ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

ANÁLISIS DE SEGURIDAD DE LAS APLICACIONES MÓVILES BASADAS EN ANDROID, UTILIZADAS EN LA BANCA ELECTRÓNICA DEL ECUADOR

PROYECTO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y REDES DE INFORMACIÓN

JOSÉ LUIS PALACIOS LÓPEZ

DIRECTOR: MSc. GABRIEL ROBERTO LÓPEZ FONSECA

CODIRECTOR: MSc. FRANKLIN LEONEL SÁNCHEZ CATOTA

Quito, junio 2018

DECLARACIÓN

Yo, José Luis Palacios López, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedo mis derechos de propiedad intelectual correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

_____

José Luis Palacios López

CERTIFICACIÓN

Certificamos que el presente trabajo fue desarrollado por José Luis Palacios López, bajo nuestra supervisión.

MSc. Gabriel Roberto López Fonseca
DIRECTOR DEL PROYECTO

MSc. Franklin Leonel Sánchez Catota
CODIRECTOR DEL PROYECTO

AGRADECIMIENTO

A Dios, quien en su infinita sabiduría ha sabido guiarme y cuidarme en el azaroso camino de la vida. Por sus cuidados y por brindarme la bendición de estar rodeado de todas las personas que más quiero.

A mi madre, quien en su arduo esfuerzo lo ha dado todo por mí y ha sido quien me ha acompañado en mis penas y alegrías, en mis triunfos y fracasos. Cada palabra de aliento, cada abrazo, me han ayudado a seguir adelante. Todo lo que soy se lo debo a ella.

A mi padre, cuyos consejos y ejemplo de valor me han ayudado a recibir cada día con alegría y entusiasmo. Todo su esfuerzo y dedicación han contribuido a mi formación personal y profesional. Para él mi sincero agradecimiento.

A mi hermana, quien con cariño siempre ha sabido animarme hasta en los momentos más difíciles. Sus oportunos consejos y apoyo incondicional me han hecho sentir lo maravilloso de tener una hermana.

Al MSc. Gabriel López y el MSc. Franklin Sánchez, por aceptar dirigir mi tesis, por todo el tiempo y esfuerzo dedicado para sacar adelante este proyecto.

Anita, a ti, por todo el cariño y comprensión que a diario me brindas. Por todo el apoyo que me has dado en los momentos más difíciles, y por compartir conmigo todos esos momentos de alegría. Gracias por ver algo en mí que nadie más pudo.

A mis amigos Ronald, Fabián, David y Jenny, con quienes he compartido alegrías y tristezas, largas jornadas de estudio y también agradables momentos de esparcimiento. Todo este tiempo ha valido la pena por tener el gusto de conocer personas como ustedes. Gracias por esa amistad que ha sabido mantenerse a través del paso del tiempo.

A todos los profesores de la carrera, quienes a diario dan una parte de sus vidas para formarnos como buenos profesionales.

A mis amigos, compañeros y a todos quienes en algún momento me brindaron su ayuda y apoyo.

DEDICATORIA

A Dios. Sin importar el nombre ni la forma que le demos, siempre su sabiduría nos ha de guiar por el camino del bien.

A mi madre, mi padre y mi hermana. Quienes a pesar de los problemas y dificultades me han brindado su apoyo incondicional. Toda su abnegación y consejos me han formado como la persona que soy ahora, a ustedes les dedico este logro.

José

"AMAT VICTORIA CURAM"

- *La victoria favorece a los que se preparan.*

CONTENIDO

ANEXOS

ORDEN DE EMPASTADO

ÍNDICE DE FIGURAS

ÍNDICE DE CÓDIGO

ÍNDICE DE TABLAS

RESUMEN

El presente trabajo presenta las bases necesarias para el análisis de la seguridad de las aplicaciones móviles basadas en Android para la banca electrónica del Ecuador.

Al inicio se revisan los fundamentos teóricos que se utilizarán en los capítulos siguientes. Por ejemplo, se trata la estructura de Android así como la metodología OWASP *Mobile Security Project*.

A continuación, se diseña un protocolo de pruebas, el cual se utiliza posteriormente para realizar el análisis de seguridad de las aplicaciones bancarias A, B y C.

Posteriormente, se implementa el análisis de seguridad sobre las aplicaciones móviles basadas en Android para la banca electrónica del Ecuador.

Luego se analizan los resultados obtenidos de las pruebas realizadas.

Finalmente, se presentan las conclusiones obtenidas como resultado de la aplicación del protocolo de pruebas sobre las tres aplicaciones bancarias.

PRESENTACIÓN

En el presente trabajo se presenta el análisis de seguridad de las aplicaciones móviles basadas en Android, utilizadas en la banca electrónica del Ecuador. Para esto, el documento se encuentra repartido en cuatro capítulos.

En el primer capítulo se aborda toda la teoría necesaria para entender los conceptos que se usan en los siguientes capítulos. Para esto se realiza una introducción a Android tomando en cuenta su arquitectura, componentes básicos, modelo de permisos, firmado de aplicaciones y Google Bouncer. Se revisa también la metodología propuesta por OWASP *Mobile Security Project* considerando los pasos que son aplicados en este trabajo.

En el segundo capítulo se seleccionan las herramientas que se utilizarán posteriormente, para ejecutar el protocolo de pruebas a las aplicaciones bancarias A, B y C. Las herramientas se seleccionan para las fases de recolección de información, análisis estático y análisis dinámico; para esto, se resumen en una tabla por cada fase, las ventajas y desventajas de cada herramienta. A continuación, se definen los pasos que componen el protocolo a seguir para el análisis de las aplicaciones bancarias en cada fase y al final de este capítulo, se incluyen los resultados que se esperan alcanzar al ejecutar el protocolo.

En el tercer capítulo se describe el procedimiento necesario, para que las herramientas seleccionadas en el segundo capítulo, estén listas para la ejecución del protocolo sobre las aplicaciones bancarias A, B y C. Para esto, se especifica dentro de las fases de recolección de información, análisis estático, y análisis dinámico, cada herramienta con los pasos y recomendaciones para su instalación y uso.

En el cuarto capítulo se realiza la ejecución del protocolo sobre las aplicaciones bancarias A, B y C. Se presenta la ejecución completa del protocolo para la aplicación bancaria A, mientras que para las aplicaciones bancarias B y C el desarrollo se encuentra en los anexos. Además, al final del capítulo se muestran los resultados encontrados para cada aplicación bancaria.

Finalmente, se presenta el capítulo 5, con las conclusiones y recomendaciones obtenidas a partir del desarrollo del presente trabajo.

CAPÍTULO 1
MARCO TEÓRICO

1.1 INTRODUCCIÓN

Desde que Google adquirió Android en el año 2005, y se hizo cargo de su desarrollo muchas cosas han cambiado especialmente en el campo de la seguridad de este sistema operativo. El desarrollo y crecimiento de esta plataforma se debe en gran parte, al apoyo de varios fabricantes de *smartphones* como LG, Samsung, Sony y HTC [1].

Android es un sistema operativo basado en el kernel de Linux y optimizado para mejorar su rendimiento en dispositivos móviles. Para esto cuenta con su propia arquitectura, la cual está organizada por capas. Cada una de estas capas tiene sus funciones bien definidas y, con excepción de la capa del tiempo de ejecución de Android, se han mantenido a lo largo de las diferentes versiones de este sistema operativo. El principal cambio que experimenta la capa del tiempo de ejecución es que deja de utilizar DVM (*Dalvik Virtual Machine*) para ser reemplazado por ART (*Android Runtime*) a partir de la versión 4.4 (KitKat) [2].

El éxito de Android se debe en gran parte a la gran cantidad de aplicaciones que ofrece en su tienda oficial: Play Store. Estas aplicaciones también tienen su estructura definida mediante varios componentes y cuentan con un ciclo de vida que permite brindar toda la funcionalidad necesaria a la aplicación [3].

La seguridad en las aplicaciones es controlada mediante varias herramientas que incluyen: buenas prácticas al momento de su desarrollo y un modelo de permisos. Los permisos que solicitan las aplicaciones permiten acceder a los recursos físicos del dispositivo, así como a datos sobre el usuario. Además, con el fin de evitar la presencia de aplicaciones maliciosas en su tienda oficial, Google ha desarrollado diversas herramientas que se encargan de esta tarea como por ejemplo: Bouncer, App Verification Service y Play Protect[4].

Si bien las herramientas de Google ayudan a controlar la presencia de aplicaciones maliciosas, no toma en cuenta un análisis más profundo sobre cada aplicación con el fin de detectar otros problemas como: malas prácticas al momento de desarrollar

la aplicación, protección de los datos del usuario, etc. Es por esto que se ha planteado la metodología OWASP *Mobile Security Project* [5], la cual brinda una guía más detallada sobre los pasos necesarios al momento de analizar una aplicación en concreto. Para esto utiliza tres fases: recolección de información, análisis estático y análisis dinámico.

Como primera fase, la recolección de información menciona los pasos que ayudan a tener una idea del funcionamiento de la aplicación bancaria al utilizarla. Esta información será de utilidad durante las fases de análisis estático y dinámico. Para la fase de análisis estático, los pasos se enfocan en el análisis del código que compone la aplicación, considerando la información recopilada en la primera fase. Finalmente, para el análisis dinámico, se busca identificar posibles vectores de ataque a la aplicación bancaria.

1.2 ARQUITECTURA DE ANDROID [2]

Android es una plataforma de código abierto basada en Linux y diseñada para una gran variedad de dispositivos y factores de forma. En la Figura 1.1 se muestra cómo está compuesta la arquitectura de Android.

Como se observa en la Figura 1.1, las capas que lo componen son las siguientes:

- Kernel de Linux.
- *Hardware Abstraction Layer* (HAL).
- Android *Runtime*.
- Bibliotecas C/C++ Nativas.
- *Framework* de la API de Java.
- Apps del sistema.

1.2.1 KERNEL DE LINUX

Como base de toda la arquitectura se encuentra el kernel de Linux, el cual ha sido modificado para brindar un mejor desempeño en entornos móviles. Además, es el encargado de interactuar directamente con todos los componentes físicos del dispositivo y por lo tanto contiene la mayoría de *drivers*. De igual manera, esta capa es responsable de la mayoría de características de seguridad presentes en Android.

Figura 1.1 Arquitectura de Android [2]

1.2.2 HARDWARE ABSTRACTION LAYER (HAL)

Esta capa provee interfaces que exponen las capacidades de hardware al *framework* de la API Java del nivel superior. Además, está compuesta por varios módulos de biblioteca, cada uno de los cuales implementa una interfaz para un tipo de componente de hardware como la cámara o el módulo Bluetooth. En el momento en el que *framework* de una API solicita el acceso al hardware del dispositivo, el sistema Android carga el módulo de biblioteca correspondiente [6].

1.2.3 ANDROID RUNTIME (ART)[7]

Para dispositivos que usan Android desde la versión 5.0 (API nivel 21) o superior cada aplicación ejecuta su propio proceso con sus propias instancias del tiempo de ejecución de Android (ART).

ART está escrito para ejecutar múltiples máquinas virtuales en dispositivos de baja memoria mediante la ejecución de archivos DEX, el cual es un formato de código de bytes optimizado para Android para ejecutarse con un impacto bajo en memoria. Las funciones principales del ART son:

- Compilación *Ahead of Time* (AOT) y *Just in Time* (JIT).
- Recolección optimizada de elementos no usados (GC).
- Mejor compatibilidad con la depuración, excepciones de diagnóstico y reportes de fallos detallados y la capacidad de establecer puntos de control para monitorear campos específicos.

Antes de la versión 5.0 de Android (API nivel 21), se utilizaba la *Dalvik Virtual Machine* (DVM) con el mismo propósito de ART.

Android incorpora además un conjunto de bibliotecas centrales en tiempo de ejecución que proveen la mayor parte de la funcionalidad del lenguaje de programación Java.

1.2.4 BIBLIOTECAS C/C++ NATIVAS

Varios componentes y servicios de Android como ART y HAL están basados en código nativo que requiere bibliotecas nativas escritas en C y C++. La plataforma Android ofrece APIs del *framework* de Java para exponer la funcionalidad de varias de estas bibliotecas nativas a las aplicaciones.

Por ejemplo, se puede acceder a OpenGL ES mediante la API Java de OpenGL para añadir soporte para el dibujo y manipulación gráficos 2D y 3D en la aplicación. Para el caso en el cual una aplicación requiera del uso de código C o C++ se puede utilizar el *kit* de Desarrollo Nativo (NDK) de Android para hacer uso de estas bibliotecas directamente. A continuación se presentan algunos ejemplos:

- ***Surface Manager*:** Se encarga del manejo de las ventanas.

- ***Media Framework*:** Permite el uso de diferentes tipos de códecs para reproducir y grabar archivos multimedia.
- **SQLite:** Es una versión más ligera de SQL usada para la administración de bases de datos.

1.2.5 FRAMEWORK DE LA API DE JAVA

El conjunto completo de características del sistema operativo Android está disponible mediante APIs escritas en lenguaje Java. Estas APIs componen la base para la creación de aplicaciones simplificando la reutilización de componentes y servicios, estos servicios se muestran en la Tabla 1.1.

Tabla 1.1 Servicios de la capa *Framework* de Android [8]

Servicio	Descripción
Activity Manager	Administra el ciclo de vida de las aplicaciones y otros componentes. Cuando una aplicación solicita iniciar una actividad por ejemplo, mediante *startActivity()*, el Administrador de Actividades provee este servicio.
Resource Manager	Provee acceso a los recursos como gráficos, *strings* y archivos de diseño.
Location Manager	Provee soporte para servicio de ubicación. (Ej. GPS)
Notification Manager	Las aplicaciones que necesitan ser notificadas acerca de ciertos eventos utilizan este servicio. Por ejemplo, si una aplicación quiere saber cuándo se ha recibido un nuevo e-mail, usa el administrador de notificaciones.
Package Manager	Este servicio, junto con *installd* que es el demonio de administración de paquetes, es responsable de instalar aplicaciones en el sistema y mantener información acerca de las aplicaciones instaladas.
Content Providers	Permite que las aplicaciones accedan a los datos desde otras aplicaciones o compartan sus propios datos con estas.
Views	Provee una amplia gama de vistas que una aplicación puede utilizar para mostrar información.

1.2.6 APPS DEL SISTEMA

Android incluye un conjunto de aplicaciones centrales para el correo, calendario, navegación en Internet y mensajería SMS. Estas aplicaciones incluidas en la

plataforma no tienen una preferencia en relación a las aplicaciones que el usuario escoge instalar. Es así que casi cualquier aplicación externa puede convertirse en la aplicación por defecto para el teclado, navegador web e incluso mensajería SMS, sin embargo, aplicaciones como la aplicación de configuración del sistema son una excepción.

1.2.7 VERSIONES DE LA PLATAFORMA

Desde su aparición, se han ido actualizando las versiones de Android introduciendo nuevas características y mejoras en temas de seguridad. Actualmente Android es el sistema operativo más utilizado en dispositivos móviles [9]. Como se muestra en la Figura 1.2, la versión más utilizada de Android es Marshmallow. Además, se muestra el porcentaje de dispositivos que tiene instalada cada versión de Android, las versiones con un porcentaje inferior al 0.1% no se incluyen en esta figura.

Version	Codename	API	Distribution
2.3.3 - 2.3.7	Gingerbread	10	0.3%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.4%
4.1.x	Jelly Bean	16	1.7%
4.2.x		17	2.2%
4.3		18	0.6%
4.4	KitKat	19	10.5%
5.0	Lollipop	21	4.9%
5.1		22	18.0%
6.0	Marshmallow	23	26.0%
7.0	Nougat	24	23.0%
7.1		25	7.8%
8.0	Oreo	26	4.1%
8.1		27	0.5%

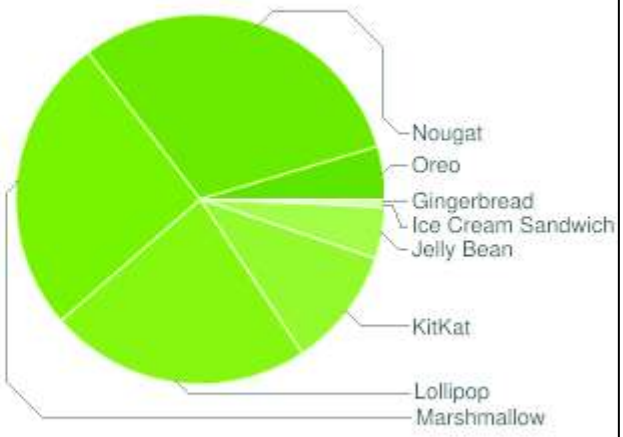


Figura 1.2 Versiones de Android a noviembre del 2017 [10]

1.3 PROCESO DE ARRANQUE Y CARGA[11]

Una de las cosas más importantes al momento de considerar la seguridad en Android el proceso de arranque. Todo el proceso inicia con el gestor de arranque (*bootloader)*, el cual inicia con el proceso *init* que es el primer proceso de usuario, este se muestra en la Figura 1.3.

De este modo, cualquier cambio en el *bootloader* o si se ha cargado otro en lugar del utilizado por defecto, se puede modificar lo que se carga en el dispositivo. El *bootloader* es específico del fabricante y por lo tanto cada fabricante tiene su propia versión modificada del mismo. Usualmente, el acceso al *bootloader* se encuentra deshabilitado por defecto mediante el bloqueo del mismo, de modo que sólo se permite correr el kernel especificado por el fabricante en el dispositivo.

Una vez que el *bootloader* arranca con el kernel y lanza el proceso *init*, se montan algunos de los directorios importantes requeridos para el funcionamiento del sistema Android como el */dev, /sys* y */proc*. Además, *init* toma sus parámetros de configuración de los archivos *init.rc* e *init.[nombre-de-dispositivo].rc*, y en algunos casos de archivos *.sh* encontrados en la misma ubicación.

Figura 1.3 Procesos init del dispositivo móvil

Al utilizar el comando *cat* sobre el archivo *init.rc* se pueden observar todas las especificaciones usadas por el proceso *init* mientras se carga, esto se muestra en la Figura 1.3.

El proceso *init* es responsable además del inicio de otros componentes necesarios como el demonio *Android Debug Bridge (ADBD)*, el cual es responsable de la comunicación ADB y el *Volume Daemon* (VOLD). Algunas de las propiedades que son utilizadas mientras se está cargando el sistema se encuentran en el archivo *build.prop*, ubicado en *system/*. El final del proceso de carga de *init* se da cuando el logo de Android se muestra en la pantalla del dispositivo.

Una vez que todo se encuentra listo se cargan otros componentes necesarios como el *Activity Manager* cuya función es descrita más adelante. Ahora que se ha completado todo el proceso de arranque, el sistema envía un mensaje *Boot_Completed* que están esperando algunas aplicaciones gracias al *Broadcast Receiver*.

1.4 ARQUITECTURA DE UNA APLICACIÓN ANDROID[12]

Toda aplicación en Android está construida de una manera similar, de modo que su estructura general se organiza en varios bloques. A continuación, se tratan estos componentes.

1.4.1 *SANDBOXING* Y COMPONENTES DE LA APLICACIÓN

Una aplicación típica de Android puede realizar diversas funciones, por ejemplo, la aplicación de reloj que incluye el sistema. Esta aplicación tiene varias funciones: mostrar la hora (se acuerdo a la zona horaria), configurar alarmas y manejar el cronómetro.

Básicamente, hay tres diferentes pantallas para la misma aplicación. Además de su obvia funcionalidad, esta aplicación necesita comunicarse con algún servidor para las actualizaciones de tiempo, ejecutar un servicio en segundo plano para las alarmas, sincronizarse con el reloj del procesador y demás. Por eso, aún una aplicación simple en Android tiene varios bloques que la componen.

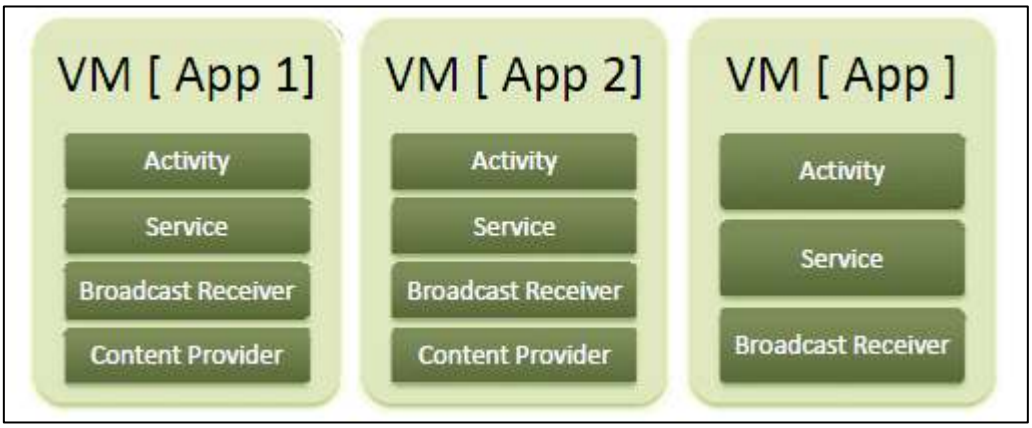


Figura 1.4 Componentes básicos de una aplicación Android [13]

Hay cuatro componentes de una aplicación: *Activities*, *Broadcast Receivers*, *Content Providers* y *Services*. Estos componentes interactúan entre ellos (o con

componentes de otras aplicaciones) mediante mensajes llamados *Intents*. La Figura 1.4 muestra los principales componentes de una aplicación Android.

Cada aplicación se ejecuta como una instancia independiente de una máquina virtual, por lo que, si una aplicación falla durante su ejecución las otras continúan con su funcionamiento normal. Es decir, cada aplicación se encuentra aislada del resto, esto se conoce como *Sandboxing*.

1.4.1.1 Activities

Las *Activities* son básicamente pantallas que el usuario visualiza o con la cual interactúa. Pueden verse como los componentes de la interfaz gráfica de usuario (UI). La mayoría de las aplicaciones cuentan con múltiples *Activities* (una por cada pantalla con la cual interactúa el usuario) y el usuario puede navegar entre ellas de acuerdo al flujo de la aplicación.

Para mejorar la experiencia de usuario, este puede abrir diferentes *Activities* para la misma aplicación según esta lo requiera. El usuario puede también lanzar una *Activity* de otra aplicación (mediante *Intents*) como se muestra en la Figura 1.5.

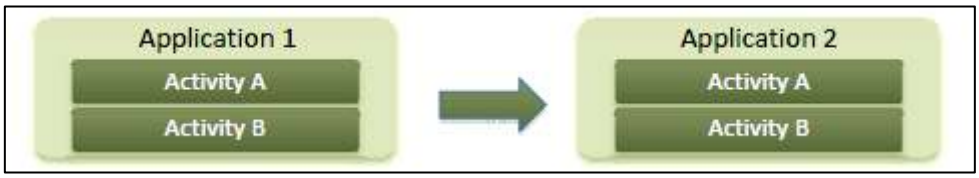


Figura 1.5 Interacción de *Activities* en Android [13]

Toda aplicación en Android tiene una actividad, la cual es lanzada a su inicio, desde esta *Activity* el usuario puede navegar por toda la aplicación. Otros ejemplos de actividades son:

- La pantalla de autenticación de una aplicación.
- La redacción de un correo electrónico.
- La edición de una imagen.

Todas las *Activities* están interconectadas para permitir la navegación; al inicio de una aplicación siempre hay una *Activity* principal para mostrar la pantalla principal al usuario. Los componentes para las interfaces de usuario se crean usando setContentView*(View)* y para cuando una *Activity* se crea, pausa, detiene o pasa a

segundo plano existen métodos que se encargan de estas tareas. Los más importantes son:

- *OnCreate(Bundle):* Este es el punto en el cual la actividad se inicia y cada *Activity Class* implementa este método. Usualmente, el método *setContentView(Int)* es llamado dentro de *OnCreate()* y define la interfaz de usuario de la pantalla o actividad. En cambio, el método *findViewById(Int)* se usa para buscar recursos e interactuar con ellos de manera programada.
- *OnPause():* Si un usuario decide abandonar una actividad, el proceso de guardar el estado e operaciones importantes se realiza por este método.

Otros métodos importantes para el control de flujo de la aplicación son: *onStart(), onRestart, onResume, onStop() y onDestroy().* Estos métodos se detallan posteriormente en el ciclo de vida de una aplicación.

1.4.1.2 Intents

Son mensajes a través de los cuales otros componentes de la aplicación (actividades, servicios y *Broadcast Receivers*) son activados. En otras palabras, son mensajes que especifican las acciones u operaciones que deben ser realizadas en determinado punto de la ejecución de la aplicación.

Mediante *Intents* Android provee un mecanismo para enlazar componentes de la aplicación luego del tiempo de ejecución (ya sea dentro de la misma aplicación o entre diferentes aplicaciones).

En resumen, los *Intents* son objetos que contienen información sobre las operaciones a ser realizadas; o en el caso de *Broadcast Receivers*, los detalles de un evento ocurrido.

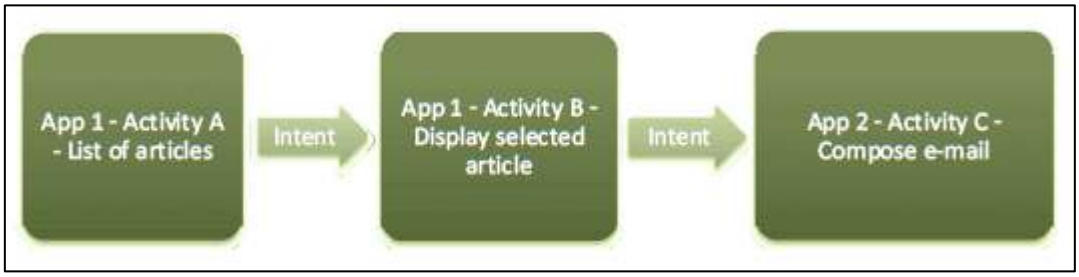


Figura 1.6 Uso de *Intents* [13]

La Figura 1.6 muestra cómo los *Intents* relacionan las diferentes actividades para una aplicación de noticias que además de leer los artículos permite enviarlos por correo electrónico.

1.4.1.3 Broadcast Receivers

Se utilizan para manejar los *Intents* y son medios utilizados por los cuales las aplicaciones Android y los componentes del sistema pueden comunicarse entre ellos. El receptor siempre está a la espera del *Intent* para realizar una acción determinada. El sistema (y aplicaciones) pueden transmitir *Intents* a cualquiera que esté interesado en recibirlas, sin embargo, esto puede controlarse mediante permisos de seguridad.

Luego de que un *Intent* se trasmite los receptores interesados que cuentan con los permisos necesarios ya pueden ser activados. Algunos ejemplos de *Intents* propios del sistema son:

- ACTION_TIME_TICK
- ACTION_TIME_CHANGED
- ACTION_BOOT_COMPLETED
- ACTION_PACKAGE_ADDED
- ACTION_PACKAGE_CHANGED

Cada uno de estos puede ser utilizado por diferentes aplicaciones, por ejemplo, para una aplicación de despertador dos *Intents* de interés son: ACTION_TIME_CHANGED y ACTION_TIMEZONE_CHANGED.

Los *Broadcast Receivers* por sí mismos no tienen un componente de interfaz de usuario y por tanto, la aplicación (mediante la *Activity*) define el método *OnReceive()* para ejecutar las acciones necesarias en función del mensaje. Para esto la *Activity* necesita heredar de la clase *Android.content.BroadcastReceiver* e implementar el método *onReceive()*. Los *Broadcast Receivers* deben estar registrados con la etiqueta <`receiver`> en el archivo `AndroidManifest.xml`. Un ejemplo de *Broadcast Receiver* se muestra en el Código 1.1.

```
<receiver
    android:name="com.google.android.gms.gcm.GcmReceiver"
    android:permission="com.google.android.c2dm.permission.SEND"
    android:exported="true"
    >
    <intent-filter
        >
        <action
            android:name="com.google.android.c2dm.intent.RECEIVE"
            >
        </action>
        <action
            android:name="com.google.android.c2dm.intent.REGISTRATION"
            >
        </action>
        <category
            android:name="com.APLICACION.GENERICA"
            >
        </category>
    </intent-filter>
</receiver>
```

Código 1.1 Registro de un *Broadcast Receiver* en el archivo AndroidManifest.xml

1.4.1.4 Services

Un servicio es un componente de la aplicación que puede realizar operaciones de larga duración en segundo plano. Un *Service* no cuenta con un componente de interfaz de usuario, pero ejecuta tareas en segundo plano, por ejemplo, un despertador o un reproductor de música. Un ejemplo de servicio declarado en el archivo `AndroidManifest.xml` se muestra en el Código 1.2.

Otras aplicaciones pueden estar corriendo aún mientras el usuario abra otra aplicación, adicionalmente, un componente de la aplicación puede limitarse a un servicio y de esta manera interactuar con esta en segundo plano. Por ejemplo, un componente de aplicación puede unirse a un reproductor de música e interactuar con este como necesite. Por lo tanto, el servicio puede estar en dos estados:

- *Started.*
- *Bound.*

Cuando un componente de la aplicación lanza un servicio, éste pasa al estado *Started.* Esto se hace mediante el método *startService().* Una vez que el servicio ha iniciado, puede continuar corriendo en segundo plano aun cuando la aplicación ya no se esté ejecutando.

Un componente de la aplicación puede enlazarse a sí mismo a un servicio mediante el método *bindService()*. En el estado *Bound* el servicio puede ser usado como un

mecanismo cliente-servidor y un componente puede interactuar con el servicio. El servicio sólo correrá si el componente se encuentra enlazado, una vez que éste se desenlaza, el servicio es destruido.

Para crear un servicio se debe crear una subclase de servicio e implementar métodos *Callback*. Los métodos más importantes de este tipo para servicios son: *onStartCommand(), onBind(), onCreate()* y *onDestroy().*

- *onStartCommand()*: Este método es llamado por el sistema cuando otro componente de la aplicación solicita el inicio de un servicio particular mediante el método *startService()*. Este servicio entonces correrá hasta encontrarse con *stopSelf()* o *stopService().*
- *onBind()*: Este método es llamado cuando otro componente quiere enlazarse al servicio mediante *bindService().*
- *onCreate()*: Cuando el servicio se crea por primera vez, este método realiza una configuración antes de llamar a *onStartCommand()* o *onBind()*.
- *onDestroy()*: Este método es llamado cuando el servicio ya no es necesario.

```
<service
    android:name="com.APLICACION.GENERICA.gcmnotifications.MyGcmListenerService"
    android:exported="false"
    >
    <intent-filter
        >
        <action
            android:name="com.google.android.c2dm.intent.RECEIVE"
            >
        </action>
    </intent-filter>
</service>
```

Código 1.2 Servicio en el archivo AndroidManifest.xml

1.4.1.5 Content Providers

Proveen a las aplicaciones de los medios necesarios para compartir datos persistentes. Un *Content Provider* puede verse como un repositorio de datos y diferentes aplicaciones pueden acceder a él. Las aplicaciones pueden compartir datos mediante *Intents* sin embargo, este no es un medio adecuado para compartir datos sensibles o persistentes y es por ello que se usan *Content Providers*.

Los datos contenidos sólo pueden ser manipulados por las aplicaciones que cuenten con los permisos correspondientes. El *Content Provider* debe ser declarado en el archivo `AndroidManifest.xml` con la etiqueta `<provider>` y su acceso se controla mediante los permisos `android:readPermission` y `android:writePermission` para controlar el tipo de operaciones que se permite realizar. Un ejemplo de esto se muestra en el Código 1.3.

Con un nivel de API de Android menor a 17, la propiedad por defecto de un *Content Provider* es siempre *exported*. Esto significa que, a menos de que el desarrollador especifique los permisos, cualquier aplicación puede acceder y consultar datos. El Código 1.3 muestra un ejemplo de un *Content Provider* que tiene la propiedad `exported="false"` para prevenir que otras aplicaciones accedan al contenido.

```
<provider
    android:name="com.google.android.gms.measurement.AppMeasurementContentProvider"
    android:exported="false"
    android:authorities="com.APLICACION.GENERICA.google_measurement_service"
    >
</provider>
```

Código 1.3 *Content Provider* en el archivo AndroidManifest.xml

Las *Activities* son pantallas con las cuales interactúa el usuario. Una aplicación típica consta de múltiples *Activities* entre las cuales el usuario puede navegar. El usuario puede también abrir una *Activity* de otra aplicación (mediante *Intents*).

Es fundamental entender el ciclo de vida de una *Activity*, ya que cuando esta es cambiada o terminada, es necesario implementar ciertos métodos *Callback*. Si una *Activity* no implementa los métodos requeridos esto puede conducir a problemas de rendimiento o confiabilidad [14].

