas en los dispositivos. El objetivo de las librerías es proporcionar funcionalidad a las aplicaciones para tareas que se repiten con frecuencia. Entre las librerías incluidas habitualmente se encuentran OpenGL, Bibliotecas multimedia, Webkit, SSL, FreeType, SQLite, entre otras.

Runtime de Android

Como se observa en la figura 1.1, el entorno de ejecución de Android no se considera una capa en sí mismo, dado que también está formado por librerías. Aquí se encuentran las librerías con las funcionalidades habituales de Java así como otras específicas de Android. Está basado en el concepto de máquina virtual utilizado en Java. El componente principal del entorno de ejecución de Android es la máquina virtual denominada **Dalvik.**

Las aplicaciones se codifican en Java y son compiladas en un formato específico para que esta máquina virtual las ejecute. La ventaja es que las aplicaciones se compilan una única vez y de esta forma estarán listas para distribuirse y podrán ser ejecutadas en cualquier dispositivo con Android que disponga de la versión mínima del sistema operativo que requiera la aplicación.

Capa de Aplicaciones

Está formada por todas las clases y servicios que utilizan directamente las aplicaciones para realizar sus funciones. La mayoría de los módulos de esta capa son librerías Java que asienten a los recursos de las capas anteriores a través de la máquina virtual Dalvik. Algunos de los componentes son:

- Views: encargada de la parte visual de los componentes.
- Resource Manager: proporciona acceso a recursos que no son en código.
- Activity Manager: manipula el ciclo de vida de las aplicaciones y proporciona un sistema de navegación entre estas.
- Notification Manager: permite a las aplicaciones mostrar alertas personalizadas en la barra de estado.
- Content Providers: es el mecanismo desarrollado para acceder a datos de otras aplicaciones.
- Windows Manager: es el encargado de organizar lo que se mostrará en pantalla. En pocas palabras crea las superficies en la pantalla que a continuación pasarán a ser ocupadas por las actividades.
- Content Provider: es la librería que crea una capa que encapsula los datos que se compartirán entre aplicaciones para controlar cómo acceder a la información.
- Views:en Android, las vistas son los elementos que ayudarán a construir las interfaces de usuario: botones, cuadros de texto, listas y hasta elementos más avanzados como un navegador web o un visor de Google Maps.
- Notification Manager: es donde se encuentran los servicios que notifican al usuario cuando algo requiera su atención y lo realiza mostrando alertas en la barra de estado.
- Package Manager: esta biblioteca permite obtener información sobre los paquetes instalados en el dispositivo Android, así como también de gestionar la instalación de nuevos paquetes.

- Telephony Manager: por medio de esta librería es posible realizar llamadas telefónicas o intercambiar SMS/MMS.
- Resource Manager: es la que permite manejar todos los elementos que forman parte de la aplicación y que están fuera del código, es decir, cadenas de texto traducidas a diferentes idiomas, imágenes, sonidos o layouts.
- Location Manager: es por medio de esta librería con la cual el dispositivo puede determinar la posición geográfica mediante GPS o redes disponibles y también permite trabajar con mapas.

Inicialización del dispositivo

Para hacer un análisis forense es indispensable entender cómo inicializa el dispositivo móvil con Android para lo cual se demostrará el proceso de inicialización [3]. Como se mencionó con anterioridad Android está basado en un Kernel de Linux muy similar al que utiliza una computadora de escritorio, pero hay que tener en cuenta que el procesador que utilizan las gran mayoría de los dispositivos móviles es ARM y el hardware es muy distinto también, por lo tanto el Kernel está configurado y adaptado para el hardware del dispositivo.

A continuación se describirá la secuencia de inicialización de un dispositivo con Android.

Boot ROM

Cuando se inicializa el dispositivo y arranca el procesador, este tiene predefinida la dirección de memoria en donde se encuentra el Bootloader y procederá a cargar dicho programa en la RAM del dispositivo para comenzar su ejecución.

Bootloader

Es el primer programa que corre cuando se enciende un dispositivo móvil y es el encargado de gestionar el arranque del sistema. También se asegura de que todos los componentes de hardware estén en correcto funcionamiento y en ese caso se encargará de hacer iniciar Android en el caso de un inicio normal o en el caso de que sea solicitado por el usuario podrá iniciar con Recovery, Fastboot u otros.

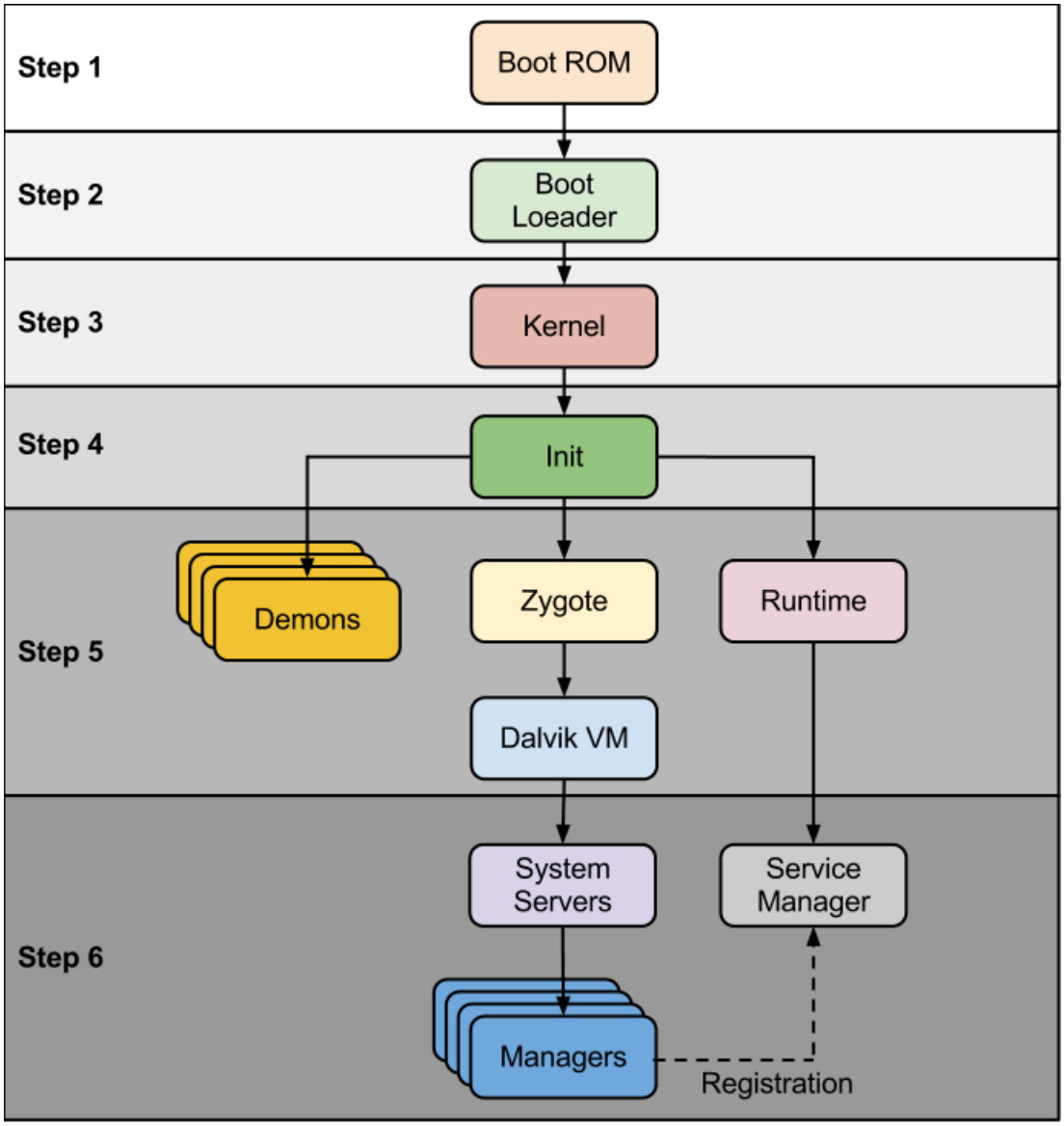


Figura 1.2 Proceso de Inicialización de Android [4]

Kernel

El Kernel de Android inicializa de igual manera que cualquier computadora de escritorio con Linux: montará los filesystem necesarios, el cache, la memoria protegida, cargará los drivers y cuando termine todas estas tareas buscará entre los archivos del sistema el programa “init” el cual será el primero en ejecutar del sistema operativo que dará inicio a Android.

El proceso Init

Es el primero de todos los que corre el sistema operativo de base, se puede decir que es el proceso raíz. Se encuentra en la base del system “/” proceso encargado de montar los principales filesystems del sistema operativo para luego dar lugar a la ejecución del archivo “init.rc” el cual se encargará de cuatro clases bien definidas de operaciones: acciones, servicios, comandos y opciones, que servirán de base para el funcionamiento de los siguientes procesos en jerarquía de Andriod.

Al realizar todas estas tareas lo primero que se podrá visualizar en el teléfono es el logo de Android o su correspondiente logo de inicialización.

Zygote y Dalvik

El servicio de Zygote es inicializado desde Init y es el encargado de precargar e inicializar las librerías de clases del núcleo de Android y es la base sobre la cual se inicializan las máquinas virtuales Dalvik.

Dalvik es la máquina virtual que se generará para cada proceso y tiene un funcionamiento muy similar al de las VM de Java.

Servicios del Sistema o Servicios

Una vez que se haya completado la etapa anterior, Zygote se encargará de inicializar los servicios de sistema. Entre los principales servicios se encuentran: telefonía, teclado, batería, alarmas, sensores, administrador de ventanas, entre otros.

Boot completo

Este es el último paso en la etapa de inicialización, cuando todas las etapas concluyeron se dispara una acción de BROADCAST denominada "ACTION_BOOT_COMPLETED" la cual indica la finalización del proceso.

Particionamiento de la memoria de Android

El particionamiento de la memoria de Android comúnmente se encuentra dividido en:

```
    512     256K bootloader      u-boot.bin
   2048       8M recovery        recovery.img
  18432       8M boot            boot.img
  34816     512M system          system.img
1083392     256M cache           cache.img
1607680     512M userdata        userdata.img
```

Figura 1.3 Tabla de particiones de Android

Como se observa en la tabla anterior se distinguen seis particiones: bootloader, recovery, boot, system, cache y userdata. Dependiendo del fabricante y el modelo del equipo esta estructura puede tener más particiones pero es la estructura básica sobre la cual se trabajará para el caso de estudio aquí planteado.

- Bootloader como bien su nombre lo indica tiene dentro de ella los programas necesarios para iniciar el bootloader que será el primer programa que cargará el teléfono apenas de inicializa.
- Recovery: es una partición de recuperación en la cual desde el bootloader puede elegirse para iniciar el dispositivo con el fin de realizar tareas de mantenimiento sobre el equipo. Al igual que la partición boot contiene un Kernel y un Ramdisk que actuará de manera distinta al ramdisk de la partición boot ya que sus funciones son limitadas. Cuando se inicia esta partición se puede visualizar una pantalla o consola que

permitirá al usuario realizar alguna actualización sobre el dispositivo o borrar el contenido de las particiones /data o /cache.

- Boot: esta partición contiene el kernel y el ramdisk de Android y es la que utiliza el arranque del teléfono cuando es inicializado en un modo "normal".
- System: dentro de esta partición se encuentra el framework de Android, o sea su entorno gráfico y también se están preinstaladas las aplicaciones del sistema.
- Cache: es la partición donde Android almacena información o datos que el usuario utiliza con más frecuencia, el fin de esta partición es agilizar la carga de aquellos a los que más se accede.
- Userdata: es en esta partición donde se almacenan todos los datos relacionados al usuario, es decir, todo aquello que es creado o modificado por él queda almacenado en esta partición:aplicaciones, contactos, información de llamadas (realizadas, perdidas y recibidas), mensajes, correos electrónicos, fondos de pantalla, fotos, cuentas de aplicaciones como Facebook, Google, Twiteer , etc.

Este último, desde el punto de vista forense, es el más rico en información y aquel desde donde podrá surgir toda la actividad del usuario con el dispositivo móvil.

Filesystem

Por medio de distintos Filesystem, Android organizará la información que contenga los dispositivos. Al estar basado en Linux soporta varios tipos de filesystems como EXT3, FAT, YAFFS2 y EXT4. El mayor reto desde el punto de vista forense son los Filesystems con YAFFS2 ya que no todas las herramientas forenses soportan este tipo de Filesystem. Lo positivo, es que con la evolución de Android cada vez son más los dispositivos que soportan EXT4 que sí está soportado por la mayoría de las herramientas.

Dentro de la estructura de Filesystems de Android se pueden distinguir las diferentes particiones con distintos Filesystems. A continuación

se muestran dos (2) capturas de distintos dispositivos donde se observa con claridad que la primera utiliza yaffs2 para sus particiones de /system , /data y /cache y la segunda imagen se muestra que las principales particiones utilizan ext4, salvo la tarjeta micro-sd que por cuestiones de compatibilidad con otros sistemas operativos suele utilizar Fat o ExFat.

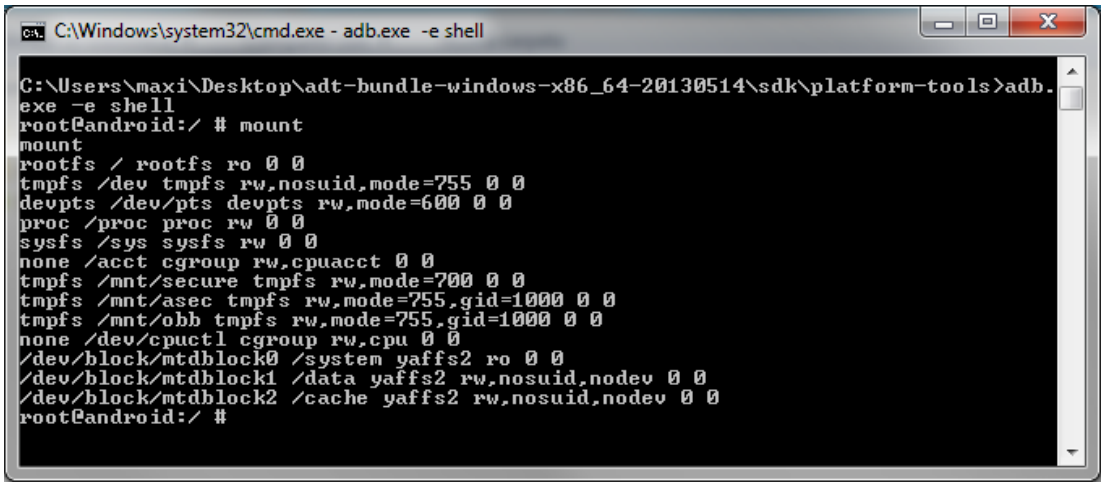


Figura 1.4 Particiones con Filesystem Yaffs2

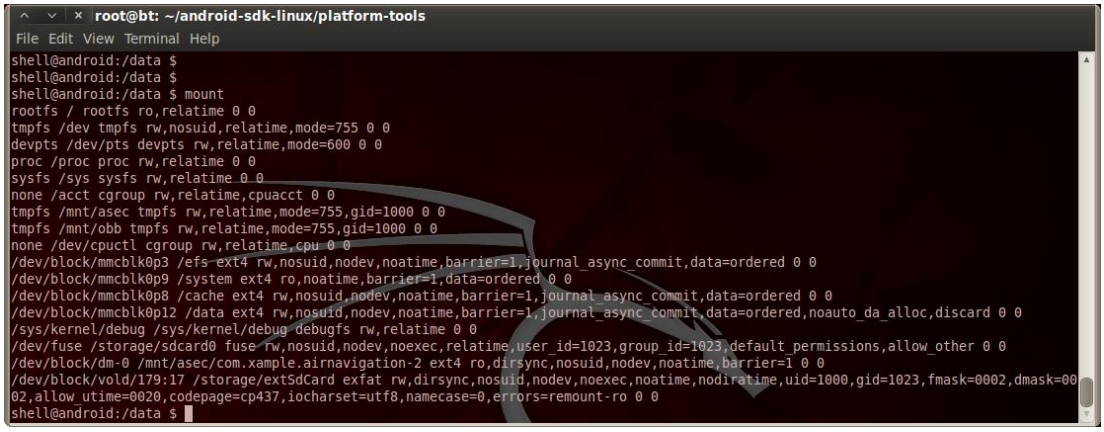


Figura 1.5 Particiones con Filesystem EXT4

¿Extracción física o lógica?

Al momento de extraer información de un dispositivo se pueden establecer dos metodologías que se definen como extracción física o

extracción lógica. Estas alternativas dependiendo del alcance de la investigación brindaran al perito la información que requiera analizar del dispositivo. A continuación se hará una breve descripción de las bondades que ofrecen cada una de estas alternativas

Extracción Física: este tipo de extracción o copia forense permitirá a quien deba examinar el dispositivo cavar en lo profundo del Filesystem del sistema operativo, permitiendo el acceso a información *"invisible"* como archivos borrados o la información contenida en los *"slack spaces"*. Para poder llevar a cabo una extracción de este tipo se deberá tener un almacenamiento igual o mayor al de origen ya que una copia de estas características tendrá como resultado un archivo (o más) que totalizará el tamaño del disco origen, este tipo de copias también insumen una gran cantidad de tiempo ya que el copiado es completo y suele ocupar varios gigabytes.

