UCLM
UNIVERSIDAD DE CASTILLA-LA MANCHA

Escuela Superior Ingeniería Informática

DSI
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS

XUNTA DE GALICIA
CONSELLERÍA DE CULTURA, EDUCACIÓN E UNIVERSIDADE

CFR FERROL

Puesta en producción segura

Tema 5. Detección de problemas de seguridad en aplicaciones para dispositivos móviles.

UNIÓN EUROPEA
Fondo Social Europeo
EL FSE invierte en tu futuro

GOBIERNO DE ESPAÑA
MINISTERIO DE EDUCACIÓN Y FORMACIÓN PROFESIONAL

Índice

- Arquitectura de los sistemas Android e iOS
- Particiones y sistema de ficheros Android e iOS
- Estructura y creación de aplicaciones Android e iOS
- Componentes y comunicaciones de aplicaciones Android e iOS
- OWASP Mobile TOP 10
- Características de seguridad de Android e iOS
- Ejercicios prácticos (PoCs)
- Análisis estático
- Análisis dinámico
- Análisis de red

Objetivos

- Conocer la arquitectura de los principales dispositivos móviles.
- Conocer la estructura y proceso de creación de las aplicaciones móviles.
- Conocer los componentes de las aplicaciones móviles.
- Enumerar y comprender las principales vulnerabilidades móviles.
- Conocer el modelo de seguridad de Android e iOS y sus características.
- Conocer los distintos tipos de análisis que pueden realizarse a una aplicación.
- Aplicar los conceptos y técnicas en un entorno práctico controlado.

UCLM
UNIVERSIDAD DE CASTILLA-LA MANCHA
Escuela Superior Ingeniería Informática
DSI
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS
XUNTA DE GALICIA
CONSELLERÍA DE CULTURA, EDUCACIÓN E UNIVERSIDADE
CFR FERROL

Tema 5. Detección de problemas de seguridad en aplicaciones para dispositivos móviles

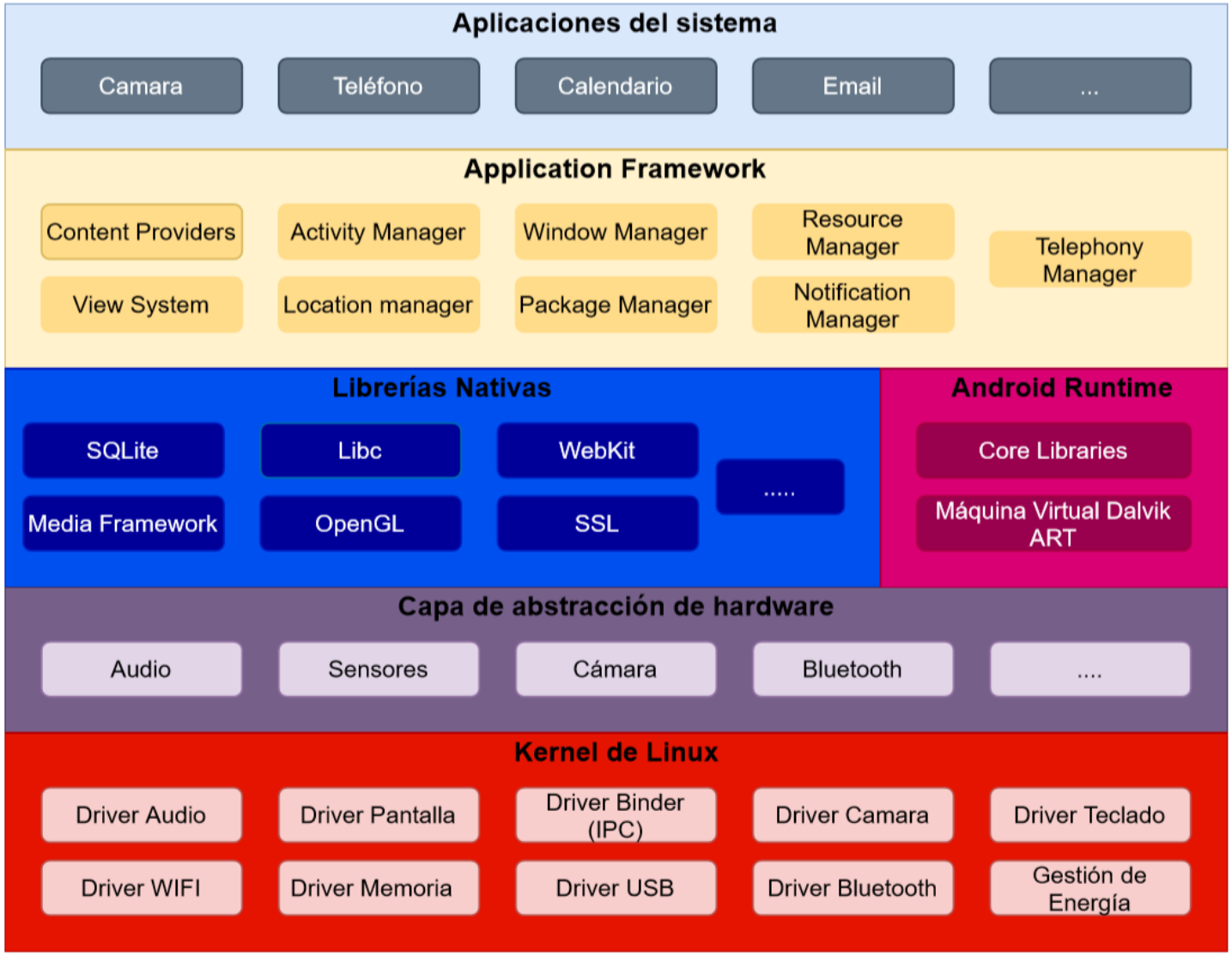
Arquitectura de los sistemas Android e iOS.

UNIÓN EUROPEA
Fondo Social Europeo
EL FSE invierte en tu futuro
GOBIERNO DE ESPAÑA
MINISTERIO DE EDUCACIÓN Y FORMACIÓN PROFESIONAL

Android

- Disponible para distintos tipos de arquitecturas → x86, x86_64, ARM…
- Basado en el kernel de Linux.
- Aplicaciones ejecutadas por el Android Runtime o ART (sucesor de Dalvik).
- Soporta varios sistemas de ficheros → EXT4, JFFS2, YAFFS2…
- Soporta cifrado de disco.
- Es open source, basado en una licencia Apache 2.0.

Arquitectura del sistema
Android



Java API Framework
Android

- Proporciona acceso a los desarrolladores a distintas funcionalidades de la plataforma:
  - **Activity Manager**: se encarga de gestionar las distintas actividades que aparecen por pantalla. Gestiona el ciclo de vida de una actividad.
  - **Content Providers**: proporciona una interfaz para almacenar y acceder a los datos de las aplicaciones. Las aplicaciones pueden crear content providers y exponer sus datos a otras apliaciones.
    - Ejemplo: acceso de WhatsApp a tus contactos.
  - **Notification Manager**: para gestionar y notificar al usuario los eventos que ocurren en las aplicaciones (información de la barra de estado y panel de notificaciones).
  - **Location Manager**: proporciona acceso a la información relacionada con la ubicación (mediante GPS, WIFI, etc).

Java API Framework

Android

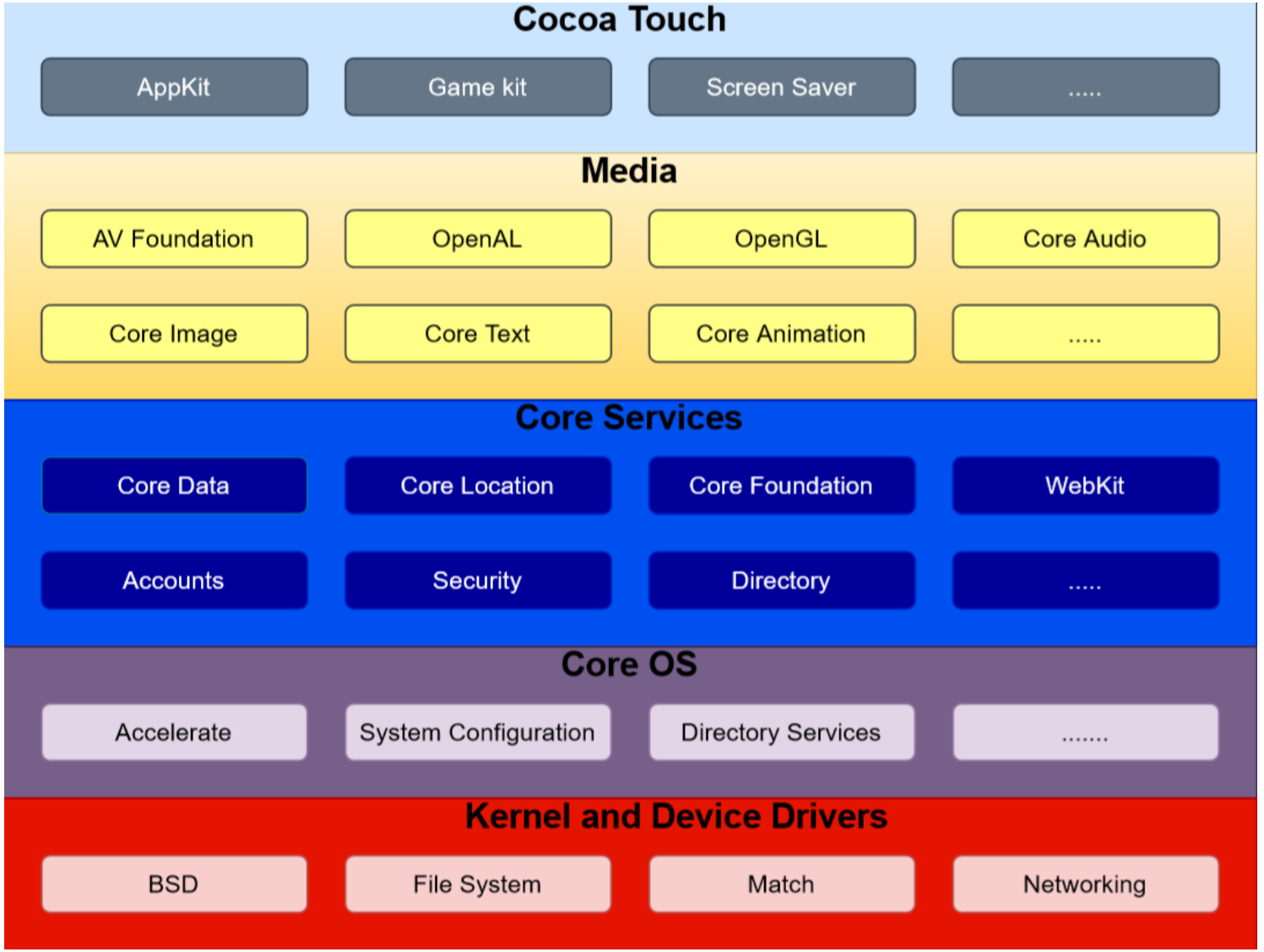
- **View System**: se encarga de gestionar las diferentes vistas (interfaces de usuario) que presenta una aplicación.
  - Listas, cuadrículas, botones, menús desplegables, cuadros de texto…
- **Package Manager**: gestiona la información relacionada con los paquetes instalados en el sistema.
- **Telephony Manager**: gestiona la información relacionada con los servicios de telefonía.
  - Subscriber ID, SIM serial number, tipo de red móvil, roaming…

- Más información sobre librerías de Android: https://developer.android.com/reference/packages

iOS

- Soporte para la arquitectura ARM.
- Está basado en el kernel de OS X → BSD (derivado de Unix).
- Sistema de ficheros HFSX hasta iOS 10.3 y APFS en la actualidad.
- Cifrado de disco por defecto (FileVault).
- Su licencia es privada e impide la instalación en cualquier equipo que no sea vendido por Apple.

Arquitectura del Sistema
iOS



API Framework
iOS

- Conjunto de librerías por defecto para la creación de aplicaciones:
  - **UIkit**: para la creación de interfaces de usuario. Se encarga del ciclo de vida de la aplicación, eventos de interacción del usuario, etc.
  - **CFNetwork:** librerías relacionadas para la creación de conexiones de red (HTTP, DNS, FTP, etc).
  - **Core Location:** librerías relacionadas con la ubicación del dispositivo.
  - **Core Data:** librerías para el almacenamiento persistente de los datos de las aplicaciones (XML, archivos binarios o SQLIte).
  - **Core Motion**: librerías relacionadas con el acceso a los datos que son generados por los sensores del dispositivo (giroscopio, acelerómetro, etc).
- Más información sobre las librerías de iOS: https://developer.apple.com/documentation/technologies

UCLM
UNIVERSIDAD DE CASTILLA-LA MANCHA

Escuela Superior Ingeniería Informática

DSI
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS

XUNTA DE GALICIA
CONSELLERÍA DE CULTURA, EDUCACIÓN E UNIVERSIDADE

CFR FERROL

Tema 5. Detección de problemas de seguridad en aplicaciones para dispositivos móviles

Particiones y sistema de ficheros Android e iOS.

UNIÓN EUROPEA
Fondo Social Europeo
EL FSE invierte en tu futuro

GOBIERNO DE ESPAÑA
MINISTERIO DE EDUCACIÓN Y FORMACIÓN PROFESIONAL

Particiones
Android

Particiones
Android

- **Boot**: partición de arranque del sistema. Contiene el kernel.
- **System**: partición del sistema operativo. Contiene los archivos del SO y las aplicaciones del sistema que vienen preinstaladas.
- **Recovery**: partición de recuperación. Alternativa a la partición boot que permite al dispositivo iniciar en modo “recovery”.
- **Data**: partición que contiene los datos de usuario como las aplicaciones instaladas, contactos, mensajes, etc.