Las *Activities* son manejadas como una pila; cuando el usuario navega por la aplicación, las *Activities* pasan por diferentes estados en su ciclo de vida. Por ejemplo, cuando una nueva *Activity* se inicia, se pone primera en la pila (es la que se muestra al usuario) y se envía a la que se encontraba corriendo anteriormente un nivel más abajo en la pila.

El sistema llama a diferentes métodos del ciclo de vida dependiendo del estado. Por ejemplo están los métodos: *onCreate, onRestart, onStart* o *onResume* dependiendo de si la *Activity* gana enfoque o pasa a segundo plano. El sistema

llama a un conjunto de diferentes métodos *Callback* (por ejemplo *onPause()*) cuando una *Activity* pierde enfoque. A continuación se definen varios estados de una *Activity* [15]:

- *Active/Running*: Una *Activity* se encuentra en este estado si se encuentra corriendo y tiene el enfoque del usuario.
- *Paused*: La *Activity* en este estado ha perdido enfoque del usuario pero aún es visible como cuando otra *Activity* que no llena completamente la pantalla toma el enfoque de usuario. La *Activity* todavía retiene información de estado y puede ser terminada en caso de que el sistema no tenga los recursos suficientes para mantenerla activa.
- *Stopped*: Si una *Activity* pierde enfoque debido a otra que llena completamente la pantalla y que ha ganado enfoque, el estado cambia a detenida. La *Activity* todavía retiene información de estado y puede ser terminada en caso de que el sistema no tenga los recursos suficientes para mantenerla activa.
- *Inactive/Killed*: El sistema puede terminar una *Activity* cuando esta se pausó o detuvo. En este caso toda información de estado se pierde y debe ser inicializada nuevamente.

En el momento en el que una *Activity* se destruye esta puede haber pasado por múltiples entre estar activa e inactiva. Resulta útil ver a la línea del tiempo de la aplicación desde tres diferentes puntos de vista:

- Línea del tiempo completa: Comprende la línea del tiempo de una *Activity* entre la primera llamada a *onCreate()* y *onDestroy()*. Esto incluye todas las iteraciones que la *Activity* atraviesa hasta que es destruida. *onCreate* configura el estado para una *Activity* (incluyendo los recursos) mientras *onDestroy* libera estos recursos.
- Línea del tiempo visible: Corresponde al tiempo en el cual el usuario ve la *Activity* en la pantalla. Esto ocurre entre un ciclo entre *onStart()* y *onStop()*.
- Línea del tiempo en Segundo Plano: Esta corresponde al tiempo entre *onResume()* y *onPause()*.

La Figura 1.7 muestra el ciclo de vida de una *Activity*. Los rectángulos representan los métodos *Callback* que se pueden implementar cuando la *Activity* cambia de estado mientras que los óvalos los diferentes estados en los que una puede estar la *Activity*.

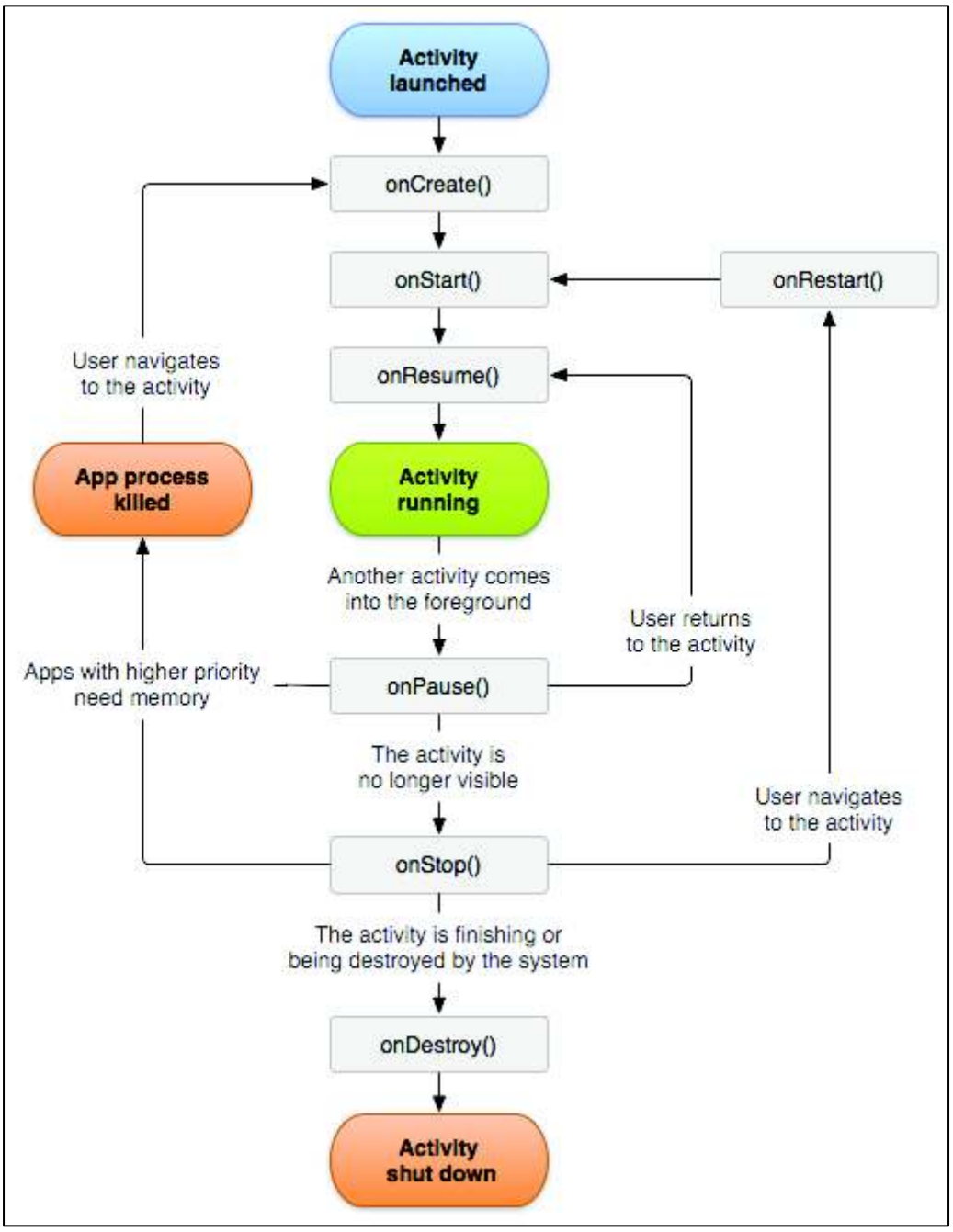


Figura 1.7 Ciclo de vida de una *Activity* y métodos *Callback* [16]

Los métodos *Callback* y sus descripciones relevantes con el ciclo de vida de la *Activity* se muestran en la Tabla 1.2.

Tabla 1.2 Descripción de los métodos *Callback* [17]

Método	Descripción
onCreate()	Se llama cuando una *Activity* se ejecuta por primera vez y se encarga de la configuración inicial.
onStart()	Se utiliza cuando una *Activity* pasa a primer plano y está disponible para interactuar con el usuario.
onRestart()	Se usa cuando una *Activity* se detiene anticipadamente y debe reiniciarse.
onResume()	Se utiliza cuando una *Activity* pasa a primer plano y empieza a interactuar con el usuario.
onPause()	Es llamado cuando el sistema necesita reanudar una *Activity* previamente pausada. Los cambios que necesitan ser guardados son realizados en este método antes de que la *Activity* se pause.
onStop()	Es llamado cuando una *Activity* ya no es visible al usuario.
onDestroy()	Es llamado cuando el sistema necesita liberar recursos.

1.5 MODELO DE PERMISOS[18]

El núcleo de la seguridad en el modelo de seguridad de Android es la separación de privilegios de Linux. Cada vez que una nueva aplicación es iniciada en el dispositivo se le asigna un identificador de usuario (UID), el cual posteriormente pasa a pertenecer a un determinado grupo predefinido.

De manera similar a Linux, todos los binarios usados como comandos se encuentran en la carpeta `/system/bin` y */system/xbin*. Además, los datos de la aplicación que se instalan desde la Play Store o cualquier otra fuente, se guardan en la carpeta */data/app* mientras que para aplicaciones que son de pago los datos se almacenan en la carpeta */data/app-private/*. Es necesario mencionar que se pueden acceder a estos datos sólo en el caso de que el dispositivo permita el acceso como superusuario (modo *root* en GNU/Linux).

Una aplicación en Android es un archivo .apk (*Android Package*), esta extensión de archivo no es más que un comprimido de todos los componentes necesarios para el funcionamiento de la aplicación y estos componentes pueden ser extraídos con cualquier herramienta como 7-Zip [19] o Winrar [20]. Para mantener la seguridad tanto para el sistema como para el usuario cada aplicación que es instalada en el dispositivo solicita previamente el acceso a un conjunto de permisos con el fin de

acceder a ciertos componentes en hardware del dispositivo o a su vez a datos almacenados. En función de la sensibilidad de la información el sistema puede conceder los permisos de manera automática o solicitar al usuario su aprobación[21].

Dado que cada aplicación tiene su propia instancia, esta no tiene acceso a los datos de otra aplicación a menos que esto sea especificado en los proveedores de contenido [22]. Todos estos permisos solicitados se pueden observar al momento de instalar una aplicación desde cualquier origen como por ejemplo la Play Store. La Figura 1.8 muestra un ejemplo al respecto.

Figura 1.8 Permisos solicitados

De acuerdo al tipo de aplicación que se instala, se solicitarán los permisos correspondientes para su correcto funcionamiento. Pedir una cantidad de permisos mayor a los necesarios para el funcionamiento de la aplicación es una práctica común de aplicaciones maliciosas. A continuación se detallan algunos de los permisos solicitados [23]:

- **Identidad:** Utiliza uno o varios de estos datos: cuentas del dispositivo, datos de perfil.
- **Historial de aplicaciones y dispositivo:** Permite que la aplicación vea uno o varios de estos datos: información sobre la actividad del dispositivo, las aplicaciones que se están ejecutando, el historial de navegación y los marcadores.

- **Ubicación:** Usa la ubicación del dispositivo.
- **SMS:** Usa una o varias de estas funciones: SMS, MMS. Es posible que se apliquen cargos.
- **Teléfono:** Usa una o varias de estas funciones como teléfono, registro de llamadas. Es posible que se apliquen cargos.
- **Fotos/multimedia/archivos:** Utiliza uno o varios archivos del dispositivo, como imágenes, vídeos o audio, o de almacenamiento externo del dispositivo.
- **Cámara:** Utiliza las cámaras del dispositivo.
- **Información sobre la conexión Wi-Fi:** Permite que la aplicación vea información sobre la conexión a redes Wi-Fi (por ejemplo, si está habilitada la conexión Wi-Fi y los nombres de los dispositivos Wi-Fi conectados.)
- **ID de dispositivo y datos de llamada:** Permite que la aplicación determine el número de teléfono y los ID del dispositivo, si una llamada está activa y el número remoto conectado por una llamada.

Un desarrollador de aplicaciones de Android debe especificar los permisos necesarios al momento de programar la aplicación, para esto se utiliza el archivo `AndroidManifest.xml`. Este archivo contiene información referente a la aplicación tal como la versión mínima de Android que requiere para su funcionamiento, el nombre del paquete, la lista de actividades (pantallas visibles al usuario), servicios (procesos en segundo plano de la aplicación), y los permisos requeridos, etc [24].

Si al momento de desarrollar la aplicación estos permisos no son especificados en el archivo `AndroidManifest.xml`, la aplicación colapsa y consulta al usuario si desea forzar el cierre de la misma. En el Código 1.4 se muestran un extracto de los diferentes permisos solicitados con las etiquetas `<uses-permission>`.

A todas las aplicaciones se les asigna un identificador único de usuario (UID) la primera vez que se instalan e inician. Todos los usuarios con un UID determinado pertenecen a un grupo en particular dependiendo se los permisos que solicita. Por ejemplo, una aplicación que sólo necesita acceso a Internet pertenece al grupo *inet*.

Una aplicación pertenece a varios grupos en función de los permisos que solicita. Dicho de otra manera, cada usuario (aplicación) puede pertenecer a múltiples grupos y cada grupo cuenta con varios usuarios. Los grupos tienen un nombre único definido por el identificador de grupo (GID). En la Figura 1.9 se muestran algunos ejemplos tomados del archivo `platform.xml` ubicado en el directorio `system/etc/permissions/`.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:versionCode="17"
    android:versionName="4.6"
    platformBuildVersionName="6.0-2704002"
    platformBuildVersionCode="23"
    package="com.APLICACION.GENERICA"
    >
    <uses-sdk
        android:minSdkVersion="16"
        android:targetSdkVersion="22"
        >
    </uses-sdk>
    <uses-permission
        android:name="android.permission.CAMERA"
        >
    </uses-permission>
    <uses-permission
        android:name="android.permission.CALL_PHONE"
        >
    </uses-permission>
    <uses-permission
        android:name="android.permission.ACCESS_NETWORK_STATE"
        >
    </uses-permission>
    <uses-permission
        android:name="android.permission.INTERNET"
        >
    </uses-permission>
```

Código 1.4 Archivo AndroidManifest.xml

Esto aclara el panorama respecto a las aplicaciones nativas que corren en los dispositivos Android, dado que estas interactúan de manera directa con el procesador en lugar de correr en una instancia de una máquina virtual lo cual no afecta al modelo de seguridad general en ninguna forma.

Cada aplicación cuenta con un directorio independiente en el cual almacena los datos necesarios para su funcionamiento, este directorio se encuentra en la ruta: `data/data/[nombre del paquete]`. De igual forma, todas las carpetas que contienen los datos de la aplicación tienen el mismo identificador de usuario (UID), el cual forma parte de la base del modelo de seguridad usado en Android [11].

Dependiendo del UID y los permisos de los archivos, se restringe tanto el acceso como la modificación desde otras aplicaciones con un UID diferente.

```
<!-- The following tags are associating low-level group IDs with
     permission names.  By specifying such a mapping, you are saying
     that any application process granted the given permission will
     also be running with the given group ID attached to its process,
     so it can perform any filesystem (read, write, execute) operations
     allowed for that group. -->

<permission name="android.permission.BLUETOOTH_ADMIN" >
    <group gid="net_bt_admin" />
</permission>

<permission name="android.permission.BLUETOOTH" >
    <group gid="net_bt" />
</permission>

<permission name="android.permission.BLUETOOTH_STACK" >
    <group gid="net_bt_stack" />
</permission>