Extracción Lógica: esta operación permitirá al investigador extraer información del dispositivo que está accesible desde el Filesystem, esto incluye todos los archivos que estén disponibles ya sean fotos, bases de datos, archivos de texto, correos electrónicos, videos, mp3, entre otros. Cabe destacar que gran parte de la información que maneja el sistema operativo Android la almacena en bases de datos del tipo SQLlite, por lo tanto acceder a estos archivos brindará al investigador información relativa a contactos, sms, mms, correos electrónicos, historial de llamados, configuraciones del dispositivo, entre otras. A diferencia de una extracción física el tiempo que insume es notablemente menor ya que solo se trabajará sobre los archivos que sean necesarios para la investigación y no la totalidad del dispositivo.

En los próximos capítulos se demostrará cómo toda esta información sale a la luz una vez que los datos son extraídos del dispositivo.

Desafíos

El principal desafío que se enfrenta en este trabajo es realizar una extracción forense (o copia bit a bit) de las particiones de la memoria que están en el dispositivo móvil y contienen información sensible del usuario sin contaminar la evidencia. Pero ¿cómo es posible extraer la información que el dispositivo posee si al iniciar el sistema operativo las funciones inherentes a su funcionamiento montan y escriben información sobre las particiones? No es intención de este trabajo hacer una simple copia del contenido que tienen las particiones, tampoco se ve viable la opción de hacer un *"chip-off"* y retirar la memoria del equipo. Por el contrario, se propone hacer una copia bit a bit de la partición *userdata* que es la que guarda información de la actividad de los usuarios de los dispositivos y que puede ser utilizada con fines legales a los efectos de resolver un caso. La ventaja que ofrece poder realizar una copia bit a bit es que se puede extraer **toda** la información de la partición, aquella que se ve por el usuario y también la que no se ve, como información borrada o información que se encuentra en lo que se denomina en informática forense "slack space" (espacio perdido).

Queda fuera del alcance de este trabajo realizar una imagen y un análisis forense sobre la tarjeta de almacenamiento externa micro-sd ya que puede ser realizada y analizada siguiendo los mismos pasos que para cualquier dispositivo interno o externo de una PC.

Capítulo 2: Extracción de imagen forense de un Android

Metodología

La metodología que se propone en este trabajo consiste en la creación de una imagen o firmware el cual será instalado en la partición de Recovery del Smartphone a ser peritado. Tiene como fin configurar el dispositivo de manera tal que inicie en modo de Recovery con un programa de inicialización realizado específicamente para realizar imágenes forenses del dispositivo teniendo en cuenta las mejores prácticas para realizar estas tareas.

El firmware desarrollado no montará ni realizará ninguna tarea que escriba o modifique datos sobre las particiones que deberán ser preservadas para un posterior análisis. Lo que aquí se propone es comparable con la tarea de iniciar una computadora personal con una unidad externa con un sistema operativo con fines forenses como por ejemplo el HELIX o CAINE (entre otros) para ordenadores.

Recovery

Los dispositivos con sistema operativo Android pueden inicializar no solo en el modo normal sino que también se le puede ordenar al Bootloader que inicialice el dispositivo en modo Fastboot, modo Flash (dependiendo del dispositivo) y modo Recovery.

El modo Recovery permite inicializar el dispositivo utilizando la información contenida en la partición de Recovery, la cual contiene el programa de inicialización (INIT) con la base del sistema operativo que permite al usuario realizar tareas de mantenimiento en el dispositivo, entre las que se encuentran actualizar el dispositivo y borrar las particiones /data y /cache como se muestra en la figura a continuación.

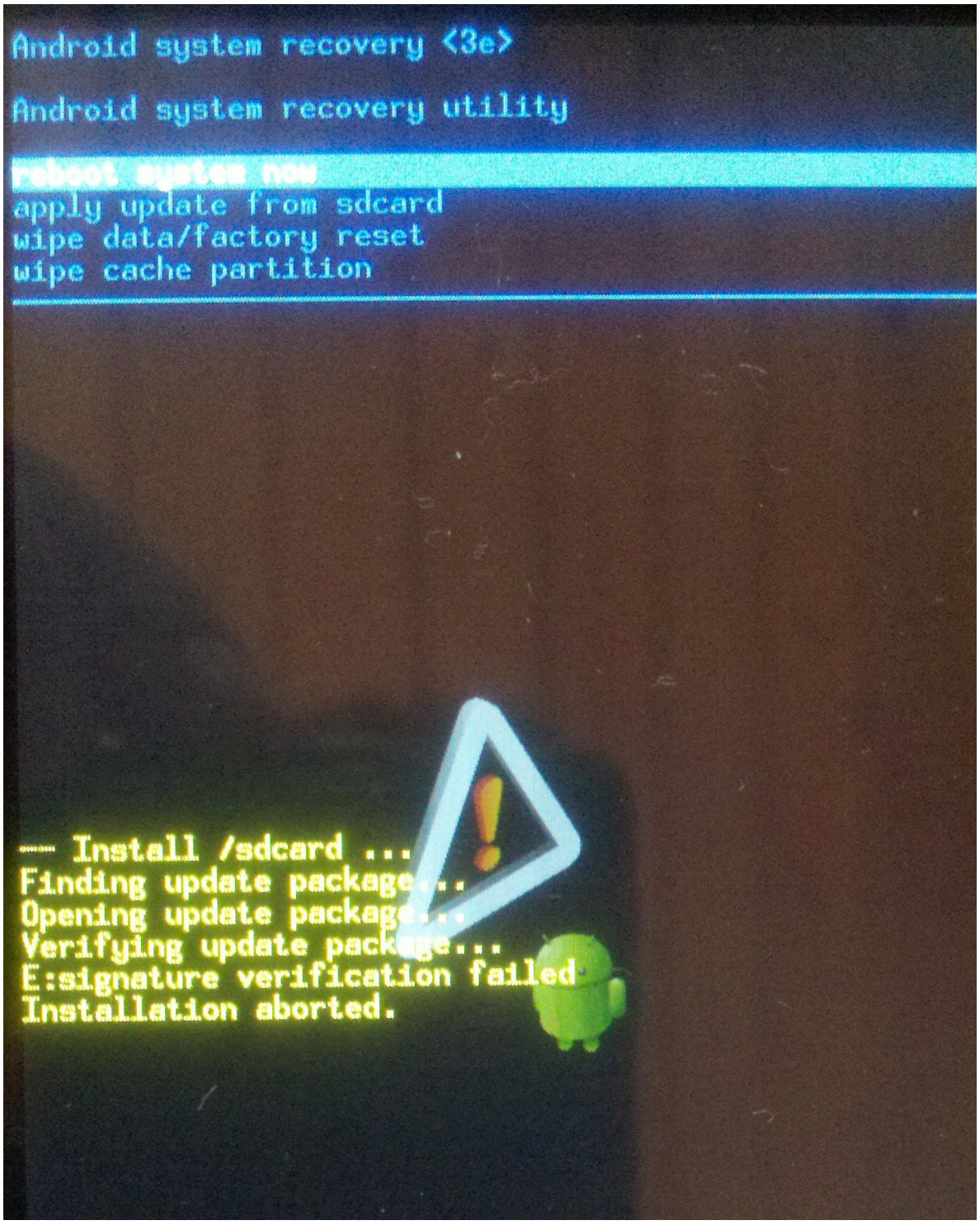


Figura 2.1 Captura de pantalla de un inicio con Recovery

Fastboot

Fastboot es un protocolo que es utilizado para actualizar los filesystems de dispositivos con Android desde una computadora por USB. A través suyo, se puede grabar en las particiones imágenes de firmware sin que estén firmadas. El principal problema que se enfrenta desde el punto de vista forense es que no todos los fabricantes incluyen fastboot en sus dispositivos por y esto es debido a que este no es un único protocolo que permite realizar estas tareas de actualización, sin embargo para este caso de estudio es posible utilizarlo.

Estructura de un firmware

La estructura de un firmware está conformada por un “boot header” (o encabezado de boot) , un kernel , un ramdisk y un carga de “second stage” , según surge de los paquetes de la SDK de Android y lo documenta y lo explica Android-dls [6] , como se puede ver a continuación:

```
+-----------------+
| boot header     | 1 pagina
+-----------------+
| kernel          | n paginas
+-----------------+
| ramdisk         | m paginas
+-----------------+
| second stage    | o paginas
+-----------------+

n = (kernel_tamaño + page_tamaño - 1) / pagina_tamaño
m = (ramdisk_tamaño + pagina_tamaño - 1) / pagina_tamaño
o = (second_tamaño + pagina_tamaño - 1) / pagina_tamaño
```

Figura 2.1.1 Estructura de un Firmware

Creando un firmware forense

Para dar comienzo a este trabajo se procedió a utilizar la imagen de un firmware oficial de Recovery del dispositivo a analizar. Para lograr esta

tarea se procede a *"desarmar"* el firmware separando su Kernel del Ramdisk para alterar el funcionamiento original del ramdisk evitando que este último, por medio del proceso init, monte las particiones que se desean extraer del sistema operativo. Sin que se haya montado la partición "userdata" y "cache" y se procede a adicionarle al proceso INIT el servicio de *"Android Debug Bridge"* para que se ejecute con los permisos de "*root"* (con el fin de acceder vía línea de comandos al sistema operativo cuando este inicializado), para luego rearmar la imagen de recovery y guardarla en el teléfono en la partición de recovery utilizando fastboot. Una vez realizado este procedimiento se iniciará el dispositivo en modo recovery (con la imagen realizada para este fin) y se ingresará al dispositivo vía una consola provista por *"Android Debug Bridge"* [5].

Desarmando el firmware de Recovery

Para desarmar la imagen de Recovery del dispositivo se procede a descargar del sitio clockworkmod [6] el firmware necesario para realizar esta tarea, y siguiendo los pasos descriptos en android-dls [7] se utiliza un script escrito por William Enck [8] denominado *"split_boot_img.pl"* con el fin de dividir el firmware en dos archivos distintos, en uno estará localizado el Kernel y el otro será el Ramdisk del Recovery.

```
root@bt:~/tesis# ../split_boot_img.pl recovery.img
Page size: 2048 (0x00000800)
Kernel size: 2823432 (0x002b1508)
Ramdisk size: 451086 (0x0006e20e)
Second size: 0 (0x00000000)
Board name:
Command line: androidboot.carrier=wifi-only product_type=w
Writing recovery.img-kernel ... complete.
Writing recovery.img-ramdisk.gz ... complete.
root@bt:~/tesis#
```

Figura 2.2 Partiendo la imagen de Recovery

Ramdisk

Alterando el Ramdisk

Con el fin de cambiar el comportamiento original del Recovery para que realice las tareas forenses, se procede a alterar el comportamiento de los programas que se ejecutan desde el ramdisk. En primer lugar, se procederá a descomprimirlo, como se muestra en la figura a continuación:

```
root@bt:~/tesis/ramdisk# gunzip -c ../recovery.img-ramdisk.gz | cpio
-i --verbose
data
default.prop
dev
etc
etc/recovery.fstab
init
init.rc
proc
res
res/images
res/images/icon_error.png
res/images/icon_installing.png
res/images/indeterminate1.png
res/images/indeterminate2.png
res/images/indeterminate3.png
res/images/indeterminate4.png
res/images/indeterminate5.png
res/images/indeterminate6.png
res/images/progress_empty.png
res/images/progress_fill.png
res/keys
sbin
sbin/adbd
sbin/recovery
sbin/ueventd
sys
system
tmp
ueventd.goldfish.rc
ueventd.rc
ueventd.stingray.rc
1275 blocks
root@bt:~/tesis/ramdisk#
```

Figura 2.3 Descomprimiendo el Ramdisk de Recovery

Luego, se procede a analizar el contenido de init.rc para conocer cuáles son los servicios que este proceso inicia y cuál es el comportamiento del Ramdisk. Se puede observar lo siguiente:

```
root@bt:~/tesis/ramdisk# cat init.rc
on early-init
    start ueventd

on init
    export PATH /sbin
    export ANDROID_ROOT /system
    export ANDROID_DATA /data
    export EXTERNAL_STORAGE /sdcard

    symlink /system/etc /etc

    mkdir /sdcard
    mkdir /system
    mkdir /data
    mkdir /cache
    mount /tmp /tmp tmpfs

on boot

    ifup lo
    hostname localhost
    domainname localdomain

    class_start default

service ueventd /sbin/ueventd
    critical

service recovery /sbin/recovery

service adbd /sbin/adbd recovery
    disabled

on property:persist.service.adb.enable=1
    start adbd

on property:persist.service.adb.enable=0
    stop adbd
root@bt:~/tesis/ramdisk#
```

Figura 2.4 Estructura del archivo init.rc

Se puede observar que entre las tareas más relevantes que realiza el *init.rc* se encuentra la de crear la estructura de directorios sobre la base o la raíz del sistema de archivos (ver comandos mkdir). Luego entre los servicios más importantes que inicia es el GUI del Recovery en la línea *"service recovery /sbin/recovery"* que se corresponde como resultado a la figura 2.1 y al final de este archivo se distingue que dependiendo el valor de la propiedad `persist.service.adb.enable` iniciará o no el servicio que se corresponde con "*Android Debug Bridge",* proceso necesario para realizar las tareas propuestas en este proyecto ya que proveerá al analista forense de un acceso por consola al dispositivo (también conocido como *shell*). El valor que tomará la propiedad `persist.service.adb.enable` estará configurado desde el archivo default.prop que se encuentra en la raíz del sistema de archivos. Como se muestra a continuación el valor que tiene esta propiedad (o variable) es 0 por lo que se interpreta desactivado.

```
root@bt:~/tesis/ramdisk# head default.prop
#
# ADDITIONAL_DEFAULT_PROPERTIES
#
ro.secure=1
ro.allow.mock.location=0
ro.debuggable=0
persist.service.adb.enable=0
# begin build properties
# autogenerated by buildinfo.sh
ro.build.id=H.6.2-21
root@bt:~/tesis/ramdisk#
```

Figura 2.5 Variables configuradas desde el archivo default.prop

Con el fin de lograr que el servicio de "*Android Debug Bridge"* inicie y poder acceder vía consola de comandos al dispositivo, se procede a alterar el valor del contenido del archivo "default.prop" y, para lograr que el acceso sea con *root* (o súper usuario), se cambiará el valor de la variable *ro.secure* a 0 ya que para poder realizar una imagen forense es necesario acceder a los dispositivos de modo directo (o *raw*) y solo es posible como *root*.

Teniendo alterados estos valores, para continuar con el proceso de adaptación del Ramdisk forense, se procede a insertar algunos comandos básicos para realizar la copia bit a bit. Como se puede observar en la imagen 2.3 del listado de archivos Ramdisk básico carece de comandos (o binarios de programas) y a los fines de agilizar las tareas, se procede a extraer de un firmware de Android los binarios necesarios y son copiados en el subdirectorio /sbin del Ramdisk que se está adaptando. Entre las herramientas que se utilizan y se agregan al Ramdisk para la creación de la imagen forense se encuentran: dcfldd, dd y dc3dd. Una vez realizado este paso se procede a comprimir nuevamente la estructura de Ramdisk.

```
root@bt:~/tesis/ramdisk# find . | cpio -o -H newc | gzip >
../newramdisk.gz
4406 blocks
root@bt:~/tesis/ramdisk# ls -la ../
total 16824
drwxr-xr-x  3 root root     4096 2013-08-07 01:01 .
drwx------ 50 root root     4096 2013-08-07 00:48 ..
-rw-r--r--  1 root root  1350520 2013-08-07 01:01 newramdisk.gz
drwxr-xr-x 11 root root     4096 2013-08-07 00:48 ramdisk
-rw-r--r--  1 root root 12582912 2013-08-05 19:35 recovery.img
-rw-r--r--  1 root root  2823432 2013-08-05 19:44 recovery.img-
kernel
-rw-r--r--  1 root root   451086 2013-08-05 19:44 recovery.img-
ramdisk.gz
root@bt:~/tesis/ramdisk#
```

Figura 2.6 Rearmando el archivo Ramdisk

Habiendo logrado re empaquetar el archivo Ramdisk se procede a realizar pruebas en el emulador de Android para analizar su comportamiento y corroborar si está funcionando de manera esperada antes de ser grabado en la memoria del dispositivo móvil.