Particiones
Android

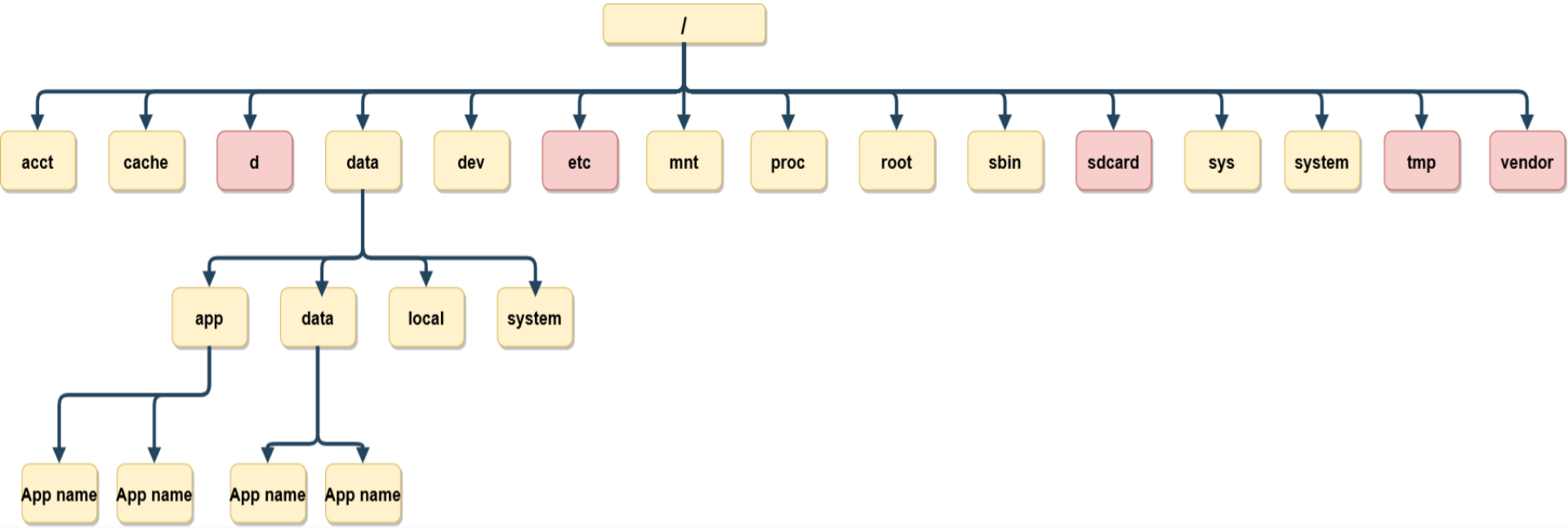
- **Cache**: para almacenar datos y componentes que son accedidos con frecuencia.

- **Misc**: para almacenar diversas opciones del sistema y aspectos relacionados con el hardware.

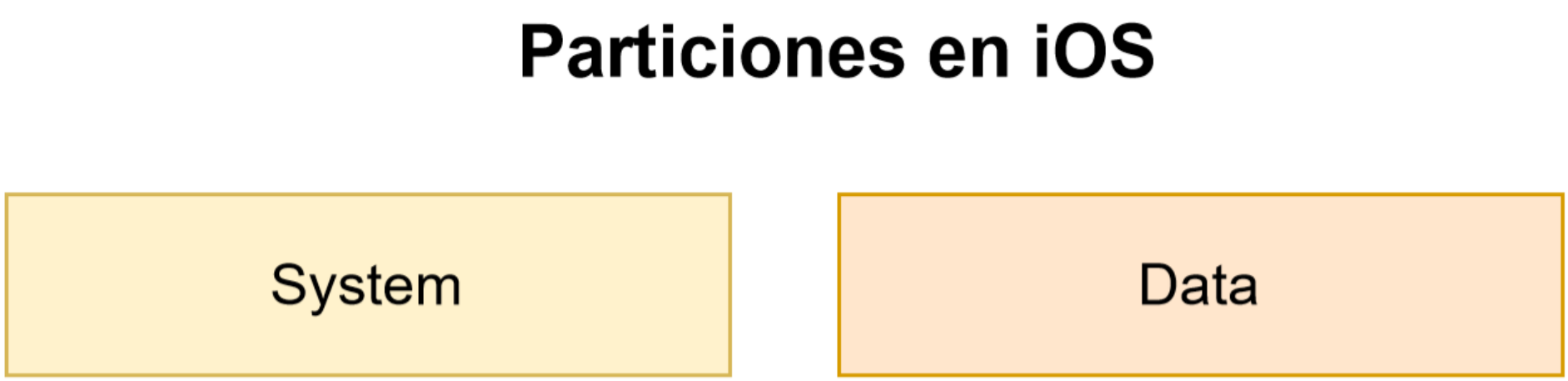
- **Sdcard**: está partición no pertenece a la memoria interna del teléfono. Para almacenar documentos, imágenes, etc.

- **Sd-ext**: no es una partición estándar pero es común en las ROMs personalizadas de Android.

Sistema de ficheros
Android



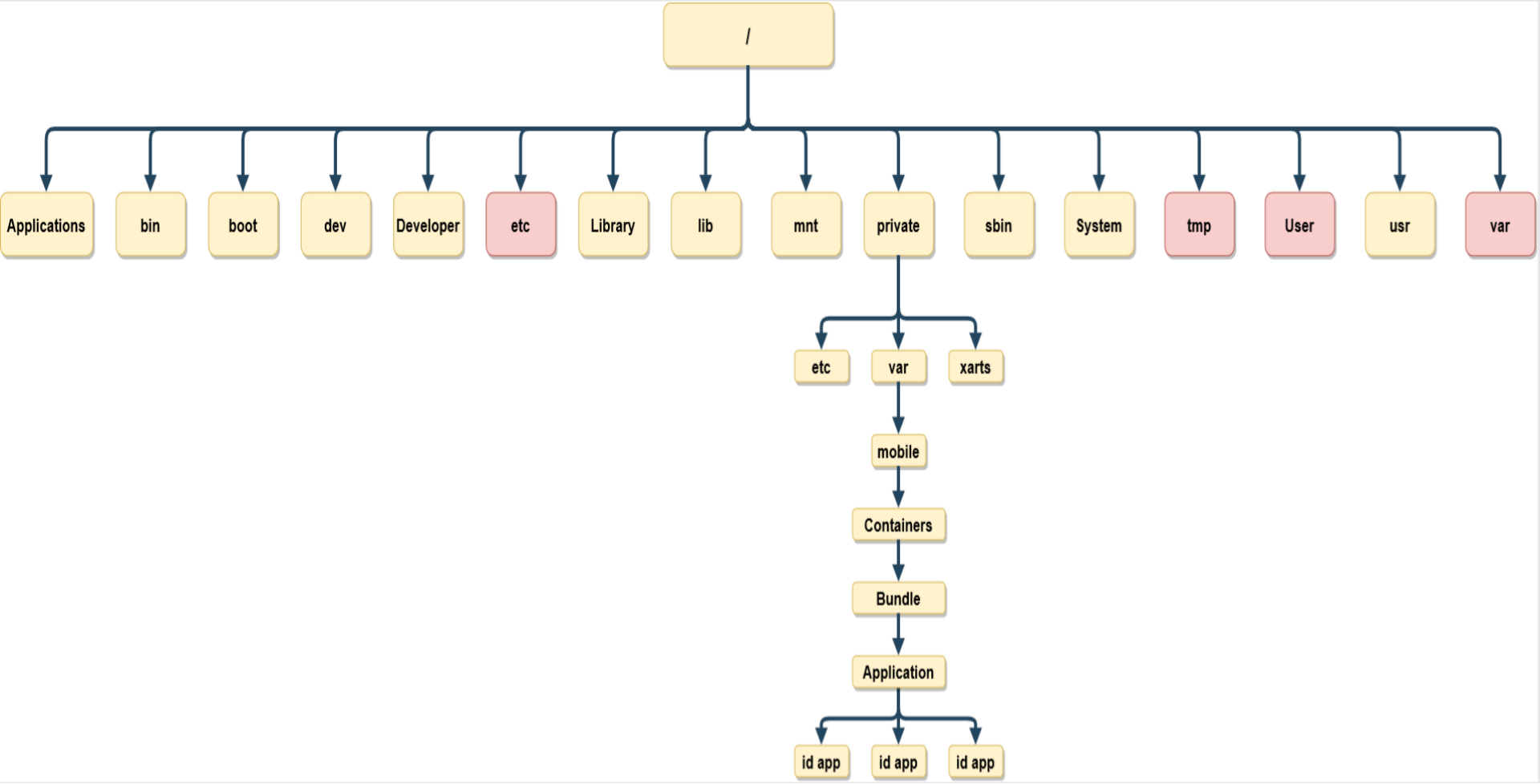
Particiones
iOS



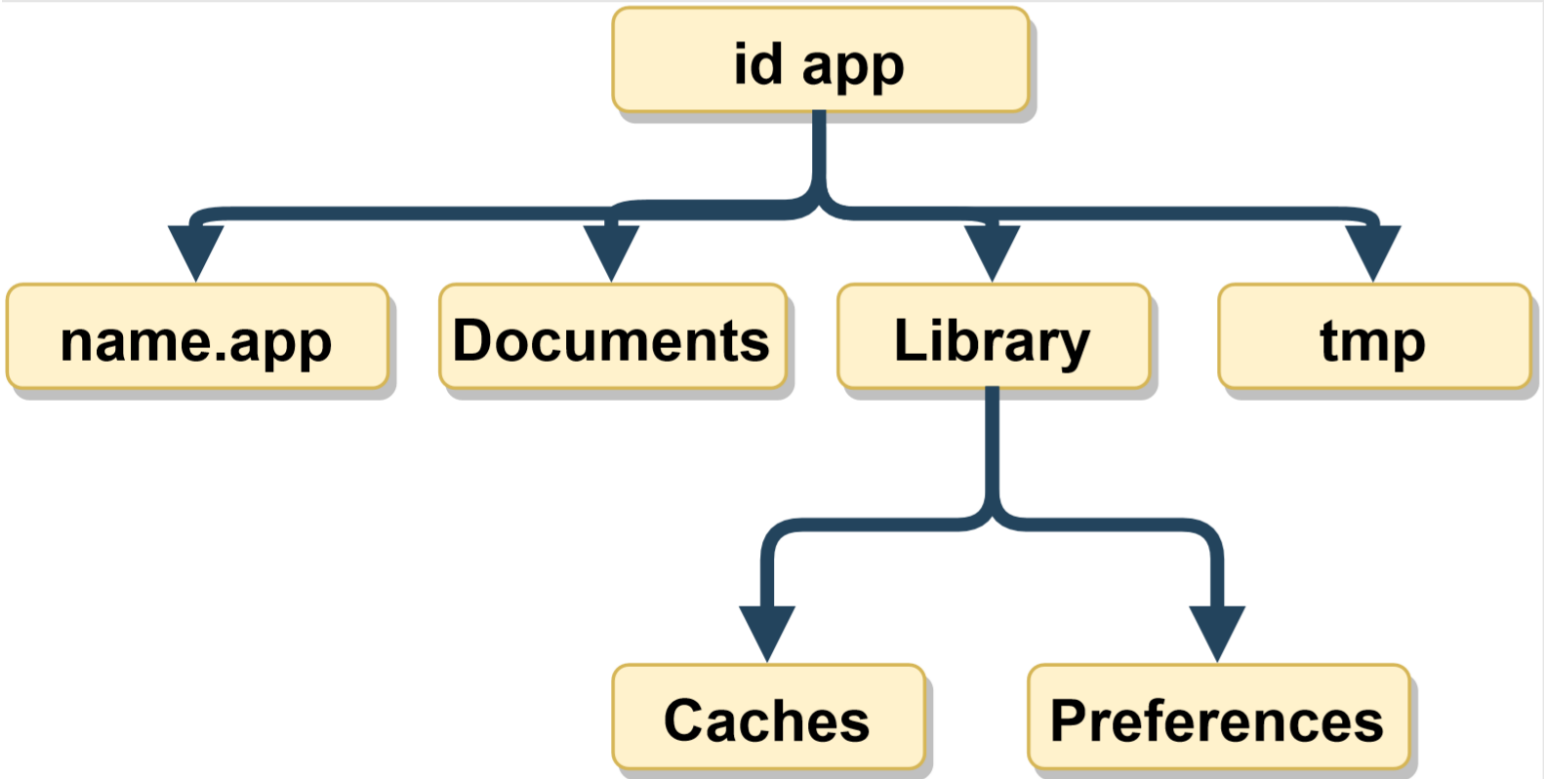
- **System**: partición de solo lectura que contiene el firmware y archivos del sistema operativo. También alberga las aplicaciones básicas que vienen preinstaladas.

- **Data**: contiene los datos del usuario y de las aplicaciones.

Sistema de ficheros
iOS



Sistema de ficheros app
iOS



Sistema de ficheros app
iOS

- ***.app**: directorio que contiene la aplicación y todos los recursos que utilice (multimedia).
  - Es de solo lectura → se encuentra firmado para evitar manipulaciones.

- **Documents**: almacena los datos generados por el usuario en la aplicación. Solo debe contener archivos que se quieran exponer al usuario, ya que este directorio permite la compartición.

- **Library**: directorio con archivos de una aplicación que no se desean exponer al usuario.

- **tmp**: almacena archivos temporales que no se necesitarán entre diferentes ejecuciones de una aplicación.

UCLM
UNIVERSIDAD DE CASTILLA-LA MANCHA

Escuela Superior Ingeniería Informática

DSI
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS

XUNTA DE GALICIA
CONSELLERÍA DE CULTURA, EDUCACIÓN E UNIVERSIDADE

CFR FERROL

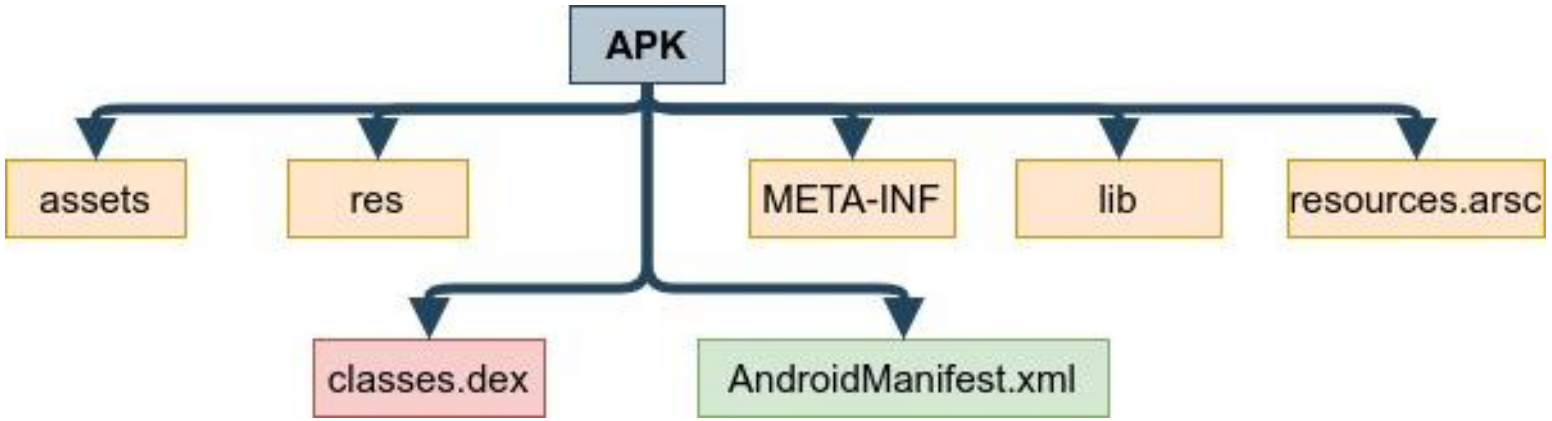
Tema 5. Detección de problemas de seguridad en aplicaciones para dispositivos móviles

Estructura y creación de aplicaciones Android e iOS.