<permission name="android.permission.NET_TUNNELING" >
    <group gid="vpn" />
</permission>
```

Figura 1.9 Archivo platform.xml

Sin embargo, es posible acceder al contenido de la tarjeta de memoria externa sin ningún tipo de permiso. De igual manera, una vez que se tiene acceso a los datos, estos se pueden enviar mediante una petición POST/GET a un servidor remoto en el cual se almacenarán. En este caso, se puede utilizar *malware* que no requiera de ningún tipo de permiso [25].

1.6 PROCESO DE FIRMADO DE APLICACIONES

El firmado de aplicaciones es una de las características de Android que ha contribuido a su éxito debido tanto a su apertura como a la comunidad de desarrolladores. Hay más de un millón de aplicaciones disponibles en la Play Store gracias a que cualquier persona puede descargar el *kit* de desarrollo (Android SDK) para crear su aplicación y posteriormente subirla a la tienda.

De manera general se consideran dos tipos de mecanismos de firma de certificados: la primera es mediante una autoridad certificada (CA) y la otra es mediante un certificado generado por el propio desarrollador [26].

El firmado mediante la autoridad certificadora es más común en el modelo de aplicaciones iOS que maneja Apple, en la cual cada aplicación que un usuario sube a la App Store es verificada y posteriormente firmada con un certificado Apple. Cuando esta es descargada a un dispositivo éste verifica si la aplicación ha sido firmada por una autoridad certificada Apple y sólo entonces permite ejecutar la aplicación [27].

Contrario a Apple, en Android no existe una autoridad certificada, en lugar de eso cada desarrollador tiene su propio certificado para firmar sus aplicaciones. Hecho esto se carga la aplicación que posteriormente es verificada por *Google Bouncer*, un entorno virtual que verifica si la aplicación es maliciosa. Luego de la verificación la aplicación aparece en la Play Store y Google no realiza ningún firmado de la misma. Cada desarrollador puede crear su propio certificado usando la herramienta *Keytool* que viene incluida en el SDK o usando la interfaz gráfica de Eclipse [28].

Una vez que el desarrollador ha firmado la aplicación con el certificado que creó, se debe mantener la llave del certificado en un sitio seguro para prevenir que alguien la utilice para firmar otras aplicaciones.

Se puede verificar la firma de una aplicación si se tiene acceso al archivo .apk usando la herramienta Jarsigner [29], también incluida en el SDK de Android.

1.7 TIPOS DE APLICACIONES MÓVILES [30]

De manera general, las aplicaciones están diseñadas para ejecutarse ya sea directamente en la plataforma para la cual fueron creadas, sobre el navegador web del dispositivo, o usando una combinación de ambas características. A continuación se describen las características de los diferentes tipos de aplicaciones móviles:

- Aplicación Móvil Nativa: Al instalarse, este tipo de aplicaciones almacenan la mayoría de su código en el dispositivo. La información que requieren estas aplicaciones es solicitada usando protocolo HTTPS [31]. Estas aplicaciones son diseñadas para un tipo de plataforma en específico, por ejemplo, Android o iOS [32].

  Las aplicaciones nativas tienen la capacidad de proveer un gran desempeño con un alto grado de confiabilidad. Como ventaja, debido a su integración cercana con el sistema operativo, este tipo de aplicaciones pueden acceder directamente a los componentes del dispositivo como: cámara, sensores, etc. La desventaja principal de este tipo de aplicaciones es que por estar diseñadas para un tipo de plataforma en específico, se deben tener bases de código para cada plataforma y el mantenimiento del código es más complejo en relación a una aplicación web.

- Aplicación Móvil Web: Son sitios web diseñados para verse como una aplicación nativa. Estas aplicaciones corren sobre el navegador web del dispositivo móvil y son desarrolladas utilizando HTML5, CSS, JavaScritpt, etc [33]. Los íconos que la componen son creados para parecer una aplicación nativa, sin embargo, estos íconos son como el enlace a una página web, simplemente abren el navegador web por defecto para cargar la página web deseada, por lo tanto, las aplicaciones web no necesitan instalarse [34].

  Las aplicaciones web tienen una integración limitada con los componentes del dispositivo y tienen un menor rendimiento en relación a una aplicación nativa. La ventaja de este tipo de aplicaciones es que está orientada a varias plataformas, por ejemplo Android e iOS. Esto permite que el costo de desarrollo de la aplicación se reduzca ya que se mantiene una misma base de código para varias plataformas.

- Aplicación Móvil Híbrida: Este tipo de aplicaciones se ejecuta como una aplicación nativa, pero la mayoría de los procesos se realizan mediante tecnologías web, esto implica que una parte de la aplicación se ejecuta en un navegador web embebido. Debido a estas características, este tipo de aplicaciones heredan las ventajas y desventajas de las aplicaciones nativas y web.

1.8 GOOGLE BOUNCER Y DETECCIÓN DE APLICACIONES MALICIOSAS EN LA PLAY STORE

El éxito y acogida que ha tenido Android se ha debido en gran parte a la facilidad con la que los desarrolladores pueden subir sus aplicaciones a la tienda oficial (Play Store), de este modo, tanto el número como la variedad de aplicaciones se ha ampliado considerablemente. Inicialmente no existía un proceso mediante el cual se revisen las aplicaciones subidas a la Play Store, pero esto ocasionó el aparecimiento de una gran cantidad de aplicaciones maliciosas. Para controlar esto Google ha creado varias herramientas, la primera de ellas es Bouncer [35].

1.8.1 GOOGLE BOUNCER[36]

Desde su aparición, este servicio fue concebido para proveer un escaneo automático de las aplicaciones disponibles en la tienda sin afectar a la experiencia de usuario ni requerir que los desarrolladores atraviesen un largo proceso de aprobación para sus aplicaciones.

El servicio realiza una serie de análisis sobre las nuevas aplicaciones, aplicaciones existentes en la tienda y sobre las cuentas de los desarrolladores. Así es como funciona: Una vez que la aplicación se ha cargado, Bouncer comienza a analizarla en busca de *malware, spyware* o Troyanos. También revisa si la aplicación tiene un comportamiento anómalo y la compara con aplicaciones previamente analizadas para detectar posibles alertas.

Además, se ejecuta cada una de las aplicaciones en la infraestructura en la nube de Google y se simula cómo sería su funcionamiento en un dispositivo Android para buscar comportamiento malicioso. Las cuentas de los desarrolladores también son revisadas para ayudar a prevenir que aquellos desarrolladores malintencionados sigan subiendo aplicaciones maliciosas.

1.8.2 GOOGLE APP VERIFICATION SERVICE[37]

Desde la versión 4.2 de Android (Jelly Bean), Google ha introducido esta nueva característica con el objetivo de detectar si las nuevas aplicaciones instaladas son o no maliciosas. Cuando una aplicación es instalada, el servicio (si se encuentra habilitado) empieza a recolectar y enviar información sobre la aplicación (nombre, tamaño, versión, etc.) así como información del dispositivo (dirección IP, identificador de dispositivo, etc) a la nube de Google. Luego de esto, la nube responde con un resultado del análisis. Si la aplicación no es segura, el usuario es notificado con una alerta calificando la aplicación como “peligrosa” o “potencialmente peligrosa”. Las aplicaciones “peligrosas” son bloqueadas antes de instalarse mientras que las “potencialmente peligrosas” alertan al usuario, pero le dan la posibilidad de instalarlas o no según lo decida. Este proceso se ilustra en la Figura 1.10.

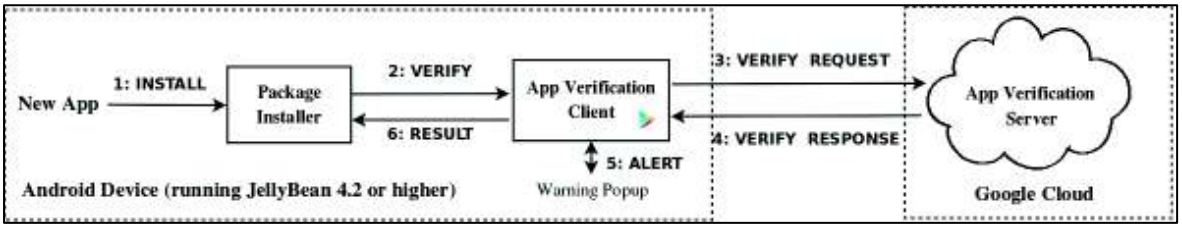


Figura 1.10 Estructura de Google App Verification Service [37]

1.8.3 GOOGLE PLAY PROTECT [38]

Google Play Protect constituye un avance importante en cuanto a temas de seguridad ya que este servicio además de proteger al usuario contra aplicaciones potencialmente dañinas, permite también controlar otras políticas como: Contenido restringido, propiedad intelectual, etc. Además de analizar el dispositivo, este servicio analiza un promedio de 50 mil millones de aplicaciones por día para constatar que no se infrinjan las políticas de Google. Una imagen de este servicio se muestra en la Figura 1.11.

En el proceso actual de publicación de una aplicación se utilizan no sólo algoritmos, sino que también hay un equipo especializado que se encarga de la verificación manual de las aplicaciones en caso de que llegaran a existir anomalías en el comportamiento de alguna de ellas.

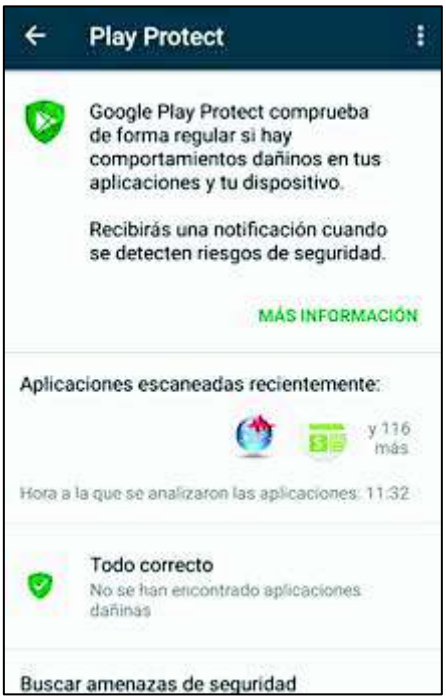


Figura 1.11 Google Play Protect

1.9 PROBLEMAS DE SEGURIDAD EN LOS DISPOSITIVOS MÓVILES [39]

La plataforma Android experimenta varios problemas de seguridad como en cualquier otro sistema operativo. Los problemas que se tratan a continuación son comunes a todas las plataformas móviles, no sólo a Android, e incluso algunos de ellos se encuentran en dispositivos tradicionales como laptops.

1.9.1 DISPOSITIVO

La mayoría de las personas cuentan con al menos un dispositivo móvil y antes de los *smartphones* lo único que preocupaba era la pérdida de los contactos.

Hoy en día un *smartphone*, además de información como mensajes de texto, historial de llamadas, etc; contiene otros datos como:

- E-mails guardados.
- Cuentas abiertas de Facebook, YouTube, Twitter, etc.
- Datos de salud.
- Información de cuentas bancarias.
- Ubicación y datos GPS.

A menos que el dispositivo se encuentre encriptado, la pérdida del dispositivo móvil conlleva también un riesgo de divulgación de datos. Para lo cual sólo es necesario conectar el dispositivo móvil a una computadora y contar con las herramientas adecuadas.

1.9.2 PATCHING

El término hace referencia a la aplicación de “parches” de seguridad, los cuales se van realizando posteriormente al lanzamiento de una versión oficial de Android. El problema es que, con la aparición de nuevas versiones del sistema operativo, los fabricantes dejan de preocuparse por los parches de las versiones anteriores y se concentran en las actuales. De este modo los dispositivos con versiones antiguas de Android quedan expuestos a vulnerabilidades.

1.9.3 ALMACENAMIENTO EXTERNO

El almacenamiento externo extraíble comprende otro problema de seguridad ya que es más fácil perder una tarjeta SD que perder un dispositivo móvil. En la mayoría de los casos, los datos de la tarjeta no se encuentran encriptados, brindando un acceso fácil a los datos del usuario. Este tipo de almacenamiento es también más frágil, por lo que es susceptible a corrupción y/o pérdida de datos.

1.9.4 TECLADO

Dado que las pantallas táctiles son muy comunes en los dispositivos móviles éstas conllevan también un riesgo. Esto ya que son propensas a que personas indeseadas miren todo cuanto se realiza en ellas sobre todo en lugares públicos ya que, con sólo mirar de reojo, una persona puede descubrir lo que el usuario está escribiendo; incluso las manchas que se crean sobre la pantalla pueden brindar pistas sobre el PIN o patrón de desbloqueo facilitando el trabajo de un atacante.

1.9.5 PRIVACIDAD DE DATOS

Una de las aplicaciones más populares de Android es Google Maps. Muchas otras aplicaciones con los permisos necesarios también pueden acceder a los datos de ubicación del usuario, por lo que con un monitoreo permanente de estos datos se pueden conocer hábitos del usuario como lugares que frecuenta, horarios, etc.

1.9.6 SEGURIDAD DE LA APLICACIÓN

Las aplicaciones móviles también son susceptibles a ataques como los realizados a aplicaciones tradicionales. Ataques de inyección SQL (SQLi), *Cross Site Scripting* (XSS), etc, no sólo que son posibles en plataformas móviles, sino que pueden conducir a ataques más serios dada la sensibilidad de la información almacenada en los dispositivos móviles.

1.9.7 CÓDIGO HEREDADO

Gran parte del código usado por los teléfonos móviles para la comunicación GSM o CDMA no ha cambiado con el paso de los años. Los *drivers* de los dispositivos fueron escritos sin tomar en cuenta prácticas de seguridad y eso los hace vulnerables. Los dispositivos actuales continúan usando este mismo código y, de hecho, el nuevo código que se desarrolla se escribe sobre el código existente.

1.9.8 EXCESO DE PERMISOS

Uno de los vectores de ataque que utilizan las aplicaciones maliciosas son los permisos que solicita para su instalación. Ya que este tipo de aplicaciones solicitan un conjunto de permisos mayor al necesario para su funcionamiento. Por ejemplo, es sospechoso que una aplicación que sólo se encarga de mostrar el calendario solicite permisos de acceso a fotos, documentos, cámara, ubicación, etc.

1.10 METODOLOGÍA OWASP MOBILE SECURITY PROJECT[40]

Este proyecto es un recurso para brindar a los desarrolladores y equipos de seguridad una guía para diseñar y mantener aplicaciones seguras. Este proyecto clasifica los riesgos en la seguridad de aplicaciones móviles y provee controles en el desarrollo para reducir el impacto o la probabilidad de explotación.

El OWASP *Mobile Security Project* se enfoca en la capa Aplicación tomando en cuenta los riesgos inherentes a la plataforma móvil. Adicionalmente, se toman en cuenta no solo las aplicaciones instaladas en los dispositivos de usuario final, sino también en la infraestructura del lado del servidor, con la cual estas aplicaciones se comunican.

La mayor prioridad del proyecto OWASP *Mobile Security Project* es ayudar a estandarizar y difundir la metodología para la evaluación de la seguridad de aplicaciones. La ventaja de esta metodología es que se puede adaptar a las necesidades que quien prueba la aplicación, mediante la selección de los pasos que se requieran, dentro de las fases siguientes:

- Recolección de información.
- Análisis Estático.
- Análisis Dinámico.

1.10.1 RECOLECCIÓN DE INFORMACIÓN

Esta fase describe los pasos y elementos a considerar cuando se inicia con el reconocimiento y manejo de la aplicación a fin de determinar las áreas más críticas. Para esta fase y las necesidades del presente trabajo se toman en cuenta los siguientes pasos:

- ❖ Identificar la funcionalidad de la aplicación y el flujo de trabajo de la misma.

- Identificar las interfaces de red que usa la aplicación.
- Determinar si la aplicación permite transacciones comerciales.
- Identificar los componentes de hardware con los que la aplicación puede interactuar, por ejemplo: GPS, Bluetooth, etc.
- Identificar si la aplicación interactúa con otras aplicaciones, servicios o datos como: llamadas, contactos, redes sociales, etc.

1.10.2 ANÁLISIS ESTÁTICO

Durante esta fase se analiza directamente el código fuente de la aplicación. Para el caso de una prueba de tipo *Black Box*, en la cual sólo se tiene acceso al archivo .apk ya compilado, se requiere de un paso previo para obtener el código fuente de la aplicación bancaria. Para esto se consideran los siguientes puntos.

1.10.2.1 Pasos Iniciales

Cuando ya se cuenta con el archivo .apk a analizar se realizan las siguientes tareas:

- Obtener el código fuente de la la aplicación bancaria.
- Revisar los permisos que la aplicación solicita en el archivo `AndroidManifest.xml`.
- Determinar el tipo de aplicación (Nativa, web o híbrida).
- Identificar las librerías utilizadas y su estado de actualización.
- Determinar qué tipo de objetos se utilizan para realizar las vistas de la aplicación, si implementan funcionalidad de navegador web o cuentan con controles de interfaz de usuario.

1.10.2.2 Autenticación

Este punto se centra en el proceso de autenticación de la aplicación hacia el servidor, ya que es un punto crítico debido a la naturaleza de los datos que se manejan (usuarios, contraseñas, etc.). Para esto se tienen los siguientes pasos:

- Localizar el código que maneja la autenticación de usuario a través de la interfaz.
- Chequear si la autenticación se realiza *online/offline*.

- Determinar si la aplicación utiliza información más allá de usuario/ *password* como, por ejemplo: identificador de dispositivo, ubicación, certificados, tokens, etc.
- Método utilizado para autenticación: si se realiza mediante patrones o mediante usuario/clave convencional.
- Bloqueo de acceso a la cuenta en caso de comportamiento anómalo en la misma.
- Uso de autenticación única al igual que se lo hace en aplicaciones como Facebook, Google Apps, etc.
- Uso de autenticación mediante SMS.

1.10.3 ANÁLISIS DINÁMICO

Con la ayuda de los datos obtenidos durante las fases de recolección de datos y análisis estático, se puede continuar con la última fase: el análisis dinámico. Una vez tomado en cuenta esto, se ejecuta la aplicación ya sea en un entorno virtual o en un dispositivo Android físico. Se interactúa con la aplicación para determinar las funciones que involucren posibles vectores de ataque a la aplicación, como por ejemplo: uso de protocolos de comunicación inseguros (por ejemplo HTTP), acceso a la información del cliente por mala configuración de Proveedores de Contenido, etc. Los pasos para esta fase son:

- Intercepción de tráfico para ver los protocolos de comunicación utilizados.
- Análisis de vulnerabilidades mediante *scripts* existentes.
- Prueba de filtrado de información del proveedor de contenido.
- Ataque de inyección desde el lado del cliente.

Sin embargo, la metodología OWASP *Mobile Security Project* sólo menciona los puntos que deben considerarse en el análisis, más no el procedimiento ni las herramientas necesarias. Por esto, en el segundo capítulo se define un protocolo de pruebas que incluya el procedimiento para cada paso, considerando las fases de recolección de información, análisis estático y análisis dinámico, así como las herramientas necesarias.

CAPÍTULO 2
DISEÑO DEL PROTOCOLO

2.1 INTRODUCCIÓN

Se utilizó como referencia la metodología OWASP *Mobile Security Project* para llevar a cabo el análisis propuesto. Si bien esta metodología menciona los puntos necesarios para el análisis, no indica ni las herramientas ni los pasos a realizar en detalle. En el presente capítulo se describe el protocolo de pruebas con sus respectivas herramientas.

En primer lugar, en la sección de virtualización se comparan dos opciones para tener un ambiente Android. Además, se describe el procedimiento necesario para tener listo el dispositivo virtual Android, tanto para la opción utilizando Virtual Box [41], como para la opción en la cual se usa el *Android Virtual Device Manager* [42].

A continuación, se presentan algunas de las herramientas disponibles para realizar tanto análisis estático, como análisis dinámico. Además, junto con las herramientas disponibles se presentan las características con base en las cuales serán o no seleccionadas para el diseño del protocolo de pruebas.

Posteriormente, se definen las herramientas a utilizar para el diseño del protocolo de pruebas, tomando en cuenta las características descritas en la sección anterior.

En la sección siguiente, se presenta el protocolo definido para cada una de las fases: recopilación de información, análisis estático y análisis dinámico, utilizando las herramientas seleccionadas en el paso anterior.

Finalmente, se describen los resultados que se espera obtener al ejecutar el protocolo de pruebas sobre las aplicaciones bancarias. Esto se resume mediante una tabla que, detalla la herramienta y comandos necesarios en cada paso.

Con el objetivo de mantener la privacidad, los nombres, datos personales y demás información específica respecto a la aplicación que se está analizando, o a datos de sus clientes, ha sido anonimizada mediante el ocultamiento de parte de las imágenes o la sustitución de la información sensible en los logs, por caracteres “X”.

2.2 VIRTUALIZACIÓN DE ANDROID

Para la selección de herramientas de virtualización se consideraron aquellas que permiten crear un dispositivo Android sobre el cual se pueda ejecutar el protocolo de pruebas. Tomando esto en cuenta se realiza la preparación del ambiente utilizando *Android Virtual Device Manager* [42] como una alternativa, dado que permite crear varios dispositivos Android con ajustes tales como: versión de Android, tipo de dispositivo, etc. Como otra opción, se utiliza una máquina virtual en Virtual Box [41] creada mediante la imagen ISO del sistema operativo Android [43]. Con esto se consigue comparar las ventajas de estas dos opciones.

Además, previo a la creación del ambiente virtualizado, es necesario crear una cuenta de Gmail, la cual es utilizada para poder acceder a la tienda oficial de aplicaciones Android, Google Play [44], desde donde serán descargadas las aplicaciones bancarias a ser evaluadas posteriormente.

2.2.1 PREPARACIÓN DEL AMBIENTE MEDIANTE UNA MÁQUINA VIRTUAL

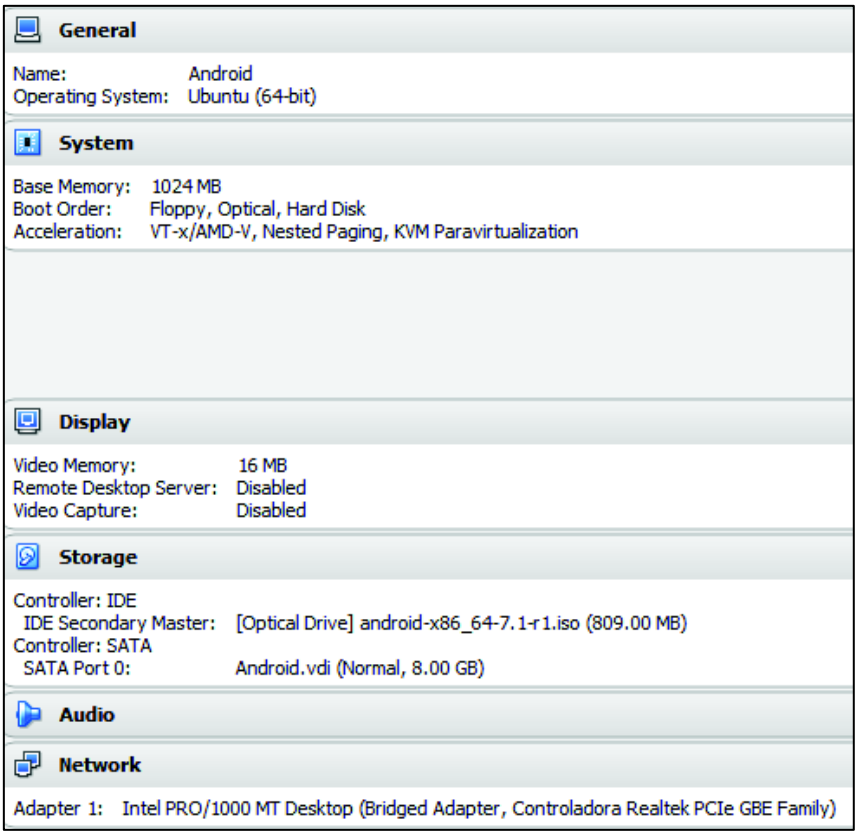


Figura 2.1 Máquina virtual con Android

En primer lugar se debe obtener la imagen del sistema operativo Android a virtualizar, esta imagen se descarga desde: [45], o a su vez desde [46]. En estas páginas existen varias versiones de Android disponibles, pero se recomienda utilizar una versión superior a la 6.0 ya que, como se ve en la Figura 1.2, es la versión más utilizada a febrero del 2018.

Una vez con la imagen de Android descargada, se procede a crear la máquina virtual, para este caso se ha utilizado Virtual Box. Se ha optado por esta opción por ser software libre, además de que provee las mismas características que VMWare [47] para crear una máquina virtual con sistema operativo Android. Como se muestra en la Figura 2.1, la máquina virtual se crea mediante la imagen del sistema operativo Android y utiliza una configuración de red tipo puente, esto facilita pasos posteriores como el análisis de tráfico.

Esto se debe a que la IP que esta máquina recibe estará en la misma red que la máquina física.

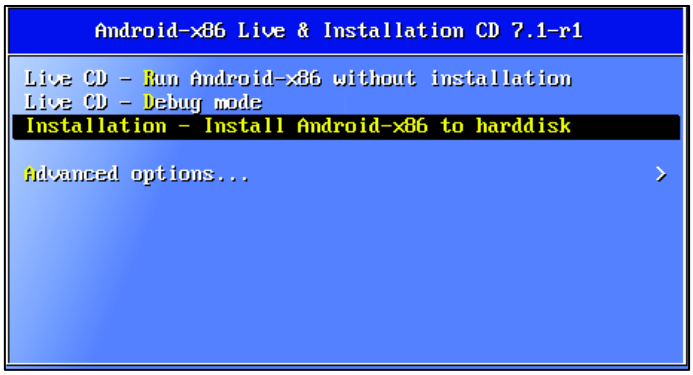


Figura 2.2 Proceso de instalación de máquina virtual Android

Posteriormente, una vez iniciada la máquina virtual, se presentan las opciones de la Figura 2.2, de la cual se selecciona la opción de instalar.

Luego, se selecciona la partición que fue creada por la máquina virtual, y se le da formato, a continuación se omite la instalación del *boot loader GRUB* ya que este sirve para brindar opciones de arranque del sistema Android y no será utilizado. Se solicita un espacio para el almacenamiento permanente de archivos, el cual se utiliza más adelante.

Una vez realizado los pasos anteriores se selecciona la opción de ejecutar Android como se muestra en la Figura 2.3.

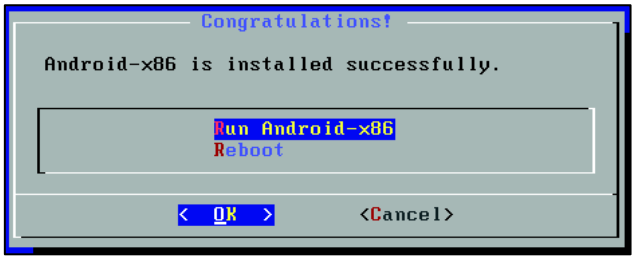


Figura 2.3 Paso final del proceso de Instalación de Android en la máquina virtual.

Hecho esto, el sistema Android inicia y solicita la configuración necesaria de idioma, red Wi-Fi, cuenta de Google, nombre del equipo y opciones de privacidad. Con esto ya se tiene el sistema operativo listo para poder descargar e instalar las aplicaciones como se muestra en la Figura 2.4.

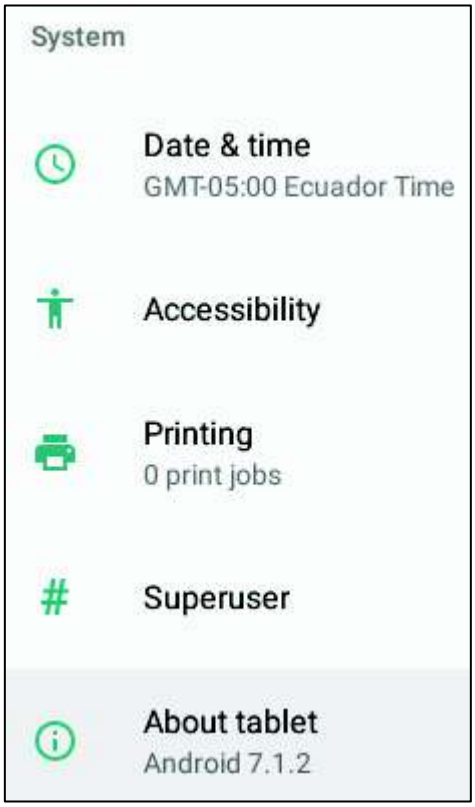


Figura 2.4 Sistema Operativo Android Listo

Para este ejemplo se ha utilizado la versión 7.1.2 de Android pero el procedimiento es el mismo para cualquiera de las versiones, desde la 4.4 hasta la 7.1. La desventaja de utilizar esta forma de virtualizar es que toma más de media hora tener lista la máquina virtual, además, no permite una conexión mediante ADB (*Android Debug Bridge*), lo cual es necesario para la fase de análisis estático.

La imagen de Android que se utilizó para la creación de esta máquina virtual corresponde a la de una *tablet*, además el sistema operativo se encuentra "rooteado", lo que podría restringir el funcionamiento de ciertas aplicaciones que controlan esta característica.

2.2.2 PREPARACIÓN DEL AMBIENTE MEDIANTE ANDROID VIRTUAL DEVICE MANAGER

Android Virtual Device Manager (AVD) [48] forma parte del Android SDK. AVD Manager crea dispositivos virtuales Android con diferentes configuraciones en cuanto a la versión de Android, tipo de dispositivo, tamaño de pantalla, memoria, etc.

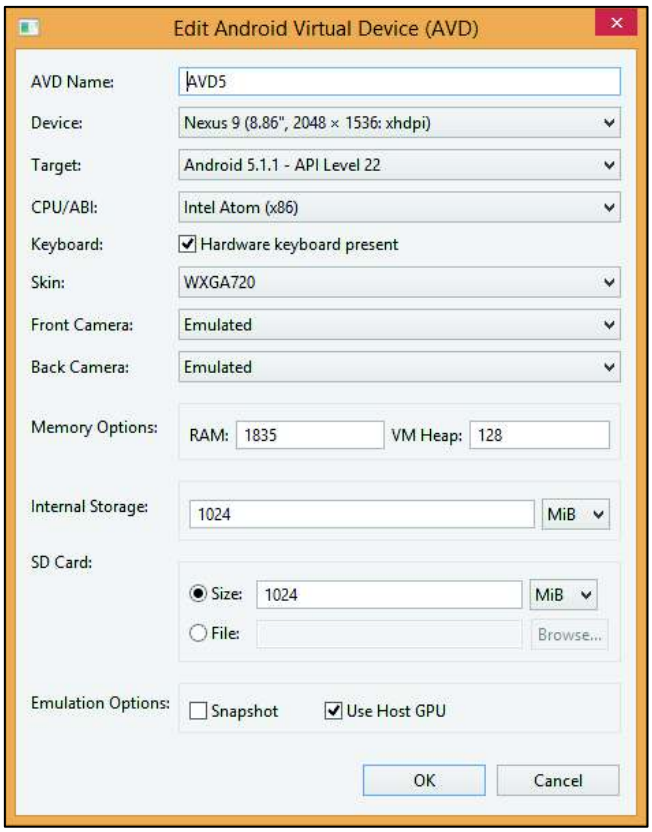


Figura 2.5 Propiedades del dispositivo virtual

Android Virtual Device Manager se encuentra en la carpeta `Android/sdk/AVD manager`, luego de Instalar Android Studio [49]. En la pantalla principal se muestran los dispositivos virtuales creados, como se ve en la Figura 2.6. Para crear un dispositivo virtual, primero se descarga la versión de Android que se desea utilizar

en la pestaña `Tools/Manage SDK`, hecho esto ya se puede crear el dispositivo virtual de acuerdo a las necesidades como se muestra en la Figura 2.5.

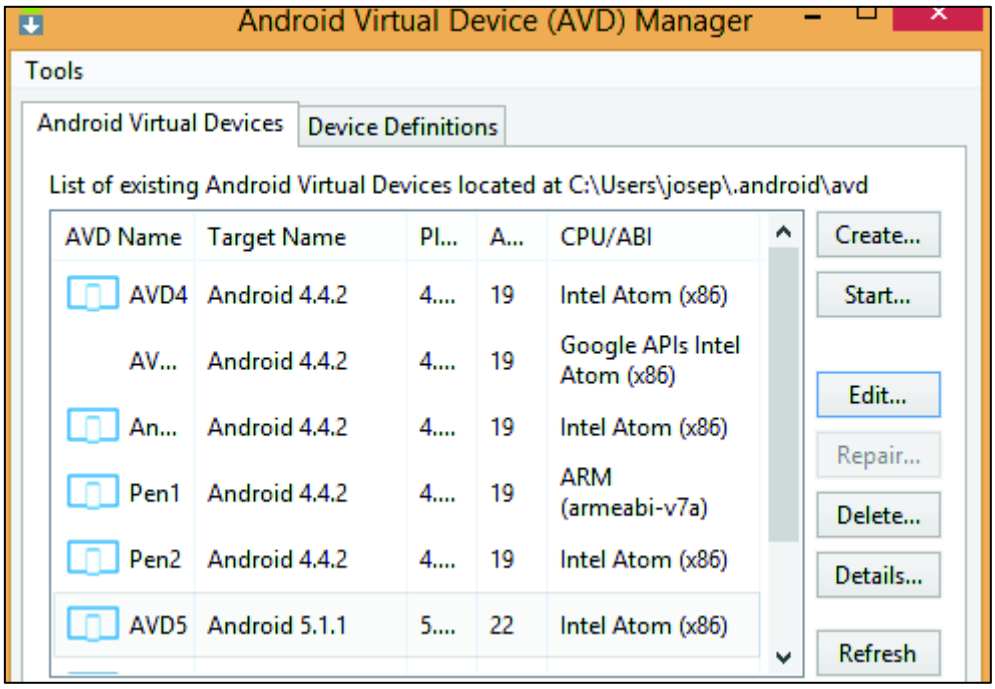


Figura 2.6 Pantalla principal del AVD Manager

2.2.3 RESUMEN DE CARACTERÍSTICAS PARA VIRTUALIZACIÓN DE ANDROID

Tabla 2.1 Características de herramientas de virtualización para Android

Herramienta	Ventajas	Desventajas
Máquina Virtual	• Independiente de la instalación de otra herramienta.	• Toma más tiempo en relación a AVD. • El funcionamiento de la máquina virtual no brinda fluidez para la navegación por el dispositivo Android.
AVD	• Más opciones configurables (Procesador, versión de Android, etc). • Se pueden configurar diferentes tipos de dispositivos (*smartphone*, *tablet*, etc). • Permite realizar cambios sobre el dispositivo virtual, aún luego de ser creado.	• Requiere de la instalación de Android Studio.

Con base en la Tabla 2.1, se optó por la utilización de un dispositivo virtual creado mediante AVD Manager, ya que es más rápido en comparación con crear una máquina virtual mediante la imagen .ISO de Android. Además, un AVD puede conectarse mediante ADB y sus características pueden ser modificadas con mayor facilidad, también, se puede seleccionar el tipo de dispositivo a crear (*tablet*, *smartphone*, etc.).

2.3 HERRAMIENTAS PARA ANÁLISIS ESTÁTICO

En esta sección se realiza la selección de herramientas para el análisis estático de las aplicaciones bancarias. La selección se realiza con base en las ventajas o desventajas que cada una ofrece, estas se encuentran resumidas en la Tabla 2.2, en la sección 2.3.6.

Para cada una de las tareas que comprenden el análisis estático, se presentan algunas de las herramientas disponibles. Se presenta además una descripción de sus características, las cuales se consideran posteriormente como ventaja o desventaja.

Finalmente, luego de la Tabla 2.2, se concluye con un párrafo indicando las herramientas que se ha seleccionado para el protocolo de pruebas con base en sus ventajas y desventajas.

2.3.1 OBTENCIÓN DEL ARCHIVO APK DE LA APLICACIÓN

Para obtener el archivo .apk de las aplicaciones bancarias se han considerado las herramientas utilizadas en documentos como: *Android Repacking Attack Lab* [50], *Security Analysis of Mobile Money Applications on Android* [51], o páginas como *Android File Managers* [52] y *Building and Running from the Command Line* [53], ya que han demostrado ser herramientas efectivas al momento de obtener el archivo .apk de las aplicaciones analizadas. Tomando en cuenta estos documentos, las herramientas consideradas son:

- **APKPure** [54]**:** Es una herramienta *online* que permite la descarga con solo buscar el nombre de la aplicación tal como se encuentra en la Play Store. Sin embargo, no es de mucha utilidad ya que tiene problemas para encontrar ciertas aplicaciones. Por ejemplo, como se muestra en la Figura 2.7, no

encontró las aplicaciones bancarias a ser analizadas en este trabajo ni introduciendo su nombre ni la URL de la aplicación en la Play Store.

Figura 2.7 Herramienta APKPure

- **Apk Extractor** [55]**:** Es una aplicación gratuita disponible en la Play Store porque se encarga de respaldar el archivo .apk correspondiente a la aplicación de interés en el almacenamiento interno o externo del dispositivo móvil, como se muestra en la Figura 2.8. Además de extraer la aplicación, también permite enviarla por correo.

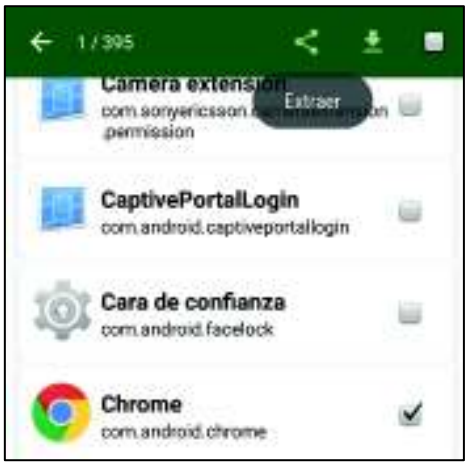


Figura 2.8 Herramienta APKExtractor

- **ES File Explorer** [56]**:** Como se muestra en la Figura 2.9, esta herramienta permite navegar por las carpetas del dispositivo, y crear un respaldo de las aplicaciones instaladas en el dispositivo Android.
- **Android Platform Tools** [57]**:** Estas herramientas se incluyen en Android Studio y permiten una conexión directa con el dispositivo, sea una *tablet* o teléfono y permite realizar la copia de seguridad de la aplicación mediante comandos. La desventaja de este método es que necesita que el dispositivo Android permita el acceso como superusuario (modo *root* en GNU/Linux).

Figura 2.9 Herramienta ES File Explorer

Para extraer el archivo .apk de un dispositivo Android se utilizan los siguientes comandos, para esto se requiere tener instalado *Android Debug Bridge* (ADB) [58]:

Conectado mediante ADB al dispositivo Android, el siguiente comando permite listar las aplicaciones instaladas:

```
>adb shell pm list packages
```

Hecho esto, se identifica el nombre de la aplicación que se desea extraer y se usa el siguiente comando para encontrar la ruta, en la cual se encuentra el archivo .apk de la aplicación deseada:

```
>adb shell pm path com.nombre.aplicacion
```

Finalmente, se copia el archivo .apk a una carpeta del PC mediante el siguiente comando, el cual incluye la ruta del archivo, obtenida del paso anterior:

```
>adb pull /data /app /com.nombre.aplicacion /base.apk
D:\APK
```

Si el comando se ejecutó exitosamente, el archivo se encontrará en la carpeta destino del paso anterior y se tendrá el resultado mostrado en la Figura 2.10

Figura 2.10 Archivo .apk obtenido mediante ADB

2.3.2 DESCOMPRIMIR EL ARCHIVO .APK

Para este paso se consideraron las herramientas 7-Zip [19] y PeaZip [59], ya que son software libre y reconocen directamente la extensión .apk como la de un archivo comprimido.

- **7-Zip:** Es una herramienta que soporta gran variedad de formatos y que brinda una tasa de compresión de entre 30 y 70%. El problema con esta herramienta es su inestabilidad al momento de descomprimir varios archivos de forma simultánea, además de que su interfaz no es muy amigable al usuario, esto se muestra en la Figura 2.11.

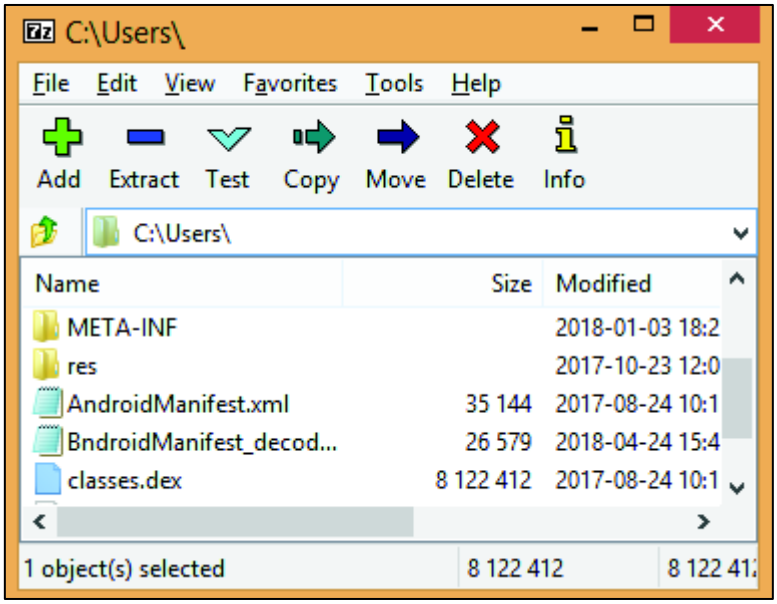


Figura 2.11 Herramienta 7 Zip

- **PeaZip:** Es una herramienta basada en 7 Zip, la cual cuenta con más de 150 formatos diferentes. Además de contar con una interfaz gráfica amigable al usuario, esta herramienta cuenta con gran estabilidad al momento de descomprimir simultáneamente varios archivos. Su interfaz se muestra en la Figura 2.12.

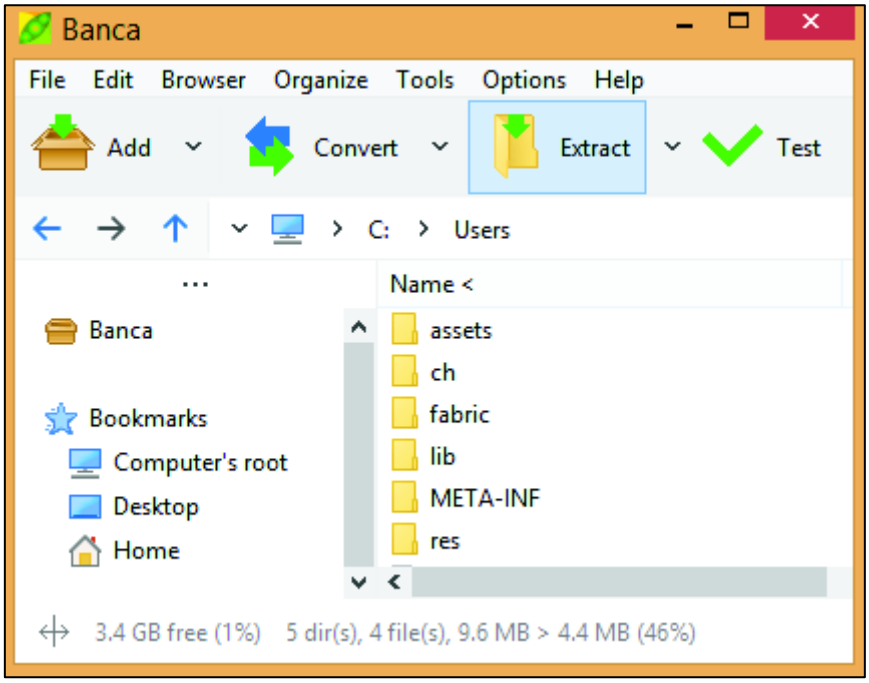


Figura 2.12 Herramienta PeaZip

2.3.3 PASAR EL ARCHIVO ANDROIDMANIFESTXML A UN FORMATO LEGIBLE

El archivo `AndroidManifest.xml`, que es el que contiene todos los permisos que solicita la aplicación para su instalación en el dispositivo Android, se encuentra al extraer el contenido del archivo .apk. Sin embargo, no es legible, por lo cual es necesario utilizar la herramienta AXMLPrinter2 [60], para hacerlo legible, de acuerdo a como se realiza en el libro: *The Mobile Application Hacker Handbook* [61], en el cual se ha conseguido obtener el archivo `AndroidManifest.xml` legible al usuario. Esta herramienta está disponible en el repositorio de Github y para su funcionamiento se requiere la instalación previa de Java [62].

2.3.4 ANÁLISIS DE CÓDIGO FUENTE

Para el análisis del código fuente se consideraron las herramientas utilizadas en trabajos como: *Android Analysis* Tools [63] y *Android Development Environment* [64]. En estos trabajos se han obtenido buenos resultados para implementar un laboratorio de pruebas para aplicaciones Android. Estas herramientas se describen a continuación.

- **APKAnalyser** [65]

Es una herramienta completa que soporta la instalación, ejecución y verificación de los resultados. Sin embargo, está más orientada a la fase de desarrollo de la aplicación y no a la aplicación que ya está publicada en Google Play. Como se muestra en la Figura 2.13, esta herramienta muestra información respecto a la estructura y vistas de la aplicación analizada.

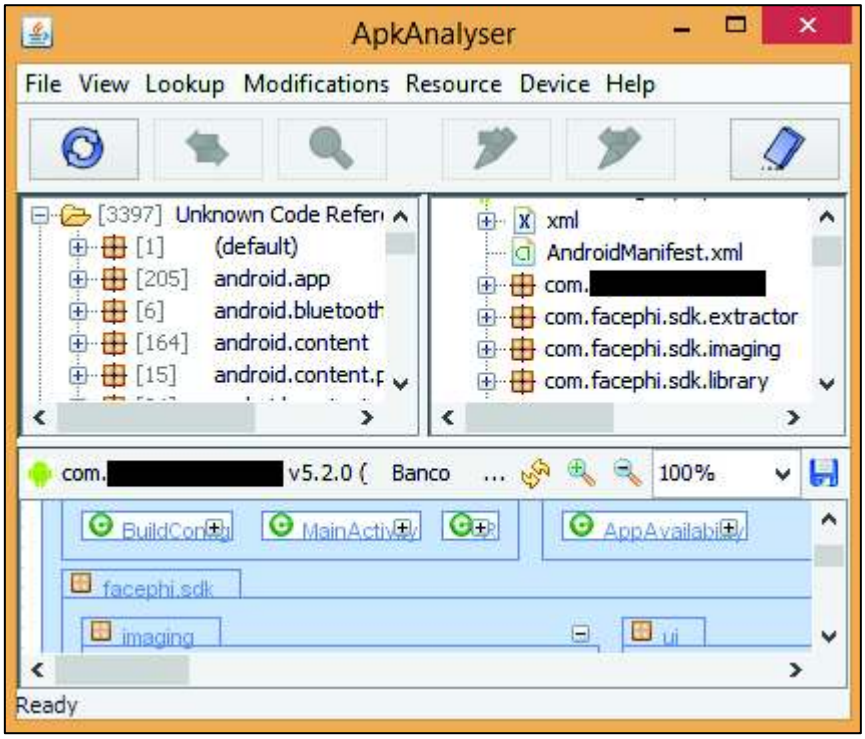


Figura 2.13 Herramienta APKAnalyser

Esta herramienta requiere la instalación de java, tanto del JDK como del JRE, la instalación de la herramienta se ha desarrollado sobre la distribución de GNU/Linux llamada Ubuntu. Para brindar control sobre el desarrollo seguro de la aplicación, esta herramienta utiliza: Logcat [66], ApkTool [67] y ADB [58].

- **Dex2jar** [68]

Esta herramienta es utilizada para obtener el código fuente de la aplicación. Sus principales componentes se detallan a continuación:

- dex-reader: está diseñado para leer el ejecutable Dalvik (.dex) contenido dentro del paquete de la aplicación (archivo .apk).

- dex-translator: se encarga del trabajo de coversión. Lee las instrucciones del archivo .dex y las convierte al formato ASM.
- dex-ir: es usado por el dex-translator, diseñado para que el archivo dex-tools pueda trabajar con archivos .class.
- d2j-smali: desensambla el archivo dex en archivos smali.
- Dex-writer: se encarga de escribir utilizando el mismo principio que dex-reader.

Como su nombre lo indica, esta herramienta permite transformar los archivos .dex en archivos .jar. De este modo se tienen los archivos listos para ser abiertos con la herramienta JD GUI.

- **JD GUI** [69]

Es una utilidad gráfica que muestra el código contenido en los archivos .class. Se puede explorar el código reconstruido mediante esta herramienta para un acceso rápido a los campos y métodos.

- **ADB**

Android Debug Bridge (ADB), es una herramienta de línea de comandos que brinda comunicación con un dispositivo Android, físico o virtual. Por medio de esta herramienta se pueden realizar múltiples tareas como instalar y depurar aplicaciones, además de proveer acceso a una terminal UNIX para ejecutar comandos en el dispositivo. Para cumplir con estas funciones esta herramienta se ayuda de los siguientes elementos:

- Un Cliente: Se encarga del envío de comandos. Este se ejecuta en la máquina del desarrollador y se invoca desde una terminal mediante los comandos ADB.
- Un Demonio (adbd): Se ejecuta en el dispositivo Android como un proceso en segundo plano.
- Un Servidor: Maneja la comunicación entre el cliente y el demonio. El servidor se ejecuta como proceso en segundo plano en la máquina del desarrollador.

Esta herramienta se incluye en el paquete de desarrollo (Android SDK) en la carpeta `Platform Tools`.

- **Logcat**

Esta herramienta permite visualizar los logs generados por el dispositivo Android, sea este físico o virtual, a medida que se ejecutan las aplicaciones a analizar.

2.3.5 ANÁLISIS ESTÁTICO AUTOMATIZADO

Para este punto se han considerado las herramientas utilizadas para realizar un análisis estático de forma automática, en documentos como: Análisis de Archivos APK con MobSF [70] y Análisis de archivos APK con AppMon [71] y Construyendo un Laboratorio de Análisis de Aplicaciones Android [72], ya que estas herramientas han demostrado ser de utilidad al momento de analizar aplicaciones Android de forma automática. A continuación se presenta una descripción de estas herramientas.

- **MobSF** [73]

Es una herramienta de análisis de aplicaciones tanto Android como iOS y Windows Mobile. Permite realizar un análisis automatizado de aplicaciones Android y generar un reporte en formato .pdf de los resultados obtenidos. Además, esta herramienta cuenta con un ambiente amigable al usuario y el análisis de aplicaciones se realiza con solo arrastrar el archivo .apk a la herramienta. Su análisis permite identificar vulnerabilidades como XML *External Entity* (XXE) [74], *Server Side Request Forgery* (SSRF) [75], etc.

- **AppMon** [76]

Esta herramienta incluye un conjunto de *scripts* que permiten al analista revisar los eventos que la aplicación testeada va generando en el sistema, y cuyos resultados se visualizan mediante una interfaz web que incluye filtros de búsqueda y ordenamiento.

Además, esta herramienta permite modificar el curso normal de la aplicación. Sin embargo, esta herramienta se centra más en modificar el comportamiento de la aplicación analizada que en analizar el código de la misma, esto se muestra en la Figura 2.14.

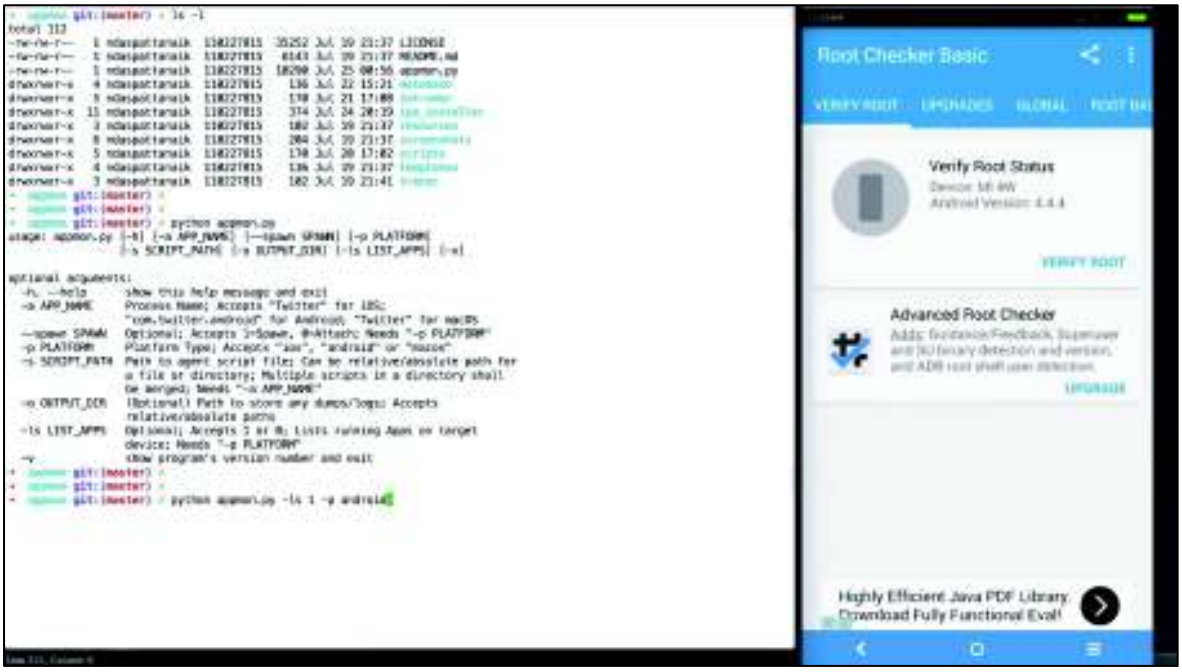


Figura 2.14 Herramienta AppMon

- **Androwarn** [77]

Es una herramienta cuyo fin principal es detectar y advertir al usuario acerca de un potencial comportamiento malicioso por parte de la aplicación analizada. La detección la realiza mediante el análisis del código que compone la aplicación.

Esta herramienta genera un reporte de manera automática, pero no funciona bien con todas las aplicaciones. Como se muestra en la Figura 2.15, al realizar el análisis de las aplicaciones bancarias no se obtienen resultados favorables.