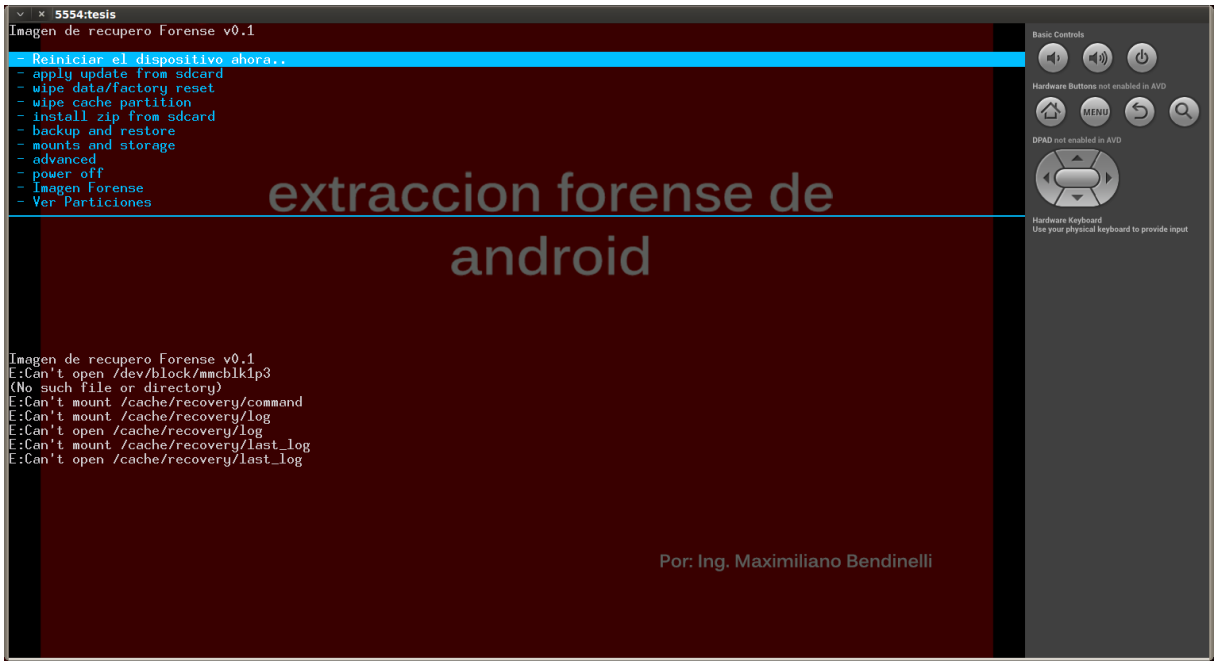


Figura 2.7 Probando Firmware Forense en el Emulador

Una vez que se ha verificado la iniciación del nuevo firmware en un entorno virtualizado para dicho fin, se procede a crear un firmware el cual contendrá el kernel del y el ramdisk creados.

```
mkbootimg --cmdline 'no_console_suspend=1 console=null' --kernel
recovery-s3.img-kernel --ramdisk newramdisk.cpio.gz -o recovery.img
```

Figura 2.7.1 Probando Firmware Forense en el Emulador

Una vez realizado el empaquetado del Kernel con el Ramdisk se procede a salvar la imagen en un dispositivo real, en este caso se utilizará una *Tablet* Marca Motorola modelo Xoom. Utilizando la herramienta Fastboot se guarda el firmware en la Tablet en la partición de Recovery y se procede a inicializar el equipo con la imagen de Recovery creada según se describió en los pasos anteriores.

Figura 2.8 Grabando el Firmware Forense usando fastboot

A continuación se puede apreciar la vista del dispositivo con el firmware siendo grabado.

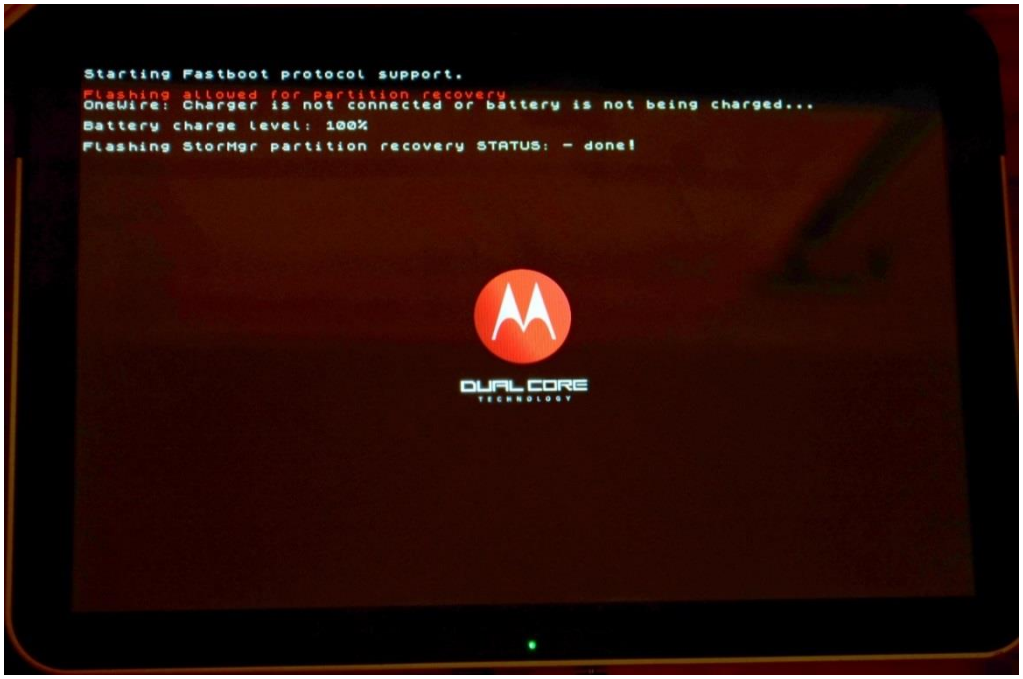


Figura 2.9 Grabando el Firmware Forense en el dispositivo

Para verificar el correcto funcionamiento del firmware creado y salvado, se inicializa la Tablet en modo Recovery y se puede observar la siguiente pantalla en el dispositivo.

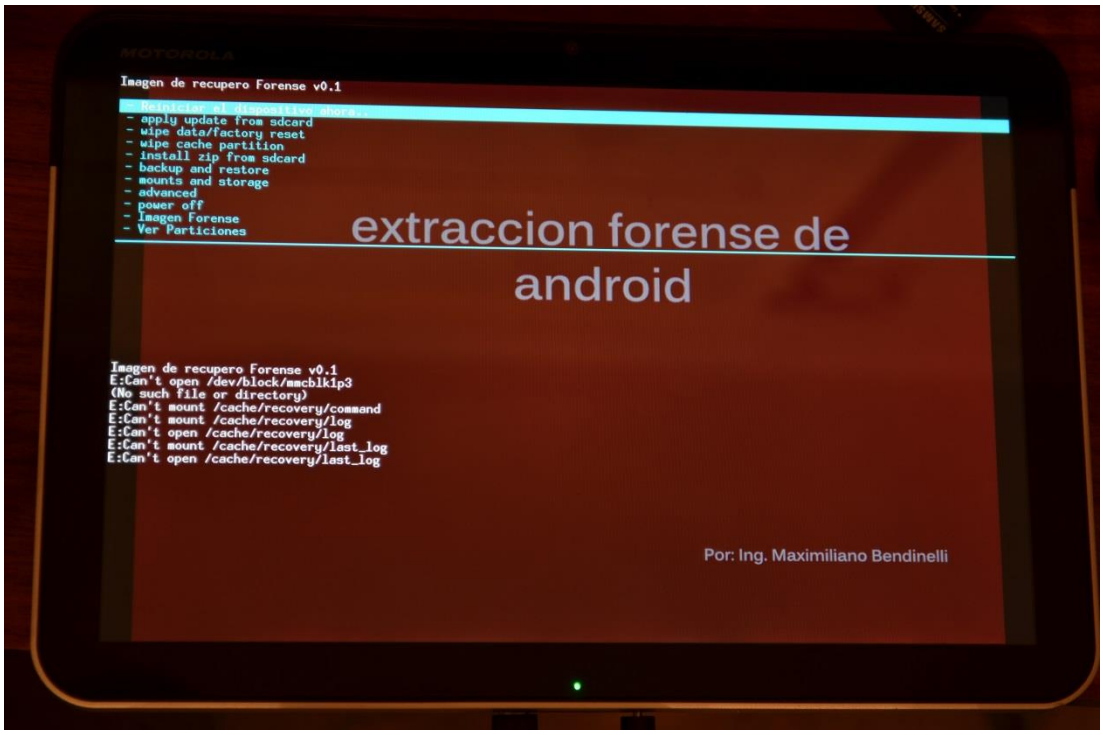


Figura 2.10 Firmware creado funcionando sobre la Tablet

Habiendo inicializado la Tablet con el firmware creado para realizar las imágenes forenses de las particiones, se procede a ingresar a la consola de comandos utilizando el ADB.

```
root@bt:~#       /root/adt-bundle-linux-x86_64-20130219/sdk/platform-
tools/adb devices
List of devices attached
037c70884080e217        recovery

root@bt:~#       /root/adt-bundle-linux-x86_64-20130219/sdk/platform-
tools/adb shell
~ # uname -a
Linux localhost 2.6.36.3 #1 SMP PREEMPT Wed Mar 23 12:15:44 CDT 2011
armv7l GNU/Linux
~ # id
uid=0(root) gid=0(root)
~ # cat /proc/cpuinfo
Processor       : ARMv7 Processor rev 0 (v7l)
processor       : 0
BogoMIPS        : 999.42

processor       : 1
BogoMIPS        : 999.42
```

```
Features        : swp half thumb fastmult vfp edsp vfpv3 vfpv3d16
CPU implementer : 0x41
CPU architecture: 7
CPU variant     : 0x1
CPU part        : 0xc09
CPU revision    : 0

Hardware        : stingray
Revision        : 8300
Serial          : 037c70884080e217
```

Figura 2.11 Ingresando vía consola al dispositivo

Se comprueban las particiones del dispositivo y se verifican que no hayan sido montadas, lo que garantiza que **no se altera en ningún momento la integridad de la información** en dichas particiones.

```
~ # cat /proc/partitions
major minor  #blocks  name

 179        0   31162368 mmcblk0
 179        1       3072 mmcblk0p1
 179        2       2048 mmcblk0p2
 179        3       2048 mmcblk0p3
 179        4       4096 mmcblk0p4
 179        5       2048 mmcblk0p5
 179        6      12288 mmcblk0p6
 179        7       8192 mmcblk0p7
 259        0     262144 mmcblk0p8
 259        1     173056 mmcblk0p9
 259        2   30663168 mmcblk0p10
~ # cat /etc/fstab
/dev/block/mmcblk1p9 /cache ext4 rw
/dev/block/mmcblk1p10 /data ext4 rw
/dev/block/mmcblk1p8 /system ext4 rw
/dev/block/mmcblk0p1 /sdcard vfat rw
~ # mount
rootfs on / type rootfs (rw)
tmpfs on /dev type tmpfs (rw,relatime,mode=755)
devpts on /dev/pts type devpts (rw,relatime,mode=600)
proc on /proc type proc (rw,relatime)
sysfs on /sys type sysfs (rw,relatime)
```

Figura 2.12 Verificación de particiones

Se procede a verificar la existencia y funcionamiento de los programas instalados para realizar la imagen forense, en particular dcfldd y dc3dd.

```
~ # ls -la /sbin/dcfldd /sbin/dc3dd
-rwxr-xr-x    1 root     root       1021168 Aug 21 02:04 /sbin/dc3dd
-rwxr-xr-x      1  root        root             804678  Aug  21  02:04
/sbin/dcfldd
~ # /sbin/dcfldd --version
dcfldd (dcfldd) 1.3.4-1
Written by: dcfldd by Nicholas Harbour, GNU dd by Paul Rubin, David
MacKenzie and Stuart Kemp.

Copyright (C) 1985-2006 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There
is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR
PURPOSE.
~ # /sbin/dc3dd --version
dc3dd (dc3dd) 7.1.614
Copyright (C) 2008 Free Software Foundation, Inc.
License     GPLv3+:     GNU     GPL     version     3     or     later
<http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Written by Paul Rubin, David MacKenzie, Stuart Kemp,
Jesse Kornblum, Andrew Medico, and Richard Cordovano.
```

Figura 2.13 Verificación de Binarios para la adquisición forense

Al llegar a este punto, se pudo crear un firmware (o imagen) de Recovery el cual una vez instalado en el dispositivo e inicializado en este modo, permitió el acceso al equipo y a ejecutar distintos comandos. En los siguientes capítulos se demostrarán distintas maneras de extraer la información contenida en el dispositivo siguiendo las mejores prácticas para la extracción de evidencia digital de un medio de almacenamiento.

Preservación de la evidencia

Previo a realizar una imagen forense de las particiones que son de interés para este trabajo (*“userdata” y “cache”*) se debe elegir el método y el

medio por el cual se almacenará o guardaráa las imágenes forenses creadas que son resultantes de la extracción de la información. Entre las opciones que se presentan existen la de almacenar la copia forense en una memoria micro-sd o trasferir la copia forense por el cable USB hacia una computadora.

Ambas opciones son válidas para resguardar y preservar las imágenes, pero presentan pros y contras. La principal ventaja que posee realizar una copia bit a bit sobre una memoria micro-sd es la velocidad de transferencia, pero debe disponerse de una memoria con la capacidad suficiente para realizar dicha tarea y con el crecimiento en almacenamiento de los teléfonos puede ser en algún caso difícil tener disponible una memoria con la capacidad requerida. La trasferencia por cable USB es más lenta pero el almacenamiento en las computadoras personales suele estar al alcance del investigador.

A continuación se demostrará las dos maneras aquí planteadas de extraer la evidencia y se podrá verificar mediante un hash del tipo MD5 que las copias, sin importar el método, son exactamente iguales.

Se procede a extraer una partición del dispositivo y guardarla en una computadora conectada vía USB y se toma el tiempo que se tarda en realizar la tarea.

```
root@bt:~/adt-bundle-linux-x86_64-20130219/sdk/platform-tools#  time
./adb  shell  "dcfldd  if=/dev/block/mmcblk0p9  hashlog=/tmp/hash.log
errlog=/tmp/error.log 2>/tmp/stderr.log" | sed 's/\r$//' > mmcblk0p9

real    3m1.130s
user    0m0.600s
sys     0m7.552s
```

Figura 2.14 Realizando una copia bit a bit por medio de una conexión USB

El tiempo que llevó realizar este procedimiento fue de 3 minutos y 1,1 segundos para copiar una partición de 170 Megabytes por USB. Acto

seguido se procede a verificar el md5 de la partición del dispositivo y luego se verifica la el mismo hash de la copia.

```
# ./adb shell md5sum /dev/block/mmcblk0p9
cf1404228092bb1ab5300128d5bef774  /dev/block/mmcblk0p9

# md5sum  mmcblk0p9
cf1404228092bb1ab5300128d5bef774  mmcblk0p9
```

Figura 2.15 Verificando la integridad de la copia.

Como se observa en recuadro anterior (Figura 2.15), la partición del dispositivo, así como también la copia realizada, son idénticas obteniéndose como resultado de esta prueba el mismo hash md5 el cual fue "cf1404228092bb1ab5300128d5bef774".-

Al chequear el contenido del archivo con el comando de Linux *"file"* se verifica que dicho archivo contiene una partición con un filesystem ext4.

```
# file mmcblk0p9
mmcblk0p9: Linux rev 1.0 ext4 filesystem data, UUID=57f8f4bc-abf4-
655f-bf67-946fc0f9f25b (extents) (large files)
```

Figura 2.16 Verificando el contenido de la copia.

A continuación se procederá a realizar la misma tarea sobre una memora micro-sd. Primero se identifican las particiones de la memoria, luego se monta y se realiza la copia bit a bit de una partición. Se muestra en los siguientes párrafos el procedimiento realizado:

```
~ # cat /proc/partitions
major minor  #blocks  name

 179        0    7761920 mmcblk0
 179        1    7757824 mmcblk0p1
 179        8   31162368 mmcblk1
 179        9       3072 mmcblk1p1
 179       10       2048 mmcblk1p2
 179       11       2048 mmcblk1p3
```

```
 179       12        4096 mmcblk1p4
 179       13        2048 mmcblk1p5
 179       14       12288 mmcblk1p6
 179       15        8192 mmcblk1p7
 259        0      262144 mmcblk1p8
 259        1      173056 mmcblk1p9
 259        2    30663168 mmcblk1p10
~ #
```

Figura 2.17 Tabla de particiones

Como se observa en la figura 2.17, se identifican claramente dos dispositivos de almacenamiento, la memoria micro-sd identificada como mmcblk0, y la memoria interna del equipo de estudio que es mmcblk1.En el siguiente paso se montará la memoria y se verificará que haya montado correctamente y que exista espacio disponible para almacenar la imagen forense.

```
~ # mount /dev/block/mmcblk0p1 /sdcard/
~ # mount
rootfs on / type rootfs (rw)
tmpfs on /dev type tmpfs (rw,nosuid,relatime,mode=755)
devpts on /dev/pts type devpts (rw,relatime,mode=600)
proc on /proc type proc (rw,relatime)
sysfs on /sys type sysfs (rw,relatime)
/dev/block/mmcblk0p1 on /sdcard type vfat
(rw,relatime,fmask=0000,dmask=0000,allow_utime=0022,codepage=cp437,i
ocharset=iso8859-1,shortname=mixed,errors=remount-ro)
~ # df -h
Filesystem               Size      Used Available Use% Mounted on
tmpfs                  359.4M     32.0K    359.4M   0% /dev
/dev/block/mmcblk0p1     7.4G      4.4G      3.0G  59% /sdcard
~ #
```

Figura 2.18 Montando la memoria externa

Los siguientes comandos se ejecutan con el fin de almacenar la imagen forense en la tarjeta micro-sd al mismo tiempo que se calcula su hash md5 y con fines puramente comparativos se calcula el tiempo que tarda en realizar las tareas. Cabe aclarar que la herramienta dcfldd que se introdujo dentro del firmware es comúnmente utilizada en tareas de adquisición forense con discos rígidos de computadoras, por lo tanto su

utilización es la misma que a la de una herramienta con una madurez reconocida.

```
~ # time dcfldd if=/dev/block/mmcblk1p9 of=/sdcard/mmcblk1p9.dd
hash=md5
5376 blocks (168Mb) written.Total (md5):
3026dc171cff839cb54765ef79babd36

5408+0 records in
5408+0 records out
real 0m 13.61s
user 0m 11.71s
sys 0m 1.89s
~ # md5sum /dev/block/mmcblk1p9
3026dc171cff839cb54765ef79babd36  /dev/block/mmcblk1p9
~ #
```

Figura 2.19 Realizando una copia bit a bit hacia una memoria externa

Del resultado obtenido, se observa el resultado md5 el cual dio **3026dc171cff839cb54765ef79babd36** y que luego con los fines de verificarlo se calculó el md5 de la partición la cual arrojo el mismo resultado y que el tiempo en realizar la tarea que vía USB demoró 3 minutos con una memoria micro-sd demoró apenas 13.61 segundos.