UNIÓN EUROPEA
Fondo Social Europeo
EL FSE invierte en tu futuro

GOBIERNO DE ESPAÑA
MINISTERIO DE EDUCACIÓN Y FORMACIÓN PROFESIONAL

Estructura de una aplicación
Android



- **Android Application Package (APK)**: paquete software que engloba todos los recursos relacionados con una aplicación Android. Es una variante del formato JAR de Java.
  - Extensión → ***.apk***

Estructura de una aplicación
Android

- **assets**: directorio con los recursos extra utilizados por la aplicación (fuentes, archivos multimedia, etc).

- **res**: directorio con los recursos Android no compilados de la aplicación (animaciones, estilos, layouts, strings, etc).

- **META-INF**: directorio con metadatos de la aplicación (certificado, firma SHA-1, lista de recursos, etc).

- **lib**: directorio donde se almacenan las librerías nativas utilizadas por la aplicación para plataformas específicas (x86, ARM, etc) → subdirectorio para cada arquitectura.

Estructura de una aplicación
Android

- **resources.arsc**: directorio con recursos pre-compilados de la aplicación (cadenas de texto, estilos, etc).

- **classes.dex**: código compilado (bytecode) de la aplicación en formato Dalvik Executable (***.dex***) → ejecutado por el Android Runtime.
  - Puede ser convertido a una representación legible (Smali) desensamblando el .dex.

Estructura de una aplicación
Android

```
.method public static aes256encrypt([B[B[B)[B
    .locals 4
    .param p0, "ivBytes"    # [B
    .param p1, "keyBytes"    # [B
    .param p2, "textBytes"    # [B
    .annotation system Ldalvik/annotation/Throws;
        value = {
            Ljava/io/UnsupportedEncodingException;,
            Ljava/security/NoSuchAlgorithmException;,
            Ljavax/crypto/NoSuchPaddingException;,
            Ljava/security/InvalidKeyException;,
            Ljava/security/InvalidAlgorithmParameterException;,
            Ljavax/crypto/IllegalBlockSizeException;,
            Ljavax/crypto/BadPaddingException;
        }
    .end annotation

    .prologue
    .line 52
    new-instance v1, Ljavax/crypto/spec/IvParameterSpec;

    invoke-direct {v1, p0}, Ljavax/crypto/spec/IvParameterSpec;-><init>([B)V

    .line 53
    .local v1, "ivSpec":Ljava/security/spec/AlgorithmParameterSpec;
    new-instance v2, Ljavax/crypto/spec/SecretKeySpec;

    const-string v3, "AES"

    invoke-direct {v2, p1, v3}, Ljavax/crypto/spec/SecretKeySpec;-><init>([BLjava/lang/String;)V
```

Estructura de una aplicación
Android

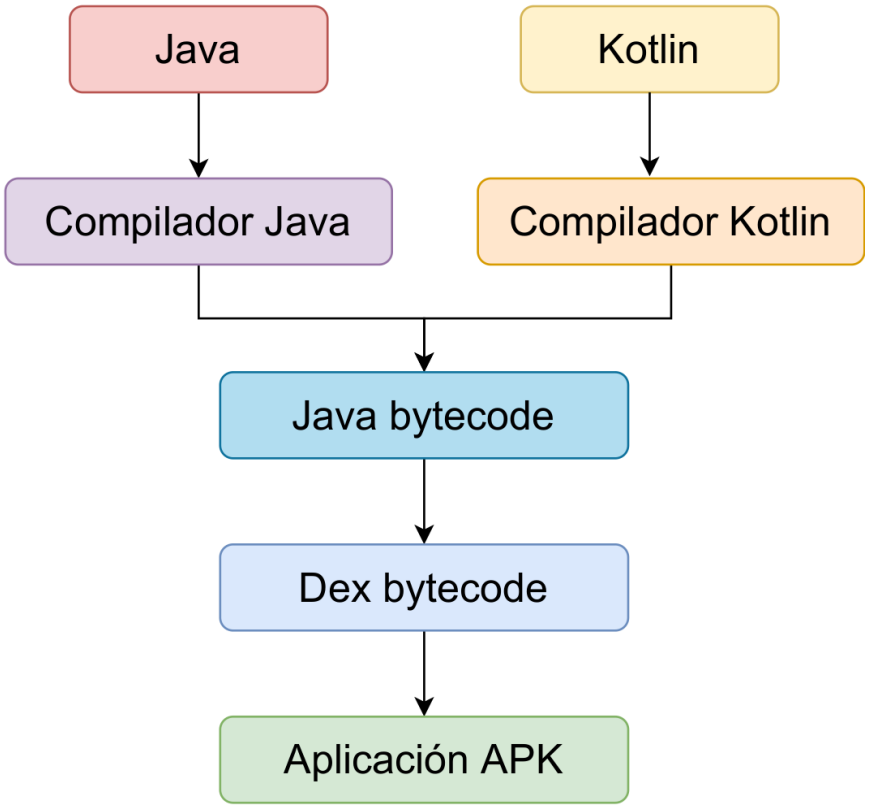
- **AndroidManifest.xml:** fichero que contiene toda la información que necesita la aplicación para su correcta instalación y ejecución:
  - Versión mínima soportada.
  - Nombre de la aplicación y versión.
  - Permisos de la aplicación que se solicitarán al usuario.
  - Librerías del sistema necesarias.
  - Componentes que incluye la aplicación → actividades servicios, etc.

Estructura de una aplicación

Android

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.android.insecurebankv2" >

    <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) as part of sign-in: -->
    <uses-permission android:name="android.permission.GET_ACCOUNTS" /> <!-- To auto-complete the email text field in the login form with the user's emails -->
    <uses-permission android:name="android.permission.READ_PROFILE" />
    <uses-permission android:name="android.permission.READ_CONTACTS" />

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

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@android:style/Theme.Holo.Light.DarkActionBar">
        <!--
        android:theme="@style/AppTheme"-->
        <activity
            android:name=".LoginActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
```

Creación de ficheros *.apk*
Android



- El código fuente escrito en lenguaje Java o Kotlin es compilado produciendo archivos ***.class*** con el bytecode de Java → se almacenan/empaquetan en un fichero ***.jar***
- El bytecode de Java se trasforma en bytecode ***.dex*** que es el código que ejecutará el Android Runtime o la máquina virtual Dalvik.
- Finalmente, este fichero ***.dex*** junto con los distintos recursos de la aplicación, el AndroidManifest, etc, se comprime en formato ***.zip*** → **.apk**
- Si se desea publicar la aplicación en Google Play, esta debe firmarse con un certificado.

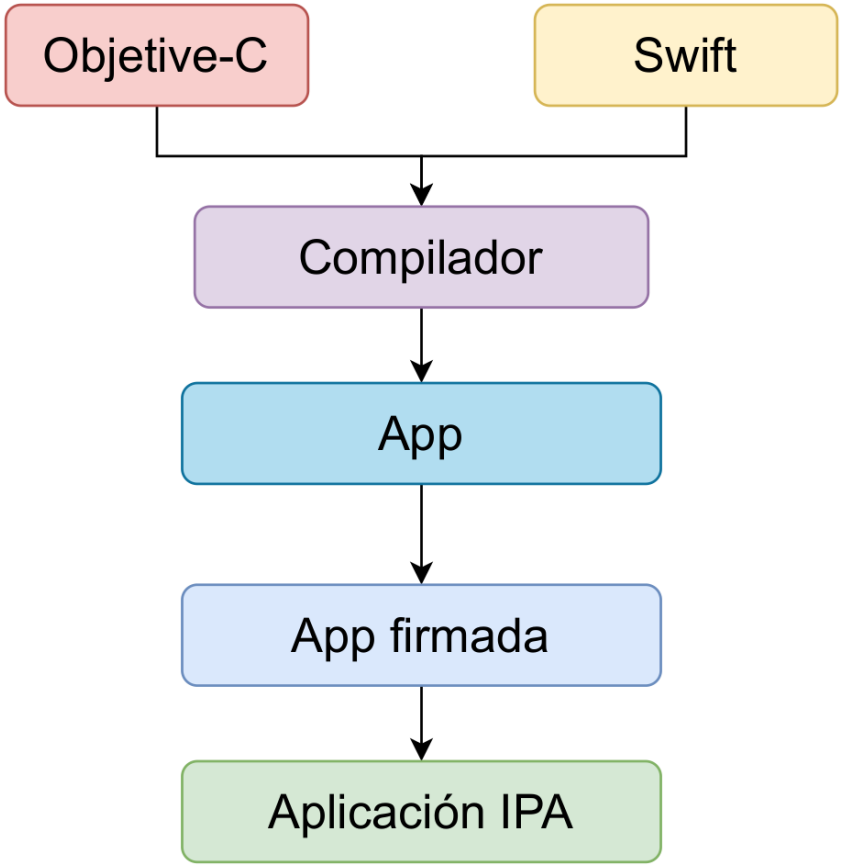
Estructura de una aplicación
iOS

- Empaquetada en un archivo comprimido ***.zip*** con la extensión ***.ipa (iOS App Store Package)***. El paquete IPA se descompone en varios ficheros y directorios entre los que se destacan:
  - **META-INF**: contiene archivos con los metadatos del fichero IPA.
  - **iTunesArtWork**: contiene la imagen que será utilizada por la App Store para visualizar la aplicación (icono).
  - **iTunesMetadata.plist**: contiene los metadatos de la aplicación que serán utilizados por la App Store:
    - ID del desarrollador, el nombre del autor, versión de la aplicación, etc.

Estructura de una aplicación
iOS

- **Payload**: contiene un directorio con el nombre de la aplicación (acabado en .app). Este a su vez contiene:
  - El código ejecutable de la aplicación.
  - Los distintos recursos que utiliza la aplicación.
  - Directorio *_CodeSignature* con un único archivo donde se almacena la firma del código de la aplicación → si el código fuese modificado, la aplicación dejaría de funcionar.
  - Fichero *Info.plist* con la información necesaria (atributos clave-valor) para la instalación y ejecución de la aplicación.

Creación de ficheros *.ipa*
iOS



- El código fuente de la aplicación (Objective-C o Swift) se compila para la arquitectura ARM.
- Se genera el paquete de la aplicación con su código ejecutable y todos los recursos que utiliza.
- Para que se pueda distribuir a través de la tienda de Apple, la aplicación tiene que ser firmada por el desarrollador.

UCLM
UNIVERSIDAD DE CASTILLA-LA MANCHA

Escuela Superior Ingeniería Informática

DSI
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS

XUNTA DE GALICIA
CONSELLERÍA DE CULTURA, EDUCACIÓN E UNIVERSIDADE

CFR FERROL

Tema 5. Detección de problemas de seguridad en aplicaciones para dispositivos móviles

Componentes y comunicaciones de aplicaciones Android e iOS.

UNIÓN EUROPEA
Fondo Social Europeo
EL FSE invierte en tu futuro

GOBIERNO DE ESPAÑA
MINISTERIO DE EDUCACIÓN Y FORMACIÓN PROFESIONAL

Componentes de una app Android

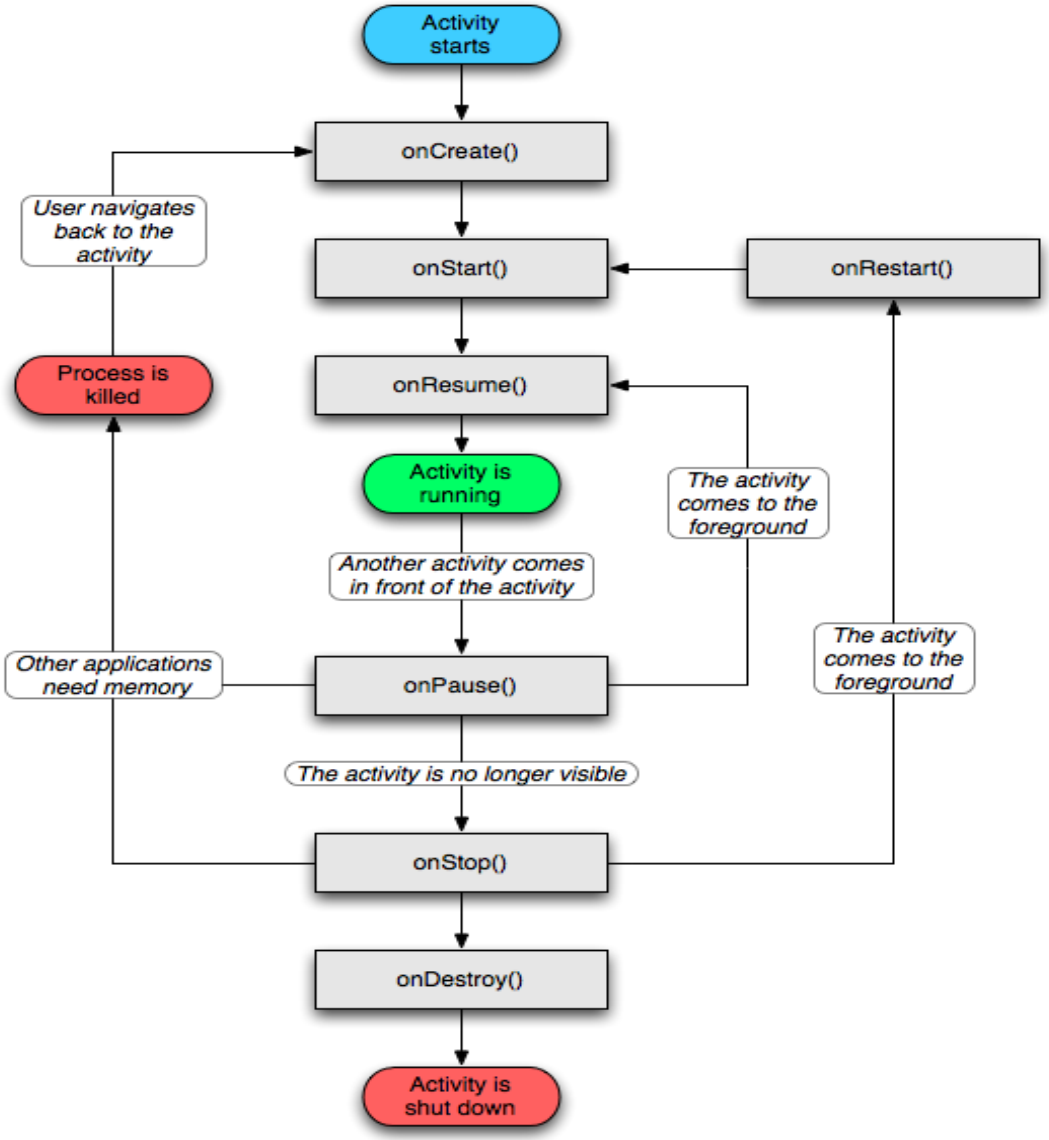
- Podemos ver los componentes de una aplicación como bloques funcionales esenciales
  - Tanto el usuario como el sistema son capaces de interactuar con estos bloques y, en definitiva, con la aplicación.

- Los componentes fundamentales son:
  - Actividades.
  - Servicios.
  - Receptores de emisiones (broadcast receivers).
  - Proveedores de contenido (content providers).

Actividades en Android

- Una **actividad** en Android es un punto de interacción con el usuario.
- Puede verse como una pantalla de interacción (interfaz de usuario) que puede ser llamada, si los permisos lo permiten, por otra actividad.
- Permiten separar las diferentes actividades o funcionalidades de una aplicación para el usuario.
  - Ejemplo de WhatsApp.
- Deben diferenciarse de los **fragmentos**: parte modular y reusable de la interfaz de usuario.

Ciclo de vida de actividades en Android



Ciclo de vida de actividades en Android