```
jose@ubuntu: ~/androwarn
  File "androwarn.py", line 99, in main
    data = perform_analysis(APK_FILE, a, d, x, no_connection)
  File "/home/jose/androwarn/androwarn/analysis/analysis.py",
rm_analysis
    ( 'device_settings_harvesting',                    gathe
harvesting(x) ),
  File "/home/jose/androwarn/androwarn/search/malicious_behavi
gs.py", line 96, in gather_device_settings_harvesting
    result.extend( detect_get_package_info(x) )
  File "/home/jose/androwarn/androwarn/search/malicious_behavi
gs.py", line 79, in detect_get_package_info
    flags = recover_bitwise_flag_settings(flag, PackageManager
  File "/home/jose/androwarn/androwarn/util/util.py", line 257
   flag_settings
    if (int(flag) & option_value) == option_value :
ValueError: invalid literal for int() with base 10: 'Lcom/goog
mon/zze;->zzan(Landroid/content/Context;)V'
```

Figura 2.15 Herramienta Androwarn

2.3.6 RESUMEN DE CARACTERÍSTICAS DE HERRAMIENTAS PARA ANÁLISIS ESTÁTICO

Tabla 2.2 Características de herramientas para análisis estático

Tarea	Herramienta	Ventajas	Desventajas
Obtener el archivo .apk de la Aplicación Bancaria	APKPure	• Descarga directa del archivo .apk	• Las aplicaciones bancarias a analizar no se encuentran disponibles.
	Apk Extractor	• Amigable al usuario. • Permite extraer al archivo .apk y enviarlo por correo electrónico.	• Ninguna
	ES File Explorer	• Ninguna	• La funcionalidad de Navegador de archivos que esta aplicación brinda, es innecesaria para la tarea de obtener el archivo .apk.
	Android Platform Tools	• Conexión directa con un dispositivo Android físico o Virtual.	• Para respaldar una aplicación, requiere que el dispositivo tenga permisos de superusuario. • Requiere de la instalación de Android Studio.
Extraer el Contenido del Archivo .apk	7-Zip	• Reconoce directamente la extensión .apk como un archivo comprimido. • Es software libre.	• Inestabilidad al descomprimir simultáneamente varios archivos.
	PeaZip	• Reconoce directamente la extensión .apk como un archivo comprimido. • Es software libre. • Soporta un mayor número de formatos que 7-Zip. • Interfaz gráfica amigable al usuario.	• Ninguna.

Tarea	Herramienta	Ventajas	Desventajas
Pasar el Archivo AndroidManifest a un formato legible	AXMLPrinter 2	• Convierte el archivo `Android Manifest.xml` a un formato legible.	• Ninguna.
Analizar el Código Fuente de la Aplicación	Apk Analyser	• Ninguna	• Es más útil durante la fase de desarrollo de la aplicación que cuando esta ya se encuentra en producción.
	Dex2Jar	• Convierte el archivo .dex en .jar	• Ninguna
	JD Gui	• Permite visualizar el contenido de los archivos .jar. • Permite visualizar los métodos y clases que compone la aplicación a analizar de forma organizada.	• No se logra recuperar totalmente el código que compone la aplicación.
	ADB	• Permite conectarse con el dispositivo Android Físico o Virtual. • Permite instalar, depurar y eliminar aplicaciones.	• Requiere la instalación de Android Studio.
	Logcat	• Permite visualizar los logs generados por el dispositivo Android. • Ayuda a identificar las porciones de código correspondientes a paso de la aplicación (autenticación, registro, etc).	• Requiere que ADB esté instalado.
Análisis Estático Automatizado	MobSF	• Análisis del código de la aplicación de manera automática. • Generación de reportes en formato .pdf. • Interfaz amigable al usuario. • Compatible con aplicaciones Android, iOS y Windows.	• Requiere instalar Python 2.7.

Tarea	Herramienta	Ventajas	Desventajas
Análisis Estático Automatizado	AppMon	• Análisis del código de la aplicación de manera automática. • Compatible con aplicaciones Android, iOS y Windows.	• Se utiliza más para modificar el comportamiento de la aplicación que para obtener información sobre el código que la compone.
	Androwarn	• Análisis del código de la aplicación de manera automática. • Generación de reportes en formato .html.	• No funcionó con las aplicaciones bancarias que se desea analizar.

De acuerdo a la Tabla 2.2, se han seleccionado las herramientas APK Extractor por su interfaz amigable al usuario. PeaZip, ya que reconoce directamente la extensión .apk como la de un archivo comprimido y ofrece mayor estabilidad al momento de descomprimir simultáneamente varios archivos. AXMLPrinter2, ya que permite obtener el archivo `AndroidManifest.xml` en un formato legible.

Dex2jar, ya que permite obtener el archivo .jar a partir del archivo .dex. JD Gui, gracias a su interfaz gráfica que muestra la estructura de la aplicación analizada. ADB, ya que permite conectarse directamente al dispositivo Android, físico o virtual. Finalmente, también se incluye Logcat, ya que mediante ADB, permite visualizar los logs generados en el dispositivo Android y MobSF, ya que permite realizar un análisis estático de la aplicación bancaria de forma simple y automática, además de que permite generar un reporte en formato .pdf.

2.4 HERRAMIENTAS PARA ANÁLISIS DINÁMICO

En este punto se presentan las herramientas consideradas para el análisis dinámico de las aplicaciones bancarias. Estas herramientas se consideran debido a que han sido utilizadas en documentos como: Security Analysis of Mobile Money Applications on Android, The Hackers mobile Application Penetration Testing Arsenal [78] y A Study of Vulnerabilities on Android Systems [79] y se han obtenido resultados favorables para el análisis dinámico de aplicaciones Android.

Figura 2.16 Herramienta AndroL4b

A continuación, se realiza una descripción de cada herramienta y en el punto 2.4.1 se resumen sus características. Finalmente, se concluye con un párrafo que indica las herramientas seleccionadas para análisis dinámico, considerando sus ventajas y desventajas.

- **AndroL4b** [80]

Es una máquina virtual basada en Ubuntu-mate, la cual tutoriales y laboratorios de diferentes expertos en seguridad para realizar el análisis de *malware* en aplicaciones Android. Como se ve en la Figura 2.16, esta máquina viene equipada con varias herramientas para el análisis.

- **AppUse** [81]

Es una máquina virtual desarrollada por AppSec Labs. Es una sola plataforma para la prueba de seguridad en entornos Android e incluye herramientas diseñadas de acuerdo a las necesidades del cliente. La versión básica es la mostrada en la Figura 2.17.

- **CobraDroid** [82]

Es una versión modificada del sistema operativo Android orientada para los analistas en seguridad o personas interesadas en el análisis de *malware*. Esta herramienta maneja dos versiones: la primera está basada en Android 2.3.7_r1 (CobraDroidGB) y la segunda basada en Android 4.3.1_r1 (CobraDroidJB). Como

se muestra en la Figura 2.18, esta herramienta se encuentra desactualizada, desde su versión beta 1.0.

Figura 2.17 Herramienta AppUse

- **Drozer** [83]

Se creó para brindar la confianza de que las aplicaciones desarrolladas posean un adecuado nivel de seguridad. Para esto cuenta con un Agente, el cual se instala en el dispositivo Android.

- **OWASP ZAP** [84]

El Zed Attack Proxy (ZAP) de OWASP constituye una de las herramientas más populares para el análisis de seguridad. Puede ayudar a encontrar problemas de seguridad de forma automática en aplicaciones web al momento de desarrollar y probar aplicaciones. Además de esto, es de gran utilidad para realizar pruebas de

seguridad de forma manual ya que permite analizar el tráfico que generan las aplicaciones en el dispositivo Android.

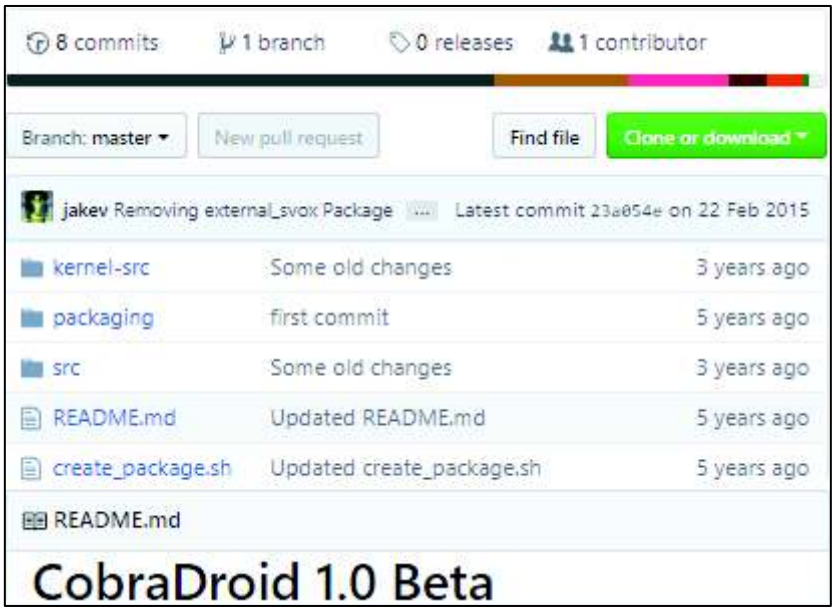


Figura 2.18 Herramienta CobraDroid

2.4.1 RESUMEN DE CARACTERÍSTICAS DE HERRAMIENTAS PARA ANÁLISIS DINÁMICO

Tabla 2.3 Características de herramientas para análisis dinámico

Herramienta	Ventajas	Desventajas
AndroL4b	• Contiene información como tutoriales y laboratorios.	• Contiene herramientas innecesariamente instaladas. • Consume más recursos físicos del PC (procesador, memoria, etc.), en relación a si se instalaran sólo las herramientas necesarias. • Es más útil para fines didácticos.
AppUse	• Permite análisis de aplicaciones Android e iOS.	• Es una herramienta de pago. • Contiene herramientas innecesariamente instaladas.
CobraDroid	• Es una versión de Android orientada al análisis de seguridad.	• Se encuentra desactualizada.
Drozer	• Permite el análisis de Proveedores de contenido para la aplicación. • Permite realizar pruebas de inyección SQL.	• Su instalación debe realizarse en una máquina virtual aislada, ya que su actividad es bloqueada por Windows.

Herramienta	Ventajas	Desventajas
OWASP ZAP	• Permite la captura de tráfico que genera la aplicación analizada en el dispositivo Android. • Muestra la información encontrada de manera organizada, permitiendo ver: URL, contenido y protocolo.	• Funciona bien sólo con un dispositivo Android físico.

Con base en el resumen presentado en la Tabla 2.3, se seleccionan las herramientas: Drozer, por su capacidad para realizar pruebas de Proveedor de Contenido y OWASP ZAP, por su capacidad para el análisis de tráfico generado por las aplicaciones a analizar.

2.5 PROTOCOLO PARA RECOLECCIÓN DE INFORMACIÓN

A continuación, se describe el protocolo para recolección de información, el cual se utiliza posteriormente para el análisis de las aplicaciones bancarias A, B y C. Para esto, las aplicaciones deben ser obtenidas desde la tienda oficial, esto con el objetivo de evitar el uso de aplicaciones maliciosas o modificadas.

Figura 2.19 Pantalla de ingreso a la aplicación

2.5.1 FUNCIONALIDAD Y FLUJO DE TRABAJO DE LA APLICACIÓN

Como se muestra en las Figuras 2.19 y 2.20, una vez instalada la aplicación, ya sea en un dispositivo Android físico o virtual, se debe navegar por la aplicación tomando en cuenta los siguientes puntos:

- Métodos de autenticación que ofrece.

- Registro de usuario y *password*.
- Qué sucede en caso de ingresar mal las credenciales.
- ¿La autenticación se realiza una única vez?
- Longitud de la contraseña.
- ¿Se notifica al usuario por correo o SMS del ingreso a la cuenta?
- ¿Se permiten transacciones comerciales? ¿Cuáles?
- ¿Qué se necesita para realizar una transacción?
- ¿Qué información sobre el cliente muestra la aplicación?

Todo esto se debe realizar de forma rápida ya que posteriormente se pondrá atención a cada punto en particular.

Figura 2.20 Transacciones que permite la aplicación

2.5.2 INTERFACES DE RED Y COMPONENTES DE HARDWARE

Cuando la aplicación se instala desde la Play Store solicita un conjunto de permisos, es necesario en este punto realizar una captura de pantalla, que permita visualizar estos permisos como se muestra en la Figura 2.21. Posteriormente, se deben listar el conjunto de permisos, separando los permisos que brindan acceso a las interfaces de red con las que interactúa con la aplicación bancaria.

Figura 2.21 Permisos solicitados por la aplicación

Posteriormente, se deben listar el conjunto de permisos, separando los permisos que brindan acceso a las interfaces de red con las que interactúa con la aplicación bancaria.

2.5.3 TRANSACCIONES COMERCIALES E INTERACCIÓN CON OTRAS APLICACIONES

Utilizando la exploración realizada en el paso 2.5.1, se debe tomar en cuenta la información sobre el cliente que exhiba la aplicación, y listar las transacciones comerciales que permite realizar. Además se debe verificar qué información es necesaria o cuál es el proceso para ejecutar estas transacciones. Por ejemplo, si las transacciones pueden ser ejecutadas sólo con acceder a la aplicación o si se requiere confirmar estas con algún código adicional. Como se muestra en la Figura 2.22, las aplicaciones bancarias pueden interactuar con otras aplicaciones como redes sociales o envío de mensajes, etc. Por esto, es necesario también listar estas aplicaciones con las que interactúa.

Figura 2.22 Aplicaciones con las que Interactúa la aplicación bancaria

2.5.4 FIRMADO DE LA APLICACIÓN

Cada aplicación viene firmada con un certificado obtenido por el desarrollador de la misma, para comprobar el origen de la aplicación es necesario verificar los datos que contiene este certificado. Para obtener esta información se utilizan dos herramientas: Jarsigner y OpenSSL.

- Jarsigner

Esta herramienta utiliza el archivo .apk directamente y muestra la información por consola. Para esto utiliza el siguiente comando:

```
>jarsigner -verify -certs -verbose AppX.apk
```

Con este comando se obtiene la información como se muestra en la Figura 2.23. Entre la información puede encontrarse la organización que desarrolló la aplicación, el país, etc.

Figura 2.23 Resultado de la herramienta Jarsigner

- OpenSSL

Esta herramienta permite obtener el archivo .cer, este archivo contiene información más detallada sobre el certificado utilizado para firmar la aplicación. Para esto se ejecuta el comando:

```
>openssl pkcs7 -in CERT.RSA -print_certs -inform DER -out
out.cer
```

El archivo .cer obtenido se muestra en la Figura 2.24. Este archivo contiene información más detallada como versión, algoritmos usados, emisor, etc.

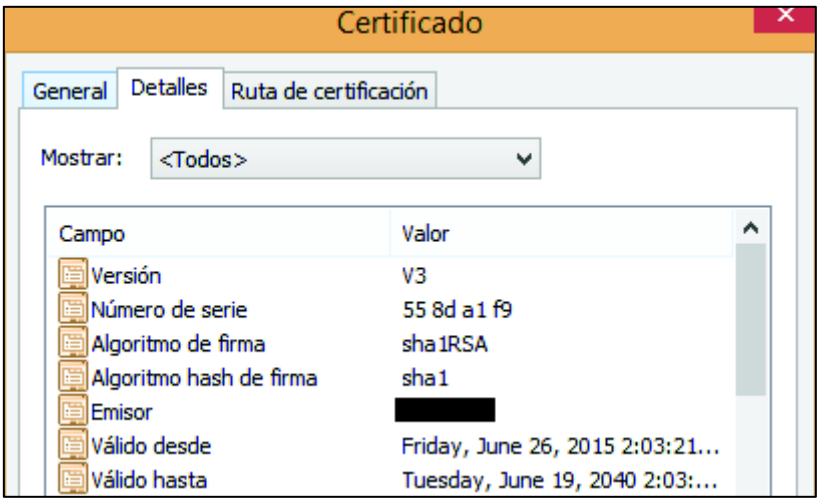


Figura 2.24 Resultado de la herramienta OpenSSL

2.6 PROTOCOLO PARA ANÁLISIS ESTÁTICO

Para esta fase las aplicaciones bancarias son instaladas en un dispositivo virtual Android, y se utilizan las herramientas seleccionadas en el punto 2.3.6. En cada tarea que compone el análisis estático, se describe la utilización de cada una de las herramientas necesarias para dicho análisis. Además, se incluyen recomendaciones que ayudarán a realizar cada tarea.

2.6.1 OBTENCIÓN DEL CÓDIGO FUENTE

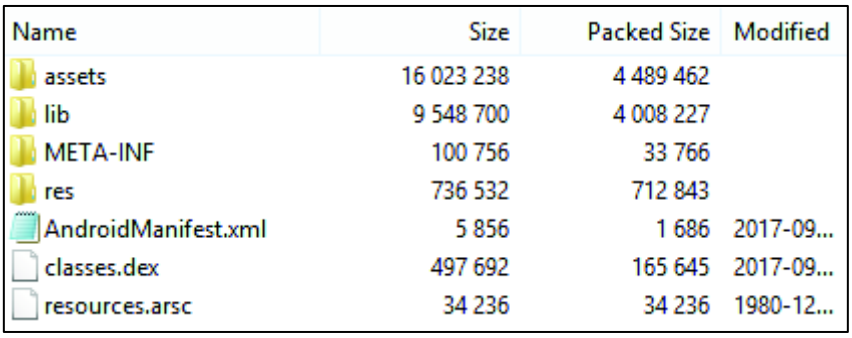
Una vez instalada la aplicación bancaria en el dispositivo virtual Android, se procede con la instalación de la aplicación Apk Extractor en el mismo.

Apk Extractor lista las aplicaciones instaladas como se muestra en la Figura 2.25. Esta herramienta extrae la aplicación bancaria que se desea analizar y obtiene su archivo .apk. Al pulsar sobre la aplicación deseada, se extrae el archivo .apk y se muestra la ruta en la cual este se ha guardado.

Figura 2.25 Aplicaciones listadas por Apk Extractor

Ahora que ya se tiene el archivo .apk se utiliza la herramienta PeaZip para visualizar los elementos que contiene este archivo. Como se muestra en la Figura 2.26, el archivo .apk es un comprimido de todo lo necesario para que funcione la aplicación.

A continuación, se extraen estos archivos en una carpeta para tenerlos disponibles para su análisis.



Name	Size	Packed Size	Modified
assets	16 023 238	4 489 462	
lib	9 548 700	4 008 227	
META-INF	100 756	33 766	
res	736 532	712 843	
AndroidManifest.xml	5 856	1 686	2017-09...
classes.dex	497 692	165 645	2017-09...
resources.arsc	34 236	34 236	1980-12...

Figura 2.26 Visualización de archivos .apk usando PeaZip

Posteriormente, se obtiene el código que compone la aplicación utilizando la herramienta Dex2Jar. Para esto, se ejecuta el comando `d2j-dex2jar.bat` desde el directorio de la carpeta `dex2jar-2.0`, ejecutando la siguiente instrucción, donde la segunda parte del comando es la ruta del archivo `classes.dex` que se obtuvo en el paso anterior.

```
>d2j-dex2jar.bat C:\Users\josep\Desktop\classes.dex
```

EL resultado se muestra en la Figura 2.27 y el archivo resultante se encuentra en la carpeta de la herramienta, este archivo será utilizado posteriormente, por lo que se recomienda ponerlo en la misma carpeta en la cual se extrajo el archivo .apk

Figura 2.27 Resultado de la Herramienta Dex2jar

2.6.2 PERMISOS EN EL ARCHIVO ANDROIDMANIFEST.XML

```
19    <uses-permission
20        android:name="android.permission.CAMERA"
21        >
22    </uses-permission>
23    <uses-permission
24        android:name="android.permission.WRITE_SETTINGS"
25        >
26    </uses-permission>
```

Figura 2.28 Extracto del archivo AndroidManifest.xml

Para realizar este punto se trabaja con el archivo `AndroidManifest.xml` que se obtuvo en el paso 2.6.1. Sin embargo, este archivo necesita ser procesado por otra herramienta antes de ser legible, para esto se utiliza AXMLPrinter2. Con el siguiente

comando se obtiene el archivo legible, la segunda parte del comando indica el nombre del archivo con el que se guardará el resultado.

```
>java -jar AXMLPrinter2.jar AndroidManifest.xml > AndroidManifest_decoded_.xml
```

Con el archivo `AndroidManifest.xml` ya en un formato legible, se utiliza cualquier editor de texto para poder leer su contenido. Se recomienda utilizar la herramienta Sublime Text [85] , ya que permite una lectura más cómoda del archivo. El resultado se muestra en la Figura 2.28.

Este archivo contiene los permisos que la aplicación solicita al momento de su instalación, estos se encuentran con la etiqueta *<uses-permission>*. Estos permisos se deben listar para comprobar que sean los mismos que la aplicación solicitó al instalarla. En la referencia [86] se muestra un detalle de los permisos más relevantes.

2.6.3 FRAMEWORK

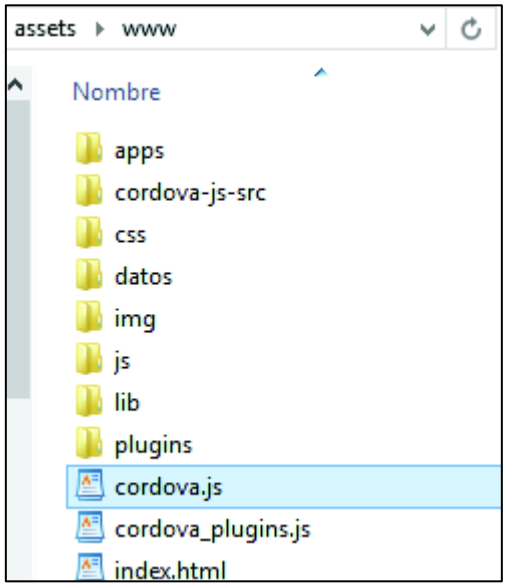


Figura 2.29 *Framework* identificado en la carpeta www

Este punto tiene por objetivo determinar el *framework* en el cual fue desarrollada la aplicación. La determinación de esta información se realiza de acuerdo a las recomendaciones de los documentos: [87] o [88], donde se especifica navegar por el contenido del archivo .apk descomprimido, obteniendo información sobre el *framework* utilizado para el desarrollo de la aplicación. En estos documentos se

recomienda buscar dentro de la carpeta `assets`, si existe una carpeta `www` que contenga el nombre de paquetes como los sugeridos en [30], como por ejemplo Apache Cordova [89], Ionic [90], etc. Esta información también se puede buscar en la carpeta `src`. La Figura 2.29 muestra un ejemplo de aplicación que utiliza el *framework* Apache Cordova, lo cual fue detectado por el contenido de la carpeta `assets/www`.

No todas las aplicaciones incluyen este tipo de información sobre el *framework* en el cual fueron desarrolladas. Por ejemplo, en la Figura 2.30 se muestra el caso de una aplicación en la cual no se ha obtenido la información del *framework,* dado que en la carpeta `assets` no se encuentra información acerca de algún tipo de paquete como los sugeridos en [30]. Esta es una de las limitaciones propias de realizar un análisis a partir de la aplicación ya compilada [91].

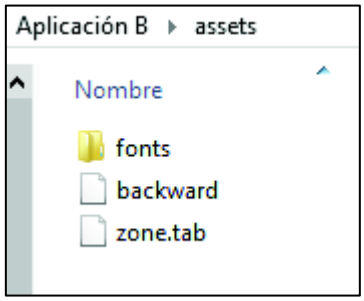


Figura 2.30 Contenido de la carpeta www sin Información sobre el *Framework*

2.6.4 LIBRERÍAS

Cada aplicación utiliza librerías que le brindan ciertas funcionalidades, estas librerías pueden introducir vulnerabilidades en la aplicación, por lo que es necesario identificarlas. Para esto se realiza el procedimiento, el cual se detalla a continuación:

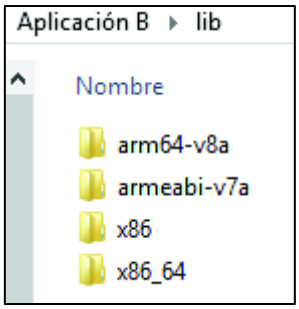


Figura 2.31 Contenido de la carpeta lib

Al descomprimir el archivo .apk se encuentra la carpeta `lib`, esta carpeta contiene las librerías que son parte de la aplicación, por ejemplo, librerías que no están incluidas en el Android SDK [30]. Para esto, existe una carpeta para cada arquitectura de procesador, como se muestra en la Figura 2.31.

Como se ve en la Figura 2.32, dentro de la carpeta *arm64-v8a*, se encuentra la librería correspondiente para esta arquitectura.

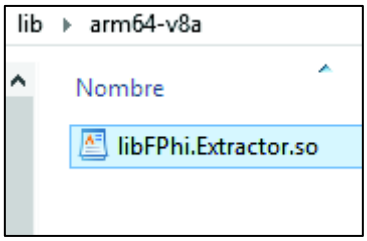


Figura 2.32 Librería encontrada

Una vez que se han identificado las librerías que son parte de la aplicación, se pueden usar referencias como: [92], [93], [94] o [95]. Estas referencias contienen una base de datos de vulnerabilidades encontradas. En la Figura 2.33 se muestra como ejemplo una librería de Android: *libxml*, la cual contiene una vulnerabilidad de ejecución de código remota.

Figura 2.33 Librería vulnerable encontrada en *National Vulnerability Database* [92]

2.6.5 VISTAS

A partir de este punto es necesario mencionar que el código obtenido mediante todo el procedimiento anterior no es exactamente igual al original que compone la aplicación, ya que una parte del código se pierde durante el proceso para crear el archivo .apk [96].

Ahora que ya se cuenta con el código de la aplicación y con los datos obtenidos en la fase de recopilación de información, se puede definir si la aplicación realiza el procesamiento de los datos de usuario, como es el caso de aplicaciones nativas e híbridas; o si solamente cuenta con las vistas necesarias para conectarse con el servidor de la entidad bancaria, y que este realice todo el proceso, como en el caso de aplicaciones web [30].

De acuerdo a lo mencionado en el punto 1.7, para el caso de una aplicación nativa o híbrida, al extraer el archivo .apk se encuentra la carpeta `res`, esta carpeta contiene las vistas de la aplicación. Un ejemplo de esto se muestra en la Figura 2.34, donde se visualiza el contenido de la carpeta res, mostrando los archivos necesarios para generar las vistas de la aplicación, en el caso de una aplicación nativa o híbrida.

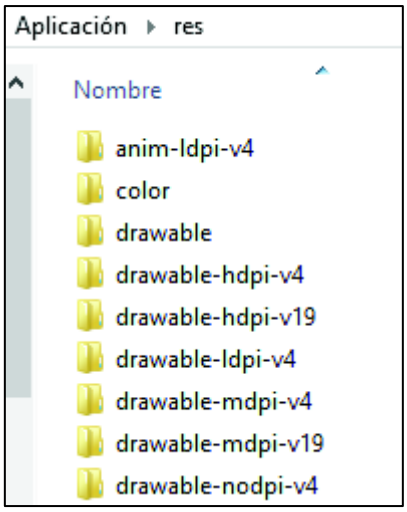


Figura 2.34 Contenido de la carpeta res para una aplicación nativa o híbrida

Para el caso de una aplicación móvil web no se guarda un archivo .apk en el dispositivo Android, como se mencionó en el punto 1.7. Por lo tanto, una aplicación de este tipo se ve como la de la Figura 2.35, en la que la URL de la página web contiene la letra `m` luego de la cadena https://. Esto indica que es un sitio web optimizado para dispositivos móviles [97]. En este caso, el análisis de seguridad se vería limitado al código fuente de la página web a la que se conecta el dispositivo móvil, en cuyo caso las técnicas a utilizar se basarían sólo en el análisis de tráfico y estudio de URLs.

Figura 2.35 Vista de una aplicación móvil web

El objetivo de este punto es verificar si dentro de la carpeta se encuentran los archivos que generan las vistas de la aplicación.

2.6.6 INGRESO DE DATOS

Para la realización de los siguientes pasos se utilizarán las herramientas JD-Gui y Logcat. JD-Gui se usa para el análisis del código, mientras que para el análisis de los logs generados por el dispositivo Android al momento de utilizar la aplicación, se utiliza Logcat. Esto facilita el análisis del código ya que se pueden visualizar los logs a medida que la aplicación se va ejecutando, así se pueden identificar las porciones de código de cada etapa.

Para el ingreso de datos, se ejecuta la aplicación y de forma paralela Logcat. Se recomienda que cada vez que se realice un paso, como el ingreso de usuario o contraseña, se registre el número de log para poder visualizarlo posteriormente, de este modo se puede conocer qué logs pertenecen a cada etapa.

Por ejemplo, para este punto se considera el proceso de una transferencia electrónica y se observan los logs generados. Como se observa en la Figura 2.36, en los logs generados aparece la Actividad `Direct Transfer Confirmation Activity`, y también la palabra `requiereValidacion` la cual se explora posteriormente con la herramienta JD-Gui. Para poder desarrollar con éxito este punto, es necesario recoger los logs generados sólo por este proceso en particular,

(para este ejemplo, una transferencia electrónica), y con esa información buscar el código relacionado mediante la herramienta JD-Gui.