Se procede a insertar la memoria micro-sd en una computadora y se verifica la existencia del archivo contenedor de la imagen forense.

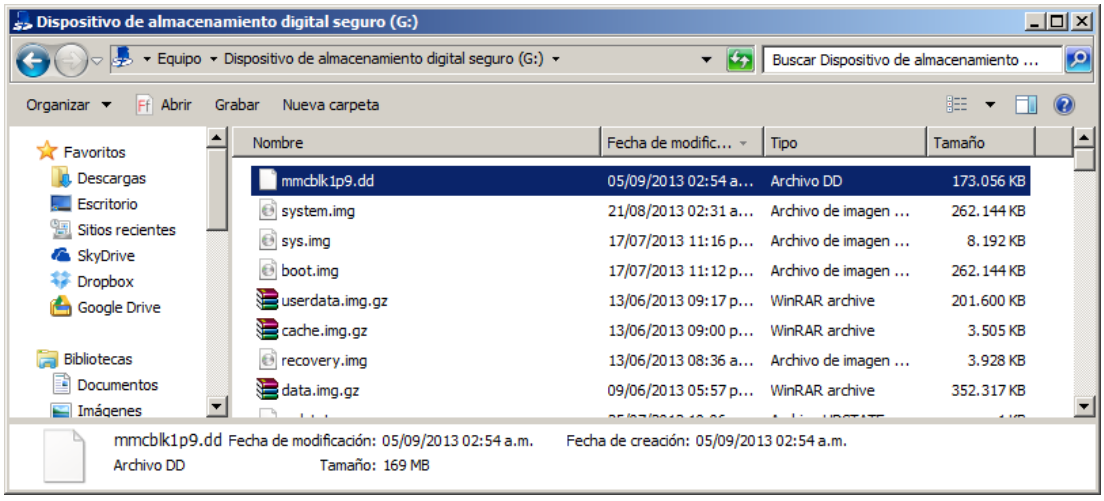


Figura 2.20 Archivo contenedor de la imagen forense

Al abrir el mencionado archivo con la herramienta FTK Imager, se puede ver el contenido de la partición resguardada.

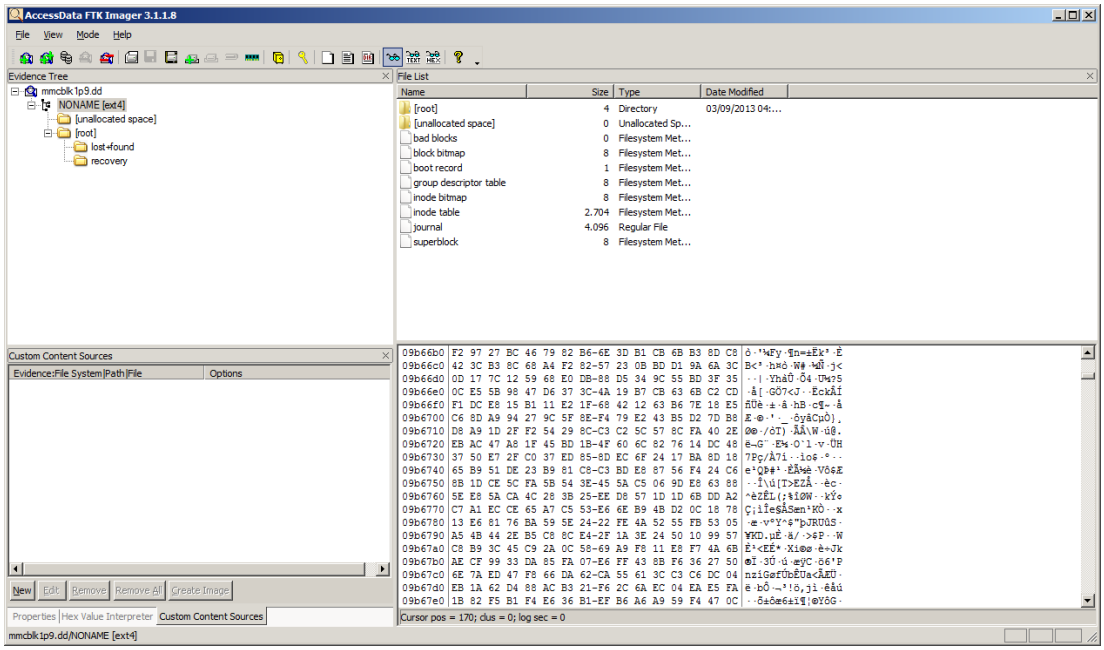


Figura 2.21 Abriendo la imagen forense con FTK Imager

Hasta aquí se ha demostrado que es posible mediante distintas herramientas de código abierto, realizar una imagen forense de un dispositivo con Android sin alterar la evidencia que contiene, manteniendo la integridad de la información en todos sus pasos y permitiendo que las copias realizadas puedan ser interpretadas por herramientas forenses para su posterior análisis.

Capítulo 3: Análisis forense Android

Para realizar un análisis forense de la copia bit a bit de la memoria interna en la cual se aloja el Filesystem del dispositivo con Android, se puede seguir las mismas prácticas forenses que para la copia de una imagen bit a bit del disco rígido de una computadora o notebook.

Hasta aquí se ha logrado resguardar y preservar la evidencia extraída de un dispositivo con Android así como también se obtuvo un hash de validación el cual le permitirá en un futuro al investigador validar la integridad de la información extraída.

Si bien objetivo de este trabajo consiste en lograr una adquisición forense de un dispositivo con Android, en este capítulo se intentará demostrar cómo con herramientas forenses se puede obtener información de la imagen (o copia forense) creada siguiendo los pasos y utilizando las herramientas utilizados en los capítulos anteriores.

Para continuar se procede a realizar una copia forense de la partición "/data" (o también conocida como "userdata"), que como se mencionó anteriormente aquí, se alojan los archivos que utilizan las distintas aplicaciones instaladas en el dispositivo como la información del usuario, contactos, mensajes, configuraciones, aplicaciones instaladas, fotos, videos, música y variada información que puede ser de utilidad para una investigación. También podrá surgir si el dispositivo a ser investigado fue víctima de alguna intrusión o malware el cual realizó actividades sin el conocimiento del usuario del dispositivo. Es por esto que a lo largo de este capítulo se demostrarán distintas formas de extraer información útil de la imagen forense realizada.

Como primer paso se procede a realizar una extracción física de una partición (unserdata) la memoria del dispositivo, para este caso de estudio se utilizó un teléfono celular Samsung i9300 (también conocido como S3). Una vez instalado el firmware creado en el Capítulo 2, se procede a ingresar al dispositivo vía *ADB* y por medio de los siguientes comandos que son

ejecutados desde una consola del dispositivo se logra la copia bit a bit y se la almacena en una memoria micro-sd:

```
~ # mount /dev/block/mmcblk0p1 /sdcard/
~ # dcfldd if=/dev/block/mmcblk0p12 hash=md5 | gzip -c >
/external_sd/userdata.gz

377856 blocks (11808Mb) written.Total (md5):
90799ea5802b69051f812fedf3778efe
377856+0 records in
377856+0 records out
```

Figura 3.1 Abriendo la imagen forense con FTK Imager

Habiendo realizado la extracción física del teléfono móvil se procede a verificar en una computadora la integridad de la copia por medio de un cálculo de hash del tipo MD5 y se verifica que los valores de cálculo hechos durante la extracción son los mismos que se obtienen en la computadora del investigador.

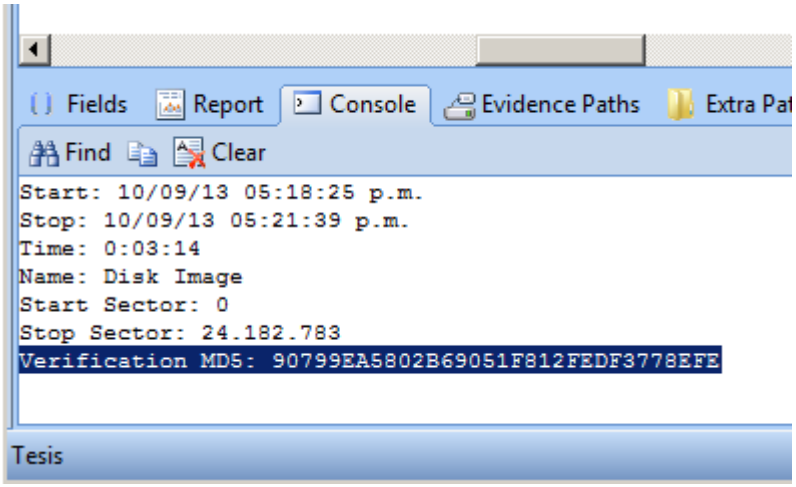


Figura 3.2 Verificando la integridad de la imagen forense

A modo de ejemplo se procede buscar en las distintas bases de datos de las aplicaciones instaladas en el sistema operativo.

Analizando SMS

La primera base que se va a verificar es la de sms realizados y recibidos desde el teléfono que está siendo investigado, para ello se abre la base llamada mmsms.db que se encuentra en "/data/com.android.providers.telephony/databases", en la figura 3.2 se puede visualizar la localización mencionada.

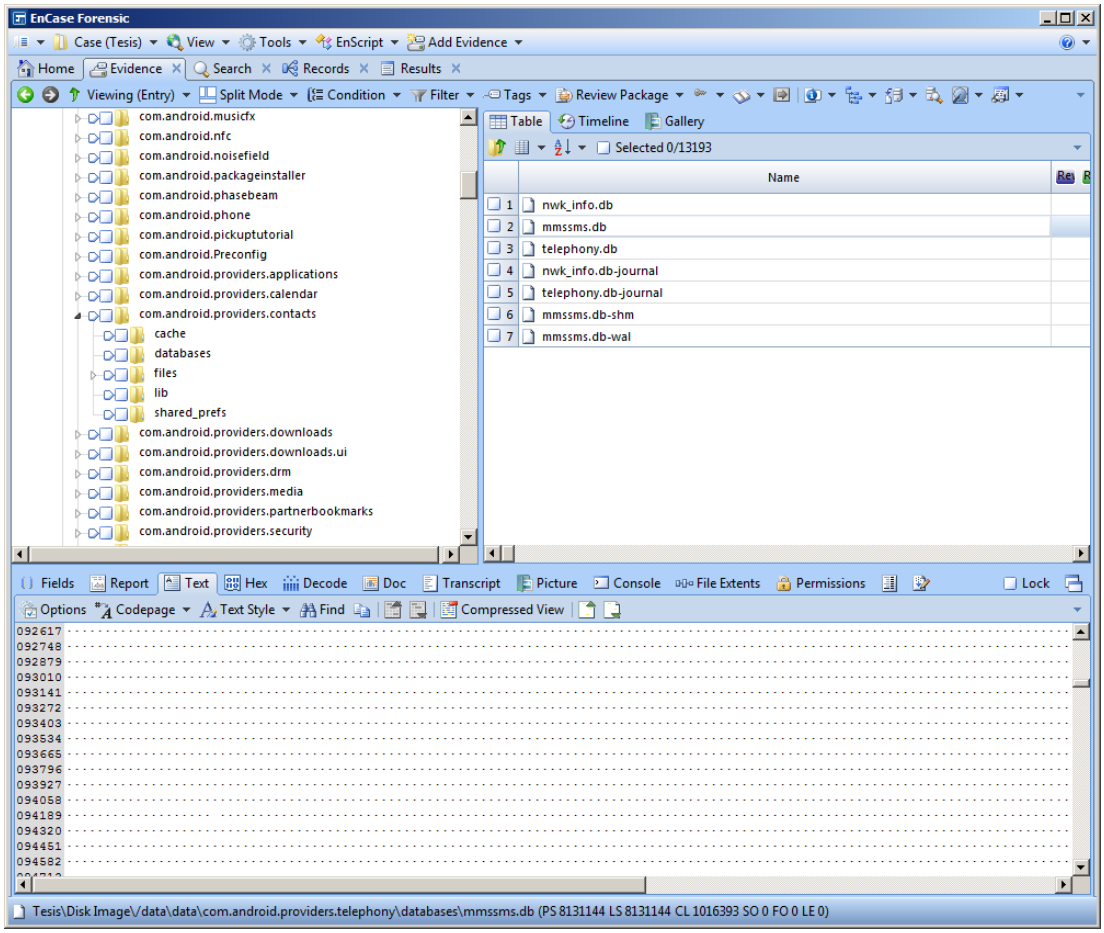
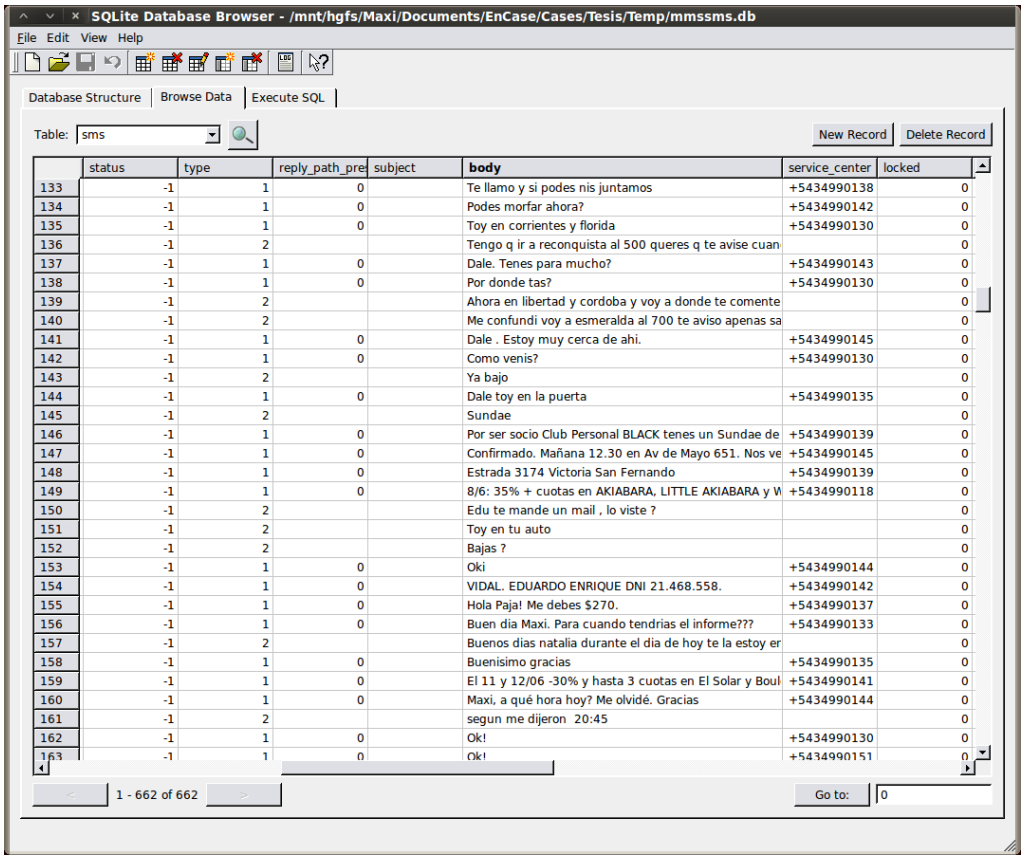


Figura 3.2 Localización de la Base de datos de SMS

Al abrir la base utilizando el programa "SQLite database Browser" (o cualquier otro cliente SQLite) se visualizan los mensajes enviados, recibidos, horarios de los mismos, si el SMS ha sido visto o no, entre otros datos de gran aporte para una investigación relativa a conocer el intercambio de SMS. Información que se puede observar en la figura 3.3.



	status	type	reply_path_pre	subject	body	service_center	locked
133	-1	1	0		Te llamo y si podes nis juntamos	+5434990138	0
134	-1	1	0		Podes morfar ahora?	+5434990142	0
135	-1	1	0		Toy en corrientes y florida	+5434990130	0
136	-1	2			Tengo q ir a reconquista al 500 queres q te avise cuan		0
137	-1	1	0		Dale. Tenes para mucho?	+5434990143	0
138	-1	1	0		Por donde tas?	+5434990130	0
139	-1	2			Ahora en libertad y cordoba y voy a donde te comente		0
140	-1	2			Me confundi voy a esmeralda al 700 te aviso apenas sa		0
141	-1	1	0		Dale . Estoy muy cerca de ahi.	+5434990145	0
142	-1	1	0		Como venis?	+5434990130	0
143	-1	2			Ya bajo		0
144	-1	1	0		Dale toy en la puerta	+5434990135	0
145	-1	2			Sundae		0
146	-1	1	0		Por ser socio Club Personal BLACK tenes un Sundae de	+5434990139	0
147	-1	1	0		Confirmado. Mañana 12.30 en Av de Mayo 651. Nos ve	+5434990145	0
148	-1	1	0		Estrada 3174 Victoria San Fernando	+5434990139	0
149	-1	1	0		8/6: 35% + cuotas en AKIABARA, LITTLE AKIABARA y W	+5434990118	0
150	-1	2			Edu te mande un mail , lo viste ?		0
151	-1	2			Toy en tu auto		0
152	-1	2			Bajas ?		0
153	-1	1	0		Oki	+5434990144	0
154	-1	1	0		VIDAL. EDUARDO ENRIQUE DNI 21.468.558.	+5434990142	0
155	-1	1	0		Hola Paja! Me debes $270.	+5434990137	0
156	-1	1	0		Buen dia Maxi. Para cuando tendrias el informe???	+5434990133	0
157	-1	2			Buenos dias natalia durante el dia de hoy te la estoy er		0
158	-1	1	0		Buenisimo gracias	+5434990135	0
159	-1	1	0		El 11 y 12/06 -30% y hasta 3 cuotas en El Solar y Boul	+5434990141	0
160	-1	1	0		Maxi, a qué hora hoy? Me olvidé. Gracias	+5434990144	0
161	-1	2			segun me dijeron 20:45		0
162	-1	1	0		Ok!	+5434990130	0
163	-1	1	0		Ok!	+5434990151	0

Figura 3.3 Contenido de la base de datos de SMS.