```
public class Activity extends ApplicationContext {
    protected void onCreate(Bundle savedInstanceState)

    protected void onStart();

    protected void onRestart();

    protected void onResume();

    protected void onPause();

    protected void onStop();

    protected void onDestroy();
}
```

Ciclo de vida de actividades en Android

```
public class CalendarActivity extends Activity {
    ...

    static final int DAY_VIEW_MODE = 0;
    static final int WEEK_VIEW_MODE = 1;

    private SharedPreferences mPrefs;
    private int mCurViewMode;

    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        mPrefs = getSharedPreferences(getLocalClassName(), MODE_PRIVATE);
        mCurViewMode = mPrefs.getInt("view_mode", DAY_VIEW_MODE);
    }

    protected void onPause() {
        super.onPause();

        SharedPreferences.Editor ed = mPrefs.edit();
        ed.putInt("view_mode", mCurViewMode);
        ed.commit();
    }
}
```

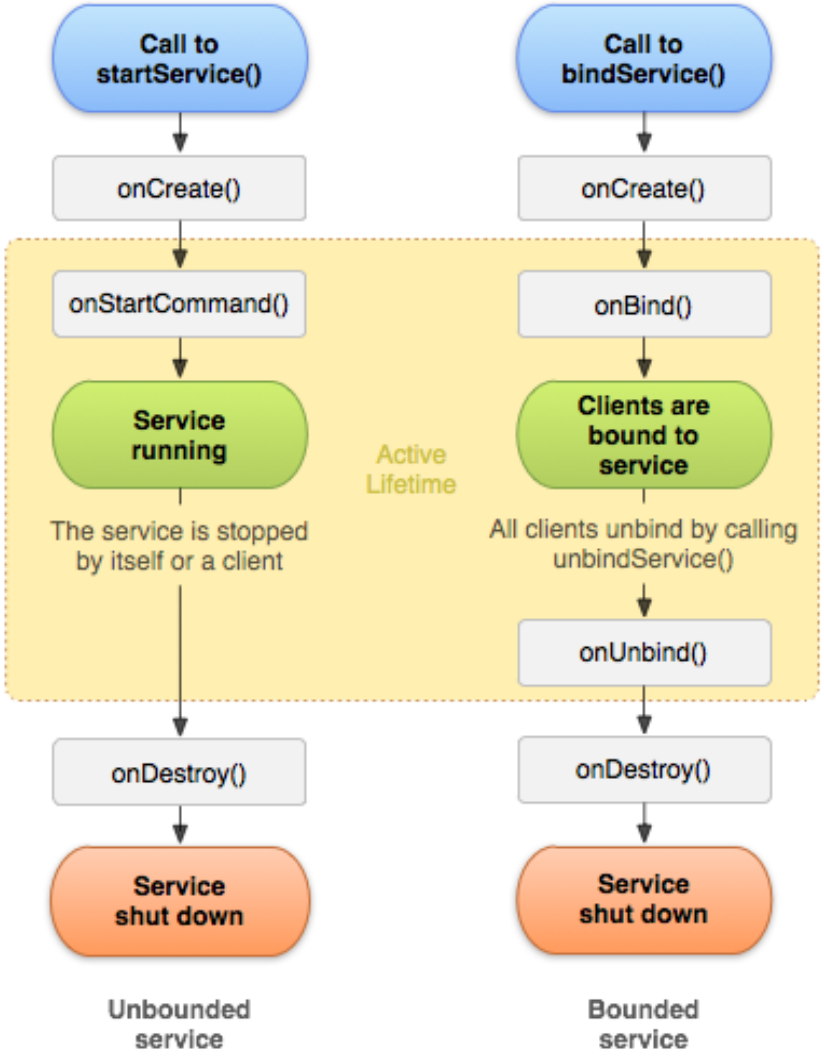
Servicios en Android

- Un **servicio** es un punto de entrada/componente/proceso que corre en segundo plano y que carece de interfaz gráfica.
- Pueden ser iniciados por e interactuar con otros componentes.
- Son empleados para realizar ciertas tareas largas y/o periódicas que no requieren de la interacción del usuario.
  - Ejemplos: notificaciones push, descarga de archivos, tareas de mantenimiento de una aplicación, reproducción de música en segundo plano, etc.

Servicios en Android

- Pueden ser de los siguientes tipos:
  - **Primer plano**: cuando es intrusivo en cierta medida (el usuario nota su presencia). Deben mostrar una notificación.
  - **Segundo plano**: cuando es transparente al usuario.
  - **Enlace**: cuando uno o varios componentes se vinculan al servicio → ofrece una interfaz cliente-servidor para que los componentes interactúen enviando solicitudes y recibiendo datos.
    - Método *bindService()*.

Ciclo de vida de servicios en Android



Ciclo de vida de servicios en Android

```
public class ExampleService extends Service {
    int startMode;       // indicates how to behave if the service is killed
    IBinder binder;      // interface for clients that bind
    boolean allowRebind; // indicates whether onRebind should be used

    @Override
    public void onCreate() {
        // The service is being created
    }
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        // The service is starting, due to a call to startService()
        return mStartMode;
    }
    @Override
    public IBinder onBind(Intent intent) {
        // A client is binding to the service with bindService()
        return mBinder;
    }
    @Override
    public boolean onUnbind(Intent intent) {
        // All clients have unbound with unbindService()
        return mAllowRebind;
    }
    @Override
    public void onRebind(Intent intent) {
        // A client is binding to the service with bindService(),
        // after onUnbind() has already been called
    }
    @Override
    public void onDestroy() {
        // The service is no longer used and is being destroyed
    }
}
```

Receptores de emisiones en Android

- Posibilita que las aplicaciones y el sistema operativo intercambien información en forma de eventos.
  - El componente se encarga de recibir y responder ante **eventos globales** (broadcast).
- Estas emisiones pueden enviarse incluso a aplicaciones que no estén en ejecución.
- Pese a no tener una interfaz asociada, pueden generar notificaciones en la barra de tareas.
- Extendemos la clase *BroadcastReceiver* y utilizamos los elementos *receiver* y *intent-filter* en el AndroidManifest.xml

Receptores de emisiones en Android

```java
public class MyBroadcastReceiver extends BroadcastReceiver {
        private static final String TAG = "MyBroadcastReceiver";
        @Override
        public void onReceive(Context context, Intent intent) {
            StringBuilder sb = new StringBuilder();
            sb.append("Action: " + intent.getAction() + "\n");
            sb.append("URI: " + intent.toUri(Intent.URI_INTENT_SCHEME).toString() + "\n");
            String log = sb.toString();
            Log.d(TAG, log);
            Toast.makeText(context, log, Toast.LENGTH_LONG).show();
        }
    }
```

```xml
<receiver android:name=".MyBroadcastReceiver"  android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.BOOT_COMPLETED"/>
        <action android:name="android.intent.action.INPUT_METHOD_CHANGED" />
    </intent-filter>
</receiver>
```

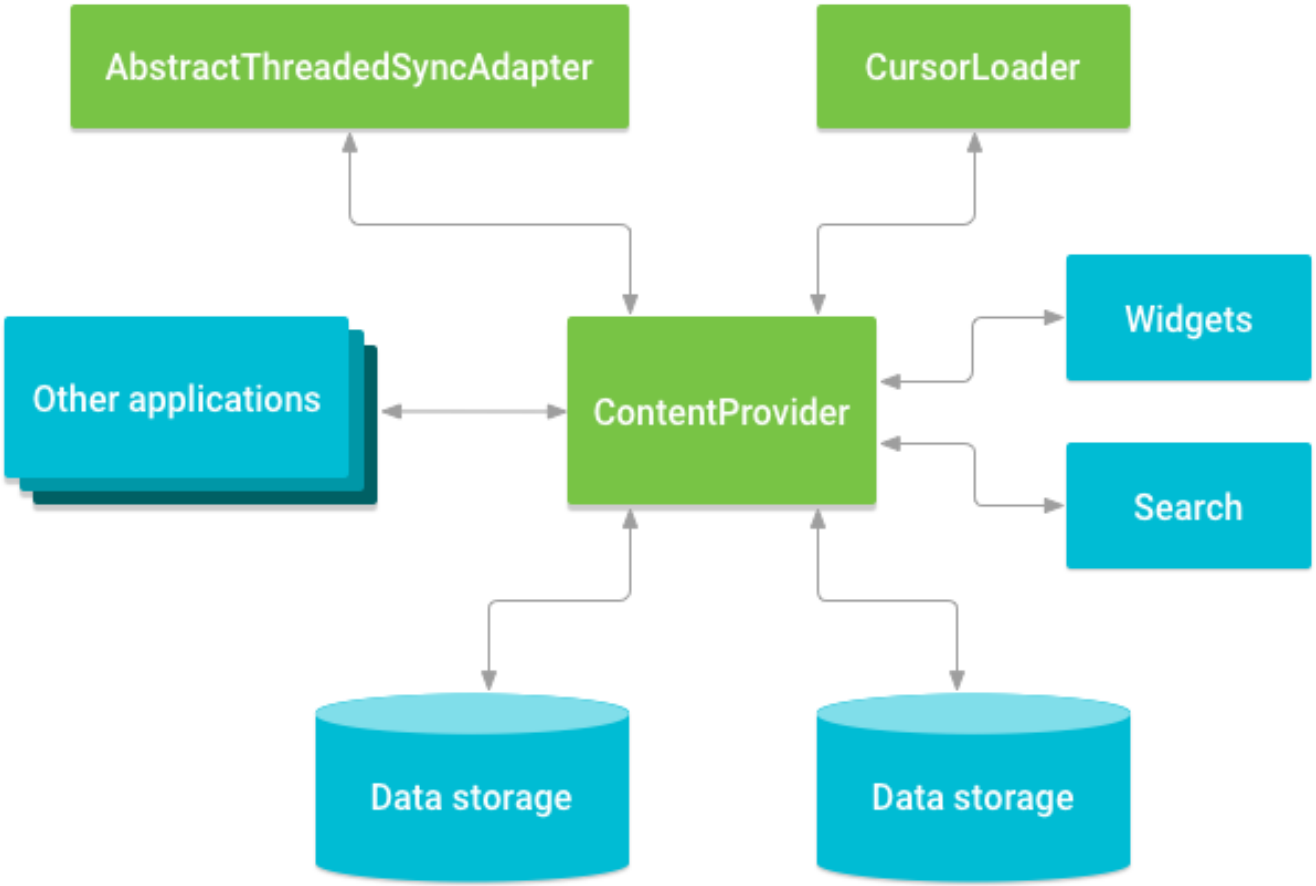
Proveedores de contenido en Android

- Los proveedores de contenido son un componente diseñado para **almacenar**, **gestionar** y **compartir** información.

- Por ejemplo, usamos un proveedor de contenido para acceder a los contactos (compartición de datos entre aplicaciones).

- El acceso a los datos se realiza mediante URIs (identificador de recursos uniforme).
  - Permiten gestionar de forma más segura los contenidos.

Proveedores de contenido en Android



Comunicación entre aplicaciones Android

- Ya que Android está basado en un núcleo UNIX, existen dos categorías de mecanismos para comunicar aplicaciones:
  - Heredados de **UNIX**:
    - Uso de sockets o el propio sistema de ficheros.
  - Exclusivos de **Android**:
    - Content providers.
      - Permiten realizar operaciones de escritura y lectura sobre datos de otras aplicaciones. Ejemplo: crear un contacto desde Telegram.
    - Intents.
      - Objeto de mensajería para lanzar servicios, actividades y broadcast receivers. Ejemplo: navegación entre actividades de tu aplicación.

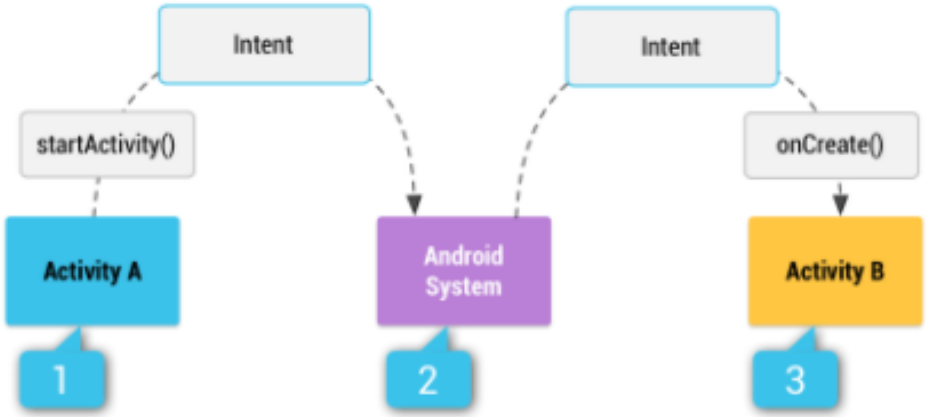
Intents en Android

- Los **Intents** son objetos empleados para solicitar una acción a otro componente. Suelen contener la siguiente información:
  - Información acerca de la operación a realizar.
  - Datos (parámetros) sobre los que realizar la operación.
  - Información adicional.

- No podemos enviar un Intent de forma directa, el sistema operativo lo hace por nosotros.

- Existen dos tipos de Intents:
  - **Explícitos**: se especifica la aplicación que gestionará el Intent (p. ej. para pasar entre actividades dentro de tu propia aplicación).
  - **Implícitos**: se especifica la acción general a llevar a cabo (p. ej. la lectura de un PDF y el listado de aplicaciones aptas para ello).

Intents en Android



- **1**. La actividad A crea el Intent y lo envía mediante el método *startActivity()*.
- **2**. Android busca la/s aplicación/es apta/s para procesar el Intent (según sea explícito o implícito).
- **3**. Android inicia la actividad concreta de la aplicación escogida mediante su método *onCreate()*.

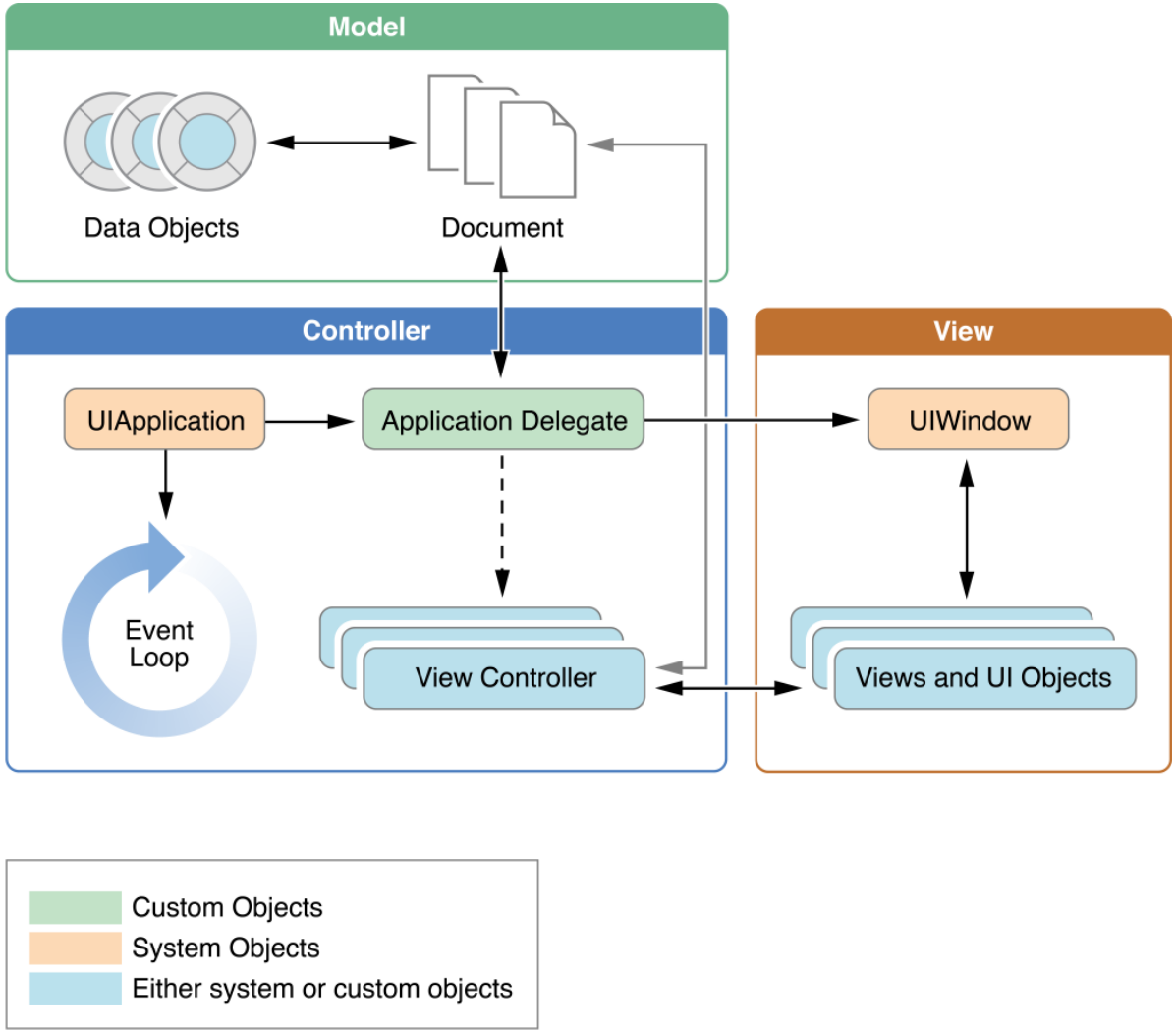
Intents en Android