```
1  04-26 21:15:29.925 I/ActivityManager( 1838): START u0
   {cmp=com.application.application/.DirectTransferConfirmationActivity (has extras)}
   from uid 10063 on display 0
2  04-26 21:15:29.947 D/Volley  ( 3406): [111] BasicNetwork.logSlowRequests: HTTP
   response for request=<[ ] https://sbapp07.application.com/
   application-mobile-services-pfm/xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
   0xe7aa217e NORMAL 10> [lifetime=8084], [size=946], [rc=200], [retryCount=0]
3  04-26 21:15:29.947 D/Response( 3406):
   {"resultadoValidaDestino":"requiereValidacion":true,"mensaje":
```

Figura 2.36 Clase identificada en los logs generados durante una transferencia electrónica

Ahora que se ha obtenido la información de una de las Actividades se buscan las palabras obtenidas en el punto anterior en el archivo .jar de la aplicación, abierto mediante la herramienta JD-Gui. Como se observa en el Código 2.1, se ha encontrado la frase `requiereValidacion`, obtenida a partir de los logs generados durante la transferencia electrónica. Gracias a esto se encuentra que la aplicación realiza validación de los datos del beneficiario de la transferencia, como por ejemplo, su nombre.

```
1  public ValidateDirectTransferParser(JSONObject paramJSONObject)
2    {
3      super(paramJSONObject);
4      try
5      {
6        Object localObject = paramJSONObject.getJSONObject("resultadoValidaDestino");
7        this.a = new DirectTransferSummary(parseProduct(((JSONObject)
         localObject).getJSONObject("producto")));
8        JSONObject localJSONObject =
         paramJSONObject.getJSONObject("tresultadoDatosBeneficiario");
9        paramJSONObject = localJSONObject.getString("nombreBenefeciario");
10       boolean bool = ((JSONObject)localObject).optBoolean("requiereValidacion");
11       this.a.setRequireValidation(bool);
12       if (bool)
```

Código 2.1 Clase ValidateDirectTransferParser y la palabra encontrada en los logs para una transferencia electrónica

2.6.7 CÓDIGO UTILIZADO PARA AUTENTICACIÓN

```
1  04-26 21:40:37.226 I/ActivityManager( 1838): Displayed
   com.aplicacion.app/.LoginOptions: +306ms
```

Figura 2.37 Log generado durante el proceso de autenticación

En esta sección se utiliza la información generada por la aplicación y recogida por Logcat para poder analizar el código correspondiente. La fase de autenticación es

la más importante ya que la aplicación debe manejar de forma adecuada la información del cliente de modo que no se exponga información sensible.

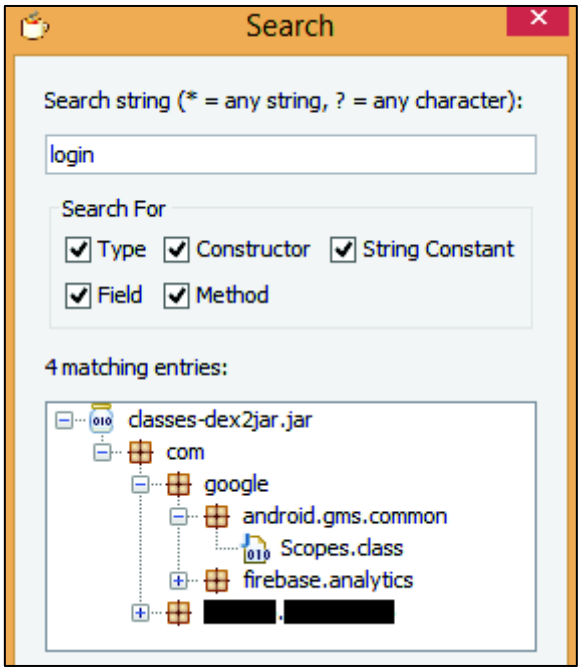


Figura 2.38 Búsqueda del código utilizado para autenticación mediante JD-GUI

En este punto se debe buscar el código correspondiente a los logs generados en el archivo .jar. Los logs se visualizan mediante Logcat mientras la aplicación se ejecuta en el dispositivo virtual Android, como se muestra en la Figura 2.37. Con esto se pueden identificar los métodos que se ejecutan y también las palabras clave que posteriormente serán buscadas mediante JD-Gui.

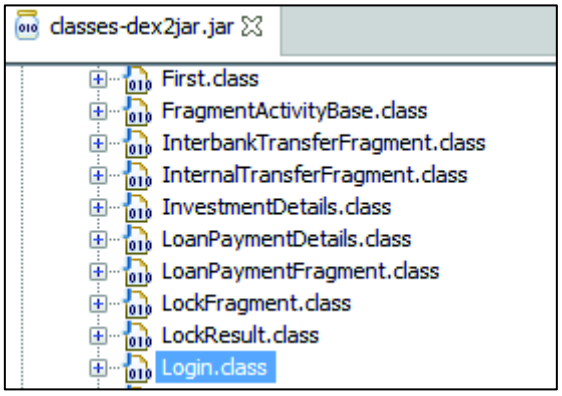


Figura 2.39 Estructura de la aplicación bancaria en JD-GUI

En la herramienta JD-Gui se buscan estas palabras, conviene incluir también palabras como: login y authentication en la búsqueda, como se muestra en la Figura

2.38. De este modo se identifica la porción de código correspondiente a la autenticación.

También conviene navegar por la estructura de la aplicación, mostrada por JD-Gui a fin de encontrar clases asociadas al proceso de autenticación, como se muestra en la Figura 2.39.

2.6.8 AUTENTICACIÓN ONLINE/OFFLINE

Para analizar este punto se hace uso de los logs generados por la aplicación bancaria, mientras esta se ejecuta en el dispositivo virtual. Mediante esto, se busca conocer si la aplicación se conecta con algún servidor externo para realizar la autenticación (*online*), o si este proceso se realiza de manera local (*offline*).

Para el caso de una aplicación que realiza el proceso de autenticación *online*, en los logs generados por la aplicación aparecen URLS a las cuales esta se conecta. Como se observa en la Figura 2.40, esta aplicación hace uso del protocolo HTTPS para conectarse con un servidor externo durante su ejecución.

```
1  04-23 20:36:43.412 W/json    ( 2452): https://www.aplicacion.com.ec/detectrd/
   image.htm?id=XXXXXXX
2  04-23 20:33:56.964 W/json    ( 2452): http://www.aplicacion.com/Files/MovilApp/
   BancoApp.jpg
```

Figura 2.40 Extracto de los logs generados por una aplicación que realiza autenticación *online*

Si la aplicación se autentica de manera *offline*, la aplicación no se conecta a ninguna URL para brindar acceso. En la Figura 2.41 se presenta un extracto de log logs generados por una aplicación que se autentica de manera *offline*, en donde se evidencia que no existe una conexión con un servidor externo.

```
1  04-26 22:09:55.926 I/ActivityManager( 1838): START u0 {act=android.intent.action.MAIN
   cat=[android.intent.category.LAUNCHER] flg=0x10200000
   cmp=com.android.music/.MusicBrowserActivity (has extras)} from uid 10007 on display 0
2  04-26 22:09:55.927 W/AudioTrack( 1838): AUDIO_OUTPUT_FLAG_FAST denied by client
3  04-26 22:09:55.972 E/libprocessgroup( 3956): failed to make and chown /acct/uid_10033:
   Read-only file system
4  04-26 22:09:55.978 I/ActivityManager( 1838): Start proc 3956:com.android.music/u0a33
   for activity com.android.music/.MusicBrowserActivity
5  04-26 22:09:56.008 I/ActivityManager( 1838): START u0 {act=android.intent.action.PICK
   dat= typ=vnd.android.cursor.dir/artistalbum flg=0x4000000
   cmp=com.android.music/.ArtistAlbumBrowserActivity (has extras)} from uid 10033 on
   display 0
```

Figura 2.41 Extracto de los logs generados por una aplicación que realiza autenticación *offline*

2.6.9 INFORMACIÓN ADICIONAL AL NOMBRE DE USUARIO/PASSWORD

```
04-12 20:48:50.962 I/***POST ( 2082): {Content-Type=application/json; charset=utf-8,
id-transaction=XXXXXXXXXXXXXXXXXX, device-id=XXXXXXXXXXXXXXX,
unique-id=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX}
```

Figura 2.42 Identificador único utilizado durante el proceso de autenticación

Dentro del proceso de autenticación las aplicaciones utilizan un nombre de usuario y una contraseña, en este punto se busca encontrar si además de esto, la aplicación utiliza información adicional. Por ejemplo, hay aplicaciones que utilizan un identificador de dispositivo, coordenadas, etc. En la Figura 2.42, se encuentra un Identificador único, generado por la aplicación, también otro identificador llamado `Device Secret`.

```
protected Map<String, String> getHeaders(Map<String, String> paramMap)
  {
    HashMap localHashMap = new HashMap(paramMap.size() + 1);
    localHashMap.putAll(paramMap);
    localHashMap.put("Content-Type", "application/json; charset=utf-8");
    localHashMap.put("id-transaction", UUIDGenerator.getCurrentUUID());
    Log.i("*** UniqueId1: ", this.b.getUniqueId1());
    Log.i("*** DeviceSecret: ", this.b.getDeviceSecret());
    paramMap = localHashMap;
    if (!Utils.isNullOrEmptyOrBlank(this.b.getUniqueId1()))
    {
      paramMap = localHashMap;
      if (!Utils.isNullOrEmptyOrBlank(this.b.getDeviceSecret())) {
        paramMap = a(localHashMap, this.b.getUniqueId1(), this.b.getDeviceSecret());
      }
    }
    return paramMap;
  }
```

Código 2.2 Información adicional al PIN durante la autenticación

Utilizando los log generados por la aplicación se busca en el código de la aplicación estos identificadores. Como se ve en el Código 2.2, durante el proceso de autenticación se usan los identificadores: `UniqueId` y `DeviceSecret`. Además, se pueden ver las funciones que generan estos elementos adicionales.

2.6.10 MÉTODOS DE AUTENTICACIÓN

Actualmente, las aplicaciones brindan más de una opción para acceder. El objetivo de este punto es determinar qué métodos utiliza la aplicación para permitir el acceso. Generalmente, las aplicaciones bancarias utilizan dos métodos de autenticación, para encontrar el código correspondiente se exploran los resultados del punto 2.6.7 en la herramienta JD-Gui. En el ejemplo del Código 2.3, la aplicación cuenta con dos métodos de autenticación: `onFPhiClick`, para reconocimiento

facial y `onPinClick`, para el ingreso del PIN. En este punto se debe tomar en cuenta que la búsqueda de JD-GUI mostrará resultados tal como está escrito en el cuadro de búsqueda, diferenciando mayúsculas y minúsculas. Para este ejemplo, se ha identificado que el fragmento `FPhi` corresponde a una librería para reconocimiento facial, al buscar este fragmento en Internet.

```
1  public void inject(ButterKnife.Finder paramFinder, final T paramT, Object paramObject)
2    {
3      super.inject(paramFinder, paramT, paramObject);
4      ((View)paramFinder.findRequiredView(paramObject, 2131624057, "method
       'onFPhiClick'")).setOnClickListener(new DebouncingOnClickListener()
5      {
6        public void doClick(View paramAnonymousView)
7        {
8          paramT.a();
9        }
10     });
11     ((View)paramFinder.findRequiredView(paramObject, 2131624058, "method
       'onPinClick'")).setOnClickListener(new DebouncingOnClickListener()
12     {
13       public void doClick(View paramAnonymousView)
14       {
15         paramT.b();
16       }
17     })
```

Código 2.3 Métodos de autenticación

2.6.11 BLOQUEO

En caso de comportamiento anómalo, por ejemplo, si un usuario ingresa varias veces credenciales inválidas en la aplicación, la aplicación debe bloquearse. En este punto se busca encontrar el código que realice esta acción. Para encontrarlo se ingresa credenciales inválidas al momento de autenticarse en la aplicación, y se observan los logs generados.

```
1  04-26 22:22:01.198 I/ErrorResponse( 3406):
   {"clientErrorCode":40003,"clientErrorMessage":"TOT-PIN INVALIDO - TIENE 2
   INTENTOS","errorCode":"TOT-12","externalErrorCode":"TOT-12",
```

Figura 2.43 Extracto de los logs generados por la aplicación al ingresar mal la contraseña

Por ejemplo, la Figura 2.43 muestra un extracto de los logs generados al momento de ingresar incorrectamente la clase de acceso a la aplicación. En estos logs se observa la frase `TOT-12`, la cual es buscada posteriormente mediante la herramienta JD-Gui. En el Código 2.4 se muestra el código encontrado al buscar la frase `TOT-12`, en este se encuentra que hay dos métodos: `decreasePinTries` y `hasLeftTries` para decrementar el número de intentos de autenticación restantes

y para verificar el número restante de intentos. Dentro de la búsqueda, resulta útil utilizar palabras como PIN, *password*, etc.

```
if (WebServiceHelper.getErrorCode(paramVolleyError).equals("TOT-12"))
    {
      SharedPreferencesManager.getInstance().decreasePinTries();
      if (SharedPreferencesManager.getInstance().hasLeftTries())
      {
        a(getString(2131165375, new Object[] { Integer.valueOf(
        SharedPreferencesManager.getInstance().getLeftPinTries()) }), true);
        d();
        return;
      }
```

Código 2.4 Código encontrado con base en los logs generados por la aplicación

2.6.12 AUTENTICACIÓN ÚNICA

En este punto se comprueba si la aplicación bancaria realiza el proceso de autenticación cada vez que se abre, o si este proceso lo realiza únicamente la primera vez que se accede a ella. Para determinar esto se utilizan los logs generados por la aplicación cuando esta se abre.

```
04-26 22:44:12.260 I/***     ( 3406): AplicacionActivity onResume
04-26 22:44:12.260 I/***     ( 3406): AplicacionActivity showRestriction
```

Figura 2.44 Extracto de los logs obtenidos al abrir la aplicación

Las aplicaciones utilizan varios métodos que controlan el comportamiento de la misma, el método que interesa para este punto es el `onResume()`, este método es el que se ejecuta cuando la aplicación se abre. Como se muestra en la Figura 2.44, esta aplicación ejecuta este método cuando la aplicación se abre. Utilizando la herramienta JD-Gui se busca la frase `onResume` para encontrar el código asociado.

```
public void onResume()
{
  RUMApplicationHook.onActivityResumed(this);
  showRestriction();
  HP_WRAP_onResume();
  ActivityHooks.onActivityResumeEndHook(this);
}

protected boolean showRestriction()
{

  showLoginScreen();

}
```

Código 2.5 Método onResume y método showRestriction

Por ejemplo, para esta aplicación se ha encontrado el método `onResume` como se muestra en el Código 2.5, este método llama al método `showRestriction`, el cual se encarga de llamar a la pantalla de Login cada vez que la aplicación se abre o regresa a primer plano. Por lo tanto, esta aplicación requiere del ingreso de credenciales cada vez que se abre.

2.6.13 AUTENTICACIÓN EN DOS PASOS

Para permitir el acceso, una aplicación bancaria debe mantener un adecuado nivel de seguridad, una de las formas de conseguirlo el mediante una autenticación en dos pasos [98].

```
04-26 22:22:04.772 I/send Email and SMS: ( 3406): success
04-26 22:22:04.774 D/Response( 3406): {"message":"La notificaciÃ³n fue enviada con Ã©xito"}
```

Figura 2.45 Logs del envío de correo y SMS durante el proceso de autenticación

Durante una autenticación en dos pasos, en la primera fase el usuario ingresa correctamente sus credenciales en la aplicación, mientras que en la segunda fase se envía un código de verificación al correo electrónico o a un número celular del cliente, este código debe ser ingresado en la aplicación móvil para acceder.

Para identificar este proceso es necesario ayudarse de la exploración inicial de la aplicación, realizada durante la fase de recolección de información, y de los logs generados por la aplicación. Como se muestra en la Figura 2.45, resulta útil buscar palabras como *code*, *activation*, *validate*, etc.

Posteriormente, las palabras identificadas en los logs se buscan en el código de la aplicación. Por ejemplo, en el Código 2.6 se muestra el caso de una aplicación que utiliza mensajes de texto para el envío del código de confirmación.

```
protected void a(Activity paramActivity, EditText paramEditText)
 {
   if (CommonUtils.checkPermission(paramActivity, "android.permission.RECEIVE_SMS"))
   {
     IntentFilter localIntentFilter = new
     IntentFilter("android.provider.Telephony.SMS_RECEIVED");
     this.f = new SmsBroadcastReceiver(paramEditText);
     paramActivity.registerReceiver(this.f, localIntentFilter);
   }
 }
```

Código 2.6 Código para recepción de SMS durante el proceso de autenticación

2.6.14 ANÁLISIS ESTÁTICO AUTOMATIZADO

Finalmente, para este punto, el archivo .apk de la aplicación que se desea analizar se arrastra hacia la interfaz web de la aplicación MobSF, como se muestra en la Figura 2.46. Una vez que el análisis ha concluido, se presentan los resultados obtenidos, los cuales se pueden guardar en formato .pdf.

Los resultados obtenidos se encuentran separados en diferentes secciones como se muestra en la Figura 2.47 y para exportarlos como archivo .pdf se utiliza la opción `Download Report` y el archivo se extrae automáticamente para poder analizarlo posteriormente.

Figura 2.46 Carga del archivo .apk para su análisis estático con la herramienta MobSF

Entre los resultados que esta herramienta ofrece están: Versión mínima de Android requerida para su funcionamiento, información del estado del certificado utilizado para firmar la aplicación, permisos solicitados por la aplicación y su estado, etc.

2.7 PROTOCOLO PARA ANÁLISIS DINÁMICO

Durante el desarrollo de este punto se describe el procedimiento para utilizar las herramientas seleccionadas en el punto 2.4.1. Para cada una de las tareas que

compone el análisis dinámico, se describe cómo utilizar las herramientas, así como las recomendaciones para obtener la información deseada.

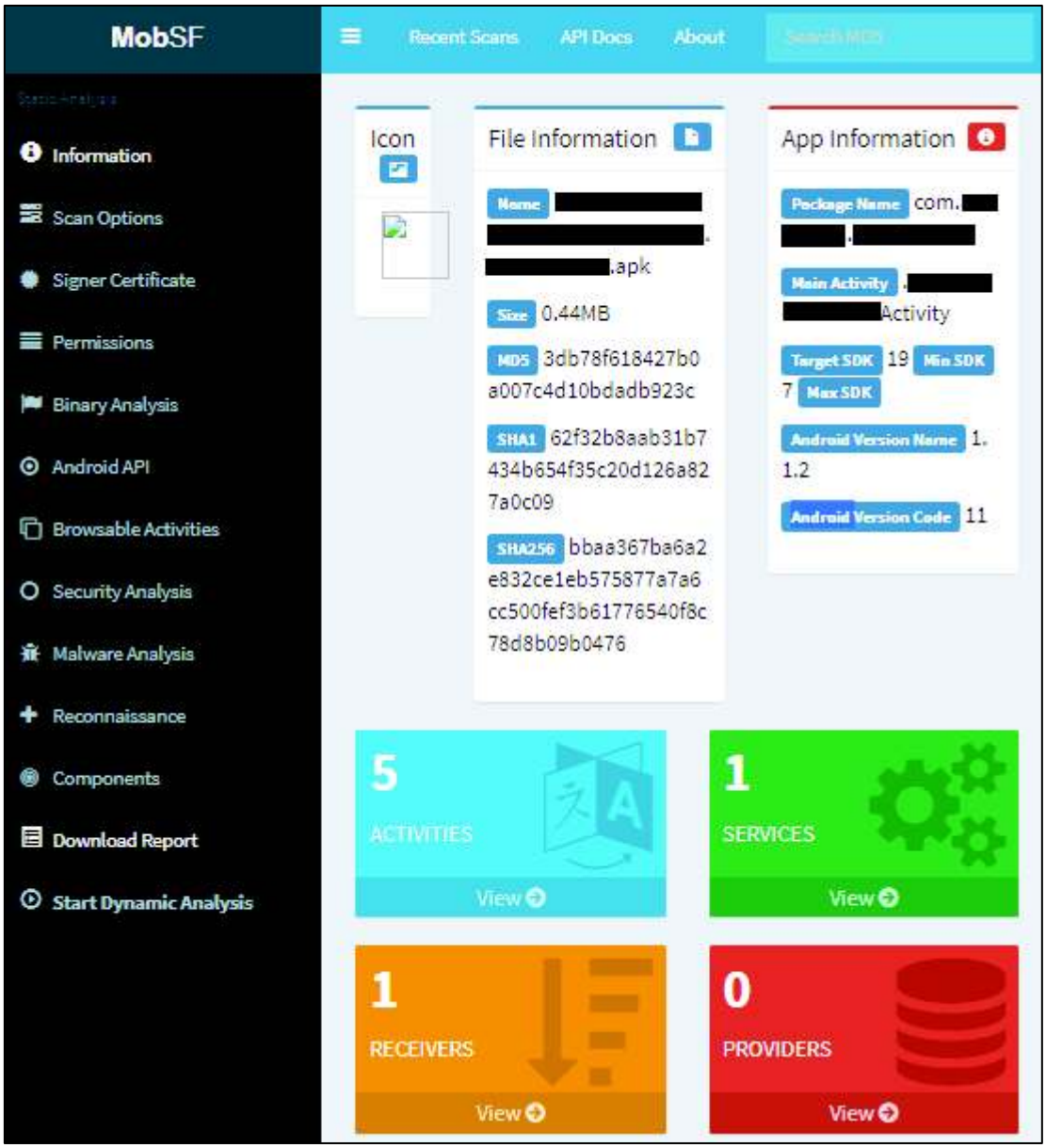


Figura 2.47 Resultados del análisis estático mediante la herramienta MobSF

2.7.1 PROTOCOLOS DE COMUNICACIÓN UTILIZADOS

Dado que las aplicaciones bancarias se comunican con un servidor web tanto para el proceso de autenticación, como para realizar transacciones, es necesario verificar que esta comunicación se realice utilizando protocolos seguros. Para este punto se utilizan dos herramientas Network Connections y OWASP ZAP.

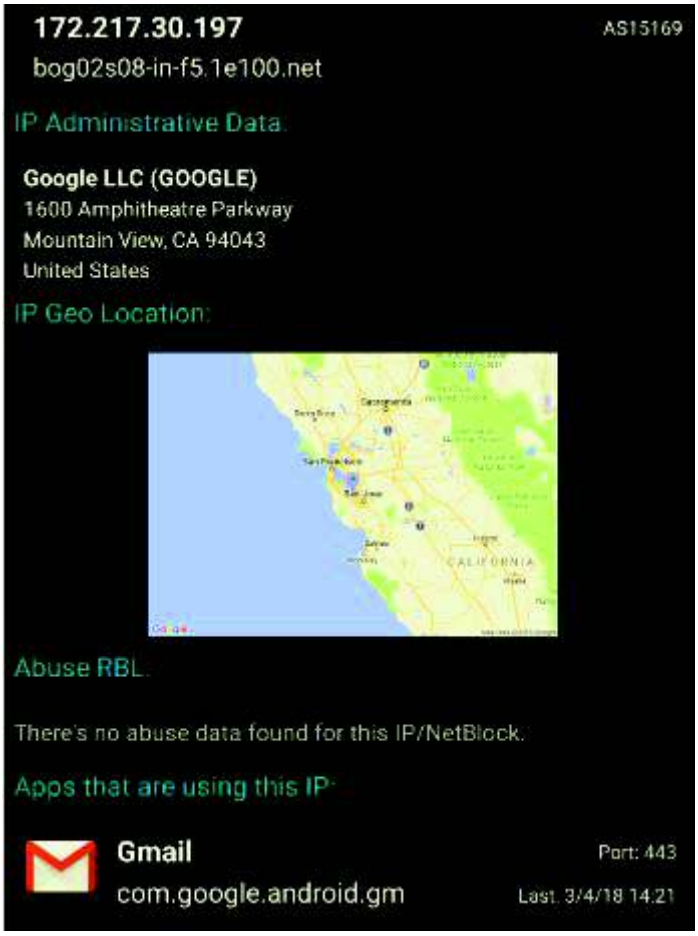


Figura 2.48 Resultado de la herramienta Network Connections

- Network Connections

Es una aplicación que, al ser instalada en el dispositivo Android, provee información respecto a las conexiones de red que mantiene el dispositivo Android.

Esta aplicación brinda información respecto a la IP del servidor con el cual se conecta la aplicación bancaria, ubicación geográfica la misma y puerto que utiliza, esto se evidencia en la Figura 2.48.

Si la aplicación bancaria no se ejecuta en segundo plano, la herramienta Network Connections ofrece la posibilidad de realizar una captura de tráfico, con esto se inicia la captura, se ejecuta la aplicación normalmente, y luego de autenticarse se detiene la captura.

De este modo ya se obtiene la información completa, en especial, la referente al número de puerto con el cual se comunica la aplicación bancaria.

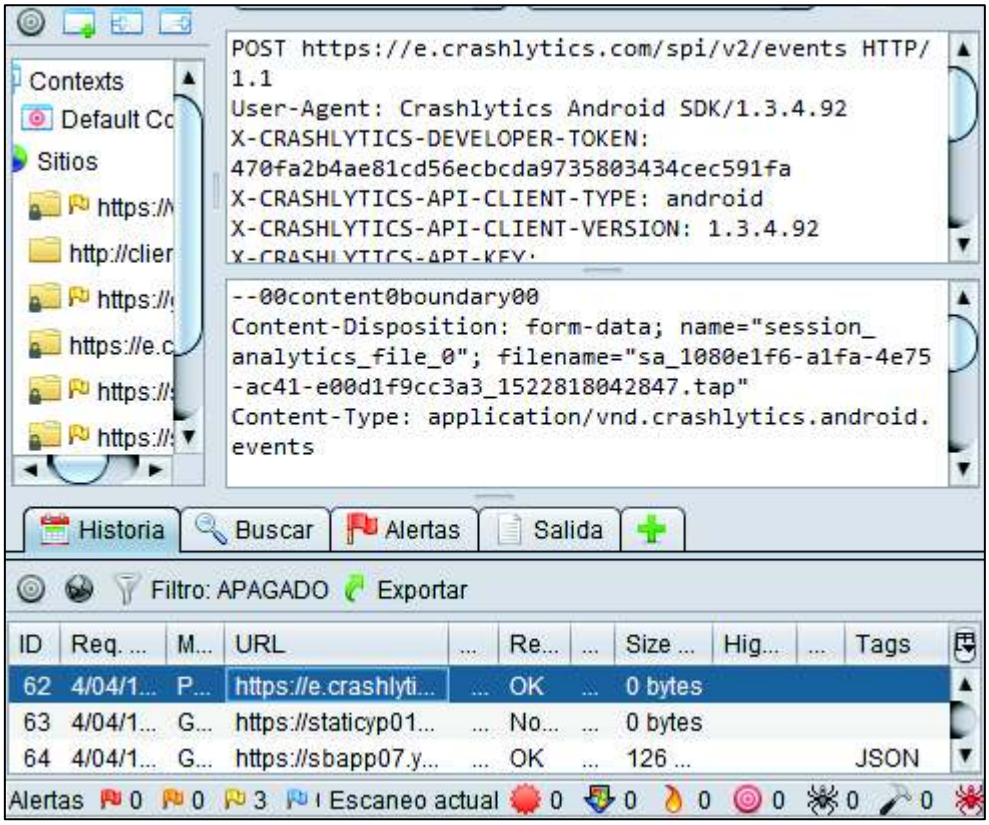


Figura 2.49 Resultado de la herramienta OWASP ZAP

- OWASP ZAP

Para realizar un análisis de tráfico más exhaustivo se utiliza la herramienta OWASP ZAP. Con la herramienta correctamente configurada, se abre la aplicación y se realiza el proceso de autenticación normalmente.

Con esto, la herramienta registra las URLs con las cuales se comunica la aplicación como lo muestra la Figura 2.49, así como el detalle del contenido de los paquetes.

Se recomienda realizar esto con un dispositivo Android físico ya que con el dispositivo virtual se presentan problemas al momento de capturar los paquetes que genera la aplicación bancaria.

2.7.2 SCRIPTS EXISTENTES PARA LA BÚSQUEDA DE VULNERABILIDADES EN APLICACIONES ANDROID

Existen *scripts* que ayudan a realizar el proceso de análisis, como Pidcat; o que brindan información acerca de vulnerabilidades que pueda contener la aplicación, como Manitree.

Para el desarrollo de este punto se han considerado los *scripts* Pidcat y Manitree de acuerdo a lo sugerido en documentos como: Cómo Evitar Errores de Desarrollo en Android [99] o *Risk Analysis of Android Based Appliance* [100], ya que estos *scripts* han demostrado ser de gran utilidad y se han obtenido buenos resultados.

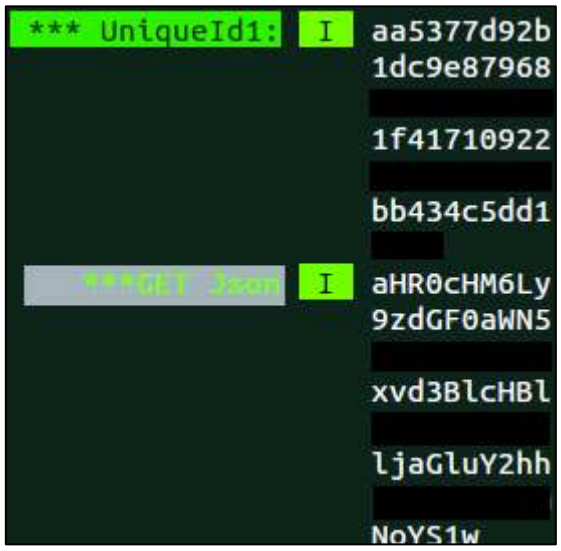


Figura 2.50 Resultado de la herramienta Pidcat

- Pidcat

Esta herramienta es una mejora de Logcat, ya que, mientras Logcat muestra todos los logs del dispositivo Android, Pidcat permite seleccionar los generados por la aplicación de interés.

Para esto se ejecuta el siguiente comando, es necesario mencionar que la información de color sólo se visualiza si se utiliza la terminal de una distribución GNU/Linux, en Windows la información de color se visualiza en la versión 10. El nombre del paquete se puede ver al ejecutar Logcat, empieza con “com.” + nombre de la aplicación. Los resultados se muestran en la Figura 2.50.

```
>C:\Python\python.exe pidcat.py com.nombre_del_paquete
```

- Manitree

Esta herramienta permite listar los permisos que la aplicación solicita directamente desde el archivo `AndroidManifest.xml`, el cual se encuentra al descomprimir el archivo .apk. Para esto se ejecuta el siguiente comando y los resultados son los mostrados en la Figura 2.51.

```
>C:\Python\python.exe manitree.py -f AndroidManifest.xml
```

Además, esta herramienta muestra los puntos vulnerables de la aplicación con su respectivo nivel de severidad.

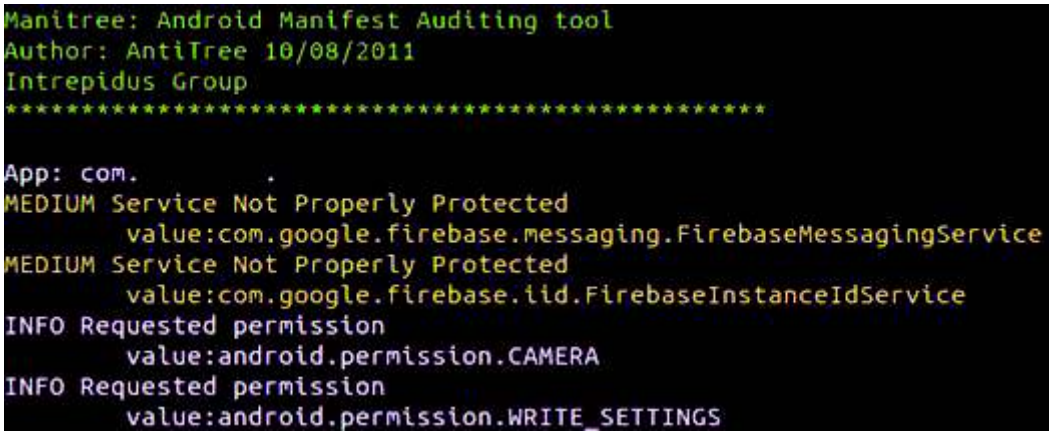


Figura 2.51 Resultado de la herramienta Manitree

2.7.3 PROVEEDOR DE CONTENIDO

Para el desarrollo de este punto se utiliza la herramienta Drozer. Un proveedor de contenido, cuando no se encuentra adecuadamente configurado, comparte la información que maneja, haciéndola accesible a otras herramientas instaladas en el mismo dispositivo Android. El procedimiento para realizar la prueba de este punto se indica a continuación.

Desde la consola Drozer, se listan las aplicaciones instaladas con el siguiente comando, en la sintaxis de este se utiliza el nombre de la aplicación, con esto se obtiene el nombre del paquete, el cual empieza con “com.”+ el nombre de la aplicación.