Analizando Contactos

Otra base de datos con información que puede ser de gran utilidad al investigador, es la base de contactos, la cual se encuentra en "/data/com.android.providers.contacts/databases" con el nombre mmssms.db como muestra la figura 3.4. Al abrir esta base se puede apreciar en la tabla "raw_contacts" los contactos agendados por el usuario del dispositivo como muestra la figura 3.5, así como también en esta misma base y en la tabla "Groups" se puede encontrar los grupos definidos por el usuario para relacionar contactos como ver observa en la figura 3.6.

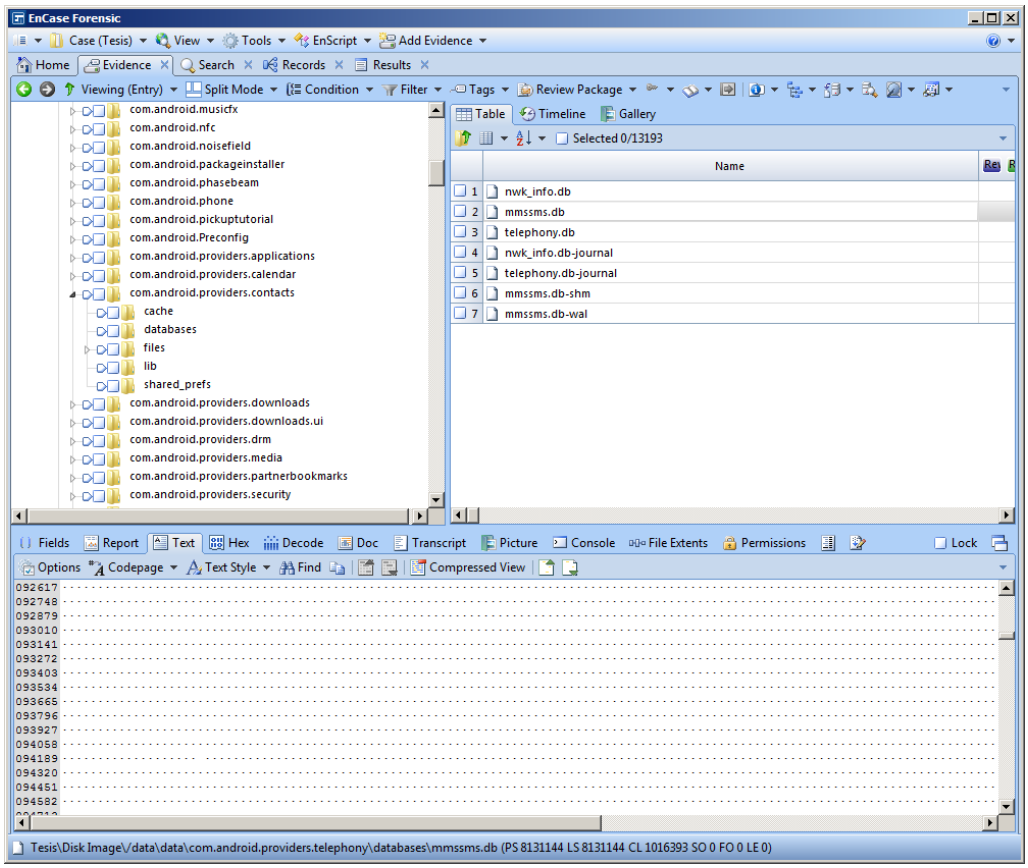


Figura 3.4 Localización de la base de datos de SMS

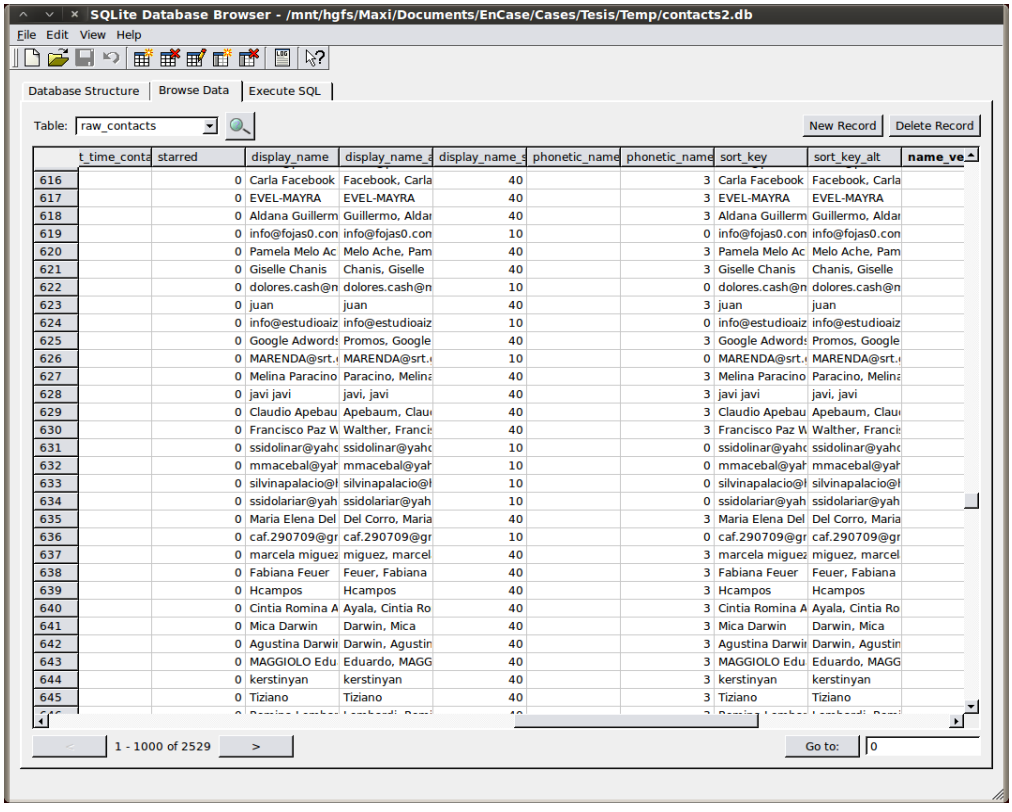


Figura 3.5 Contenido Contactos de la base de datos.

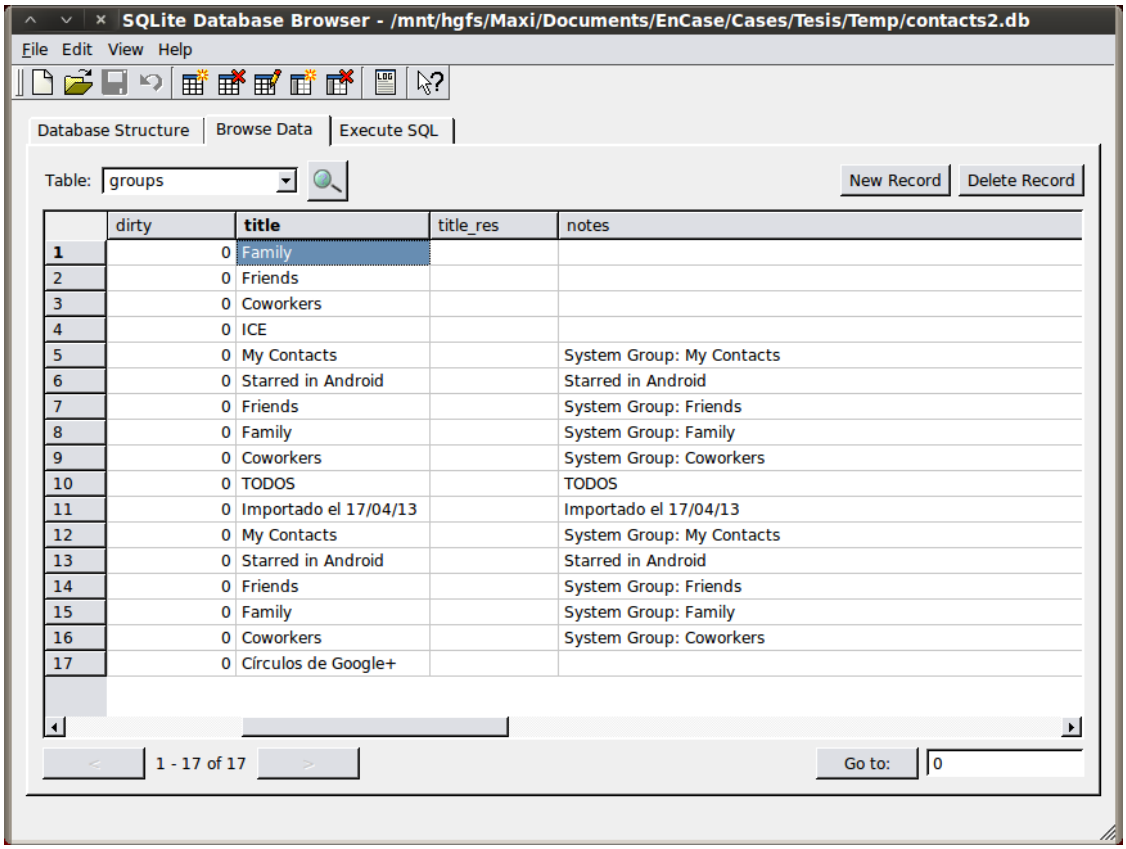


Figura 3.6 Contenido grupos de la base de datos de contactos.

Analizando Llamadas

A continuación se puede observar en la figura 3.7 la tabla "*logs*" de la base de datos *"logs.db"* ubicada en /data\data\com.sec.android.provider.logsprovider\databases contiene un registro de todas las llamadas y sms emitidas, recibidas y perdidas por el dispositivo así como también el tiempo de duración, el contacto vinculado con la llamada y la fecha y el horario entre otros datos de importancia. Se consultarápor medio de una sentencia sql: *"SELECT datetime(date/1000, 'unixepoch','localtime') as fecha, name, number, duration, type FROM logs WHERE (logtype=100 OR logtype=500);"*.

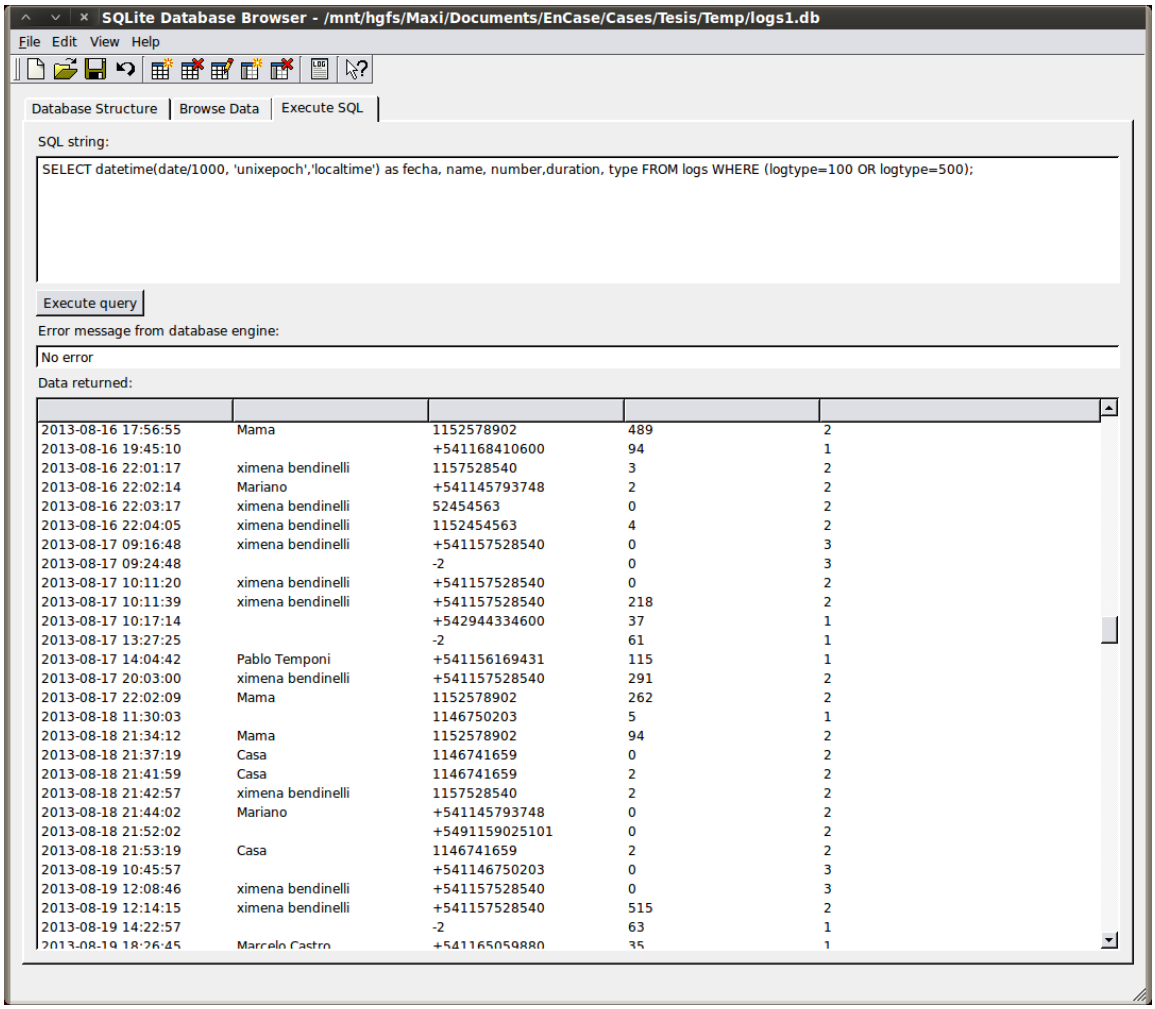


Figura 3.7 Registro de llamadas en el dispositivo.

Analizando Redes WiFi

Es posible también saber a cuáles redes WiFi estuvo conectado el dispositivo, ya que la información se encuentra dentro de la partición data (o userdata). Dentro del directorio /misc/wifi existe un archivo llamado wpa_supplicant.conf (***/misc/wifi/supplicant.conf***) que contiene información de las redes a las cuales estuvo conectado el dispositivo, asi como también sus claves de acceso en texto plano, como se muestra a continuación una porción de dicho archivo.

```
ctrl_interface=wlan0
update_config=1
device_name=m0ub
```

```
manufacturer=samsung
model_name=GT-I9300
model_number=GT-I9300
serial_number=4df113e12c0f4f31
device_type=10-0050F204-5
config_methods=physical_display virtual_push_button keypad
p2p_listen_reg_class=81
p2p_listen_channel=1
p2p_oper_reg_class=115
p2p_oper_channel=48
bss_expiration_scan_count=1

network={
        ssid="Personal EAP"
        scan_ssid=1
        key_mgmt=WPA-EAP IEEE8021X
        eap=SIM

identity="1722341212108215@wlan.mnc034.mcc722.3gppnetwork.org"
        priority=1
        vendor_spec_ssid=1
}

network={
        ssid="casita"
        psk="estaesunaclavedeprueba"
        key_mgmt=WPA-PSK
        priority=2
}

network={
        ssid="ING"
        scan_ssid=1
        psk="1ngen1er1a"
        key_mgmt=WPA-PSK
}

network={
        ssid="juani"
        key_mgmt=NONE
        auth_alg=OPEN SHARED
        wep_key0=XXXXXXXXX
        priority=3
}
```

Figura 3.8 Listado de redes wifi donde se conectó el dispositivo.