```
// Create the text message with a string
Intent sendIntent = new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.putExtra(Intent.EXTRA_TEXT, textMessage);
sendIntent.setType("text/plain");

// Verify that the intent will resolve to an activity
if (sendIntent.resolveActivity(getPackageManager()) != null) {
    startActivity(sendIntent);
}
```

Componentes de una app iOS

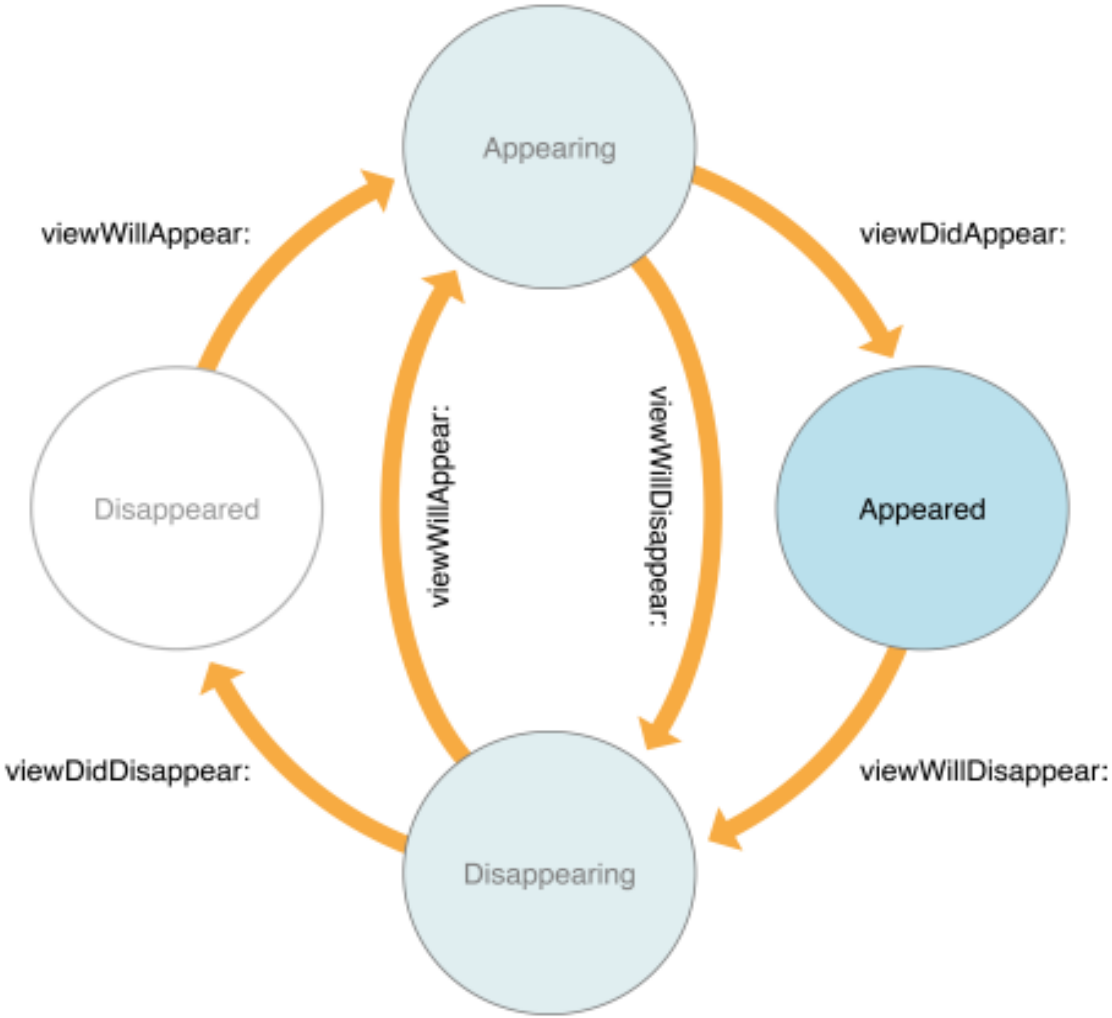
- Los componentes de una aplicación iOS son diferentes respecto a los de Android. Podemos distinguir los siguientes:
  - **AppDelegate**: objeto principal de una aplicación. Se encarga de gestionar el ciclo de vida de la aplicación y de mediar entre aplicación y sistema.
  - **ViewControlers**: se encargan de gestionar el ciclo de vida de las diferentes vistas o interfaces de usuario de una aplicación.
  - **Storyboard**: permite crear las interfaces de la aplicación y sus transiciones de forma visual…
    - …aunque también puede hacerse programáticamente (o mediante archivos *.xib*).

Componentes de una app iOS



Fuente: stackoverflow.com

Componentes de una app iOS



Fuente: medium.com

Componentes de una app iOS

Fuente: raywenderlich.com

Comunicación entre aplicaciones iOS

- El sistema iOS ofrece diferentes mecanismos para comunicar aplicaciones entre ellas. El usuario siempre debe autorizar dichas comunicaciones.
  - **AirDrop**: solo permite la comunicación entre dispositivos, no entre aplicaciones del mismo dispositivo.
  - **Esquemas URL**: parecidos a las URI de Android. Permiten enlazar contenido dentro de una aplicación.
  - **App Extensions**: permiten acceder a diferentes funcionalidades adicionales del sistema mediante APIs específicos (p. ej. File Provider).

UCLM
UNIVERSIDAD DE CASTILLA-LA MANCHA

Escuela Superior Ingeniería Informática

DSI
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS

XUNTA DE GALICIA
CONSELLERÍA DE CULTURA, EDUCACIÓN E UNIVERSIDADE

CFR FERROL

Tema 5. Detección de problemas de seguridad en aplicaciones para dispositivos móviles

OWASP Mobile TOP 10.

UNIÓN EUROPEA
Fondo Social Europeo
EL FSE invierte en tu futuro

GOBIERNO DE ESPAÑA
MINISTERIO DE EDUCACIÓN Y FORMACIÓN PROFESIONAL

OWASP Mobile Top 10

https://owasp.org/www-project-mobile-top-10

M1. Uso indebido de la plataforma

- Decimos que la vulnerabilidad se debe al uso indebido de la plataforma cuando:
  - No se utilizan las características de seguridad que implementa la plataforma.
  - Se emplean las características de seguridad pero se hace de forma indebida.

- Algunas características que pueden usarse de forma incorrecta son:
  - Intents en Android.
  - Mal uso del Touch ID o del Keychain en iOS.
  - Permisos de la plataforma.

M1. Uso indebido de la plataforma

- **Caso real**: bypass del Touch ID (autenticación) en la app Citrix Worx.
- La vulnerabilidad CVE-2016-5109 es un caso real de uso indebido de la plataforma.
- Podemos ver la PoC aquí:
  - **https://vimeo.com/167255641**
  - Minuto 00:20.

M1. Uso indebido de la plataforma

- **Caso real**: Fitness Balance app, Heart Rate Monitor y Calories Tracker app.
- Estas aplicaciones empleaban el Touch ID para cargar compras en la aplicación dificultando la visualización de la ventana emergente de la compra.
- Similar a un clickjacking.
- *https://www.wired.com/story/iphone-touch-id-scam-apps/*

M1. Uso indebido de la plataforma

- ***Contramedidas:***
  - Para Android:
    - Fijar muy bien los permisos para componentes externos a nuestra aplicación.
  - Para iOS:
    - Utilizar el servicio Keychain para autenticarse.
    - Mantener las claves en un solo dispositivo (tener cuidado con lo que compartimos con iCloud).

M2. Almacenamiento inseguro de datos

- Esta vulnerabilidad puede ser muy heterogénea en su explotación:
  - Robo físico del dispositivo, ejecución de malware, exposición indebida/involuntaria de datos, rooteado/jailbreak…
- En general, hace referencia a una mala gestión de datos que pueden ser potencialmente sensibles.
- Esta mala gestión puede ser en el almacenamiento, pero también en el procesado de los datos.

M2. Almacenamiento inseguro de datos

- **Caso real**: Tinder, aplicación de citas y red social.
- Un fallo en Tinder permitía visualizar la localización actual exacta de las personas logueadas.
- Dichos datos de localización no se encontraban debidamente cifrados → error en el proceso.
- Con diferentes datos de localización se permitía triangular a los usuarios, lo que atenta contra su privacidad.
- *https://www.abine.com/blog/2014/tinder-app-vulnerability/*

M2. Almacenamiento inseguro de datos

- ***Contramedidas:***
  - Es conveniente analizar qué información es la más sensible.
  - Es importante comprobar los permisos que configuran las aplicaciones y cómo se almacenan y envían los datos catalogados como sensibles (almacenamiento y procesado).
  - Podría ser interesante emplear herramientas de análisis de memoria para comprobar si existe información sensible en la misma.

M3. Comunicación insegura

- Esta categoría engloba las vulnerabilidades que afectan a la seguridad de las comunicaciones:
  - No utilizar cifrado en las comunicaciones (HTTP).
  - Emplear versiones obsoletas o con vulnerabilidades.
  - Usar handshakes poco seguros (establecimiento de la comunicación entre pares).
  - Utilizar cifrados débiles.
  - Ataques MITM y monitorización de paquetes Wi-Fi.

M3. Comunicación insegura

- **Caso real**: MiSafes, relojes inteligentes para niños.

- Se trata de un smartwatch infantil con tarjeta SIM que incorporaba funciones de rastreo, envío de mensajes, llamadas, etc…
  - …adoptando medidas de seguridad muy deficientes.

- El problema está en que los relojes no cifraban ninguno de los datos anteriores, por lo que a la hora de enviar dichos datos a sus padres y mediante ataques IDOR, se podía acceder al reloj y a sus datos.

- El atacante, de esta forma, era capaz de:
  - Obtener información del niño, su número de teléfono y el de sus padres.
  - Rastrear a los niños.
  - Escucharlos, obtener fotos...

- *https://nakedsecurity.sophos.com/2018/11/16/hacking-misafes-smartwatches-for-kids-is-childs-play/*

M3. Comunicación insegura

- **Caso real**: MiSafes Mi-Cam.

- Se trata de una cámara infantil para vigilar niños de forma remota que implantaba medidas de seguridad nulas:
  - En primer lugar, se podía interceptar las peticiones HTTP entre el dispositivo móvil y la nube.
  - Además, era muy fácil realizar fuerza bruta porque emplean contraseñas de 4 dígitos y una caducidad de 30 minutos si solicitábamos un cambio de contraseña.

- *https://nakedsecurity.sophos.com/2018/02/22/another-baby-monitor-is-allowing-strangers-to-spy-on-children/*

M3. Comunicación insegura

- ***Contramedidas:***
  - Emplear algoritmos como TLS o SSL utilizando certificados.
  - Proteger también las comunicaciones a nivel de aplicación (capa de seguridad adicional) → cifrado extremo a extremo.
  - Si necesitamos mucho rendimiento, podemos separar un canal para cifrar solo información muy sensible.

M4. Autenticación insegura

- Engloba las vulnerabilidades relacionadas con los mecanismos de autenticación no efectivos o insuficientes. Algunas vulnerabilidades pueden ser:
  - Políticas de seguridad débiles (contraseñas).
  - Fallos en la gestión de la sesión (tokens de sesión).
  - Fallos al solicitar la autenticación (APIs externas).

M4. Autenticación insegura

- **Caso real:** MiSafes Mi-Cam.
- Usar contraseñas de 4 dígitos y que estas no caducasen en 30 minutos, provoca que esta cámara sea tremendamente fácil de atacar.
- En este caso, hay varios problemas:
  - La política de seguridad es nula.
  - No se maneja correctamente la validez de las contraseñas.
- *https://www.forbes.com/sites/thomasbrewster/2018/02/21/50000-mi-cam-baby-cams-vulnerable-to-simple-spy-attacks/?sh=22d563be1c7e*

M4. Autenticación insegura

- **Caso real:** Grab.
- Aplicación para pedir comida en Android e iOS.
- En 2017 se reportó que era posible saltarse la verificación en dos pasos (2FA).
- El problema era que esta verificación consistía en un código de 4 dígitos que no caducaba.
- Esto facilita enormemente los ataques de fuerza bruta.
- *https://hackerone.com/reports/202425*

M4. Autenticación insegura

- ***Contramedidas:***
  - Almacenar también la sesión en el servidor y relacionarla con el usuario autenticado.
  - Es conveniente no cargar datos sensibles en la aplicación hasta que se termine el proceso de autenticación.
  - La autenticación debe pedirse periódicamente.

M5. Cifrado insuficiente

- Esta categoría engloba los fallos de seguridad derivados de emplear algoritmos criptográficos vulnerables.
  - Implementaciones inseguras de los algoritmos.
  - Algoritmos débiles y/o poco consistentes.
  - Implementaciones desactualizadas.

M5. Cifrado insuficiente

- **Caso real:** Ola App.
- Ola App es una aplicación estilo Uber que opera en India. Se encontraron vulnerabilidades en su implementación de cifrado en 2015.
- El problema es que empleaban una función de cifrado que no se considera fuerte (*AES/ECB/PKCS5Padding*).
- Los investigadores consiguieron obtener la clave con la que cifraban todas las cuentas: *"PRODKEYPRODKEY12".*
- *https://www.appknox.com/blog/major-bug-in-ola-app-can-make-you-either-rich-or-poor*

M5. Cifrado insuficiente

- ***Contramedidas:***
  - Usar algoritmos estándar, consistentes y debidamente testeados.
  - Emplear algoritmos actualizados y nuevas versiones mejoradas.
  - No dejar secretos que sean accesibles.
    - Claves de cifrado.
    - Relacionado con la exposición de datos sensibles.

M6. Autorización insegura

- Aquí se engloban los fallos en el proceso de autorización, es decir, el control de privilegios y acciones de usuarios ya identificados.
- Categoría relacionada con vulnerabilidades que permitan realizar acciones por encima del rol del usuario correspondiente (escalada de privilegios).
- Estas pueden ser muy heterogéneas y dependen de la implementación.

M6. Autorización insegura

- **Caso real:** Pandora.
- Aplicación de control para alarmas de coches.
- Un atacante podía robar la cuenta de otro usuario:
  - El atacante se creaba una cuenta en Pandora (no necesitaba tener la alarma) y cambiaba el email de la víctima por uno propio mediante peticiones al servidor debidamente formateadas.
- Al cambiar el email, el atacante podía pedir un restablecimiento de contraseña, el cual llegaría al nuevo email.
- *https://www.kaspersky.com/blog/hacking-smart-car-alarm-systems/26014/*

M6. Autorización insegura

- ***Contramedidas:***
  - Comprobar, con pruebas unitarias y de integración, los esquemas de autorización desarrollados.
  - Establecer una política de roles y privilegios para los diferentes tipos de usuarios de una aplicación.