```
>run app.package.list -f nombre_aplicación
```

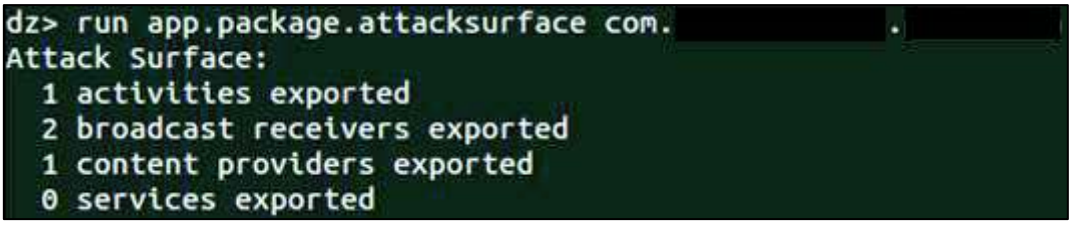


Figura 2.52 Resultado de la herramienta Drozer

Ahora que ya se conoce el nombre del paquete, se ejecuta el siguiente comando para conocer si existen proveedores de contenido exportados, además, la

aplicación también muestra si hay actividades, *broadcast receivers*, o servicios que sean exportados.

Figura 2.53 URIs de la aplicación analizada

Dentro de la sintaxis del comando, el nombre del paquete es el obtenido en el paso anterior. Los resultados obtenidos se muestran en la Figura 2.52.

```
>run app.package.attacksurface nombre_del_paquete
```

A continuación, se obtiene la información de los permisos que tiene el proveedor de contenido con el siguiente comando:

```
>run app.provider.info -a nombre_del_paquete
```

De este modo se confirma o no si la información que maneja el proveedor de contenido es accesible por otras aplicaciones. Por ejemplo, para el caso de que un permiso tenga el valor de `null`, significa que este puede ser accedido por otra aplicación.

2.7.4 INYECCIÓN DESDE EL LADO DEL CLIENTE

Finalmente, el archivo .apk contiene las URIs con las cuales se comunica la aplicación y, por medio de la herramienta Drozer, se realizan pruebas para conocer si se pueden realizar o no consultas directamente a estas direcciones. Para esto se procede con los siguientes pasos:

Desde la consola de Drozer, se ejecuta el siguiente comando para obtener las URIs con las cuales se comunica la aplicación. Al mismo tiempo, Drozer intenta realizar consultas a estas direcciones y reporta los resultados como se ven en la Figura 2.53.

```
>run scanner.provider.finduris –a nombre_del_paquete
```

Si se encontraran URIs accesibles, se utiliza el siguiente comando para obtener la información que contiene.

```
>run app.provider.query uri
```

En cambio, para comprobar si existen vulnerabilidades del tipo SQL Injection, se utiliza el siguiente comando:

```
>run scanner.provider.injection –a nombre_del_paquete
```

Con esto se conoce si la aplicación contiene o no, URIs que son vulnerables a este tipo de ataque, como se muestra en la Figura 2.54.

Figura 2.54 Pruebas de inyección SQL

2.8 RESULTADOS ESPERADOS

En este punto se realiza un resumen de los resultados esperados, así como las herramientas y comandos necesarios para la ejecución del protocolo. Esto se encuentra en la tabla del Anexo B.

CAPÍTULO 3

INSTALACIÓN DEL AMBIENTE DE PRUEBAS

3.1 INTRODUCCIÓN

En este capítulo se describe el procedimiento necesario para utilizar las herramientas necesarias para la ejecución del protocolo, de acuerdo a lo definido en el capítulo 2. El procedimiento para tener listas las herramientas está dividido de acuerdo a las fases de recopilación de información, análisis estático y análisis dinámico.

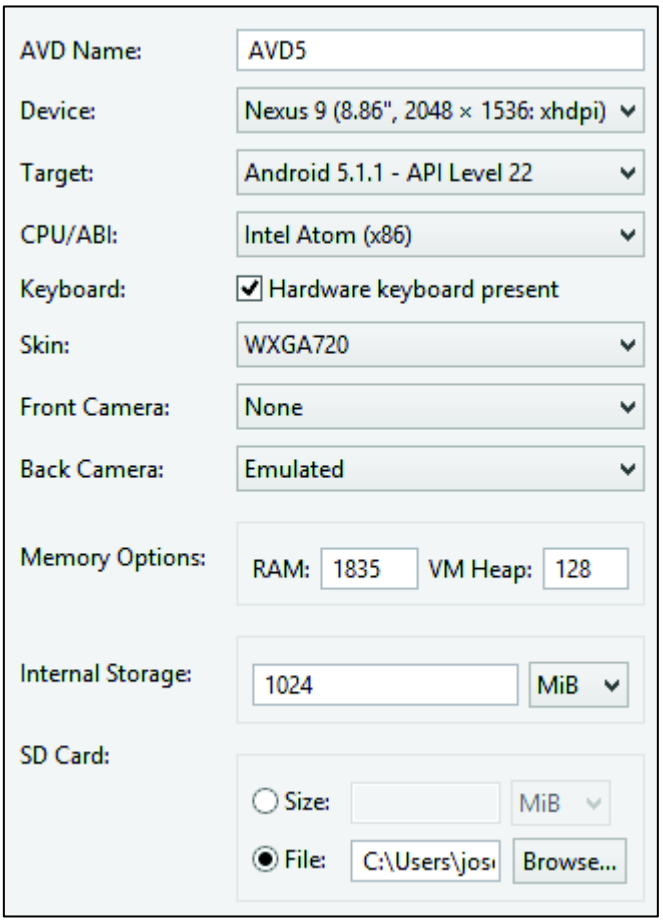


Figura 3.1 Android Virtual Device

3.2 HERRAMIENTAS PARA RECOPILACIÓN DE INFORMACIÓN

A continuación, se presenta el procedimiento para dejar listas las aplicaciones para el desarrollo de la fase de recopilación de información del capítulo 4. Previo a realizar todo este proceso es necesario tener instalado Java y JDK [101].

3.2.1 ANDROID VIRTUAL DEVICE

Figura 3.2 Dispositivo Android listo para instalar las aplicaciones a analizar

Como se describió en el capítulo 2, en la sección 2.5.1, se utiliza un dispositivo virtual Android para explorar la aplicación. Como muestra la Figura 3.1, se recomiendan algunas configuraciones [102], las cuales se describen a continuación:

- Utilizar una CPU Intel Atom (x86) ya que es más rápida y brinda un mejor desempeño del dispositivo virtual.
- Utilizar un tamaño de memoria de al menos 1GB y marcar la casilla “Use Host GPU” ayudan a que el manejo del dispositivo sea más fluido.
- Seleccionar un buen tamaño de pantalla (entre 5 y 10 pulgadas), permite visualizar mejor los menús de la aplicación.
- Se recomienda que el skin del dispositivo sea WXGA720, ya que se puede aprovechar de mejor manera la pantalla del dispositivo.
- Para la versión de Android, se recomienda utilizar una versión de Android a partir de Lollipop.

3.2 se muestra el dispositivo virtual listo para instalar las aplicaciones bancarias que serán analizadas.

3.2.2 JARSIGNER

Esta herramienta viene incluida al momento de instalar Android Studio o el JDK [103] de Java. Para utilizarla, se abre el directorio que contiene el archivo jarsigner.jar, ubicado en `C:\Program Files\Android\Android Studio\jre\bin` o en `C:\Program Files\Java\jdk1.8.0_101\bin` respectivamente, esto se muestra en la Figura 3.3.

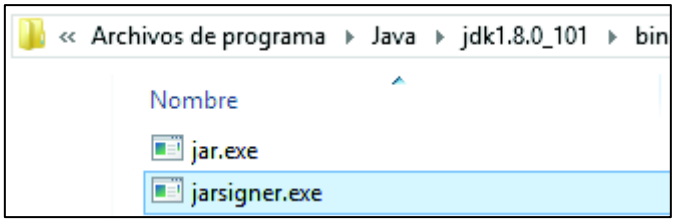


Figura 3.3 Herramienta Jarsigner

Para abrir esta herramienta, se abre una consola al dar *click* derecho dentro de la carpeta mientras se mantiene presionada la tecla *shift* izquierda como se ve en la Figura 3.4.

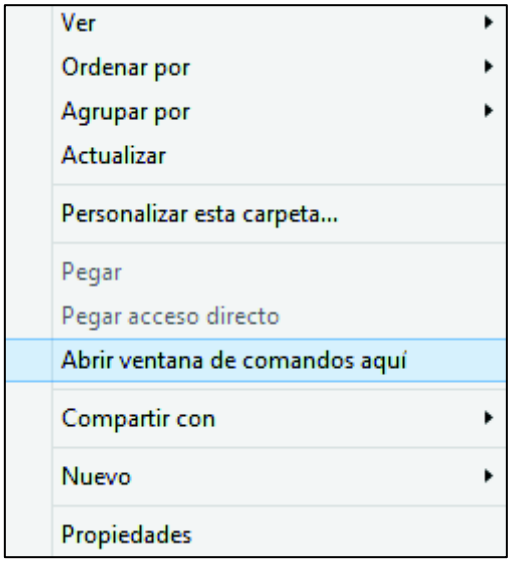


Figura 3.4 Opción para abrir la consola de comandos

También se puede configurar la variable *Path* con la ruta del archivo Jarsigner.exe, como se muestra en la Figura 3.5, para abrir la terminal de Windows desde cualquier ubicación. Cabe mencionar que si el archivo a analizar con Jarsigner no

se encuentra en la misma carpeta, se debe incluir en la sintaxis del comando del punto 2.5.4 la ruta completa del archivo .apk.

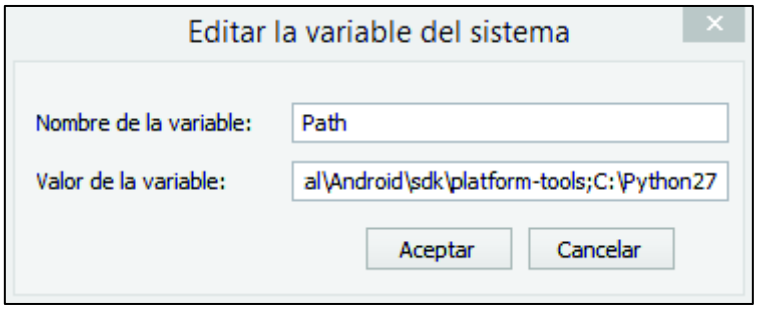


Figura 3.5 Variable Path

3.2.3 OPENSSL

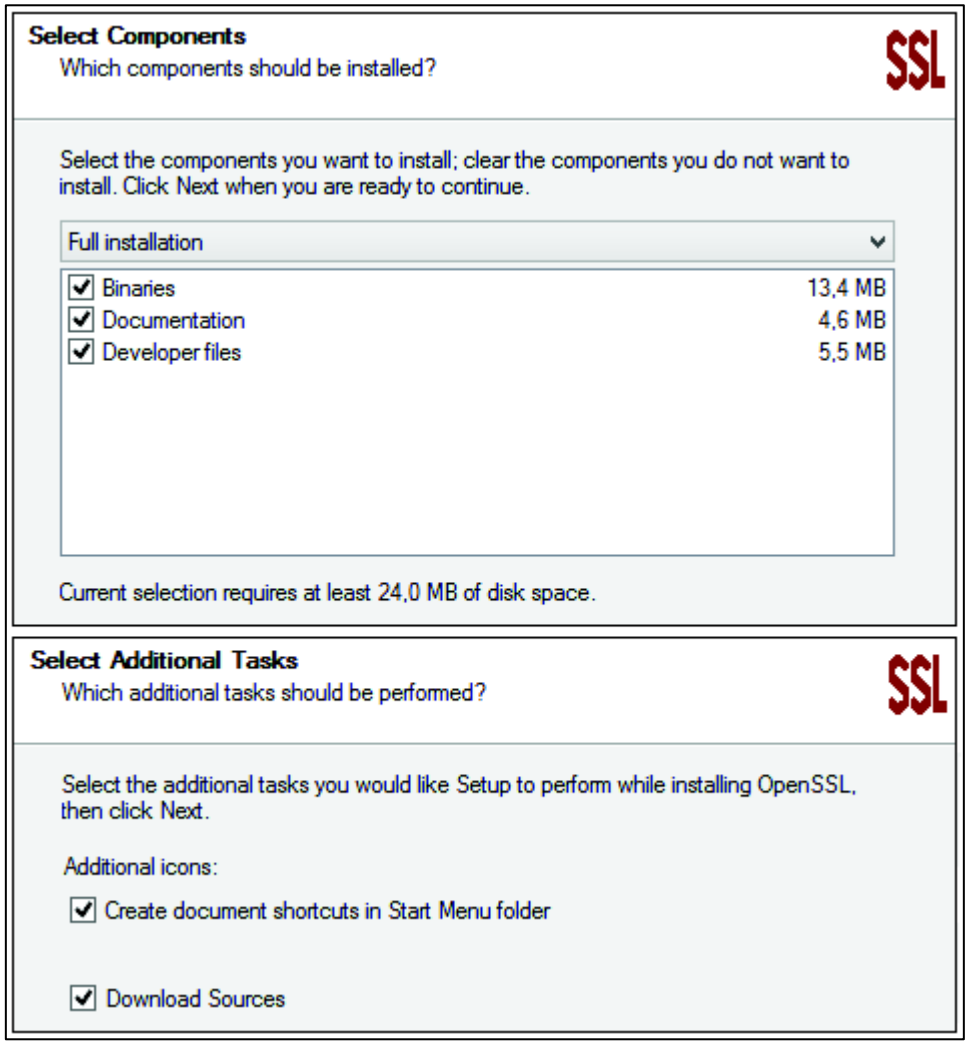


Figura 3.6 Instalación de OpenSSL para Windows

Para Windows, esta herramienta se descarga desde [104]. Una vez que se obtiene el ejecutable, este debe instalarse como Administrador, dentro de este proceso es importante seleccionar la instalación completa como se muestra en la Figura 3.6.

Luego de esto es importante seleccionar la opción `Download Sources`, como se muestra en la Figura 3.6, para que la herramienta tenga todos archivos necesarios para su funcionamiento. Además, se debe configurar la variable *Path* como en la Figura 3.5, para la ruta `C:\Program Files (x86)\GnuWin32\bin` para utilizar la herramienta desde cualquier lugar que se abra la terminal de Windows.

El archivo .RSA que será analizado con esta herramienta se encuentra en la carpeta `META-INF`, la cual aparece como resultado de descomprimir el archivo .apk de la aplicación bancaria a analizar.

3.3 HERRAMIENTAS PARA ANÁLISIS ESTÁTICO

Para la ejecución de la fase análisis estático, como se especifica en el protocolo del capítulo 2, en la sección 2.6, se utilizan las herramientas que se detallan a continuación:

3.3.1 APK EXTRACTOR

Esta herramienta se descarga directamente desde la Play Store, como se muestra en la Figura 3.7 y su instalación es rápida y sencilla. La extracción del archivo .apk de la aplicación seleccionada se realiza a partir de una lista de aplicaciones instaladas en el dispositivo, su extracción es sencilla y el archivo obtenido se guarda en la carpeta `ExtractedApks` del dispositivo.

Figura 3.7 APK Extractor

3.3.2 ANDROID DEBUG BRIDGE (ADB)

Esta herramienta es muy útil ya que permite una comunicación directa con el dispositivo Android físico o virtual, ya que mediante comandos se pueden abrir aplicaciones, instalarlas, verificar dispositivos conectados, reiniciar el dispositivo, etc. Para la ejecución del protocolo de pruebas del capítulo 2, el principal interés sobre esta herramienta es instalar las aplicaciones bancarias A, B y C en el dispositivo virtual.

Luego de la instalación de Android Studio, esta herramienta se encuentra en la carpeta: `C:\Users\usuario\AppData\Local\Android\sdk\platform-tools`, para mayor comodidad se recomienda configurar la variable `Path` con esta ruta.

Una vez que el dispositivo físico Android esté conectado mediante USB al PC, o que a su vez, el dispositivo virtual del punto 3.2.1 se encuentre listo, se utiliza el siguiente comando para verificar que este se encuentre debidamente conectado:

```
>adb devices
```

Con esto, se muestra una lista de los dispositivos conectados y que pueden comunicarse al PC mediante ADB como se muestra en la Figura 3.8. Para este punto es recomendable que sólo haya un dispositivo Android conectado.

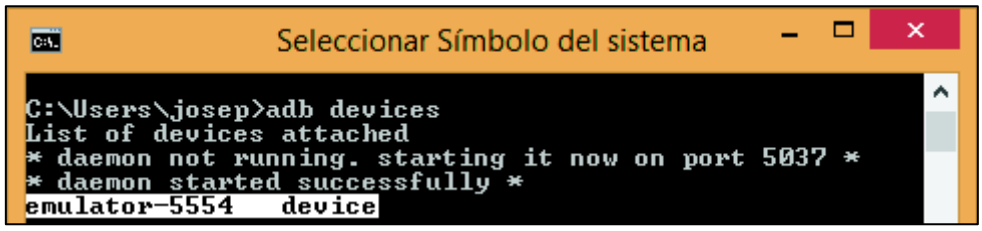


Figura 3.8 Dispositivos conectados

Posteriormente, se instala la aplicación en el dispositivo virtual mediante el siguiente comando, para esto, se usa la ruta absoluta del archivo .apk de la aplicación bancaria a analizar.

```
>adb install ruta_del_archivo\archivo.apk
```

Si la instalación se ha dado con éxito, el resultado será el de la Figura 3.9.

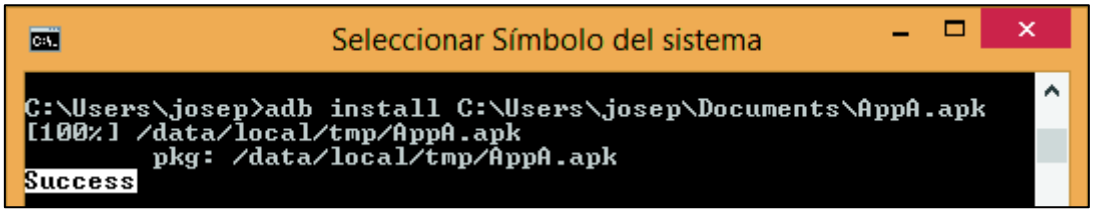


Figura 3.9 Aplicación instalada exitosamente

3.3.3 PEAZIP

Esta herramienta se obtiene desde su página oficial en [19]. Su instalación no ofrece mayor complicación. Sólo siguiendo el asistente se obtiene la herramienta lista para la ejecución del protocolo de pruebas del capítulo 2.

Además de ser una herramienta de código abierto, PeaZip reconoce el formato .apk directamente como un archivo comprimido, por lo que no es necesario cambiar la extensión del archivo .apk para descomprimirlo, como tendría que hacerse usando por ejemplo Winrar.

3.3.4 DEXJAR

Para utilizar esta herramienta, se la descarga desde [68]. Una vez extraído el contenido del archivo, dentro de la carpeta `dex2jar-2.0` se encuentra el archivo `clases-dex2jar.jar` como se muestra en la Figura 3.10.

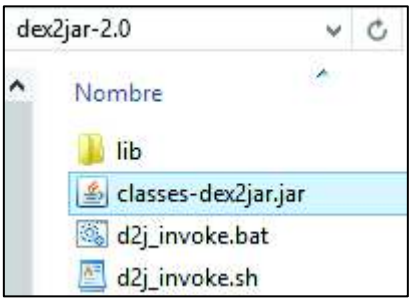


Figura 3.10 Herramienta Dex2jar

Para utilizar esta herramienta se recomienda abrir una consola de comandos en la misma carpeta en donde se encuentra el archivo clases-dex2jar.jar.

3.3.5 AXMLPRINTER

Esta herramienta es un archivo .jar, el cual se encuentra disponible en [60], para su descarga. Para utilizarlo sólo se necesita abrir una consola de comandos en la misma ubicación del archivo como en el punto 3.2.2.

3.3.6 JD-GUI

En [105], se obtiene esta herramienta, luego de su descarga, se descomprime y dentro de la carpeta se tiene el archivo de interés `jd-gui.exe`. Esto se muestra en la Figura 3.11.

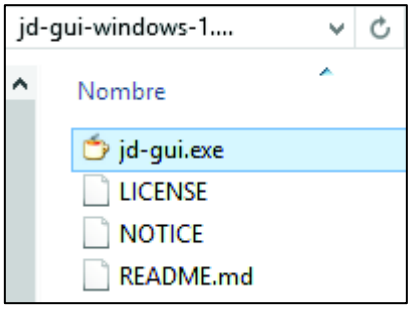


Figura 3.11 Herramienta JD-GUI

Ahora ya se tiene la aplicación lista para ser utilizada. Esta se abre como cualquier otro ejecutable de Windows y una vez dentro, se abre el archivo .jar obtenido mediante Dex2jar.

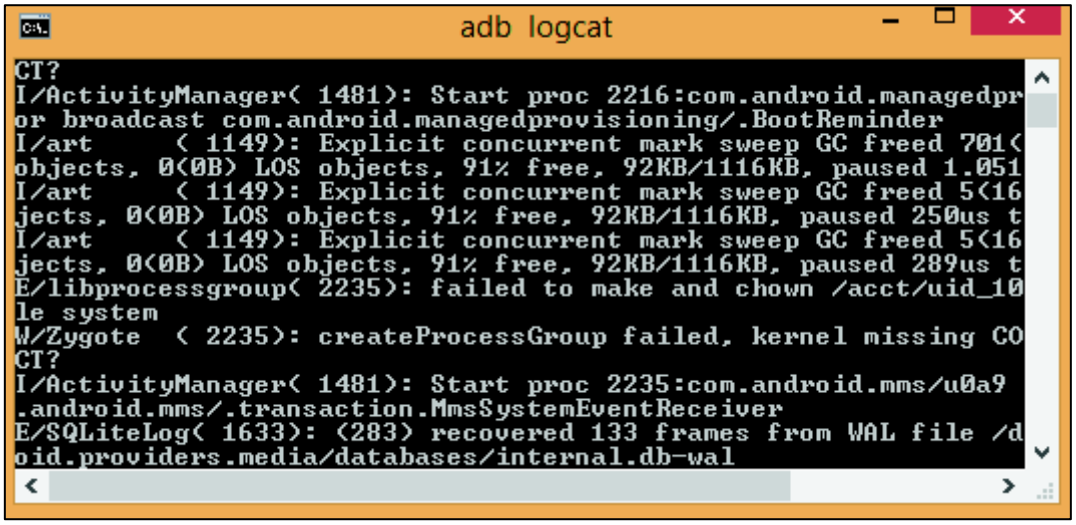


Figura 3.12 Herramienta Logcat

3.3.7 LOGCAT

Para utilizar esta herramienta no se requiere descargar ningún archivo adicional. Esta utiliza ADB y viene incluida en las herramientas de Android. Para iniciarla se ejecuta el siguiente comando en una terminal de Windows:

```
>adb logcat
```

Como resultado se visualizan los logs generados en el dispositivo Android como se muestra en la Figura 3.12. Si se utiliza un dispositivo Android físico, es necesario habilitar la Depuración de USB para visualizar los logs. Esto se consigue habilitando las Opciones de Desarrollo, para esto se va a Ajustes/Acerca del Dispositivo/ y se pulsa 7 veces seguidas sobre el Número de compilación. Como resultado, dentro de Ajustes ya se tiene Opciones de Desarrollador como se muestra en la Figura 3.13.

Figura 3.13 Opciones de desarrollo habilitadas

Hecho esto, dentro de Opciones Desarrollo se habilita la Depuración USB y con esto los logs del dispositivo Android físico ya se visualizan mediante Logcat.

3.3.8 MOBSF

Esta herramienta también utiliza Python 2.7, para su instalación en Windows se la descarga desde [106]. Una vez extraído el contenido, el archivo que interesa es el `requirements.txt`, ya que se encarga de la instalación de todo lo necesario para que la herramienta MobSF funcione, para esto se usa la consola de comandos en la carpeta en la cual se extrajeron los archivos de MobSF. El comando utilizado es:

```
>python –m pip install –r requirements.txt
```

Luego, que la instalación ha finalizado con éxito, se ejecuta el siguiente comando para que se inicie MobSF:

```
>python manage.py runserver
```

Finalmente, ya se tiene la herramienta lista, para utilizarla se abre un navegador y se introduce la dirección `http://localhost:8000`, esto se muestra en la Figura 3.14. Para analizar una aplicación sólo se necesita arrastrar el archivo .apk hacia el área marcada en la página de la herramienta.

MobSF inicia automáticamente el análisis de la aplicación y al terminar presenta un resumen de los resultados obtenidos. Adicionalmente, la herramienta tiene la capacidad de generar un reporte en PDF de los resultados.

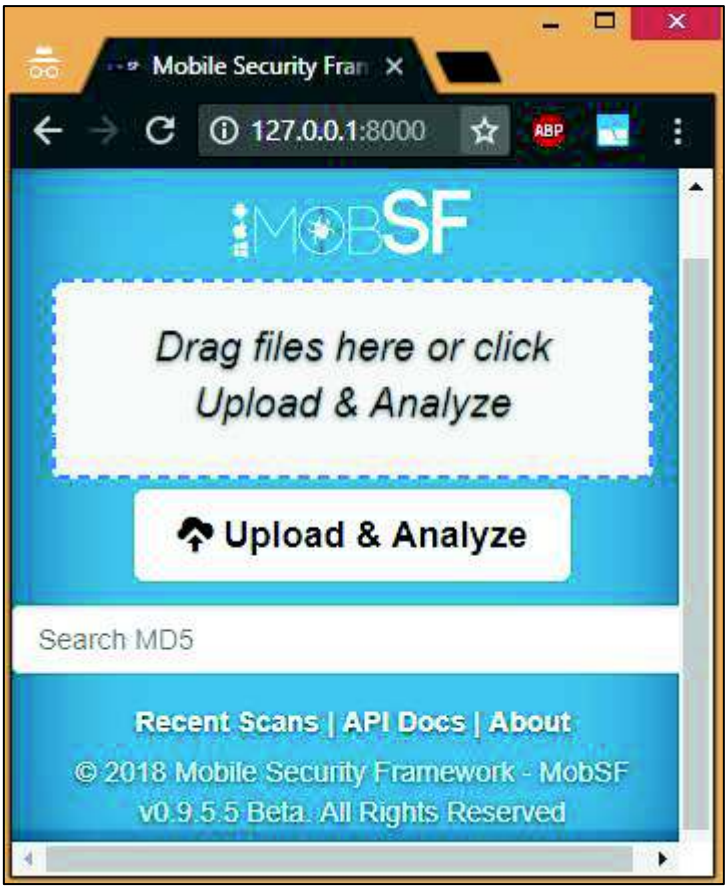


Figura 3.14 Herramienta MobSF

3.4 HERRAMIENTAS PARA ANÁLISIS DINÁMICO

Para la fase final de la ejecución del protocolo de pruebas, de acuerdo a lo descrito en el capítulo 2, en la sección 2.7, se configuran las herramientas necesarias como se detalla a continuación.

3.4.1 NETWORK CONNECTIONS

Esta aplicación se encuentra en la Play Store, y permite visualizar la información acerca de las conexiones generadas por las aplicaciones activas en el dispositivo.

De este modo se obtiene la IP destino, su ubicación geográfica y el número de puerto que usa la aplicación.

3.4.2 OWASP ZAP

Al obtener esta herramienta en [107], se abre el ejecutable y el asistente de configuración se encarga de todo el proceso. Sin embargo, para realizar la captura de tráfico del dispositivo se requiere instalar un certificado en el dispositivo Android y configurar la herramienta OWASP ZAP. Para este punto se recomienda utilizar un dispositivo Android físico, ya que con el dispositivo virtual no se obtuvieron resultados favorables.

Al abrir OWASP ZAP, se solicita un nombre y ubicación para guardar el archivo de la captura, para esto se utiliza el directorio por defecto y se recomienda utilizar un nombre de archivo que ayude a su descripción, por ejemplo, el mismo nombre de la aplicación bancaria analizada.

Hecho esto, en la opción de: Herramientas/Opciones/Certificados SSL Dinámicos, se encuentra el certificado que debe instalarse en el dispositivo Android, como se muestra en la Figura 3.15. Este certificado se guarda y posteriormente se envía al dispositivo Android para instalarlo.

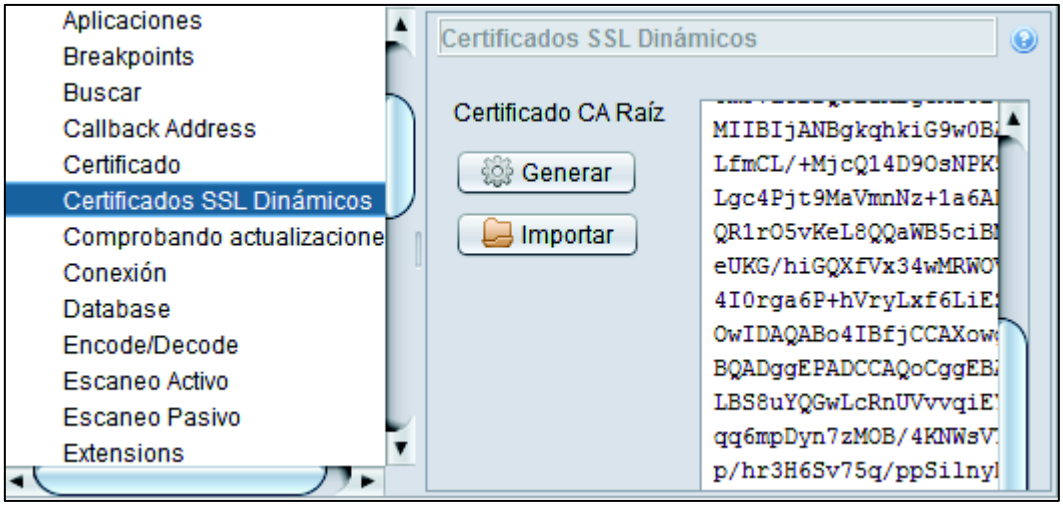


Figura 3.15 Certificado SSL

Para enviar este archivo al dispositivo Android existen varias opciones, se lo puede hacer adjuntándolo en un correo electrónico o, se utiliza ADB para copiarlo directamente al dispositivo.

Figura 3.16 Instalación del certificado

Para copiar el archivo mediante ADB se utiliza el siguiente comando:

```
>adb push Ruta_Origen_Archivo\Archivo.cer Ruta_Destino\
```

Posteriormente, se instala el certificado, se le asigna un nombre cualquiera, como se muestra en la Figura 3.16.

Figura 3.17 Configuración de la conexión

Luego de la instalación del certificado, el dispositivo Android muestra una notificación acerca de que la red puede estar siendo monitoreada, esto es normal. A continuación, se configura la conexión de red del dispositivo para que se conecte a Internet a través de la PC en la cual está instalada la herramienta OWASP ZAP.

Para que la conexión de red se realice a través de la PC, se modifica la conexión WiFi como se ve en la Figura 3.17.

La IP corresponde a aquella de la PC que tiene la herramienta OWASP ZAP instalada y el puerto debe ser el 8080. Además, es conveniente que el *Firewall* de la PC esté desactivado. Con esto ya se tiene lista la herramienta, además, se recomienda que se abra una aplicación bancaria por sesión, para que así no se mezclen los resultados obtenidos.

3.4.3 PIDCAT

A partir de este punto, es necesario tener instalado Python en su versión 2.7 [108], y añadir la ruta del archivo `Python.exe` a la variable `Path`. Una vez instalado se descarga la herramienta Pidcat desde su página oficial en [109]. Esta herramienta trabaja con ADB, por lo que para que funcione se debe extraer todo el contenido del archivo descargado en la misma carpeta que ADB (`Platform Tools`).

Dado que la consola de comandos de Windows no muestra la información de color que genera Pidcat, se recomienda correr este *script* desde una terminal de cualquier sistema operativo GNU/Linux.

3.4.4 MANITREE

Al igual que Pidcat, esta herramienta utiliza Python. Para obtenerla, se la descarga desde su página oficial en [110], y al igual que Pidcat, para utilizarla, se extrae su contenido en la misma carpeta que ADB.

Dado que esta herramienta también muestra información a color, se recomienda el uso de una terminal de cualquier sistema operativo GNU/Linux.

Sin embargo, la información más relevante de los resultados de esta herramienta, se encuentra en las primeras líneas luego de ejecutar el comando del punto 2.7.2, por lo que aún en la consola de comandos de Windows se puede aprovechar la información obtenida.

3.4.5 DROZER

Para utiliza esta herramienta se la descarga desde [111]. En su página oficial se encuentran las instrucciones necesarias para su instalación, sin embargo, se

recomienda utilizar una máquina virtual de Windows o de algún sistema operativo GNU/Linux.

Figura 3.18 Herramienta Drozer

Esto se debe a que esta herramienta utiliza algunos *exploits*, los cuales son reconocidos por herramientas como *malware*. También, se puede utilizar la herramienta AppUse, que ya incluye Drozer.

Para utilizar esta herramienta se activa el servidor, para esto se abre la aplicación en el dispositivo virtual Android y se verifica que el estado del servidor este en “On”, como se muestra en la Figura 3.18.

Posteriormente, se abre la consola de la herramienta, como se muestra en la Figura 3.19. Desde aquí se ejecutarán los comandos descritos en el punto 2.7.

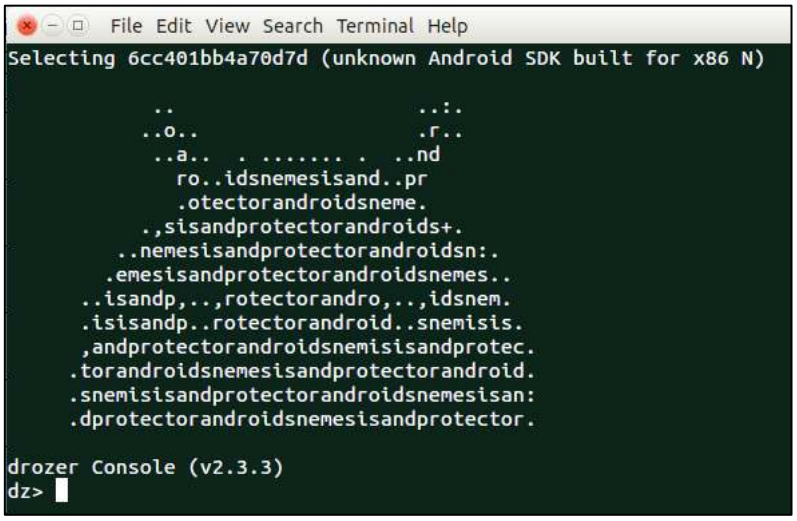


Figura 3.19 Consola de Drozer

CAPÍTULO 4
PRUEBAS Y RESULTADOS

4.1 INTRODUCCIÓN

En este capítulo se ejecuta el protocolo de pruebas definido en el capítulo dos. Este protocolo se aplica a las aplicaciones móviles de los bancos A, B, C tomando en cuenta tres fases: recolección de información, análisis estático y análisis dinámico. Para la ejecución del protocolo de pruebas se han utilizado los archivos .apk de las aplicaciones, obtenidos en el mes de octubre del 2017.

Todas las aplicaciones móviles analizadas requieren de un registro previo en la página web de su respectivo banco. Para realizar el proceso de registro, una vez que se tiene la cuenta bancaria abierta, se requiere ingresar un nombre de usuario, contraseña y preguntas de verificación, así como una imagen de seguridad. Este proceso es muy similar para las tres entidades bancarias que se usan en este trabajo, el detalle de los pasos se encuentra en el Anexo A.

Para la fase de recolección de información se realiza una exploración inicial de las aplicaciones, tomando en cuenta los permisos que solicitan al momento de su instalación. Además, se verifican los componentes de hardware e interfaces de red con los cuales puede interactuar la aplicación. También, se toman en cuenta las transacciones que permite realizar la aplicación así como la interacción con otras aplicaciones.

En la fase de análisis estático se revisa el código que compone la aplicación, incluyendo el archivo `AndroidManifest.xml`, para verificar los permisos que la aplicación solicita durante su instalación. Dentro del código se centra la atención en el proceso de autenticación que realiza cada aplicación móvil analizada (A, B, C).

Finalmente, para la fase de análisis dinámico se analiza el tráfico de las aplicaciones para constatar los protocolos que usan. Además, se utilizan herramientas para el análisis de vulnerabilidades. También, se prueba el filtrado de información del proveedor de contenido y se realiza un ataque de inyección sobre cada aplicación.

4.2 EJECUCIÓN DEL PROTOCOLO PARA LA APLICACIÓN A

4.2.1 RECOLECCIÓN DE INFORMACIÓN

Para la fase de recolección de información, la aplicación A se instala desde la tienda oficial de Google. La aplicación se instala sobre un dispositivo físico o virtual ya que esto no afecta al análisis que se realiza. A continuación, se presentan los puntos que se analizan para esta fase.

4.2.1.1 Funcionalidad y Flujo de Trabajo de la Aplicación

La primera vez que el usuario accede a la aplicación se solicita el ingreso de las mismas credenciales que utiliza para banca electrónica. Adicionalmente, se solicita el ingreso de un PIN de cuatro dígitos, el cual será utilizado posteriormente para el ingreso a la aplicación. De este modo el nombre de usuario y contraseña se utilizan sólo la primera vez que el usuario utiliza la aplicación, además se asigna un nombre al dispositivo. También, se envía un código de activación al número celular del usuario para activar la aplicación.

Una vez terminado este proceso ya se puede ingresar a la aplicación, la pantalla principal (Figura 4.1) muestra las opciones de autenticación disponibles: reconocimiento facial y PIN.

Figura 4.1 Pantalla inicial de la aplicación móvil A

Figura 4.2 Ingreso del PIN

Esta aplicación trabaja con autenticación *online*, por lo cual el dispositivo móvil debe tener acceso a Internet para que el usuario pueda acceder. El PIN es un conjunto de cuatro dígitos que fueron configurados la primera vez que se ingresó a la aplicación.

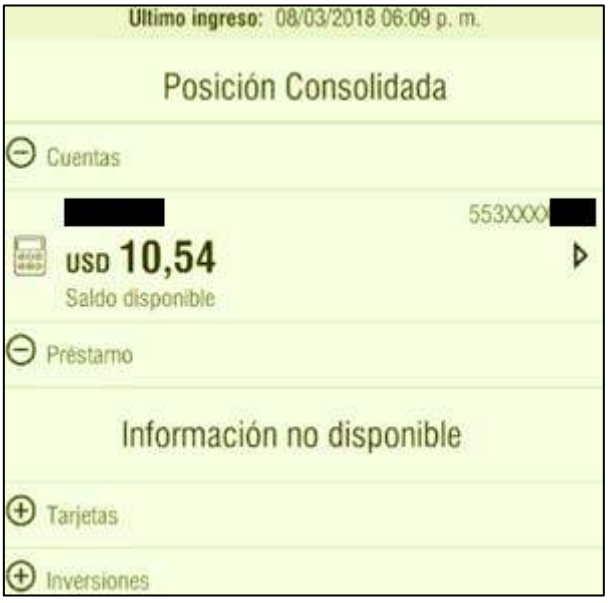


Figura 4.3 Pantalla inicial luego de la autenticación

En la pantalla de ingreso del PIN se muestra el teclado numérico solamente para el ingreso del PIN, este teclado no utiliza el diccionario del teléfono para recordar secuencias de caracteres insertadas anteriormente como se muestra en la Figura 4.2.

Una vez que el ingreso se ha dado de manera exitosa se muestra la pantalla de la Figura 4.3 y no se envía ningún tipo de notificación adicional ni al correo electrónico registrado ni al número celular del usuario.

Figura 4.4 Permisos solicitados por la aplicación A

Si se ingresa incorrectamente el PIN se envía un correo electrónico a la dirección registrada del usuario y se indica en pantalla que se pueden realizar dos intentos más para ingresar correctamente el PIN. Si se excede el número de intentos permitidos el ingreso a la aplicación se bloquea por un día. Luego de ingresar correctamente a la aplicación se muestra en la pantalla la última fecha de ingreso, el nombre de usuario, parte del número de cuenta y el saldo disponible.

4.2.1.2 Interfaces de Red y Componentes de Hardware

Al momento de instalar la aplicación se ven los permisos que solicita la misma, mediante los cuales se determinan las interfaces de red con las que la aplicación interactúa. Estos permisos también se encuentran detallados en el archivo `AndroidManifest.xml`.

A continuación, se detallan los permisos que solicita la aplicación A y esto también se muestra en la Figura 4.4.

La aplicación A como muestra la Figura 4.4, usa las siguientes interfaces de red:

- Comunicación móvil.
- Comunicación WLAN Wi-Fi.

Los componentes de hardware con los que la aplicación interactúa son:

- GPS.
- Cámara.

4.2.1.3 Transacciones Comerciales e Interacción con Otras Aplicaciones

Figura 4.5 Canales oficiales del banco A

Esta aplicación no sólo brinda la información acerca del estado de cuenta del usuario, transacciones realizadas, etc. También, permite realizar transacciones comerciales como:

- Transferencias directas e interbancarias.
- Pago de tarjetas y servicios básicos.
- Consulta del estado de cheques.
- Bloqueo de tarjetas.
- Recargas.

Estas transacciones se pueden realizar directamente desde la aplicación y no requieren de ningún código de confirmación. Por ejemplo, en el caso de una transferencia directa, sólo se requiere ingresar el monto a transferir y el número de cuenta destino para completar la transacción. Además, esta aplicación puede interactuar con las aplicaciones de redes sociales (Facebook, Twitter y YouTube), para acceder a sus canales oficiales. Esto se muestra en la Figura 4.5.

También, permite compartir un mensaje (mediante correo electrónico o mediante SMS) invitando a descargar la aplicación.

```
X.509, OU=
[certificate is valid from 6/26/15 2:03 PM to 6/19/40 2:03 PM]
```

Figura 4.6 Resultado de Jarsigner para la aplicación A

4.2.1.4 Firmado de la Aplicación

Figura 4.7 Información del archivo .cer para la aplicación A

Gracias a la herramienta Jarsigner se obtiene información básica acerca del certificado que se utilizó para firmar la aplicación. Como se ve en la Figura 4.6, para la aplicación A la información que se obtiene no es tan abundante, sólo provee el campo OU (Organizational Unit).

Gracias a la herramienta Openssl se obtiene información adicional mediante el archivo .RSA de la carpeta `META-INF`, como se muestra en la Figura 4.7. La información obtenida incluye la fecha de creación y caducidad del certificado, llave pública, etc.

4.2.2 ANÁLISIS ESTÁTICO

Durante esta fase se realizan los pasos necesarios para el análisis del código que compone la aplicación, de acuerdo a lo especificado en el protocolo de pruebas. Para esto, se toma en cuenta el proceso de autenticación con los pasos que se especifican para cada etapa.

4.2.2.1 Obtención del Código Fuente

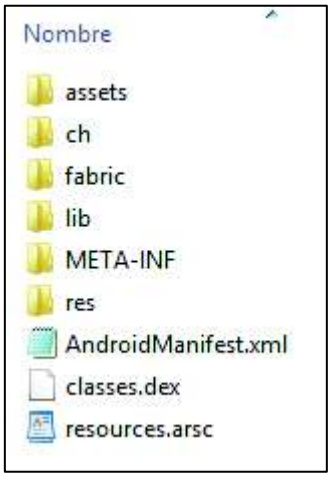


Figura 4.8 Contenido del archivo .apk de la aplicación A

Como se ha definido en el protocolo, el proceso se realiza empezando por descomprimir el archivo .apk previamente obtenido. Una vez descomprimido el contenido, ya se puede visualizar como lo muestra la Figura 4.8. Dentro de todos los archivos que se obtienen se encuentra el archivo `clases.dex`, este archivo es el que se utiliza para analizar a profundidad la estructura de la aplicación.

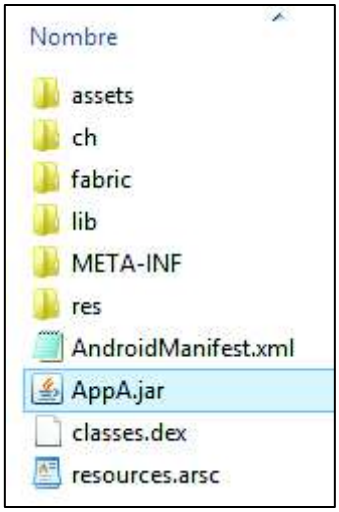


Figura 4.9 Archivo .jar de la aplicación A

Utilizando la herramienta Dex2Jar se obtiene el archivo .jar, el cual se abre con la herramienta JD-Gui. El archivo obtenido se muestra en la Figura 4.9.

4.2.2.2 Permisos en el Archivo AndroidManifest.xml

Luego de descomprimir el archivo .apk mediante PeaZip ya se tiene acceso al archivo AndroidManifest.xml. Para poder visualizar su contenido se hace uso de la herramienta Sublime Text, pero si se abre el archivo tal como se encuentra en la carpeta descomprimida este no es legible.

Transformando el archivo a un formato legible como se define en el protocolo, se obtiene la información necesaria para el desarrollo de este punto. Dentro del archivo se observan las etiquetas `<uses-permission>`, las cuales indican los permisos que la aplicación solicita para su instalación. A continuación, se presenta un extracto de los permisos solicitados, más relevantes que utiliza la aplicación A:

- CAMERA.
- CALL_PHONE.
- ACCESS_NETWORK_STATE.
- READ_GSERVICES.
- WRITE_EXTERNAL_STORAGE.
- GET_TASKS.
- READ_PHONE_STATE.
- ACCESS_WIFI_STATE.
- SEND_SMS.
- ACCESS_WIFI_STATE.
- SEND_SMS.
- ACCESS_FINE_LOCATION.
- WAKE_LOCK.
- READ_SMS.
- WRITE_SMS.
- READ_CONTACTS.

El contenido completo del archivo `AndroidManifest.xml` para la aplicación bancaria A, en formato legible, se encuentra en el ANEXO C.1.

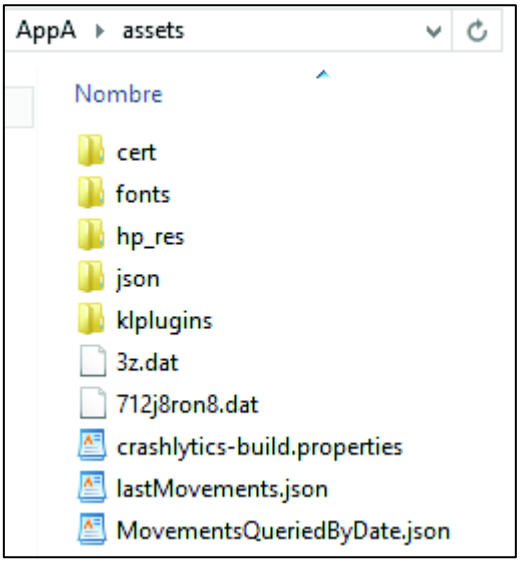


Figura 4.10 Contenido de la carpeta assets de la aplicación A

4.2.2.3 Framework

De acuerdo a lo especificado en el protocolo de pruebas para este punto se ha navegado por el contenido de la carpeta `assets`, ya que no existe dentro del archivo .apk un carpeta `src`. Dentro de la carpeta `assets` se encuentra lo que se muestra en la Figura 4.10. En este caso, la carpeta `assets` no contiene información acerca del *framework* en el cual fue desarrollada la aplicación.

4.2.2.4 Librerías

En este punto, se ha explorado el contenido de la carpeta `lib` en el archivo .apk, el cual se muestra en la Figura 4.11. Como se observa, esta carpeta contiene las librerías para las arquitecturas armeabi, armeabi-v7a y x86.

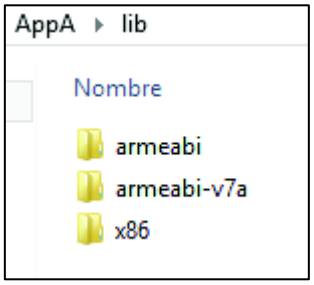


Figura 4.11 Contenido de la carpeta lib de la aplicación A

Revisando el contenido de la carpeta lib, se han encontrado las siguientes librerías:

- libapp_services

- libdump_writer
- libdump_writer_pie
- libFPhi.Extractor
- libgnustl_shared
- libwd
- libwd_pie

Al buscar estas librerías en las referencias recomendadas en el protocolo de pruebas, en la sección 2.6.4, se encuentra que no existen vulnerabilidades reportadas para ninguna de estas. Esto se muestra en la Figura 4.12.

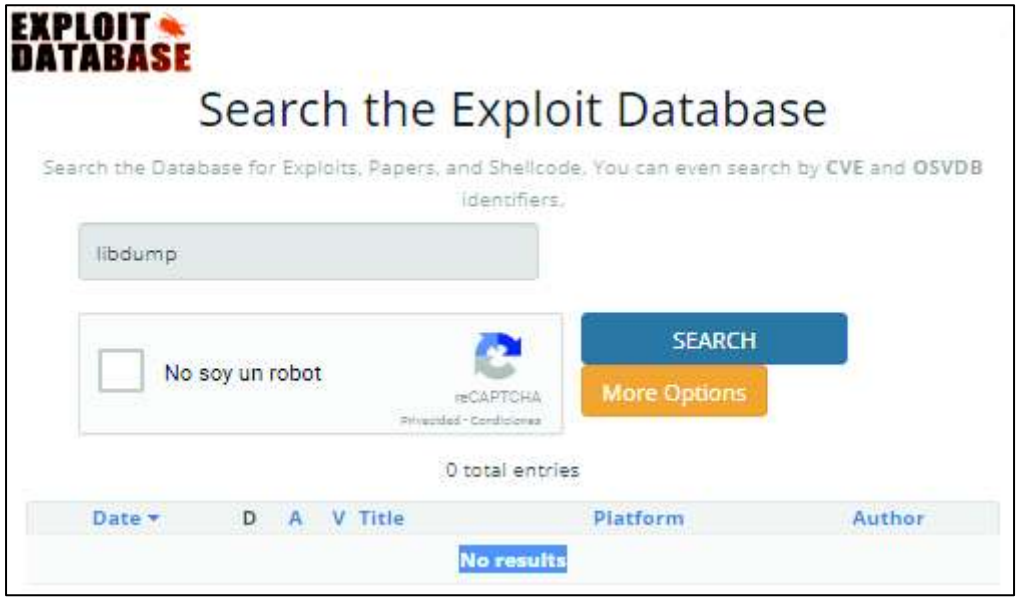


Figura 4.12 Resultados obtenidos para una de las librerías de la aplicación A

4.2.2.5 Vistas

En la Figura 4.13 se muestra un extracto del contenido de la carpeta `res`, obtenida al descomprimir el archivo .apk de la aplicación A. Dentro de esta carpeta se encuentran los archivos necesarios para generar las vistas de la aplicación, por tanto, estos archivos podrían ser utilizados para generar una aplicación maliciosa con el mismo aspecto que la original.

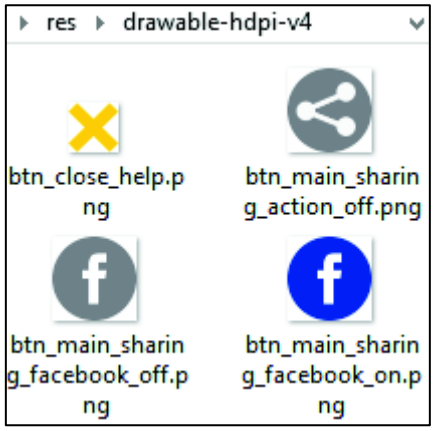


Figura 4.13 Extracto del contenido de la carpeta res de la aplicación A

4.2.2.6 Ingreso de Datos

```
04-27 16:54:37.262 D/Response( 2103): {"resultadoValidaDestino":{"tiposDeSeguridades":[
{"tipoSeguridad":"PIN","mensajeSeguridad":"Estimado Cliente
```

Figura 4.14 Extracto de los logs generados por la aplicación A durante una transferencia directa

Para este punto, se ha analizado el proceso de transferencia electrónica entre dos cuentas pertenecientes a la misma entidad bancaria. Para esto se ha ejecutado la aplicación en un dispositivo virtual Android, para observar los logs generados durante el proceso. En la Figura 4.14 se muestra un extracto de los logs obtenidos. En este extracto se ha identificado la frase `resultadoValidadestino`, a continuación se busca esta frase en el código de la aplicación con el fin de identificar la porción de código referente al proceso de transferencia electrónica.

```
1  public ValidateDirectTransferParser(JSONObject paramJSONObject)
2    {
3      super(paramJSONObject);
4      try
5      {
6        Object localObject = paramJSONObject.getJSONObject("resultadoValidaDestino");
7        this.a = new DirectTransferSummary(parseProduct(((JSONObject)
         localObject).getJSONObject("producto")));
8        JSONObject localJSONObject =
         paramJSONObject.getJSONObject("tresultadoDatosBeneficiario");
9        paramJSONObject = localJSONObject.getString("nombreBenefeciario");
10       boolean bool = ((JSONObject)localObject).optBoolean("requiereValidacion");
11       this.a.setRequireValidation(bool);
12       localObject = localJSONObject.getString("tipoCuenta");
13       this.a.setNameBeneficiary(paramJSONObject);
14       this.a.setDestinationAccountType((String)localObject);
15       return;
16     }
```

Código 4.1 Extracto de la clase validateDirectTransferParser

En el Código 4.1 se muestra la frase `resultadoValidadestino` , la cual se encontró en los logs generados por la aplicación A, durante el proceso de transferencia electrónica entre dos cuentas de la misma entidad bancaria. Se observa que existe el método `validateDirectTransferParser`, el cual maneja los datos del beneficiario de la transferencia, de esta forma se valida que el beneficiario de la transferencia sea el correcto.

4.2.2.7 Código Utilizado para Autenticación

```
1  04-12 15:34:39.714 W/ActivityManager( 1487): Activity pause timeout for
   ActivityRecord{1c9fe787 u0 com.appa.appa/.LoginActivity t2}
2  04-12 15:34:40.864 I/ActivityManager( 1487): Displayed
   com.appa.appa/.PinSetUpActivity: +1s149ms
3  04-12 15:35:14.491 I/***POST ( 2176): {Content-Type=application/json; charset=utf-8,
   id-transaction=XXXXXXXXXXXXXXXXXXXX, device-id=XXXXXXXXXXXXXX,
   unique-id=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX}
```

Figura 4.15 Logs generados por la aplicación A durante el proceso de autenticación

Con base en la sección 2.6.7, se obtienen los logs generados por la aplicación bancaria A en el dispositivo virtual Android, mientras se realiza el proceso de autenticación. Como se muestra en la línea 1 de la Figura 4.15, en primer lugar se identifica la frase LoginActivity, la cual se busca en el código de la aplicación bancaria mediante la herramienta JD-Gui.

```
1  void a()
2  {
3    boolean bool = true;
4    if (SharedPreferencesManager.getInstance().isFPhiRegistered())
5    {
6      Runnable localRunnable = this.o;
7      if (!this.d) {}
8      for (;;)
9      {
10       showFPhiScreen(localUCMode, localRunnable, bool);
11       return;
12       bool = false;
13     }
14   }
15   UUIDGenerator.generateUUID();
16   FPhiRegistrationActivity.start(true, false, getActivity(), true);
17 }
18
19 void b()
20 {
21   showPINScreen();
22 }
```

Código 4.2 Extracto de la clase LoginActivity

Utilizando la palabra `LoginActivity` y `PinsetUpActivity`, identificadas en los logs de la Figura 4.15, se busca el código correspondiente mediante la herramienta

JD-Gui. En el Código 4.2, se muestra un extracto de la clase `LoginActivity`. En este extracto se encuentra que la aplicación maneja dos opciones de autenticación: La opción `FPhi`, para la cual utiliza el método `a` (línea 1), donde primero verifica si el usuario ya está registrado en esta opción, y la opción `PIN`, para la cual utiliza el método `b` (línea 19).

Posteriormente, utilizando la línea 21 del Código 4.2, se encuentra que este pertenece a la clase `AppAActivity`. En donde se ha encontrado el código necesario para autenticación mediante reconocimiento facial (Código 4.3), lo cual fue determinado al buscar la palabra FPhi en Internet y encontrar que esta es una librería utilizada para reconocimiento facial.

```
1  public void showFPhiScreen()
2  {
3    if (!Utils.hasFrontCamera())
4    {
5      DialogManager.getInstance().createAppADialog(this, getString(2131165368), new
       View.OnClickListener()
6        return;
7    }
8  }
```

Código 4.3 Opción de autenticación mediante reconocimiento facial, para la aplicación A

Por otra parte, en la clase `AppAActivity` también se encuentra el método showPinScreen, el cual se encarga de presentar la pantalla de autenticación mediante PIN, lo cual se muestra en el Código 4.4. Además, este método realiza el llamado a la clase `PinSetupActivity` (línea 4), la cual se encarga de la autenticación mediante PIN.

```
1  public void showPINScreen()
2  {
3    Intent localIntent = new Intent(this, PinSetUpActivity.class);
4    localIntent.putExtra(PinSetUpActivity.PARAM_MODE,
     PinSetUpActivity.PinSetupMode.PIN_SETUP_MODE_LOGIN);
5  }
```

Código 4.4 Opción de Autenticación Mediante PIN, para la Aplicación A

Como se muestra en la Figura 4.16, cuando la aplicación A regresa de segundo plano (línea 1) se ejecuta el método shouldShowPin, el cual se busca en el código de la aplicación mediante JD-Gui.

```
1  04-12 20:45:58.498 I/***     ( 2082): AppAActivity onResume
2  04-12 20:46:14.155 I/***AppAActivity( 2082): showRestriction() if (
   shouldShowPin()) {
```

Figura 4.16 Logs generados cuando la aplicación A regresa a primer plano

En el Código 4.5 se muestra un extracto de la clase `AppAActivity`, en la cual se encuentra el método `showRestricion()`, identificado en la Figura 4.16. Como se observa en la línea 9, este método se encarga de regresar al usuario a la pantalla de autenticación regresa a primer plano.

```
1  protected boolean showRestriction()
2  {
3    Log.i("*** ", "AppAActivity showRestriction");
4    if (!SharedPreferencesManager.getInstance().isRegistrationFinished()) {}
5    while (!shouldShowPin()) {
6      return false;
7    }
8    Log.i("***AppAActivity", "showRestriction() if (shouldShowPin()) {");
9    showLoginScreen();
10   return true;
11 }
```

Código 4.5 Método ShowRestriction

4.2.2.8 Autenticación Online/Offline

Gracias al paso de la sección 4.2.2.7 se encuentra el código que se encarga de la autenticación. Para determinar si la autenticación se realiza *online* u *offline* se observan los logs generados por la aplicación bancaria A durante el proceso de autenticación.

Como se observa en la Figura 4.17, existe una URL con la cual se comunica la aplicación bancaria A durante el proceso de autenticación, por lo tanto, el proceso de autenticación se realiza *online*.

```
1  04-09 18:17:18.148 D/Response(23141):     "baseAppAURL": "https://sbapp07.appa.com/
   appa-mobile-services-pfm/rest/",
```

Figura 4.17 Logs obtenidos por la herramienta Logcat para el proceso de autenticación *online*

Ahora que se ha encontrado la URL con la cual se comunica la aplicación bancaria, la misma que se muestra en la Figura 4.17, se la busca en el código y de la aplicación y se encuentra que esta se ubica en la clase `Global`, en el método `getRESTBaseURL`, como se muestra en la línea 15 del Código 4.6.

```
1  public static String getRESTBaseURL()
2  {
3    if (!Utils.isNullOrEmptyOrBlank(NetworkConfig.getInstance().getBaseAppAURL())) {
4      return NetworkConfig.getInstance().getBaseAppAURL();
5    }
6    switch (1.a[ENVIRONMENT.ordinal()])
7    {
8    default:
9      return null;
10   case 1:
11     return "https://yepexdev03.appa.com/appa-mobile-services-pfm/rest/";
12   case 2:
13     return "https://yepexstg04.appa.com/appa-mobile-services-pfm/rest/";
14   }
15   return "https://sbapp07.appa.com/appa-mobile-services-pfm/rest/";
16 }
```

Código 4.6 URL con la cual se comunica la aplicación A

4.2.2.9 Información Adicional al Nombre de Usuario/Password

Al momento de explorar los resultados que muestra Logcat, de acuerdo con el procedimiento de la sección 2.6.9, se encuentra que la aplicación además de utilizar el PIN de acceso a la cuenta, utiliza elementos adicionales como: un identificador de transacción, un identificador de dispositivo y un identificador único, como se ve en la Figura 4.18.

```
1  04-09 17:16:35.814 I/***POST ( 8781): {Content-Type=application/json; charset=utf-8,
   id-transaction=XXXXXXXXXXXXXXXXXX, device-id=XXXXXXXXXXXXXX,
   unique-id=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX}
```

Figura 4.18 Identificadores encontrados en los logs de la aplicación A

4.2.2.10 Métodos de Autenticación

La aplicación cuenta con dos formas de autenticación, la primera mediante un PIN de cuatro dígitos y la segunda mediante reconocimiento facial, esta información se obtuvo a partir de la exploración inicial realizada en la sección 4.2.1. Esto se corrobora mediante la herramienta JD-Gui tal como se muestra en el Código 4.3 (Reconocimiento facial) y Código 4.4 (Ingreso mediante PIN).

4.2.2.11 Bloqueo

Con base en el procedimiento de la sección 2.6.11, se analizan los logs generados mientras se realiza un ingreso de credenciales de acceso erróneas en la aplicación bancaria A. Los logs obtenidos en la Figura 4.19 mediante Logcat, muestran que la aplicación A tiene un número limitado de intentos de acceso. Ante el ingreso erróneo de credenciales deja dos intentos restantes y genera el error con código `TOT-12`, el cual se busca en el código que compone la aplicación.

```
04-10 08:12:32.236 I/ErrorResponse( 2106):
{"clientErrorCode":40003,"clientErrorMessage":"TOT-PIN INVALIDO - TIENE 2
INTENTOS","errorCode":"TOT-12","externalErrorCode":"TOT-12","errorMessages":["Lo
sentimos, el sistema no se encuentra disponible. Por favor revise sus movimientos
antes de realizar otra transacciÃ³n.","Lo sentimos, el sistema no se encuentra
disponible. Por favor revise sus movimientos antes de realizar otra
transacciÃ³n.","TOT-PIN INVALIDO - TIENE 2 INTENTOS"]}
```

Figura 4.19 Extracto de los logs generados por la aplicación A durante el ingreso erróneo de credenciales

Al buscar `TOT-12`, utilizando JD-Gui, en el código que compone la aplicación bancaria A, se encuentra esta frase en la clase `PinSetUpFragment`. Como se muestra en el Código 4.7, se encuentra la frase `TOT-12`, además de un método para disminuir el número de intentos restantes (`decreasePinTries`) y para detectar si quedan intentos restantes (`hasLeftTries`).

```
if (WebServiceHelper.getErrorCode(paramVolleyError).equals("TOT-12"))
    {
      SharedPreferencesManager.getInstance().decreasePinTries();
      if (SharedPreferencesManager.getInstance().hasLeftTries())
      {
        a(getString(2131165375, new Object[] { Integer.valueOf(
        SharedPreferencesManager.getInstance().getLeftPinTries()) }), true);
        d();
        return;
      }
```

Código 4.7 Manejo del número de intentos restantes

4.2.2.12 Autenticación Única

Una vez que el usuario se ha autenticado, la aplicación lo mantiene habilitado para realizar transacciones. La aplicación no controla que luego de un tiempo de inactividad se solicite nuevamente autenticación, por lo que mientras la pantalla del dispositivo móvil se encuentre activa, cualquier persona que tenga acceso al dispositivo puede realizar transacciones.

```
protected boolean shouldShowPin()
{
  return (SharedPreferencesManager.getInstance().isRegistrationFinished()) &&
  (!UserManager.getInstance().isUserLocked()) &&
  (!UserManager.getInstance().hasExceedPinTries()) && (App.NEED_REQUIRE_LOGIN);
}
```

Código 4.8 Método ShouldShowPin

Cuando la pantalla del dispositivo es bloqueada y desbloqueada nuevamente la aplicación solicita nuevamente que se realice el proceso de autenticación. Esto se

consigue mediante el método `shouldShowPin` contenido en la clase `AppAActivity`, este se muestra en el Código 4.8.

Este método es llamado dentro del método `showRestriction`, mostrado en el Código 4.9, el cual a su vez es utilizado dentro del método `onResume` del Código 4.10, el cual se ejecuta cuando la aplicación regresa a primer plano. Todos estos métodos se encuentran contenidos en la clase `AppAActivity`.