Analizando Bluetooth

Al igual que en los anteriores casos de análisis, la actividad bluetooth queda almacenada en una base sqlite localizada en data/com.android.bluetooth/databases cuyo nombre es btopp.db. En su interior se puede identificar la actividad que tuvo el dispositivo bluetooth el cual muestra la dirección MAC del dispositivo con el que se conectó así como también cuál fue la actividad realizada. Estainformación puede ayudar a vincular el dispositivo en investigación con otros ya sean celulares, computadoras, tablets,entre otros. A continuación se muestra el contenido del archivo btopp.db:

```
sqlite> select * from btopp;
1|content://media/external/images/media/2063|20130805_215124.jpg||im
age/jpeg|0|00:1D:4F:87:F7:35|0|2|497|2764058||1375750344613|0
2|content://media/external/images/media/2062|20130805_215109.jpg||im
age/jpeg|0|00:1D:4F:87:F7:35|0|2|497|2646730||1375750360247|0
3|content://media/external/images/media/2073|20130806_101603.jpg||im
age/jpeg|0|00:1D:4F:87:F7:35|0|2|497|1913774||1375795051047|0
```

Figura 3.9 Actividad Bluetooth del dispositivo.

Analizando Google Maps

Es posible establecer alguna de las búsquedas o localizaciones del usuario del dispositivo por medio de las bases de datos de la aplicación Google Maps. En la base de datos localizada en data/com.google.android.apps.maps/databases *search_history.db* se puede encontrar búsquedas de posibles destinos del dispositivo como se observa en la figura 3.10.

También es posible establecer lugares con fecha y horario que fueron cargados como destino en el Google Navigator, dicha información surge de la base de datos localizada en el mismo lugar que la anterior pero denominada *da_destination_history*, que al abrilapermite apreciar esa información como se observa en la figura 3.11.

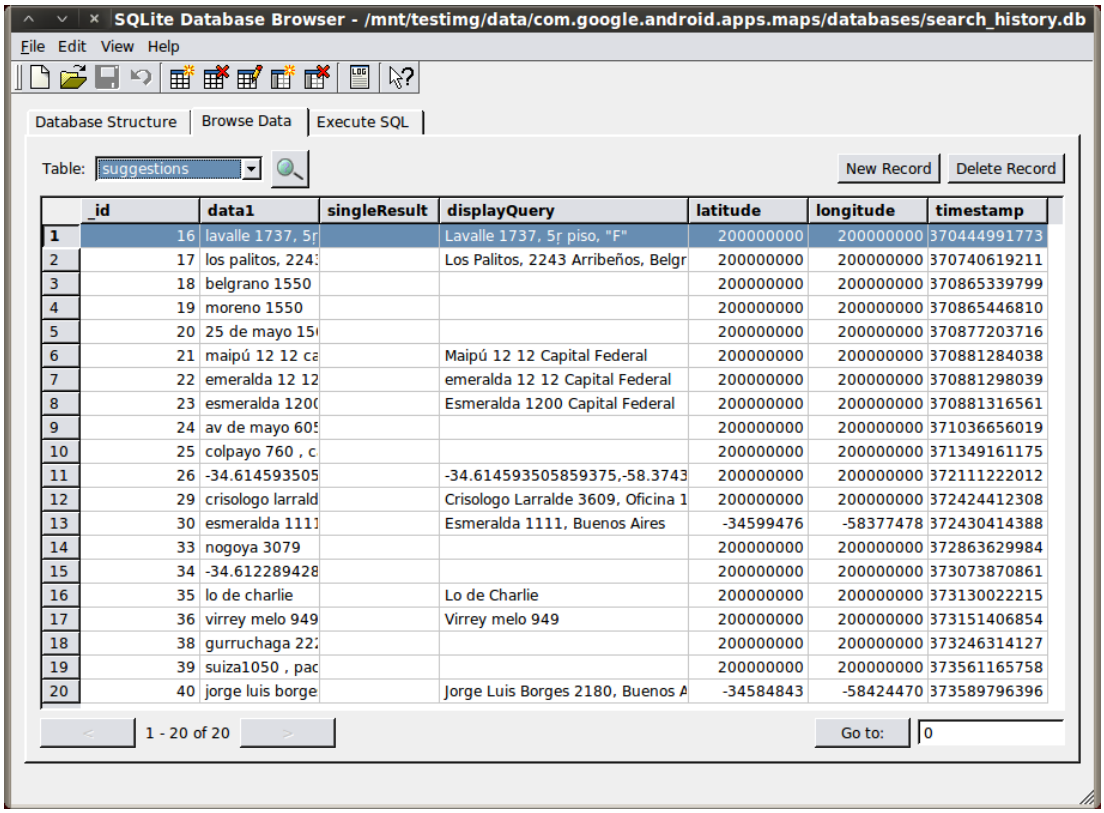


Figura 3.10 Direcciones Buscadas por Google Map.

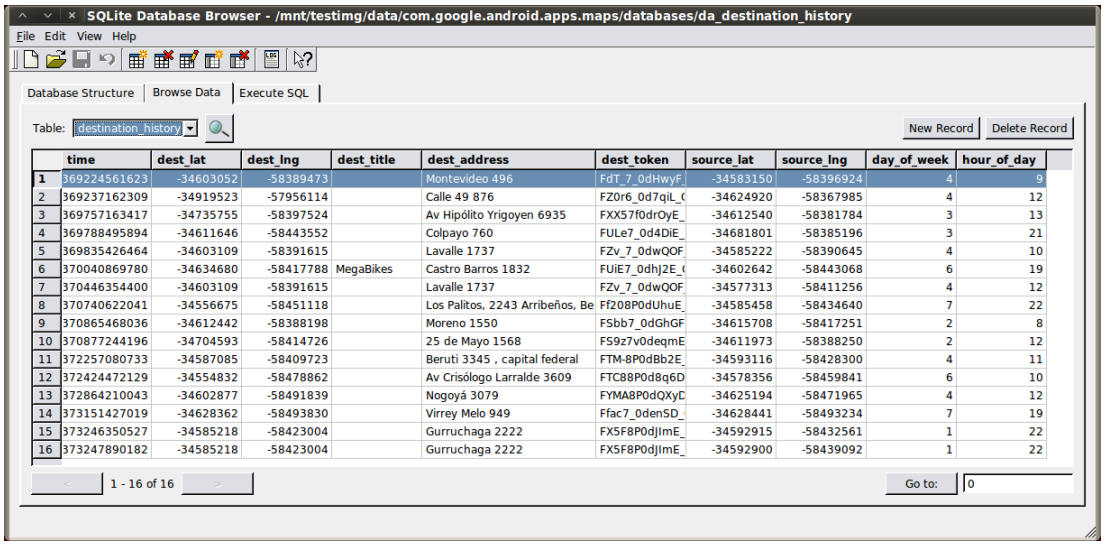


Figura 3.11 Destino de Navegación cargados en Google Navigator

Analizando Twitter

Es posible conocer por las bases de datos de Twitter la actividad del dispositivo con esta red social, se pueden analizar los mensajes públicos así como los mensajes privados que son enviados a la cuenta configurada en el dispositivo. En el directorio data/com.twitter.android/databases se encuentran las bases de datos con los contenidos de interés como *"twits"* que eran leídos desde el dispositivo y también los mensajes privados del usuario con otros usuarios de la red social.

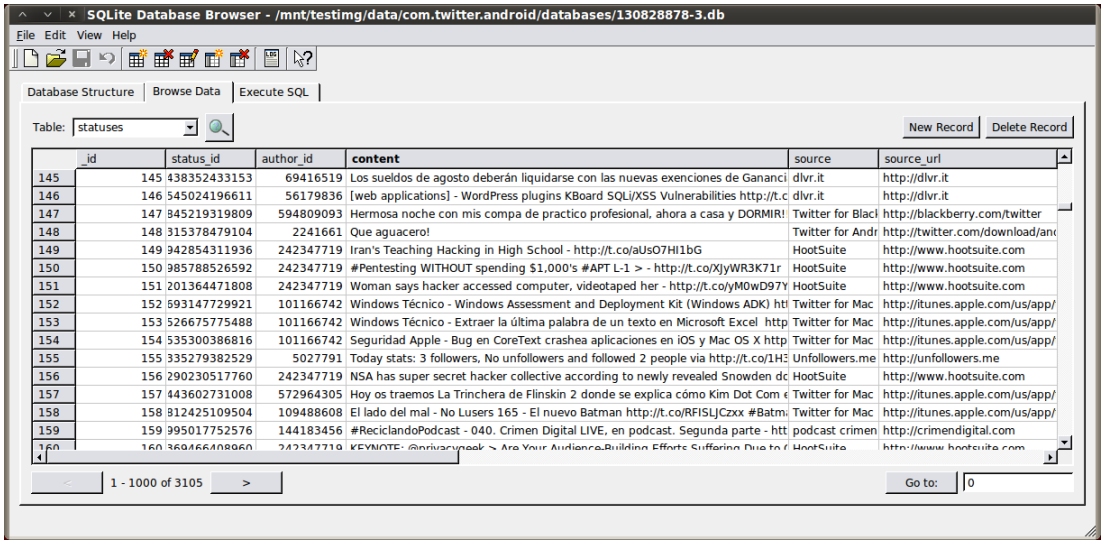


Figura 3.12 Lectura de "Twits" del usuario.

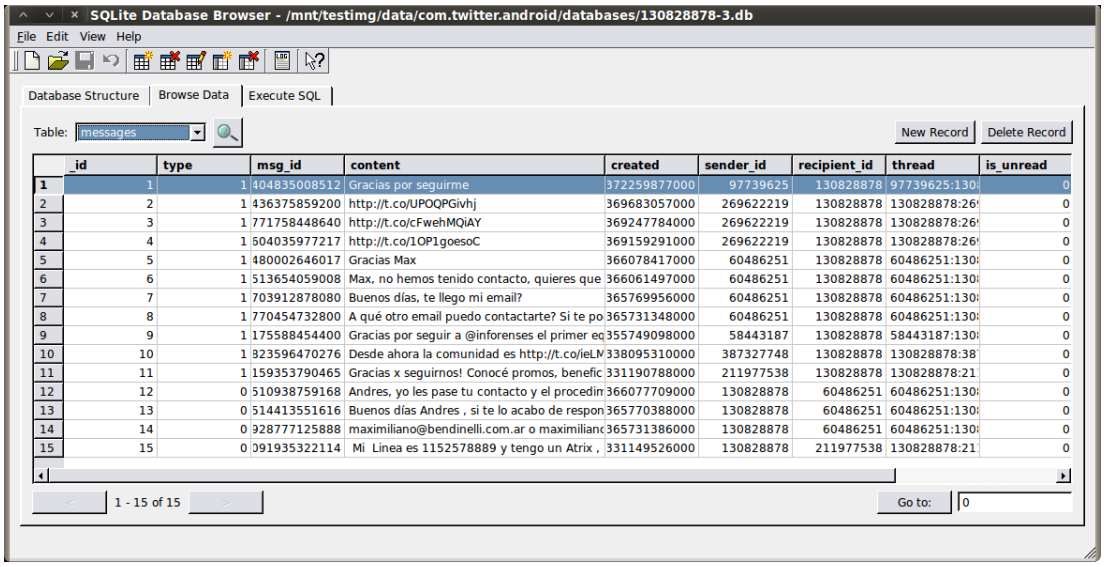


Figura 3.13 Mensajes privados del usuario con otros de la red social.

Analizando Whatsapp

Al igual que en los casos de análisis anterior es posible por medio de las bases de datos de esta aplicación conocer el intercambio de mensajes de usuario del dispositivo móvil, dicha información se encuentra en la base de datos denominada *msgstore.db* el directorio /data/com.whatsapp/databases, allí se podrá observar el horario, el mensaje, el status del mensaje y quién es el que envía o recibe dicho mensaje. Es importante aclarar que si los mensajes no fueron archivados se encuentran en texto en la base de datos, como se puede apreciar en la imagen que se encuentra a continuación:

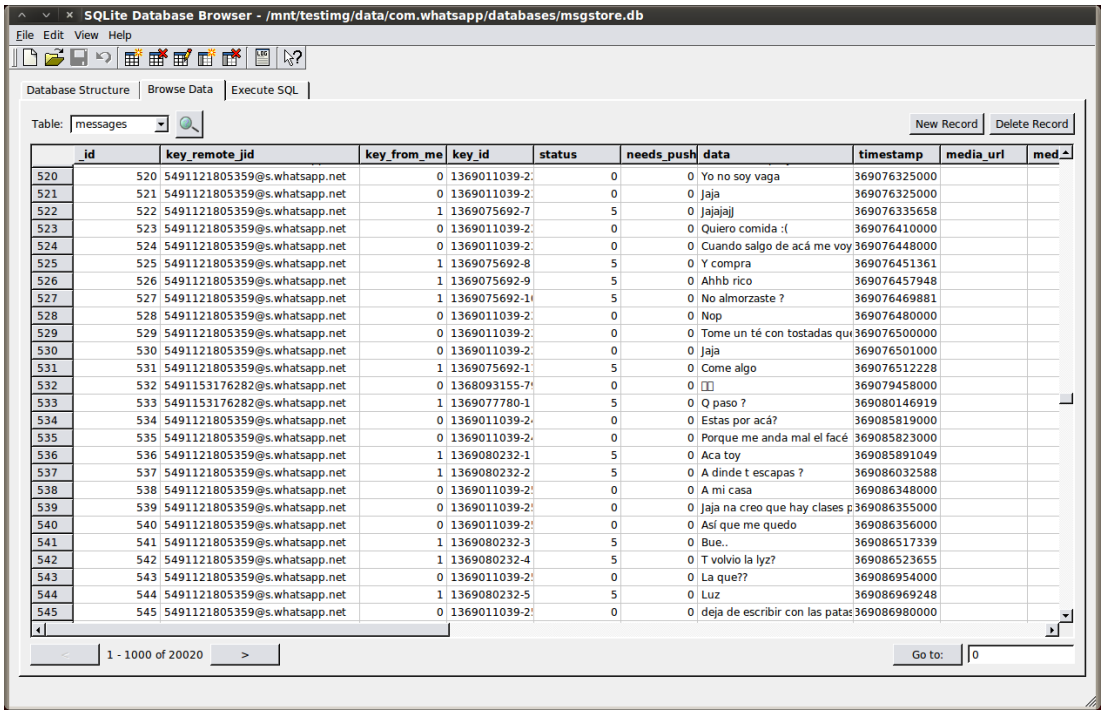


Figura 3.14 intercambio de mensajes de Whatsapp.

También fue posible hallar en directorio data\media\WhatsApp\Databasesbackups distintos archivos correspondientes a backups (o copias de resguardo) de las bases de datos contenedoras de mensajes como se aprecia en la figura 3.15, pero los

mismos se encuentran encriptados y debido a que no es parte de este trabajo no se continúa con la investigación de dichos archivos.

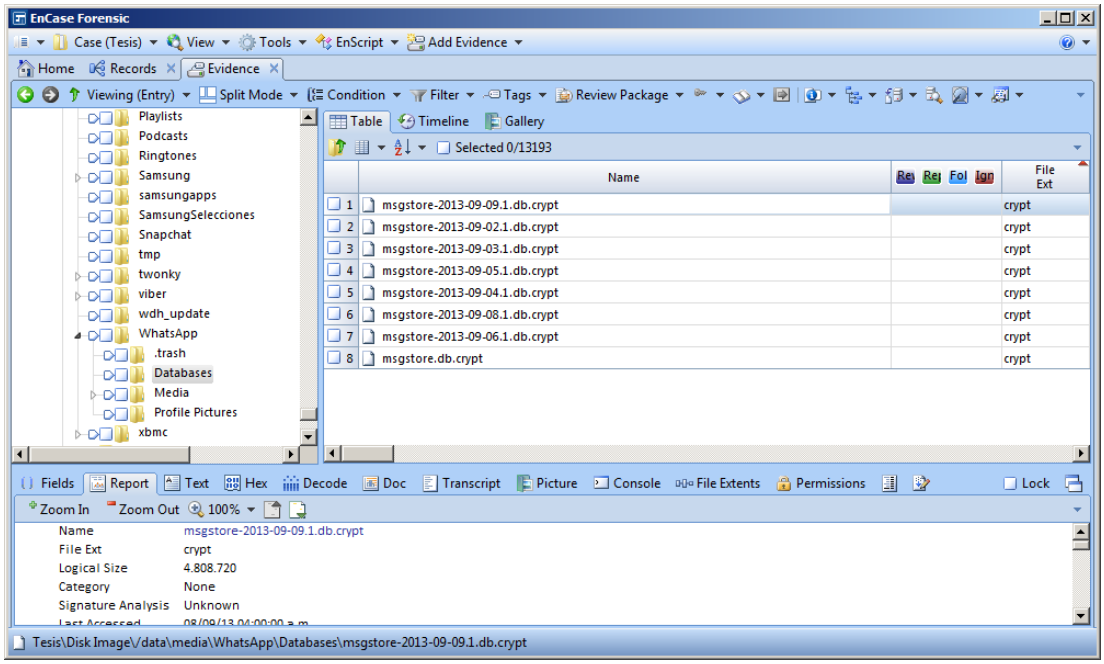


Figura 3.15 Backups de bases de datos encriptadas de Whassup.