    - Privilegios mínimos.

M7. Calidad del código del cliente

- Esta categoría engloba las vulnerabilidades derivadas de la mala calidad del código de una aplicación.
  - Esto es, la mala utilización de las características del lenguaje de programación que estamos utilizando.

- Depende mucho de la aplicación, aunque hay vulnerabilidades típicas como los buffer overflow, los errores de tipado o las fugas de memoria.

M7. Calidad del código del cliente

- **Caso real:** WhatsApp.
- La vulnerabilidad CVE-2019-3568 describía un buffer overflow en su módulo VoIP que permitía a los atacantes ejecutar código arbitrario en la víctima.
- Esta vulnerabilidad era empleada para introducir spyware en clientes de la aplicación.
- *https://thehackernews.com/2019/05/hack-whatsapp-vulnerability.html*

M7. Calidad del código del cliente

- ***Contramedidas:***
  - Emplear herramientas de análisis de código estático.
  - Auditar y testear la parte del cliente y del servidor por separado.
  - Comprobar actualizaciones de las bibliotecas utilizadas por nuestra aplicación.

M8. Manipulación de código

- Esta categoría hace referencia a las vulnerabilidades derivadas de la manipulación del código de una aplicación.

- Una práctica común es la inyección de malware, el reempaquetado de la aplicación (.apk o .ipa) y la redistribución de esta con el malware.

- Otros vectores de ataque:
  - Parcheo de binarios.
  - Modificación de memoria dinámica.

M8. Manipulación de código

- **Caso real:** Pokémon Go.
- Juego que emplea la geolocalización del dispositivo móvil cliente.
- Los investigadores realizaron un análisis de código mediante ingeniería inversa para comprender el funcionamiento.
  - Descubrieron que era posible modificar el código para emplear localizaciones falsas.
- *https://nordicapis.com/how-pokemon-go-fans-hacked-em-all-and-how-to-prevent-similar-reverse-engineering/*

M8. Manipulación de código

- ***Contramedidas:***
  - Si han conseguido “clonar” nuestra aplicación, debemos desechar las API keys actuales y utilizar otras.
  - Podemos comprobar si el dispositivo ha sido rooteado.
  - El uso de checksums y firmas es muy útil, la aplicación puede comprobar si ha sido modificada (p. ej. directorio *_CodeSignature en iOS*).

M9. Ingeniería inversa

- Hace referencia al análisis de aplicaciones mediante herramientas específicas. El objetivo es conocer el funcionamiento interno del binario. Entre otras cosas se puede buscar:
  - Algoritmos y funcionamiento de la aplicación.
  - Librerías utilizadas.
  - Recursos internos.
  - Código fuente.

- Herramientas: Apktool, dex2jar, MobSF, etc.

M9. Ingeniería inversa

- **Caso real**: Pokémon Go (II).

- En el caso de Pokémon Go, la ingeniería inversa se empleaba para comprender el funcionamiento de la aplicación.

M9. Ingeniería inversa

- ***Contramedidas:***
  - La mejor recomendación es ofuscar el código.
  - Tratar de hacer reversing a nuestras propias aplicaciones para conocer la información que un posible atacante podría obtener (o modificar).
    - Finalmente, podemos intentar segurizar nuestra aplicación.

M10. Funcionalidad inesperada

- Hace referencia a bloques de código o características en proceso de desarrollo/testing.

- Los desarrolladores las dejan expuestas y pueden servir como entradas a los atacantes
  - **Backdoors por accidente!**

M10. Funcionalidad inesperada

- ***Contramedidas:***
  - Las pruebas software (en concreto las de integración) son ideales.
  - Buenas prácticas a la hora de construir entornos de prueba.
  - No añadir partes en desarrollo o a medio probar en las versiones release de las aplicaciones.

UCLM
UNIVERSIDAD DE CASTILLA-LA MANCHA

Escuela Superior Ingeniería Informática

DSI
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS

XUNTA DE GALICIA
CONSELLERÍA DE CULTURA, EDUCACIÓN E UNIVERSIDADE

CFR FERROL

Tema 5. Detección de problemas de seguridad en aplicaciones para dispositivos móviles

Características de seguridad de Android e iOS.

UNIÓN EUROPEA
Fondo Social Europeo
EL FSE invierte en tu futuro

GOBIERNO DE ESPAÑA
MINISTERIO DE EDUCACIÓN Y FORMACIÓN PROFESIONAL

Necesidad de las características de seguridad

- Los sistemas operativos Android e iOS cuentan con medidas de seguridad integradas.

- Aunque cada uno las implementa de forma diferente, los mecanismos de seguridad que incorporan, tienen las mismas finalidades.

- La clave fundamental es, por defecto, no confiar en **aplicaciones externas** y no asumir que el usuario que tiene el móvil es el **usuario legítimo**.

Mecanismos de seguridad en sistemas móviles

- Algunos mecanismos que se emplean para proteger los sistemas operativos móviles son:
  - Control de acceso.
  - Arranque seguro.
  - Aislamiento de aplicaciones.
  - Sistema de permisos.
  - Aplicaciones firmadas.
  - Cifrado de datos en aplicaciones.

Control de acceso

- El sistema debe incorporar mecanismos para evitar que un usuario ilegítimo utilice el dispositivo.

- El mecanismo fundamental de control de acceso es el **bloqueo del dispositivo**.

- Ciertas características de seguridad no pueden emplearse si este mecanismo no está habilitado. Por ejemplo:
  - Almacén de certificados de Android.
  - Protección de datos en iOS.

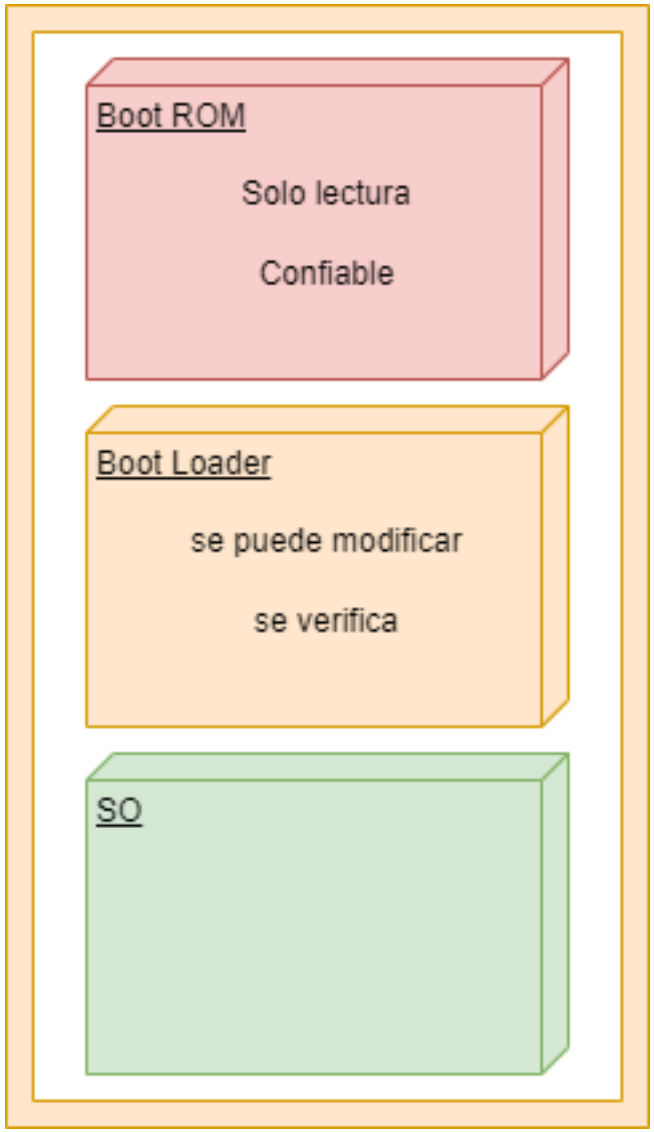
Mecanismos de bloqueo del dispositivo

- **Pin numérico**.
  - Puede limitar los intentos (en el tiempo) en caso de múltiples fallos.
  - Se puede configurar para borrar el dispositivo en caso de múltiples intentos fallidos.
- **Contraseña alfanumérica**.
  - Similar a la de cualquier otro dispositivo, servicio o aplicación.
  - Mayor seguridad a cambio de peor usabilidad.
- **Biométrica**.
  - Lectores de huellas dactilares, sistemas de reconocimiento facial (Touch ID, Face ID)…
- **Patrón de desbloqueo**.
  - En ocasiones pueden ser predecibles.

Arranque seguro

- En la etapa de arranque los sistemas pueden ser **muy vulnerables**.
  - Las características de seguridad todavía no están en funcionamiento.
- Los dispositivos deben asegurarse que el código que se ejecuta **está verificado y no ha sido modificado**.
- Si alguna verificación falla, el dispositivo debe ser restaurado de fábrica.
- El esquema de arranque suele ser parecido entre dispositivos, dividiéndose en **etapas**.

Etapas del arranque seguro



Aislamiento de aplicaciones

- Debemos desconfiar de las aplicaciones por defecto.
- Un mecanismo para evitar que las aplicaciones realicen acciones indeseadas es el **aislamiento**.
  - Las aplicaciones se ejecutan en un entorno aislado (sandbox).
  - Pueden acceder a sus recursos de forma directa, pero no a los de otras aplicaciones.
- Si se quieren emplear **recursos externos compartidos** se solicita permiso al sistema operativo (cámara, sensores, etc).
- La implementación de este mecanismo en Android e iOS es diferente.

Aislamiento de aplicaciones en Android

- La naturaleza intrínseca de Java hace que sea sencillo aislar en una **máquina virtual** cada aplicación que se instala en el dispositivo.

- Además, Android asocia un **user ID** a cada aplicación/máquina virtual.
  - Dicho user ID impide a otras aplicaciones que accedan a sus recursos.

- Este mecanismo se basa en **SELinux**.

Aislamiento de aplicaciones en iOS

- iOS asigna **directorios aleatorios** a las aplicaciones.
  - Por defecto, una aplicación solo puede acceder a su directorio.
- Las aplicaciones utilizan un usuario con bajos privilegios (‘*mobile*’).
- Para acceder a recursos externos debe pedirse permiso al SO:
  - Solo ciertas aplicaciones firmadas pueden acceder a ciertos recursos (p. ej. las aplicaciones del sistema).

Sistema de permisos

- El sistema de permisos permite al usuario gestionar el acceso a los recursos del dispositivo **para cada aplicación**.

- Si no se aceptan los permisos que require una aplicación, la funcionalidad de esta se verá limitada → arma de doble filo.

- Algunos recursos que están protegidos por permisos son:
  - Realizar llamadas.
  - Gestionar SMS.
  - Usar cámaras.
  - Utilizar los sensores.
  - El acceso a Internet.
  - El acceso al almacenamiento externo.

Aplicaciones firmadas

- Por defecto, en ambas plataformas se require una firma previa para poder instalar una aplicación.
  - Lo que permite conocer su procedencia.

- En **Android** este certificado puede ser autofirmado.
  - Se emplea para identificar qué aplicaciones han sido creadas por un mismo desarrollador.

- En **iOS** las aplicaciones tienen que estar firmadas por una autoridad de certificación (la AC de la propia Apple).

Cifrado de datos en aplicaciones

- Los sistemas operativos móviles modernos soportan de **forma nativa** el cifrado de datos.
- Este cifrado es útil cuando el móvil está **bloqueado**.
  - Ya que las aplicaciones acceden al sistema de ficheros sin cifrar.
- Android e iOS implementan el cifrado de forma ligeramentre diferente, aunque ambos usan el cifrado AES-256 bits.

UCLM
UNIVERSIDAD DE CASTILLA-LA MANCHA

Escuela Superior Ingeniería Informática

DSI
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS

XUNTA DE GALICIA
CONSELLERÍA DE CULTURA, EDUCACIÓN E UNIVERSIDADE

CFR FERROL

Tema 5. Detección de problemas de seguridad en aplicaciones para dispositivos móviles

Ejercicios prácticos (PoCs).

UNIÓN EUROPEA
Fondo Social Europeo
EL FSE invierte en tu futuro

GOBIERNO DE ESPAÑA
MINISTERIO DE EDUCACIÓN Y FORMACIÓN PROFESIONAL

PoC InsecureBankv2

- Vamos a realizar un ejercicio práctico de cara a utilizar algunas herramientas que necesitaremos en el taller:
  - Kali Linux.
  - Docker.
  - Genymotion (funciona sobre VirtualBox).
  - MobSF.
  - ADB.

- La PoC consiste en una vulnerabilidad en la aplicación InsecureBankv2 que nos permite saltar o bypassear la pantalla de login.

PoC InsecureBankv2

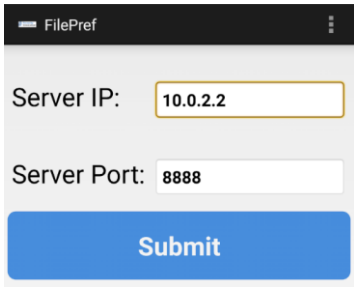
- En nuestra máquina Kali Linux, clonamos el repositorio de GitHub:
  - *git clone https://github.com/dineshshetty/Android-InsecureBankv2*

- Utilizamos Docker para ejecutar el servidor backend al que se conectará la aplicación vulnerable.

- Creamos el *Dockerfile* en el directorio raíz del proyecto descargado:
  - **FROM python:2**
  - **WORKDIR /app**
  - **COPY ./AndroLabServer/requirements.txt .**
  - **RUN pip install --no-cache-dir -r requirements.txt**
  - **COPY ./AndroLabServer .**
  - **EXPOSE 8888**
  - **CMD [“python”,”./app.py”]**

PoC InsecureBankv2

- Utilizamos Genymotion para emular un dispositivo Android.
- Instalamos la aplicación en el dispositivo virtual arrastrando su APK.
- Abrimos la aplicación, '*Preferences*' y configuramos la IP y puerto del servidor backend.



- Ahora, si introducimos un usuario y contraseña y pulsamos el botón de login, deberíamos ver la petición en el servidor.