Realizando Carving

Dado que en el capítulo anterior se procedió a realizar una copia forense (o copia bit a bit), este tipo de extracciones en las cuales toda la información contenida en la memoria interna del teléfono (visible o no para el Filesystem) es extraída, permite al investigador encontrar archivos que hayan sido eliminados y que no estén siendo mostrados por el Filesystem. Por lo tanto, por medio de la utilización de herramientas de “data carving” se procederá a recuperar el contenido que haya sido eliminado de la memoria. A continuación se muestra cómo luego de haber realizado el procedimiento para la recuperación de imágenes, se puede acceder a aquellas que no podrían haber identificarsea simple vista por quien esté realizando el análisis del dispositivo, en este ejemplo se pudo recuperar de la copia forense un total de 1669 imágenes como se puede apreciar en la figura 3.16.

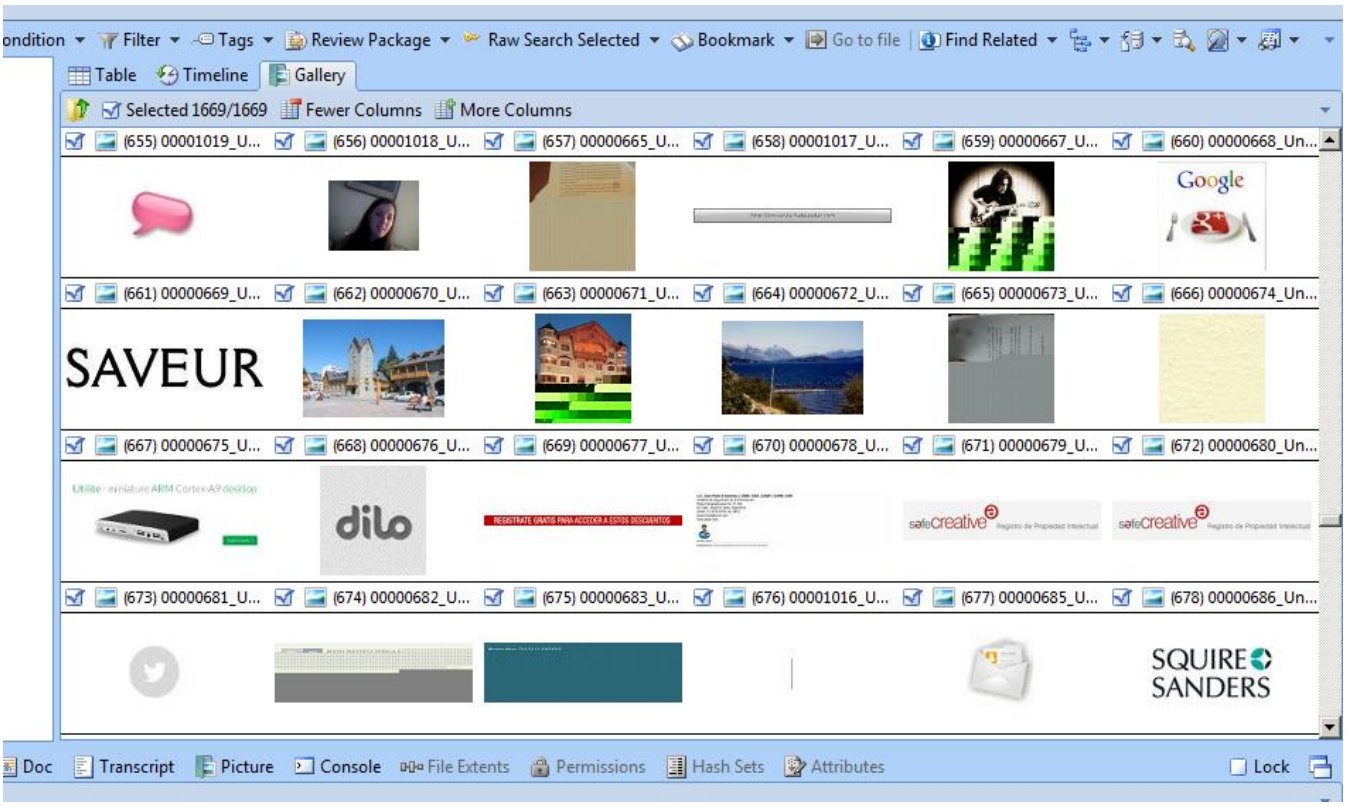


Figura 3.16 Recuperación de Archivos de Imagen de la copia Forense

Las técnicas de análisis mencionadas en este capítulo son sólo algunas de las tareas que se pueden realizar para analizar información del dispositivo móvil que ayudará a investigadores a la recopilación de información que será de utilidad para una causa.

Capítulo 4: ¿Virtualización?

La virtualización en entornos de informática forense es de utilidad para poder inicializar o “encender de manera virtual” con la copia bit a bit (o imagen forense) el dispositivo, computadora, notebook, pc, etc que se este analizándo. Además, debido a que en ciertas situaciones es necesario poder inicializar el dispositivo sin escribir información en el disco o contaminar la evidencia, se procede a virtualizar el entorno y trabajar con la copia forense de manera que ningún dato sea alterado. Para realizar las tareas con imágenes de computadoras (notebooks, PCs, netbooks) existe una herramienta denominada LiveView [9] que permite poder virtualizar dichos entornos.

Los usos de esta clase de herramientas son múltiples, entre algunas situaciones donde se puede necesitar se encuentran:investigar el comportamiento del dispositivo con la red con el fin de identificar si alguna aplicación o malware esta interactuando con el equipo y la red, tener una visión del escritorio de usuario y sus accesos directos o simplemente tener una visión general de la utilización del usuario con el dispositivo que está en análisis.

Semejantes herramientas no existen momentáneamente para dispositivos con sistema operativo Android, por esta razón, en este capítulo se intentará realizar una virtualización del entorno. Para lograr este cometido, será necesario realizar una imagen forense de la partición ***system***, la cual contiene el sistema operativo o firmware que se está utilizando en el dispositivo, y ***userdata*** que es donde se encuentra toda la parametrización del usuario así como toda la información y aplicaciones del dispositivo a analizar y se utiliza el emulador de Android [10] cambiando su ejecución *“normal”* con el fin de que tome como memoria (o disco interno) la que previamente se extrajo como la adquisición forense, y se intentará virtualizar el entorno con las imágenes del dispositivo analizado.

A continuación se demostrará paso a paso las tareas que serán realizadas para lograrlo.

Extrayendo las imágenes forenses

Para dar comienzo a esta etapa se procede a realizar una copia bit a bit de las particiones *userdata* y *system*, la cuales se almacenaron en una memoria micro-sd para luego ser copiadas el equipo que será el encargado de ejecutar la máquina virtual. Como se explicó en el punto 2 utilizando las herramientas dd, dfcldd o d3cdd se procede a extraer las particiones mencionadas almacenándolas en una memoria externa micro-sd.

La primer partición a extraer es la denominada system, la cual tiene el entorno de ejecución de Android.

```
~ # dcfldd if=/dev/block/mmcblk1p9 of=/sdcard/mmcblk1p9.dd hash=md5
5376        blocks        (168Mb)        written.Total        (md5):
3026dc171cff839cb54765ef79babd36

5408+0 records in
5408+0 records out
```

Figura 4.1 Extracción de la partición system

La segunda partición a extraer es la denominada data, la cual posee todos los datos relacionados con el usuario y las customizaciones del dispositivo.

```
~   #   dcfldd   if=/dev/block/mmcblk1p10   of=/sdcard/mmcblk1p10.dd
hash=md5
937500        blocks        (30Gb)        written.Total        (md5):
2a1946fc3bb0a952d9a1006f0d911a07
```

Figura 4.2 Extracción de la partición data

Ejecutando el entorno virtual

Una vez que las copias han sido almacenadas en el equipo encargado de la virtualización se procede a crear un perfil de emulación en la herramienta de desarrollo denominada ADV (Android Virtual Device). En este caso será llamada “tesis3” como se puede observar en la figura 4.1.

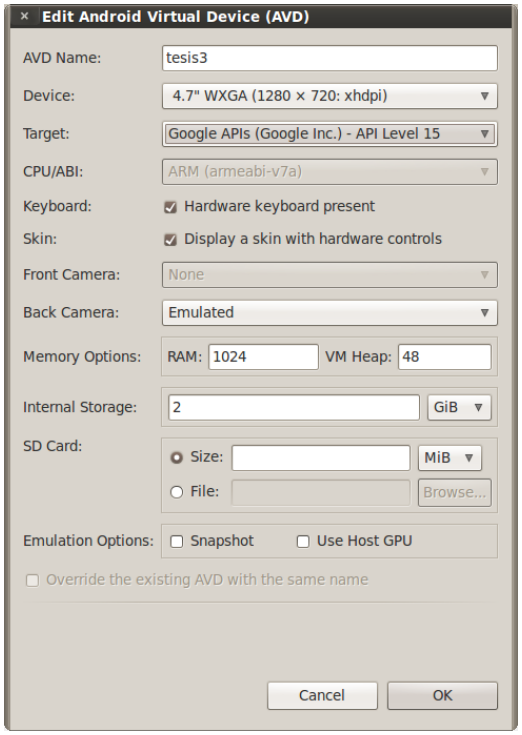


Figura 4.3 Creando un Virtual Device

Luego, se procede a ejecutar la línea de comando como se muestra en la figura 4.2, utilizando como parámetros el sistema extraído del dispositivo invocándolo por medio del parámetro –system y parametrizando por medio del parámetro *“datadir”* el archivo que contendrá la copia de la partición userdata y se agregan los parámetros “-verbose”, “-show-kernel” y “-shell” con fines de depuración y de ingreso a la plataforma por medio de una consola (o Shell).

```
emulator @tesis3 -ramdisk /root/roms/emulator/newramdisk.cpio.gz -
system /root/roms/xoom/system-test.img -datadir /mnt/datadir -
verbose -show-kernel -shell
```

Figura 4.4 Comando ejecutado para la inicialización de la "Virtualizacion"

Habiendo ejecutando la máquina virtual se puede observar desde la ventana del emulador que inicia el sistema operativo y queda el logo animado mostrándose en la pantalla sin que el estado cambie en ningún otro momento.

Figura 4.5 Emulador iniciando con las imágenes forenses

Al verificar el log de ejecución por medio del comando "*adb -e logcat*" se puede observar la siguiente salida:

```
...
W/dalvikvm( 138): threadid=11: thread exiting with uncaught
exception (group=0x40a191f8)
E/AndroidRuntime( 138): *** FATAL EXCEPTION IN SYSTEM PROCESS:
android.server.ServerThread
E/AndroidRuntime( 138): dalvik.system.StaleDexCacheError:
/system/framework/com.motorola.android.frameworks.jar
E/AndroidRuntime( 138): at
dalvik.system.DexFile.isDexOptNeeded(Native Method)
```

```
E/AndroidRuntime(         138):                                        at
com.android.server.pm.PackageManagerService.<init>(PackageManagerSer
vice.java:990)
E/AndroidRuntime(         138):                                        at
com.android.server.pm.PackageManagerService.main(PackageManagerServi
ce.java:838)
E/AndroidRuntime(         138):                                        at
com.android.server.ServerThread.run(SystemServer.java:172)
E/AndroidRuntime(  138): Error reporting crash
E/AndroidRuntime(  138): java.lang.NullPointerException
E/AndroidRuntime(         138):                                        at
com.android.internal.os.RuntimeInit$UncaughtHandler.uncaughtExceptio
n(RuntimeInit.java:72)
E/AndroidRuntime(         138):                                        at
java.lang.ThreadGroup.uncaughtException(ThreadGroup.java:693)
E/AndroidRuntime(         138):                                        at
java.lang.ThreadGroup.uncaughtException(ThreadGroup.java:690)
I/Process (  138): Sending signal. PID: 138 SIG: 9
I/Zygote  (    37): Exit zygote because system server (138) has
terminated
```

4.6 Salida del debug de la máquina virtual

De esta depuración se observa con claridad que el proceso Zygote se termina por una excepción y por lo tanto no puede continuar.

Resultado

Como resultado se observa que al configurar el emulador con las particiones que se extrajeron como parte de su entorno real inicia el proceso de emulación, se visualiza el logo del fabricante del firmware (Motorola) quedando el proceso de inicialización detenido en esta fase del inicio. Al analizar los logs del entorno se puede apreciar que el sistema comienza su proceso de iniciación pero el DAVLINK comienza a tener un gran número de errores que provocan la "*caída*" del proceso en sí. Se vuelve a ejecutar automáticamente y se obtiene el mismo resultado del comienzo, una y otra vez, entrando en un loop (o ciclo) infinito sin que la máquina virtual logre terminar su ejecución "normal". Por este motivo, se decide cancelar el proceso de emulación sin lograr el objetivo planteado al comienzo de este capítulo.

Dado que no era el principal objetivo de este trabajo realizar una virtualización y por cuestiones que exceden las metas planteadas, al no ser posible logar el funcionamiento de esta tarea, se deja en este estado de estudio.-

Desarrollo del software

Actualmente se está trabajando en realizar una distribución gratuita de este software para que pueda ser utilizada por toda la comunidad de investigadores forenses y fuerzas de la ley sin costo alguno. Dicho proyecto ha sido llamado “Android Forensic Toolkit”.-

Conclusiones

La falta de herramientas de código abierto o gratuito que ayuden a peritos o investigadores a extraer la información que está contenida en dispositivos móviles con sistema operativo Android siguiendo las mejores prácticas forenses, fue la inquietud que motivó la realización de este trabajo. A partir de este cuestionamiento, se procedió a verificar la factibilidad de realizar la extracción y, al constatarse que dicha tarea era posible, se desarrolló una metodología que mediante el uso de herramientas nativas de Unix así como también herramientas de código abierto, de código libre o gratuito, permitió la extracción forense de la información contenida en la memoria interna del dispositivo o teléfono. Al mismo tiempo, se demostró que con la extracción es posible realizar un análisis de la actividad del usuario con el dispositivo y extraer información sensible como goeposicionamiento, llamados, mensajes, sms, chat, imágenes, videos, elementos eliminados y actividad en redes sociales.

Adicionalmente se intentó realizar una virtualización del dispositivo mediante la utilización de las imágenes forenses extraídas del equipo en conjunto con el emulador de Android provisto por las herramientas de desarrollo de esta plataforma. Como se pudo observar en el capítulo 4, si bien es posible hacer iniciar un entorno virtual con las copias de la memoria, no es factible por el momento emular todo el hardware del equipo original. Además, debido a que algunas aplicaciones instaladas requieren acceso directo a equipamiento físico que, por razones inherentes a la emulación (o virtualización) no se encuentran disponibles, esta tarea se torna por el momento difícil ya que el sistema es inestable cuando logra arrancar y en algunos casos *no arranca*, Si bien no es objetivo inicial de este trabajo, se deja en este estado de situación y se abre la posibilidad de que la cuestión sea objeto de investigación a futuro por quien desee realizarlo ya que se convertiría en una excelente herramienta para investigadores forenses que interactúen con esta clase de tecnología.

Anexos

Ejecución del emulador para virtualización

```
root@bt:~/adt-bundle-linux-x86_64-20130219/sdk/tools#./emulator
@tesis3    -ramdisk  /root/roms/emulator/newramdisk.cpio.gz  -system
/root/roms/xoom/system-test.img  -datadir  /mnt/datadir   -verbose  -
show-kernel -shell
emulator:   found   SDK   root   at   /root/adt-bundle-linux-x86_64-
20130219/sdk
emulator:      Android      virtual      device      file      at:
/root/.android/avd/tesis3.ini
emulator: virtual device content at /root/.android/avd/tesis3.avd
emulator:         virtual         device         config         file:
/root/.android/avd/tesis3.avd/config.ini
emulator:       using       core       hw       config       path:
/root/.android/avd/tesis3.avd/hardware-qemu.ini
emulator: Found AVD target API level: 17
emulator: 'magic' skin format detected: 1280x720
emulator: autoconfig: -skin 1280x720
emulator: autoconfig: -skindir (null)
emulator: keyset loaded from: /root/.android/default.keyset
emulator:   found   SDK   root   at   /root/adt-bundle-linux-x86_64-
20130219/sdk
emulator: trying to load skin file '/root/adt-bundle-linux-x86_64-
20130219/sdk/tools/lib/emulator/skins/dynamic//layout'
emulator: loaded dynamic skin width=1280 height=720 bpp=16

emulator:    autoconfig:    -kernel    /root/adt-bundle-linux-x86_64-
20130219/sdk/system-images/android-17/armeabi-v7a//kernel-qemu
emulator:  Using  initial  system  image:  /root/roms/xoom/system-
test.img
emulator: autoconfig: -data /root/.android/avd/tesis3.avd/userdata-
qemu.img
emulator:                    autoconfig:                    -initdata
/root/.android/avd/tesis3.avd/userdata.img
emulator: autoconfig: -cache /root/.android/avd/tesis3.avd/cache.img
emulator: Physical RAM size: 1024MB

Content of hardware configuration file:
  hw.cpu.arch = arm
  hw.cpu.model = cortex-a8
  hw.ramSize = 1024
  hw.screen = touch
  hw.mainKeys = yes
  hw.trackBall = no
  hw.keyboard = yes
  hw.keyboard.lid = no
  hw.keyboard.charmap = qwerty2
  hw.dPad = no
  hw.gsmModem = yes
  hw.gps = yes
```

```
  hw.battery = yes
  hw.accelerometer = yes
  hw.audioInput = yes
  hw.audioOutput = yes
  hw.sdCard = no
  disk.cachePartition = yes
  disk.cachePartition.path = /root/.android/avd/tesis3.avd/cache.img
  disk.cachePartition.size = 66m
  hw.lcd.width = 1280
  hw.lcd.height = 720
  hw.lcd.depth = 16
  hw.lcd.density = 320
  hw.lcd.backlight = yes
  hw.gpu.enabled = no
  hw.camera.back = emulated
  hw.camera.front = none
  vm.heapSize = 48
  hw.sensors.proximity = yes
  hw.sensors.magnetic_field = yes
  hw.sensors.orientation = yes
  hw.sensors.temperature = yes
  kernel.path  =  /root/adt-bundle-linux-x86_64-20130219/sdk/system-
images/android-17/armeabi-v7a//kernel-qemu
  kernel.parameters =  androidboot.console=ttyS2 android.checkjni=1
  disk.ramdisk.path = /root/roms/emulator/newramdisk.cpio.gz
  disk.systemPartition.initPath = /root/roms/xoom/system-test.img
  disk.systemPartition.size = 253m
  disk.dataPartition.path  =  /root/.android/avd/tesis3.avd/userdata-
qemu.img
  disk.dataPartition.size = 2g
  avd.name = tesis3
.
QEMU options list:
emulator: argv[00] = "./emulator64-arm"
emulator: argv[01] = "-show-kernel"
emulator: argv[02] = "-serial"
emulator: argv[03] = "stdio"
emulator: argv[04] = "-android-hw"
emulator:   argv[05]   =   "/root/.android/avd/tesis3.avd/hardware-
qemu.ini"
Concatenated QEMU options:
 ./emulator64-arm   -show-kernel   -serial   stdio   -android-hw
/root/.android/avd/tesis3.avd/hardware-qemu.ini
emulator: registered 'boot-properties' qemud service
emulator:                                                nand_add_dev:
system,size=0xfd00000,initfile=/root/roms/xoom/system-test.img
emulator: mapping 'system' NAND image to /tmp/android-root/emulator-
WLLjr3
emulator: rounding devsize up to a full eraseunit, now fd0b000
```

```
emulator:                                                    nand_add_dev:
userdata,size=0x80000000,file=/root/.android/avd/tesis3.avd/userdata
-qemu.img
emulator: rounding devsize up to a full eraseunit, now 80010000

emulator: registered 'boot-properties' qemud service
emulator: Adding boot property: 'dalvik.vm.heapsize' = '48m'
emulator: Adding boot property: 'qemu.sf.lcd_density' = '320'
emulator: Adding boot property: 'qemu.hw.mainkeys' = '1'
emulator: Adding boot property: 'qemu.sf.fake_camera' = 'back'
emulator:                                                    nand_add_dev:
cache,size=0x4200000,file=/root/.android/avd/tesis3.avd/cache.img
emulator: Initializing hardware OpenGLES emulation support
Failed to create Context 0x3005
emulator: Can't start OpenGLES renderer?
emulator: WARNING: Could not initialize OpenglES emulation, using
software renderer.
emulator: Kernel parameters: qemu.gles=0 qemu=1 console=ttyS0
android.qemud=ttyS1 androidboot.console=ttyS2 android.checkjni=1
ndns=1
emulator: Trace file name is not set

emulator: autoconfig: -scale 1
emulator: Could not open file: (null)/system/build.prop: No such
file or directory
emulator: control console listening on port 5554, ADB on port 5555
emulator: sent '0012host:emulator:5555' to ADB server
emulator:      ping      program:      /root/adt-bundle-linux-x86_64-
20130219/sdk/tools/ddms
emulator:      ping      command:      /root/adt-bundle-linux-x86_64-
20130219/sdk/tools/ddms ping emulator 22.0.4.0 "" "" ""
Uncompressing
Linux.....................................................................
..................................... done, booting the kernel.
goldfish_fb_get_pixel_format:167: display surface,pixel format:
  bits/pixel:  16
  bytes/pixel: 2
  depth:       16
  red:         bits=5 mask=0xf800 shift=11 max=0x1f
  green:       bits=6 mask=0x7e0 shift=5 max=0x3f
  blue:        bits=5 mask=0x1f shift=0 max=0x1f
  alpha:       bits=0 mask=0x0 shift=0 max=0x0
Initializing cgroup subsys cpu
Linux version 2.6.29-gea477bb (kroot@kennyroot.mtv.corp.google.com)
(gcc version 4.6.x-google 20120106 (prerelease) (GCC) ) #1 Wed Sep
26 11:04:45 PDT 2012
CPU: ARMv7 Processor [410fc080] revision 0 (ARMv7), cr=10c5387f
CPU: VIPT nonaliasing data cache, VIPT nonaliasing instruction cache
Machine: Goldfish
Memory policy: ECC disabled, Data cache writeback
```

```
Truncating RAM at 00000000-3fffffff to -33ffffff (vmalloc region
overlap).
Built 1 zonelists in Zone order, mobility grouping on.  Total pages:
211328
Kernel command line: qemu.gles=0 qemu=1 console=ttyS0
android.qemud=ttyS1 androidboot.console=ttyS2 android.checkjni=1
ndns=1
Unknown boot option `qemu.gles=0': ignoring
Unknown boot option `android.qemud=ttyS1': ignoring
Unknown boot option `androidboot.console=ttyS2': ignoring
Unknown boot option `android.checkjni=1': ignoring
PID hash table entries: 4096 (order: 12, 16384 bytes)
Console: colour dummy device 80x30
Dentry cache hash table entries: 131072 (order: 7, 524288 bytes)
Inode-cache hash table entries: 65536 (order: 6, 262144 bytes)
Memory: 832MB = 832MB total
Memory: 840320KB available (2900K code, 707K data, 124K init)
Calibrating delay loop... 519.37 BogoMIPS (lpj=2596864)
Mount-cache hash table entries: 512
Initializing cgroup subsys debug
Initializing cgroup subsys cpuacct
Initializing cgroup subsys freezer
CPU: Testing write buffer coherency: ok
net_namespace: 936 bytes
NET: Registered protocol family 16
bio: create slab <bio-0> at 0
NET: Registered protocol family 2
IP route cache hash table entries: 32768 (order: 5, 131072 bytes)
TCP established hash table entries: 131072 (order: 8, 1048576 bytes)
TCP bind hash table entries: 65536 (order: 6, 262144 bytes)
TCP: Hash tables configured (established 131072 bind 65536)
TCP reno registered
NET: Registered protocol family 1
checking if image is initramfs... it is
Freeing initrd memory: 160K
goldfish_new_pdev goldfish_interrupt_controller at ff000000 irq -1
goldfish_new_pdev goldfish_device_bus at ff001000 irq 1
goldfish_new_pdev goldfish_timer at ff003000 irq 3
goldfish_new_pdev goldfish_rtc at ff010000 irq 10
goldfish_new_pdev goldfish_tty at ff002000 irq 4
goldfish_new_pdev goldfish_tty at ff011000 irq 11
goldfish_new_pdev goldfish_tty at ff012000 irq 12
goldfish_new_pdev smc91x at ff013000 irq 13
goldfish_new_pdev goldfish_fb at ff014000 irq 14
goldfish_new_pdev goldfish_audio at ff004000 irq 15
goldfish_new_pdev goldfish_memlog at ff006000 irq -1
goldfish_new_pdev goldfish-battery at ff015000 irq 16
goldfish_new_pdev goldfish_events at ff016000 irq 17
goldfish_new_pdev goldfish_nand at ff017000 irq -1
goldfish_new_pdev qemu_pipe at ff018000 irq 18
goldfish_new_pdev goldfish-switch at ff01a000 irq 19
```

```
goldfish_pdev_worker registered goldfish_interrupt_controller
goldfish_pdev_worker registered goldfish_device_bus
goldfish_pdev_worker registered goldfish_timer
goldfish_pdev_worker registered goldfish_rtc
goldfish_pdev_worker registered goldfish_tty
goldfish_pdev_worker registered goldfish_tty
goldfish_pdev_worker registered goldfish_tty
goldfish_pdev_worker registered smc91x
goldfish_pdev_worker registered goldfish_fb
goldfish_pdev_worker registered goldfish_audio
goldfish_pdev_worker registered goldfish_memlog
goldfish_pdev_worker registered goldfish-battery
goldfish_pdev_worker registered goldfish_events
goldfish_pdev_worker registered goldfish_nand
goldfish_pdev_worker registered qemu_pipe
goldfish_pdev_worker registered goldfish-switch
goldfish_pdev_worker registered goldfish-switch
ashmem: initialized
Installing knfsd (copyright (C) 1996 okir@monad.swb.de).
yaffs Sep 26 2012 11:04:43 Installing.
msgmni has been set to 1642
alg: No test for stdrng (krng)
io scheduler noop registered
io scheduler anticipatory registered (default)
io scheduler deadline registered
io scheduler cfq registered
allocating frame buffer 1280 * 720, got ffa00000
console [ttyS0] enabled
brd: module loaded
loop: module loaded
nbd: registered device at major 43
goldfish_audio_probe
tun: Universal TUN/TAP device driver, 1.6
tun: (C) 1999-2004 Max Krasnyansky <maxk@qualcomm.com>
smc91x.c: v1.1, sep 22 2004 by Nicolas Pitre <nico@cam.org>
eth0 (smc91x): not using net_device_ops yet
eth0: SMC91C11xFD (rev 1) at f480c000 IRQ 13 [nowait]
eth0: Ethernet addr: 52:54:00:12:34:56
goldfish nand dev0: size f560000, page 2048, extra 64, erase 131072
goldfish nand dev1: size 7c200000, page 2048, extra 64, erase 131072
goldfish nand dev2: size 4000000, page 2048, extra 64, erase 131072
mice: PS/2 mouse device common for all mice
*** events probe ***
events_probe() addr=0xf4814000 irq=17
events_probe() keymap=qwerty2
input: qwerty2 as /devices/virtual/input/input0
goldfish_rtc goldfish_rtc: rtc core: registered goldfish_rtc as rtc0
device-mapper: uevent: version 1.0.3
device-mapper: ioctl: 4.14.0-ioctl (2008-04-23) initialised: dm-
devel@redhat.com
logger: created 64K log 'log_main'
```

```
logger: created 256K log 'log_events'
logger: created 64K log 'log_radio'
Netfilter messages via NETLINK v0.30.
nf_conntrack version 0.5.0 (13312 buckets, 53248 max)
CONFIG_NF_CT_ACCT is deprecated and will be removed soon. Please use
nf_conntrack.acct=1 kernel paramater, acct=1 nf_conntrack module
option or
sysctl net.netfilter.nf_conntrack_acct=1 to enable it.
ctnetlink v0.93: registering with nfnetlink.
NF_TPROXY: Transparent proxy support initialized, version 4.1.0
NF_TPROXY: Copyright (c) 2006-2007 BalaBit IT Ltd.
xt_time: kernel timezone is -0000
ip_tables: (C) 2000-2006 Netfilter Core Team
arp_tables: (C) 2002 David S. Miller
TCP cubic registered
ip6_tables: (C) 2000-2006 Netfilter Core Team
IPv6 over IPv4 tunneling driver
NET: Registered protocol family 17
NET: Registered protocol family 15
RPC: Registered udp transport module.
RPC: Registered tcp transport module.
802.1Q VLAN Support v1.8 Ben Greear <greearb@candelatech.com>
All bugs added by David S. Miller <davem@redhat.com>
VFP support v0.3: implementor 41 architecture 3 part 30 variant c
rev 0
goldfish_rtc goldfish_rtc: setting system clock to 2013-10-06
21:42:47 UTC (1381095767)
Freeing init memory: 124K
init: cannot open '/initlogo.rle'
yaffs: dev is 32505856 name is "mtdblock0"
yaffs: passed flags ""
yaffs: Attempting MTD mount on 31.0, "mtdblock0"
yaffs_read_super: isCheckpointed 0
save exit: isCheckpointed 0
yaffs: dev is 32505857 name is "mtdblock1"
yaffs: passed flags ""
yaffs: Attempting MTD mount on 31.1, "mtdblock1"
hrtimer: interrupt too slow, forcing clock min delta to 26386311 ns
yaffs_read_super: isCheckpointed 0
yaffs: dev is 32505858 name is "mtdblock2"
yaffs: passed flags ""
yaffs: Attempting MTD mount on 31.2, "mtdblock2"
yaffs_read_super: isCheckpointed 0
eth0: link up
shell@android:/ $ warning: `rild' uses 32-bit capabilities (legacy
support in use)
goldfish_fb_pan_display: timeout wating for base update
request_suspend_state: wakeup (3->0) at 120060793619 (2013-10-06
21:44:46.941441593 UTC)
init: untracked pid 33 exited
init: untracked pid 39 exited
```

```
request_suspend_state:  wakeup  (0->0)  at  165640770376  (2013-10-06
21:45:32.521654980 UTC)
init: untracked pid 146 exited
init: untracked pid 145 exited
request_suspend_state:  wakeup  (0->0)  at  212188669882  (2013-10-06
21:46:19.069552871 UTC)
init: untracked pid 216 exited
init: untracked pid 215 exited
```

Bibliografía

[1] "Open Handset Alliance," 12 11 2007. [Online]. Available: http://www.openhandsetalliance.com/. [Accessed 12 03 2013].

[2] Lookout, "State of Mobile Security," 2012. [Online]. Available: https://www.lookout.com/static/ee_images/lookout-state-of-mobile-security-2012.pdf. [Accessed 10 07 2013].

[3] K. Parmar, "In Depth : Android Boot Sequence / Process," 08 11 2012. [Online]. Available: http://www.kpbird.com/2012/11/in-depth-android-boot-sequence-process.html. [Accessed 02 04 2013].

[4] K. Parmar, "In Depth : Android Boot Sequence / Process," 08 11 2012. [Online]. Available: http://www.kpbird.com/2012/11/in-depth-android-boot-sequence-process.html. [Accessed 20 03 2013].

[5] Google, "Android Debug Bridge," [Online]. Available: http://developer.android.com/tools/help/adb.html. [Accessed 15 03 2013].

[6] "ClockworkMod," [Online]. Available: http://www.clockworkmod.com/. [Accessed 20 03 2013].

[7] Desconocido, "HOWTO: Unpack, Edit, and Re-Pack Boot Images," 1 2 2013. [Online]. Available: http://android-dls.com/wiki/index.php?title=HOWTO:_Unpack,_Edit,_and_Re-Pack_Boot_Images. [Accessed 20 03 2013].

[8] W. Enck, "William Enck - Tools," [Online]. Available: http://www.enck.org/tools.html. [Accessed 16 03 2013].

[9] tvidas, "Live View," Carnegie Mellon University, [Online]. Available: http://liveview.sourceforge.net/.

[10] Google, "Android Emulator," [Online]. Available: http://developer.android.com/tools/help/emulator.html.

[11] A. Hoog, Android Forensics: Investigation, Analysis, and Mobile Security for Google Android, ELSEVIER, 2011.

[12] J. Lessard and G. C. Kessler, "Android Forensics: Simplifying Cell Phone Examinations VOL. 4, NO.1," Septiembre 2010.

[13] W. Jansen and R. Ayers, "Guidelines on Cell Phone Forensics, Recommendations of the National Institute of Standards and Technology, Special Publication 800-101," [Online]. Available: http://csrc.nist.gov/publications/nistpubs/800-101/SP800-101.pdf.

[14] M. Bjornheden, "The Android boot process from power on," 11 06 2009. [Online]. Available: http://www.androidenea.com/2009/06/android-boot-process-from-power-on.html. [Accessed 01 03 2013].

[15] V. VIJAYAN, "Android Forensic Capability and Evaluation of Extraction Tools," 04 2012. [Online]. Available: http://academia.edu/1632597/Android_Forensic_Capability_and_Evaluation_of_Extraction_Tools. [Accessed 10 03 2013].

[16] Laurent, "Xoom," 15 03 2012. [Online]. Available: http://p.quinput.eu/qwiki/Wiki.jsp?page=Xoom. [Accessed 25 03 2013].

[17] "Under the hood of Android Emulator (appcert)," 21 07 2011. [Online]. Available: https://wiki.diebin.at/Under_the_hood_of_Android_Emulator_(appcert). [Accessed 15 04 2013].

[18] "ISO/IEC 27037:2012 Information technology -- Security techniques -- Guidelines for identification, collection, acquisition and preservation of digital evidence," ISO, 15 10 2012. [Online]. Available: http://www.iso.org/iso/catalogue_detail?csnumber=44381. [Accessed 01

03 2013].