```
^Cjavier@javier-PC:~/Descargas/Android-InsecureBankv2-master$ docker run -p 8888
:8888 -it androlab
The server is hosted on port: 8888
u= None
{"message": "User Does not Exist", "user": "prueba"}
```

PoC InsecureBankv2

- Utilizamos el framework MobSF para analizar la aplicación Android en cuestión (en concreto, su *.apk*).

- Una vez analizada, accedemos a su *AndroidManifest.xml*.
  - En este podemos encontrar los diferentes permisos, actividades y otros componentes de la aplicación.
  - La vulnerabilidad se encuentra en la actividad ***PostLogin*****,** ya que el componente se exporta (se comparte con las demás aplicaciones), por lo que podríamos llamar a esta actividad sin credenciales en la aplicación.

- Para conectarnos al dispositivo virtual de Genymotion, utilizamos ADB desde Kali Linux.

- Finalmente, mediante la CLI del Activity Manager dentro del dispositivo móvil, llamamos a la actividad vulnerable.
  - ***am start -n com.android.insecurebankv2/com.android.insecurebankv2.PostLogin***

PoC DIVA

- En este caso, vamos a utilizar MobSF con una aplicación llamada DIVA (Damn Insecure and Vulnerable App).

- En el taller nos centraremos en la aplicación PIVAA (Purposefully Insecure and Vulnerable Android Application), versión que reemplaza a DIVA.
  - Por tanto, en esta PoC vamos a centrarnos en identificar posibles partes de código vulnerables.

Logging inseguro

- Los logs son accesibles mediante ADB y Logcat.

```
public class LogActivity extends AppCompatActivity {
    /* access modifiers changed from: protected */
    @Override // android.support.v7.app.AppCompatActivity, android.support.v4.app.FragmentActivity, android.support.v4.app.BaseFragmentActivityDonu
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_log);
    }

    public void checkout(View view) {
        EditText cctxt = (EditText) findViewById(R.id.ccText);
        try {
            processCC(cctxt.getText().toString());
        } catch (RuntimeException e) {
            Log.e("diva-log", "Error while processing transaction with credit card: " + cctxt.getText().toString());
            Toast.makeText(this, "An error occured. Please try again later", 0).show();
        }
    }

    private void processCC(String ccstr) {
        throw new RuntimeException();
    }
}
```

Almacenamiento sin cifrado

- No deberíamos almacenar información sensible mediante *SharedPreferences* (objetos key-value) sin cifrar.

```
public class InsecureDataStorage1Activity extends AppCompatActivity {
    /* access modifiers changed from: protected */
    @Override // android.support.v7.app.AppCompatActivity, android.support.v4.app.FragmentActivity, android.supp
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_insecure_data_storage1);
    }

    public void saveCredentials(View view) {
        SharedPreferences.Editor spedit = PreferenceManager.getDefaultSharedPreferences(this).edit();
        spedit.putString("user", ((EditText) findViewById(R.id.ids1Usr)).getText().toString());
        spedit.putString("password", ((EditText) findViewById(R.id.ids1Pwd)).getText().toString());
        spedit.commit();
        Toast.makeText(this, "3rd party credentials saved successfully!", 0).show();
    }
}
```

Almacenamiento sin cifrado

- El mismo problema pero usando SQLite.

```
public class InsecureDataStorage2Activity extends AppCompatActivity {
    private SQLiteDatabase mDB;

    /* access modifiers changed from: protected */
    @Override // android.support.v7.app.AppCompatActivity, android.support.v4.app.FragmentActivity, a
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        try {
            this.mDB = openOrCreateDatabase("ids2", 0, null);
            this.mDB.execSQL("CREATE TABLE IF NOT EXISTS myuser(user VARCHAR, password VARCHAR);");
        } catch (Exception e) {
            Log.d("Diva", "Error occurred while creating database: " + e.getMessage());
        }
        setContentView(R.layout.activity_insecure_data_storage2);
    }

    public void saveCredentials(View view) {
        try {
            this.mDB.execSQL("INSERT INTO myuser VALUES ('" + ((EditText) findViewById(R.id.ids2Usr))
            this.mDB.close();
        } catch (Exception e) {
            Log.d("Diva", "Error occurred while inserting into database: " + e.getMessage());
        }
        Toast.makeText(this, "3rd party credentials saved successfully!", 0).show();
    }
```

Almacenamiento sin cifrado

- Lo mismo ocurre si utilizamos un fichero temporal.

```
public class InsecureDataStorage3Activity extends AppCompatActivity {
    /* access modifiers changed from: protected */
    @Override // android.support.v7.app.AppCompatActivity, android.support.v4.app.FragmentActivity, an
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_insecure_data_storage3);
    }

    public void saveCredentials(View view) {
        EditText usr = (EditText) findViewById(R.id.ids3Usr);
        EditText pwd = (EditText) findViewById(R.id.ids3Pwd);
        try {
            File uinfo = File.createTempFile("uinfo", "tmp", new File(getApplicationInfo().dataDir));
            uinfo.setReadable(true);
            uinfo.setWritable(true);
            FileWriter fw = new FileWriter(uinfo);
            fw.write(usr.getText().toString() + ":" + pwd.getText().toString() + "\n");
            fw.close();
            Toast.makeText(this, "3rd party credentials saved successfully!", 0).show();
        } catch (Exception e) {
            Toast.makeText(this, "File error occurred", 0).show();
            Log.d("Diva", "File error: " + e.getMessage());
        }
    }
}
```

Almacenamiento sin cifrado

- También puede darse este problema utilizando el almacenamiento externo.

```
public class InsecureDataStorage4Activity extends AppCompatActivity {
    /* access modifiers changed from: protected */
    @Override // android.support.v7.app.AppCompatActivity, android.support.v4.app.FragmentActivity, android.supp
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_insecure_data_storage4);
    }

    public void saveCredentials(View view) {
        EditText usr = (EditText) findViewById(R.id.ids4Usr);
        EditText pwd = (EditText) findViewById(R.id.ids4Pwd);
        try {
            File uinfo = new File(Environment.getExternalStorageDirectory().getAbsolutePath() + "/.uinfo.txt");
            uinfo.setReadable(true);
            uinfo.setWritable(true);
            FileWriter fw = new FileWriter(uinfo);
            fw.write(usr.getText().toString() + ":" + pwd.getText().toString() + "\n");
            fw.close();
            Toast.makeText(this, "3rd party credentials saved successfully!", 0).show();
        } catch (Exception e) {
            Toast.makeText(this, "File error occurred", 0).show();
            Log.d("Diva", "File error: " + e.getMessage());
        }
    }
}
```

Problemas de validación

- Al igual que ocurre en aplicaciones web, debemos validar correctamente los parámetros de entrada del usuario.

```
public class InputValidation2URISchemeActivity extends AppCompatActivity {
    /* access modifiers changed from: protected */
    @Override // android.support.v7.app.AppCompatActivity, android.support.v4.app.FragmentActivity, android.support.v4.a
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_input_validation2_urischeme);
        ((WebView) findViewById(R.id.ivi2wview)).getSettings().setJavaScriptEnabled(true);
    }

    public void get(View view) {
        ((WebView) findViewById(R.id.ivi2wview)).loadUrl(((EditText) findViewById(R.id.ivi2uri)).getText().toString());
    }
}
```

Control de acceso inseguro

- Es peligroso permitir que los componentes sean exportables (actividades, content providers, etc).

```
<provider android:name="jakhar.aseem.diva.NotesProvider" android:enabled="true" android:exported="true" android:authorities="jakhar.aseem.diva.provider.notesprovider" />
```

```
public void accessNotes(View view) {
    EditText pinTxt = (EditText) findViewById(R.id.aci3notesPinText);
    Button abutton = (Button) findViewById(R.id.aci3naccessbutton);
    if (pinTxt.getText().toString().equals(PreferenceManager.getDefaultSharedPreferences(this).getString(getString(R.string.pkey), ""))) {
        ((ListView) findViewById(R.id.aci3nlistView)).setAdapter((ListAdapter) new SimpleCursorAdapter(this, R.layout.notes_entry, getContentResolver().query(NotesProvider.CO
        pinTxt.setVisibility(4);
        abutton.setVisibility(4);
        return;
    }
    Toast.makeText(this, "Please Enter a valid pin!", 0).show();
}
```

UCLM
UNIVERSIDAD DE CASTILLA-LA MANCHA

Escuela Superior Ingeniería Informática

DSI
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS

XUNTA DE GALICIA
CONSELLERÍA DE CULTURA, EDUCACIÓN E UNIVERSIDADE

CFR FERROL

Tema 5. Detección de problemas de seguridad en aplicaciones para dispositivos móviles

Análisis estático.

UNIÓN EUROPEA
Fondo Social Europeo
EL FSE invierte en tu futuro

GOBIERNO DE ESPAÑA
MINISTERIO DE EDUCACIÓN Y FORMACIÓN PROFESIONAL

Análisis estático

- Consiste en el análisis del código de un programa sin llevar a cabo la ejecución del mismo. Es decir, se trata de analizar o inspeccionar el propio binario de la aplicación.

- Elementos a analizar:
  - Código de la aplicación (conexiones, algoritmos criptográficos, contraseñas hardcodeadas, APIs utilizadas, etc).
  - Metadatos.
  - Ficheros de configuración.
  - Recursos utilizados.

Análisis estático

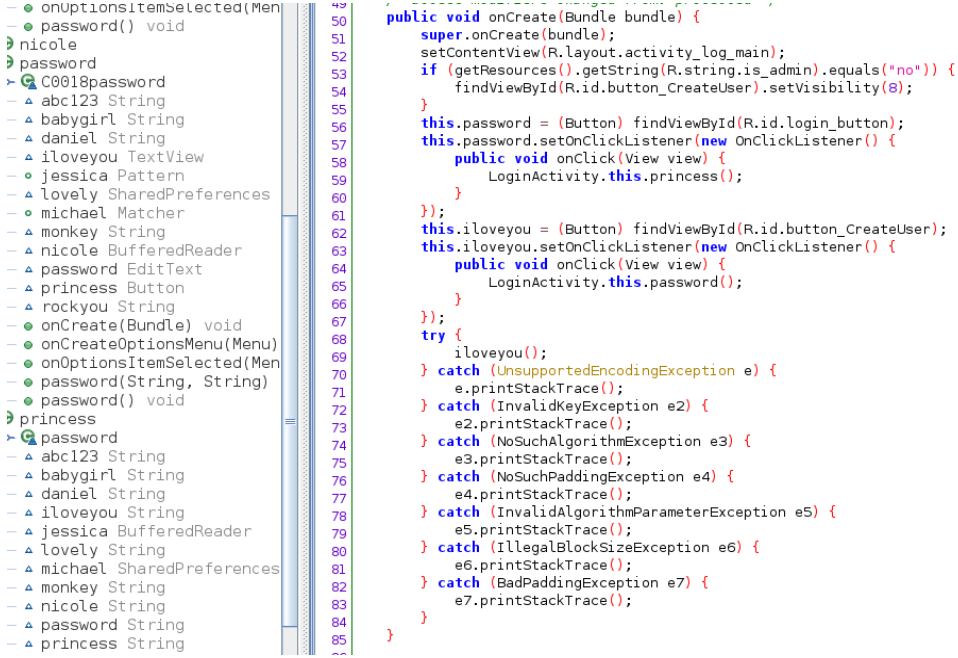
- Principales herramientas de análisis estático:
  - **apktool** → permite desempaquetar y reempaquetar APKs.
  - **dex2jar** → transforma un fichero *.dex* en un fichero *.jar* obteniendo el bytecode de Java y permitiendo así decompilar los *.class* en código Java.
  - **enjarify** → similar a dex2jar.
  - **IDA, Ghidra, Radare2** → para analizar librerías nativas.
  - **MobSF** → analiza ficheros de manera estática y automática. También permite realizar análisis dinámico.
  - **Androguard** → framework para analizar de manera estática aplicaciones en Android. Permite la automatización de tareas con Python.

Análisis estático

- **Escenario ideal**: la aplicación no utiliza ninguna librería nativa y el decompilado es correcto y completamente legible.

- Esto no siempre es así, el código puede ser **ofuscado,** dificultando su comprensión.

- Después de ser ofuscado, el código podría tener renombradas las variables, clases, etc.



Análisis estático

- Las **librerías nativas** son código que el desarrollador escribe y compila para una arquitectura específica (ARM, x86, MIPS, etc).

- Generalmente, este código es escrito en un lenguaje de bajo nivel como **C/C++**. Esto suele realizarse para tareas complejas o que requieren un alto rendimiento.
  - Operaciones matemáticas, motores gráficos, etc.

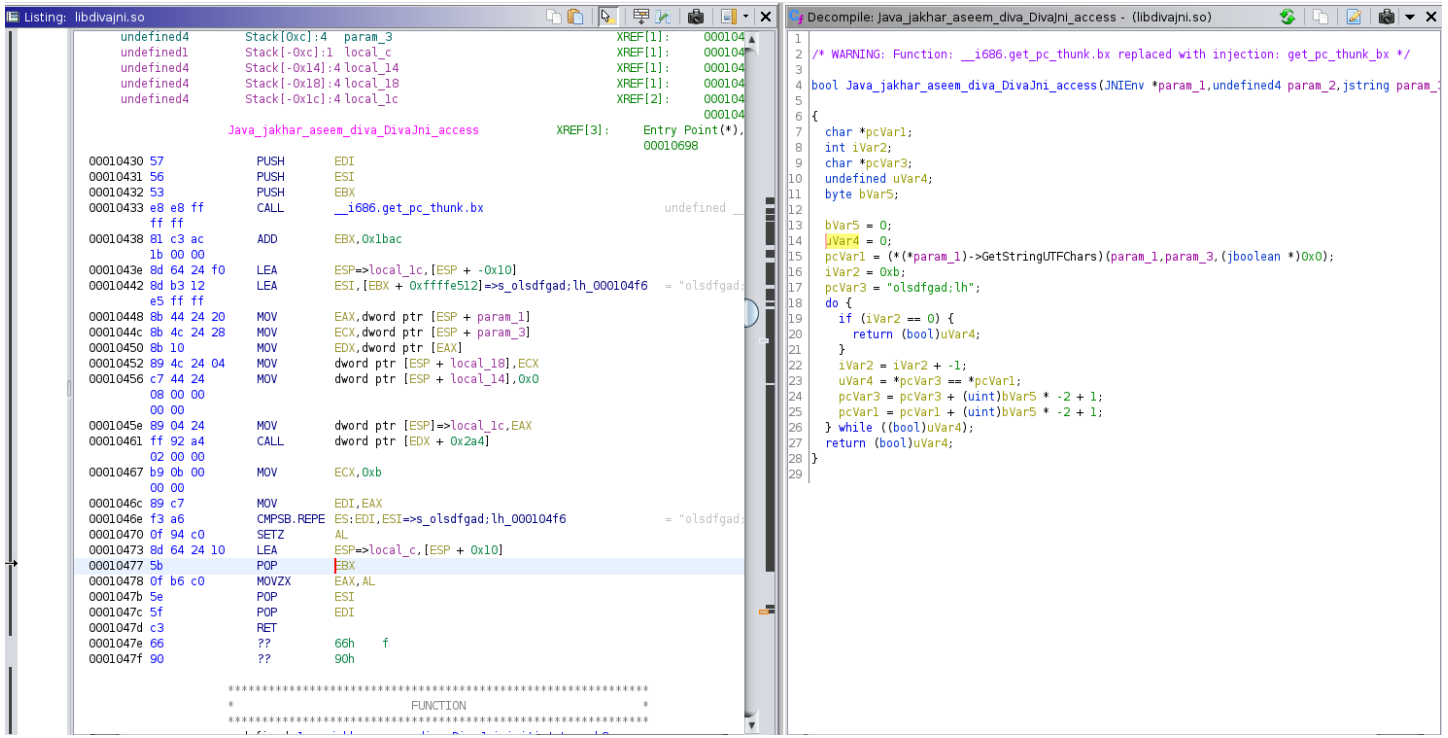
- Sin embargo, se puede escribir código nativo para otras tareas y, al ser un lenguaje compilado, su análisis es más complicado.
  - Es necesario entender las instrucciones del lenguaje ensamblador de la arquitectura, el manejo interno de la memoria, etc.

- A través del framework ***Java Native Interface (JNI),*** es posible para los desarrolladores declarar métodos que son implementados en código nativo e invocar tales métodos desde Java.

Análisis estático

```java
public class DivaJni {
    private static final String soName = "divajni";

    public native int access(String str);

    public native int initiateLaunchSequence(String str);

    static {
        System.loadLibrary(soName);
    }
}
```

```java
private DivaJni djni;

/* access modifiers changed from: protected */
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView((int) R.layout.activity_hardcode2);
    this.djni = new DivaJni();
}

public void access(View view) {
    if (this.djni.access(((EditText) findViewById(R.id.hc2Key)).getText().toString()) != 0) {
        Toast.makeText(this, "Access granted! See you on the other side :)", 0).show();
```



UCLM
UNIVERSIDAD DE CASTILLA-LA MANCHA

Escuela Superior Ingeniería Informática

DSI
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS

XUNTA DE GALICIA
CONSELLERÍA DE CULTURA, EDUCACIÓN E UNIVERSIDADE

CFR FERROL

Tema 5. Detección de problemas de seguridad en aplicaciones para dispositivos móviles

Análisis dinámico.

UNIÓN EUROPEA
Fondo Social Europeo
EL FSE invierte en tu futuro

GOBIERNO DE ESPAÑA
MINISTERIO DE EDUCACIÓN Y FORMACIÓN PROFESIONAL

Análisis dinámico

- Analizar la aplicación **durante su ejecución**, comprobando las distintas acciones que realiza en el sistema.
  - Conexiones de red.
  - Flujo de ejecución.
  - Información que almacena la aplicación.
  - Datos en memoria.

- Se necesita tener un **dispositivo o emulador** capaz de ejecutar la aplicación (interesante en el ámbito del malware).

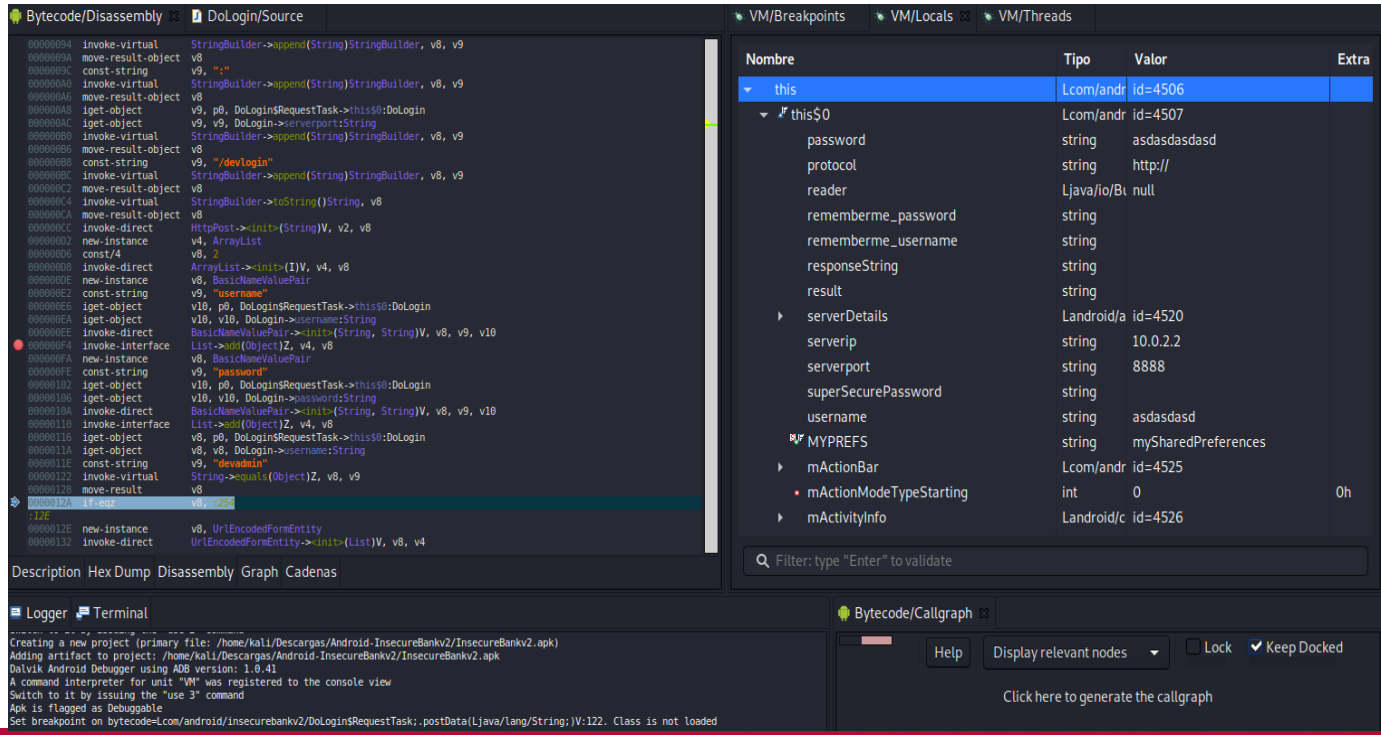
- La mayoría de las veces, es necesario tener **permisos de superusuario** para poder analizar todos los elementos.

Análisis dinámico

- A través del análisis dinámico podemos comprobar:
  - Si las transmisiones de red se realizan de manera segura.
  - Cómo se almacenan los datos en el dispositivo.
  - Problemas de configuración.
  - Problemas en los endpoints a los que se conecta la aplicación.
  - La validación de los datos que introduce el usuario.

Análisis dinámico

- Principales herramientas para el análisis dinámico de código:
  - JEB Decompiler.
  - IDA Pro.
  - Frida.
  - Android Studio.
  - Drozer.



UCLM UNIVERSIDAD DE CASTILLA-LA MANCHA

Escuela Superior Ingeniería Informática

DSI DEPARTAMENTO DE SISTEMAS INFORMÁTICOS

XUNTA DE GALICIA | CONSELLERÍA DE CULTURA, EDUCACIÓN E UNIVERSIDADE

CFR FERROL

Tema 5. Detección de problemas de seguridad en aplicaciones para dispositivos móviles

Análisis de red.

UNIÓN EUROPEA
Fondo Social Europeo
EL FSE invierte en tu futuro

GOBIERNO DE ESPAÑA
MINISTERIO DE EDUCACIÓN Y FORMACIÓN PROFESIONAL

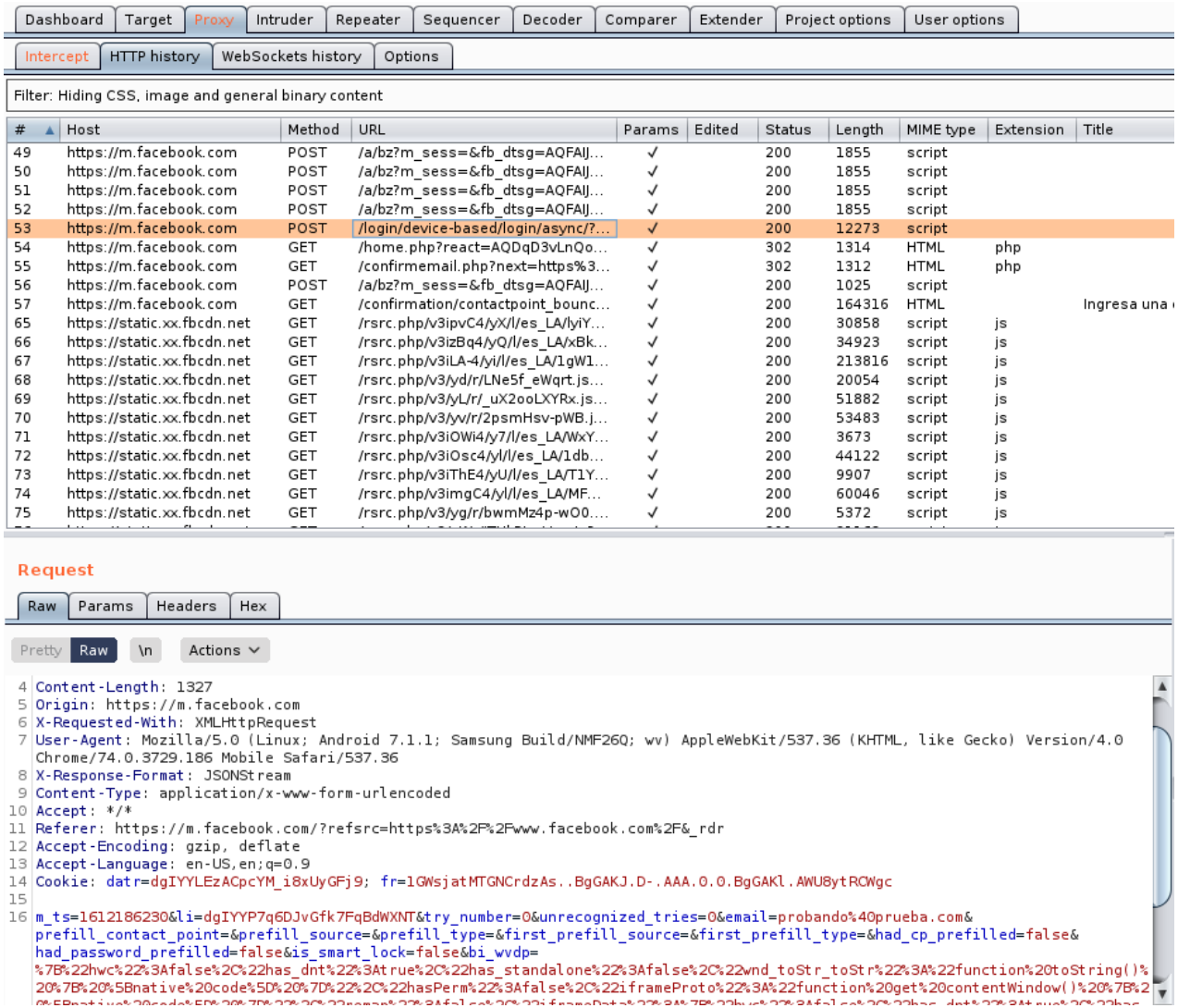
Análisis de red

- Analizar las conexiones de red que realiza la aplicación y/o el dispositivo.

- La mayoría de las conexiones serán peticiones HTTP/HTTPS, pero es importante comprobar si existen otro tipo de conexiones.

- Principales herramientas:
  - **Wireshark** → análisis y captura de paquetes de red.
  - **Servidor proxy** → inspeccionar y modificar las peticiones que realiza la aplicación (OWASP ZAP, Burp Suite, etc).

Análisis de red



UCLM
UNIVERSIDAD DE CASTILLA-LA MANCHA

Escuela Superior Ingeniería Informática

DSI
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS

XUNTA DE GALICIA
CONSELLERÍA DE CULTURA, EDUCACIÓN E UNIVERSIDADE

CFR FERROL

Puesta en producción segura

Tema 5. Detección de problemas de seguridad en aplicaciones para dispositivos móviles.

UNIÓN EUROPEA
Fondo Social Europeo
EL FSE invierte en tu futuro

GOBIERNO DE ESPAÑA
MINISTERIO DE EDUCACIÓN Y FORMACIÓN PROFESIONAL