```
1  protected boolean showRestriction()
2  {
3    Log.i("*** ", "AppAActivity showRestriction");
4    if (!SharedPreferencesManager.getInstance().isRegistrationFinished()) {}
5    while (!shouldShowPin()) {
6      return false;
7    }
8  }
```

Código 4.9 Método showRestriction

La aplicación está diseñada de forma que se manejan los métodos: `onReceive`, `onCreate`, `onResume` y `onStart`, los cuales se encuentran en la clase `MainActivity`. El método que interesa para el análisis de este punto es el `onResume`, este método es el llamado cuando la aplicación solicita autenticación, así como cuando esta se minimiza y luego vuelve a primer plano.

```
1  private void HP_WRAP_onResume()
2  {
3    super.onResume();
4    this.wasPaused = SharedPreferencesManager.getInstance().getWasPAUSED();
5    stopService(new Intent(getApplicationContext(), NotificationService.class));
6    Log.i("*** ", "AppAActivity onResume");
7    localIntentFilter.addAction("NOTIFICATION_ACTION");
8    if (this.wasPaused)
9    {
10     showRestriction();
11     this.wasPaused = false;
12     SharedPreferencesManager.getInstance().setWasPaused(this.wasPaused);
13   }
14   updateNotificationsCounter();
15 }
```

Código 4.10 Método onResume

4.2.2.13 Autenticación en dos Pasos

La aplicación bancaria A utiliza autenticación en dos pasos, enviando un código de activación durante la primera vez que el usuario accede a la aplicación. Este código de activación se envía tanto al email como al número celular del cliente. Sin embargo, este proceso se realiza únicamente la primera vez, mientras que para

futuras ocasiones, el usuario utiliza sólo un PIN de cuatro dígitos. Esto se determinó en la sección 4.2.1.1.

```
1  04-11 18:15:45.609 D/Response( 2700): {"userStatus":"RegisteredAll","uniqueId1":"XXXXXX
   XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX","email":"usuario@email.com"
   ,"phone":"09XXXXXXXX"}
2  04-11 18:17:49.162 W/ActivityManager( 1488): Activity pause timeout for
   ActivityRecord{32334edf u0 com.appa.appa/.RegisterConfirmationActivity t6}
3  04-11 18:19:50.712 I/send Email and SMS: ( 2700): success
```

Figura 4.20 Extracto de los logs generados durante el primer acceso a la aplicación bancaria A

La Figura 4.20 muestra un extracto de los logs generados durante la primera vez que se realiza el proceso de autenticación. Como se observa en esta figura, la aplicación utiliza la dirección de correo electrónico del usuario y su número celular (línea 1). Posteriormente, utiliza la clase `RegisterConfirmationActivity` (línea2), y finalmente indica que el envío del código de activación mediante SMS y correo electrónico se ha realizado con éxito (línea 3).

```
1  WebServiceHelper.getInstance().postAppARequestWithHeader(String.format("users/%s/
   send-activation-code", new Object[] { this.b }), localHashMap, null, new
   Response.Listener()new Response.ErrorListener
```

Código 4.11 Extracto del código de la clase RegisterConfirmationActivity

En el Código 4.11 se muestra un extracto del contenido de la clase `RegisterConfirmationActivity`, en él se observa que se utiliza la clase `WebServiceHelper` para el envío del código de activación.

```
1  public void sendEmailAndSms(String paramString1, String paramString2, String
   paramString3)
2    {
3      postAppARequest("message-notifications/send", new SendReceiptByEmailRequest(
       paramString1, paramString2), new Response.Listener()new Response.ErrorListener
4      {
5        public void a(JSONObject paramAnonymousJSONObject)
6        {
7          Log.i("send Email and SMS: ", "success");
8        }
9      }, new Response.ErrorListener()
10     {
11       public void onErrorResponse(VolleyError paramAnonymousVolleyError)
12       {
13         WebServiceHelper.getErrorMessage(paramAnonymousVolleyError);
14         Log.i("send Email and SMS: ", "fail");
15       }
16     }, paramString3);
17   }
```

Código 4.12 Extracto del código de la clase WebServiceHelper

En el código 4.12 se muestra un extracto de la clase `WebServiceHelper`, en la cual se utiliza el método `sendEmailAndSms` para el envío del código de activación (línea 1), mientras que en la línea 7 se encuentra el log generado cuando el envío se ha dado de manera exitosa, lo cual se encontró en la línea 3 de la Figura 4.20.

4.2.2.14 Análisis Estático Automatizado

De acuerdo con el procedimiento de la sección 2.6.14, se utiliza la herramienta MobSF para realizar el análisis automatizado de la aplicación bancaria A. Un extracto del resultado obtenido se muestra en la Figura 4.21.

Con base en el análisis realizado por la herramienta MobSF, se ha encontrado que la aplicación bancaria A está diseñada para ejecutarse en dispositivos con una versión de Android superior a Jelly Bean (API 16). Además, se encuentra que esta aplicación tiene un proveedor de contenido, el cual se encuentra exportado. Este punto se analiza posteriormente en la sección 4.2.3.3.

Otro de los resultados más relevantes que se encuentra es que la aplicación ha sido firmada utilizando el algoritmo SHA1 con RSA, del cual se conoce que existen problemas de colisión.

Dentro de los permisos considerados como peligrosos por la herramienta MobSF, la aplicación bancaria A incluye:

- Lectura de los logs generados por el dispositivo.
- Ubicación.
- Listado de las aplicaciones en ejecución.
- Realizar llamadas, etc

En la sección Code Analysis del Anexo D.1, se encuentra que la aplicación bancaria A utiliza la función `Java Hash Code`, la cual es débil y no se recomienda su uso en implementaciones de seguridad. Se encuentra también que la aplicación utiliza un generador de números aleatorios inseguro y que la aplicación registra en logs información sensible.

El resultado detallado del análisis automatizado de la aplicación A se encuentra en el Anexo D.1.

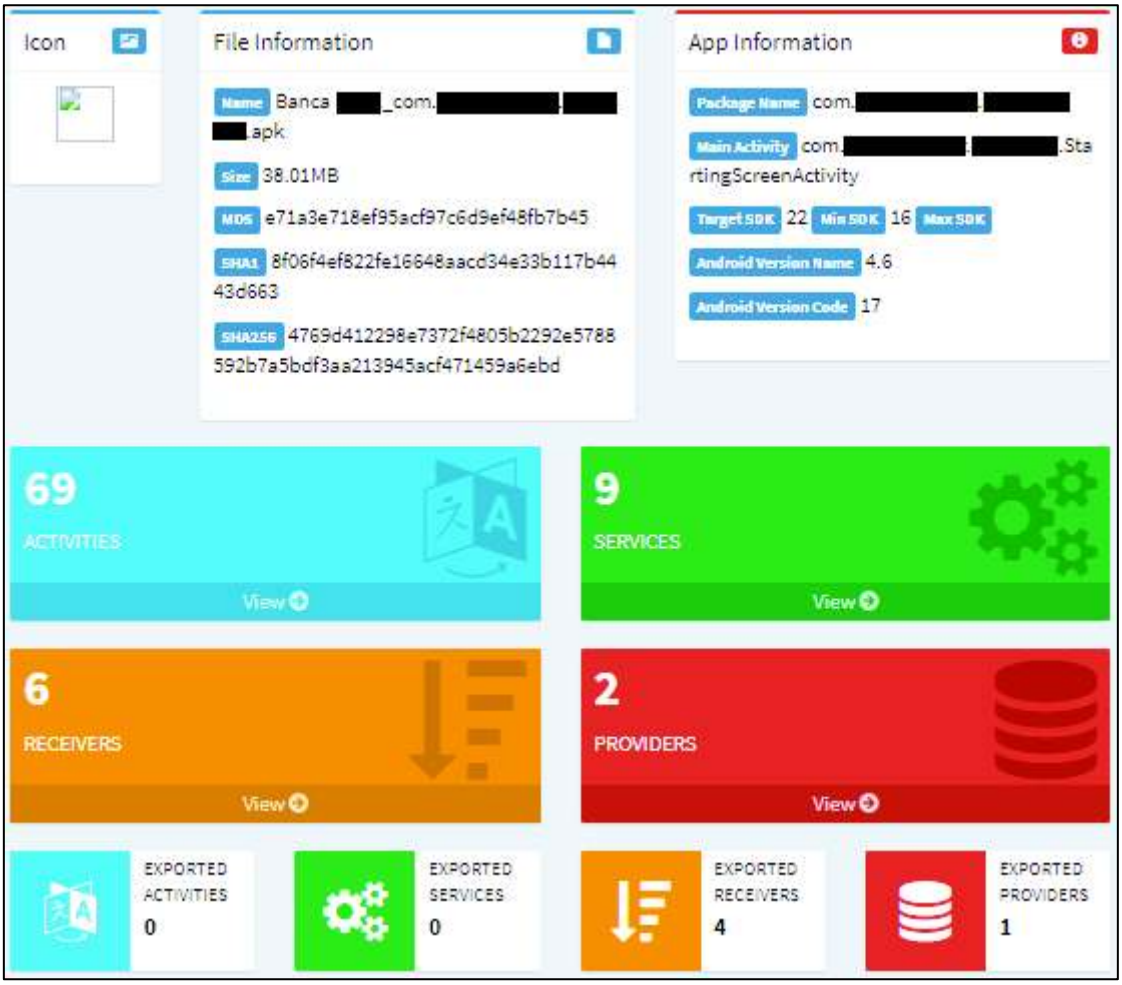


Figura 4.21 Resultado de la herramienta MobSF para la aplicación A

4.2.3 ANÁLISIS DINÁMICO

Durante esta fase se ejecuta el protocolo de acuerdo a lo definido en el segundo capítulo, en la sección 2.7.

Para el punto 4.3.2.1 se utiliza un dispositivo Android físico, mientras que para los otros puntos de la sección 4.2.3 se utiliza un dispositivo Android virtual.

4.2.3.1 Protocolos de Comunicación Utilizados

Para este punto se observan los protocolos que usa la aplicación A para comunicarse con el servidor.

La verificación de los protocolos utilizados se realiza mediante dos opciones: en la primera se utiliza la aplicación Network Connections, disponible en la Play Store, mientras que para la segunda opción se utiliza la herramienta OWASP ZAP.

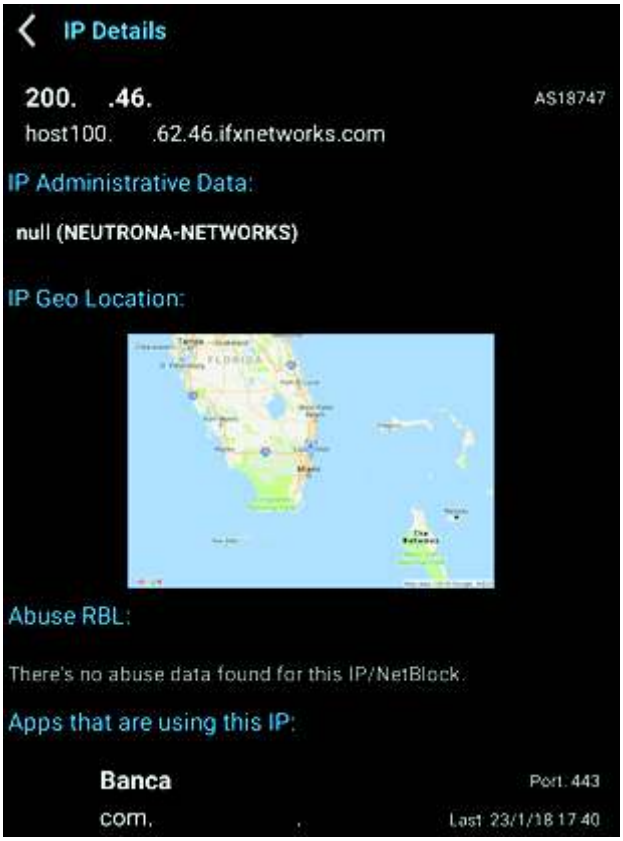


Figura 4.22 Resultado de la herramienta Network Connections para la aplicación A

- Network Connections

Una vez descargada e instalada, esta aplicación muestra las conexiones de red que se realizan con el dispositivo Android. En su versión gratuita, esta aplicación brinda información respecto a la dirección IP con la que se comunica la aplicación A, propietario de esta dirección IP y ubicación geográfica de la misma. Los resultados obtenidos se muestran en la Figura 4.22.

- OWASP ZAP

Ahora que se han realizado los pasos necesarios para configurar adecuadamente la herramienta OWASP ZAP, se procede con el análisis de tráfico que genera la aplicación A.

En la Figura 4.23 se observan las URLs con las cuales se comunica la aplicación A. Además, se observa que durante todo el proceso, la aplicación A se comunica utilizando el protocolo HTTPS.

ID	Req. Tim...	Mét...	URL	C...	Reas...	...	Size Res...	Highe...
6	3/04/18 2...	GET	https://sbapp07.[redacted]...	401	Unau...	5...	37 bytes	Bajo
33	3/04/18 2...	GET	https://sbapp07.[redacted]...	200	OK	7...	133 bytes	
66	4/04/18 0...	GET	https://sbapp07.[redacted]...	200	OK	1...	133 bytes	
37	3/04/18 2...	GET	https://sbapp07.[redacted]...	200	OK	1...	668 bytes	
70	4/04/18 0...	GET	https://sbapp07.[redacted]...	200	OK	1...	668 bytes	
12	3/04/18 2...	GET	https://sbapp07.[redacted]...	200	OK	5...	126 bytes	
64	4/04/18 0...	GET	https://sbapp07.[redacted]...	200	OK	4...	126 bytes	
31	3/04/18 2...	PO...	https://sbapp07.[redacted]...	200	OK	1...	111 bytes	
65	4/04/18 0...	PO...	https://sbapp07.[redacted]...	200	OK	1...	111 bytes	

Figura 4.23 Extracto del resultado de OWASP ZAP para la aplicación A

Sin embargo, OWASP ZAP obtiene información adicional, en el lado izquierdo de la pantalla muestra un resumen de las URLs que identifica, como se muestra en la Figura 4.24, mientras que en la parte derecha muestra en detalle el contenido de los paquetes capturados. Para la aplicación A, las URLs con las cuales se comunica la aplicación se muestran en la Figura 4.24.

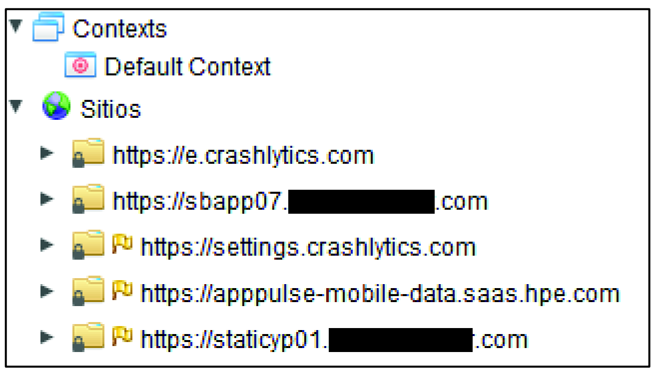


Figura 4.24 URLs con las que se comunica la aplicación A

Al abrir el contenido de los paquetes capturados se encuentra información como la que muestra la Figura 4.25. La información incluye identificador de transacción, versión y modelo de dispositivo desde el cual se ejecuta la aplicación, etc.

```
GET https://sbapp07.appa.com/appa-mobile-services-pfm/rest/
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX HTTP/1.1
Content-Type: application/json; charset=utf-8
id-transaction: 043bdd532XXXXXXXXXXX
Authorization: Basic YWE1Mzc3ZDkyYjFkYzllODc5NjhkZGQ3ZGY3NWRlMWY0MTcxMDkyMjQzNGYyMjXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXTNkMWUwO
TA2N2U5ODM1MGRlNDIx
User-Agent: Dalvik/2.1.0 (Linux; U; Android 6.0.1; D6503 Build/23.5.A.1.291)
```

Figura 4.25 Datos encontrados en los paquetes capturados, generados por la aplicación A

Analizando el contenido de los paquetes capturados mediante OWASP ZAP, se ha encontrado que el PIN que utiliza la aplicación A para el proceso de autenticación, no se envía en texto plano. Esto se evidencia en la Figura 4.26.

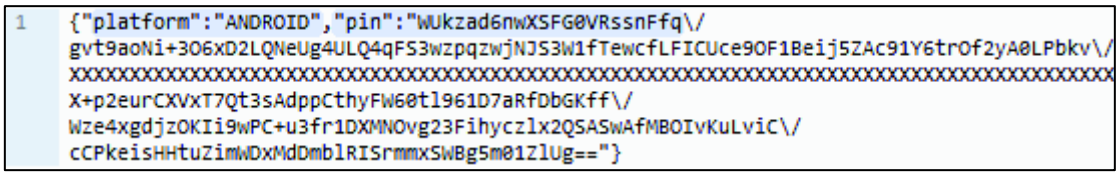


Figura 4.26 Extracto del contenido de los paquetes capturados, generados por la aplicación A para el proceso de autenticación

4.2.3.2 Scripts Existentes para la Búsqueda de Vulnerabilidades en Aplicaciones Android

En este punto se utilizan herramientas existentes para realizar el análisis de la aplicación A, e identificar posibles vulnerabilidades. Como se muestra a continuación, analizando la aplicación A tanto con el *script* Pidcat como con el *script* Manitree, se han logrado identificar datos del usuario así como permisos de la aplicación A, los cuales son potencialmente peligrosos.

Figura 4.27 Extracto de los resultados de la herramienta Pidcat para la aplicación A

- Pidcat

Mediante esta herramienta se analizan los logs generados por la aplicación A, como se ve en la Figura 4.27, esta herramienta tiene la misma funcionalidad que Logcat, pero añade color y agrupa los logs por Actividades, Procesos, etc.

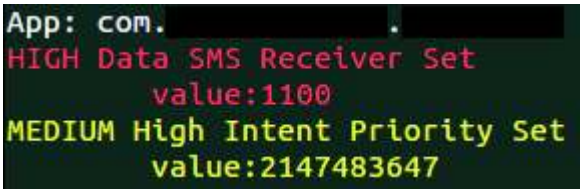


Figura 4.28 Extracto de los resultados de la herramienta Manitree para la aplicación A

Como se muestra en la Figura 4.27, se obtuvieron los logs generados por la aplicación A durante el proceso de autenticación. Como se indicó en la sección 2.7.2, esta herramienta es una versión mejorada de Logcat.

- Manitree

Para la aplicación A, la herramienta Manitree presenta los resultados que se muestran en la Figura 4.28. Como se observa, la información obtenida es la que contiene el archivo `AndroidManifest.xml`. Presenta el resumen de los permisos que solicita la aplicación A, así como los puntos de posibles vulnerabilidades. Para la aplicación A existe un permiso considerado como de alto nivel de vulnerabilidad (Data SMS Receiver Set) y un permiso con un nivel de vulnerabilidad media - alta (*Intent Priority Set*).

4.2.3.3 Proveedor de Contenido

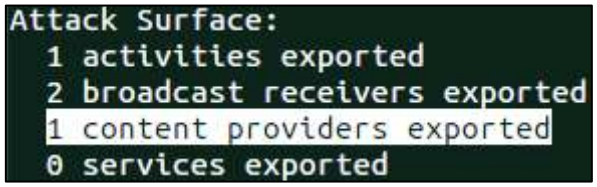


Figura 4.29 Resultados de Drozer para la aplicación móvil A

Los resultados que se obtienen mediante la herramienta Drozer para la aplicación móvil A se muestran en la Figura 4.29. Al utilizar Drozer, mediante el procedimiento de la sección 2.7.3, se encuentra que la aplicación A contiene un proveedor de contenido exportado, lo cual ya fue detectado en el análisis automático realizado en la sección 4.2.2.14.

Figura 4.30 Información del proveedor de contenido

Una vez identificado el proveedor de contenido exportado, se identifican los permisos que este posee, de este modo se conoce si la información manejada por este puede o no ser accedida por otra aplicación. Como indica la Figura 4.30, el

proveedor de contenido identificado (FacebookContentProvider), cuenta con los permisos de lectura y escritura con el valor de `null`. Por lo cual, la información de este proveedor de contenido podría ser accedida por otra aplicación.

4.2.3.4 Inyección desde el lado del Cliente

Para asegurarse que no exista información a la cual se accede desde cualquier URI presente en el código de la aplicación bancaria A, se ejecuta el comando correspondiente, y los resultados obtenidos se muestran en la Figura 4.31. Esta aplicación no tiene URIs accesibles ya que, al final del análisis se encuentra el mensaje `No accesible content URIs found`.

A continuación, utilizando la herramienta Drozer, se comprueba si alguna de las URLs contenidas en la aplicación es susceptible a ataque de Inyección SQL. Como muestra la Figura 4.32, no existen vulnerabilidades de este tipo.

Figura 4.31 Consulta a los URIs encontradas en la aplicación A

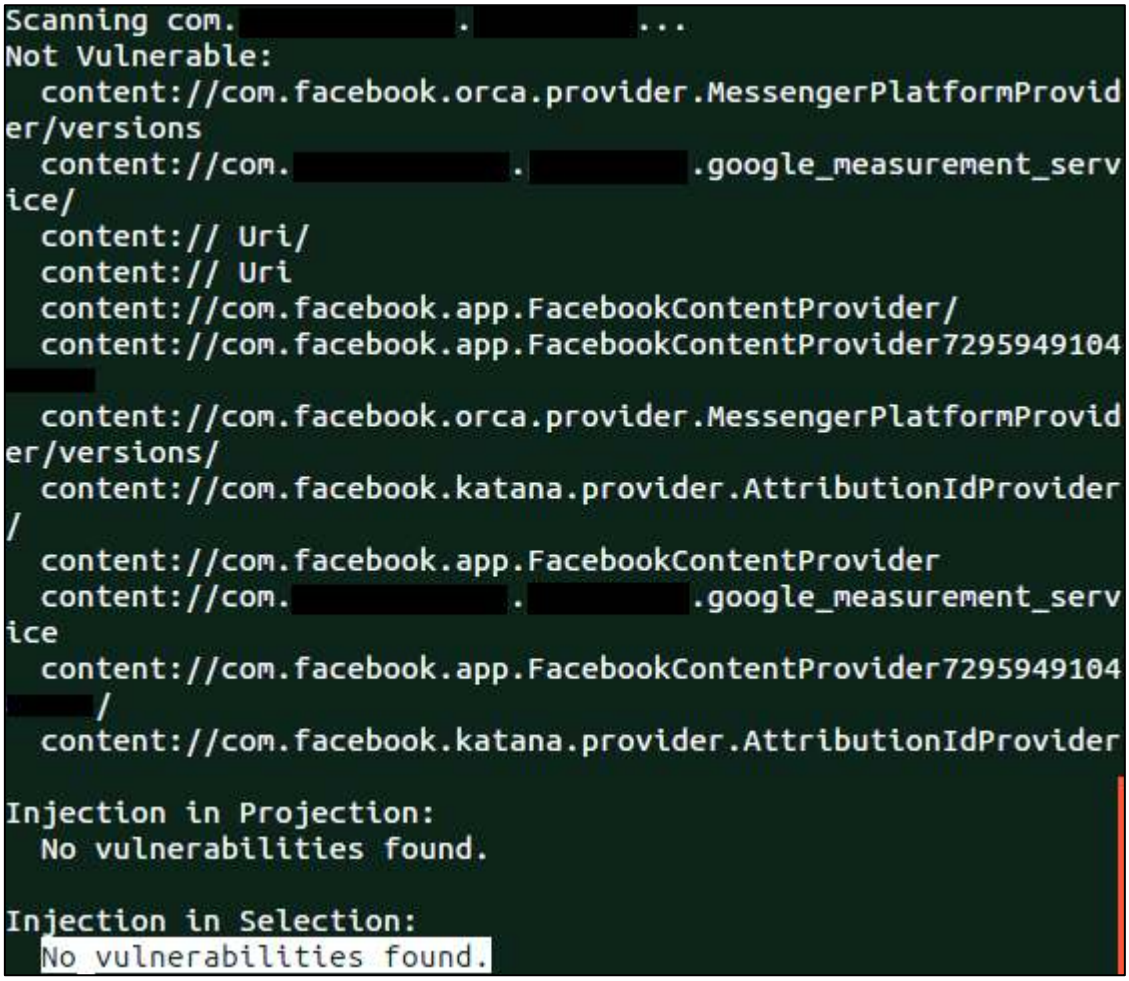


Figura 4.32 Prueba de Inyección SQL para la aplicación A

4.3 EJECUCIÓN DEL PROTOCOLO PARA LA APLICACIÓN B

La ejecución del protocolo para la aplicación bancaria B ha permitido validar su estado de seguridad. Durante la fase de recolección de información se han logrado identificar las transacciones que puede realizar, los permisos que solicita para su instalación, la información del certificado, etc. Para todo esto se utilizaron herramientas como PeaZip, OpenSSL, Jarsigner, etc

En la fase de análisis estático se ha identificado el código, así como las opciones para el proceso de autenticación, además, se ha determinado que la aplicación B trabaja en modo *online* y que no usa un proceso de autenticación en dos pasos. Para el análisis estático se utilizaron herramientas como: DEx2jar, JD-Gui y MobSF. No se encontró el código asociado al proceso de autenticación única, sin embargo, se determinó experimentalmente el tiempo de inactividad dentro del cual la aplicación solicita el ingreso de credenciales nuevamente.

Durante la fase de análisis dinámico, se ha conseguido evaluar la seguridad de la aplicación B mediante herramientas como OWASP ZAP y Drozer, además de *scripts* como Pidcat y Manitree.

El desarrollo de la ejecución del protocolo para la aplicación bancaria B, se encuentra en el Anexo E.1.

4.4 EJECUCIÓN DEL PROTOCOLO PARA LA APLICACIÓN C

La ejecución del protocolo para la aplicación bancaria C ha permitido determinar el estado en cuanto a la seguridad que ofrece. Durante la fase de recolección de información se han encontrado los permisos que solicita la aplicación, así como información del certificado que evidencia la ciudad y nombre del desarrollador de la aplicación.

Durante el proceso de análisis estático se encontró que el procedimiento de análisis de los logs generados por la aplicación C, sólo brinda información de la ubicación de los archivos que utiliza para sus distintos procesos como: autenticación, transferencias, pagos, etc. Para determinar esto, se utilizaron herramientas como el explorador de archivos y MobSF.

Finalmente, se ha conseguido determinar el estado de la seguridad de la aplicación C mediante herramientas como OWASP ZAP y Drozer, además de *scripts* como Pidcat y Manitree. El desarrollo de la ejecución del protocolo para la aplicación bancaria C, se encuentra en el Anexo E.2.

4.5 RESULTADOS OBTENIDOS

A continuación, se presentan los resultados de la ejecución del protocolo de pruebas para las aplicaciones bancarias A, B y C.

4.5.1 RESULTADOS OBTENIDOS PARA LA APLICACIÓN A

Tabla 4.1 Resultados para la aplicación A

Fases	Actividades	Resultados Encontrados
Recolección de Información	Funcionalidad y Flujo de Trabajo	• La aplicación utiliza usuario/*password* sólo la primera vez. Para las futuras ocasiones usa un PIN de cuatro dígitos. • No se envía SMS de verificación al autenticar.

Fases	Actividades	Resultados Encontrados
Recolección de Información	Funcionalidad y Flujo de Trabajo	• Una vez autenticado, el usuario puede realizar todas las transacciones sin confirmación adicional.
	Interfaces de Red y Componentes de Hardware	• La aplicación pide acceso a SMS, sin embargo el envío de SMS de confirmación se realiza sólo la primera vez que se utiliza la aplicación. • La aplicación utiliza la ubicación del cliente.
	Transacciones Comerciales e Interacción con otras Aplicaciones	• Esta aplicación permite varias transacciones sin confirmación adicional.
	Firmado de la Aplicación	• Se encuentra el nombre de la Unidad Organizacional, el cual es el mismo de la aplicación. • El Certificado se encuentra dentro de su período de validez.
Análisis Estático	Obtención del Código Fuente	• El archivo .apk se obtuvo sin problema. • El contenido se extrajo sin problemas. • La conversión se realiza de manera exitosa, sin embargo, hay código que no se ha recuperado.
	Permisos	Los permisos más relevantes identificados son: • CAMERA. • CALL_PHONE. • ACCESS_NETWORK_STATE. • READ_GSERVICES. • WRITE_EXTERNAL_STORAGE. • GET_TASKS. • READ_PHONE_STATE. • ACCESS_WIFI_STATE. • SEND_SMS. • ACCESS_WIFI_STATE. • SEND_SMS. • ACCESS_FINE_LOCATION. • WAKE_LOCK. • READ_SMS. • WRITE_SMS. • READ_CONTACTS.
	Framework	• No ha sido posible determinar el *framework*.

Fases	Actividades	Resultados Encontrados
Análisis Estático.	Librerías	Las librerías encontradas son: • libapp_services • libdump_writer • libdump_writer_pie • libFPhi.Extractor • libgnustl_shared • libwd • libwd_pie No existen vulnerabilidades reportadas.
	Vistas	• El archivo .apk cuenta con los archivos necesarios para generar las vistas de la aplicación.
	Ingreso de Datos	• Se ha encontrado que antes de realizar una transferencia, se validan los datos del destinatario.
	Código para Autenticación	• Se encontró que la aplicación permite el acceso por reconocimiento facial y por PIN. • Se identificaron los métodos onPinClick() y onFPhiClick, para autenticación por PIN y reconocimiento facial respectivamente.
	Autenticación *Online*/*Offline*	• Esta aplicación realiza autenticación *online* a la URL del Código 4.6. Esto se refleja tanto en los Logs generados como en el código de la aplicación.
	Información Adicional al Usuario/*Password*	• La aplicación genera un Identificador de único, el cual se usa durante el proceso de autenticación. Además, utiliza un identificador de dispositivo y uno de transacción, los cuales son utilizados al realiza transacciones.
	Métodos de Autenticación	• La aplicación brinda dos opciones de autenticación: PIN de cuatro dígitos y Reconocimiento Facial
	Bloqueo	• Ante un ingreso incorrecto del PIN por tres veces, la aplicación se bloquea.
	Autenticación Única	• La aplicación requiere que el usuario se autentique cada vez que la aplicación regresa a primer plano. Para esto se usa el método showRestriction().
	Autenticación en dos Pasos	• La autenticación en dos pasos se realiza sólo la primera vez que el usuario accede a la aplicación. • Para las siguientes ocasiones sólo se notifica al usuario al correo, si las credenciales ingresadas han sido incorrectas.

Fases	Actividades	Resultados Encontrados
Análisis Estático.	Análisis Estático Automatizado	• La aplicación ha sido firmada utilizando SHA1, cuyo algoritmo es conocido. • La aplicación utiliza un generador de números aleatorios inseguro. • El detalle del análisis se encuentra en el Anexo D.1.
Análisis Dinámico	Protocolo Utilizados	• Mediante la herramienta Network Connections se encontró que la aplicación utiliza el puerto 443, correspondiente a HTTPS. • Mediante OWASP ZAP, se encontró que la aplicación no envía las credenciales en texto plano.
	Herramientas Existentes	• Mediante Manitree se encontró que la aplicación contiene un receptor de SMS configurado y que la aplicación puede modificar la prioridad anulando otras peticiones. • Mediante Pidcat se obtuvieron los logs generados sólo por la aplicación A.
	Proveedor de Contenido	• Existe un proveedor de contenido exportado y la configuración de sus permisos de lectura y escritura permiten que otra aplicación pueda interactuar con este.
	Inyección SQL	• La aplicación no contiene URLs susceptibles a Inyección SQL.

4.5.2 RESULTADOS OBTENIDOS PARA LA APLICACIÓN B

Tabla 4.2 Resultados para la aplicación B

Fases	Actividades	Resultados Encontrados
Recolección de Información	Funcionalidad y Flujo de Trabajo	• La aplicación utiliza usuario/*password* siempre, para el proceso de autenticación. • No se envía SMS de verificación al autenticar. • Una vez autenticado, el usuario puede realizar transacciones con cuentas previamente registradas en la banca electrónica mediante un código de confirmación.
	Interfaces de Red y Componentes de Hardware	• La aplicación utiliza conexión mediante Wi-Fi, y datos móviles. Además, cuenta con acceso al GPS, almacenamiento del dispositivo y cámara.

Fases	Actividades	Resultados Encontrados
Recolección de Información	Transacciones Comerciales e Interacción con otras Aplicaciones	• Esta aplicación permite realizar varias transacciones a cuentas previamente registradas en banca electrónica.
	Firmado de la Aplicación	• Se encuentra el nombre de la empresa que desarrolló la aplicación. • El certificado se encuentra dentro de la fecha de validez.
Análisis Estático	Obtención del Código Fuente	• El archivo .apk se obtuvo sin problema. • El contenido se extrajo sin problemas. • La conversión se realiza de manera exitosa, sin embargo, hay código que no se ha recuperado.
	Permisos	Los permisos más relevantes identificados son: • CAMERA • WRITE SETTINGS • INTERNET • GET_ACCOUNTS • WAKE_LOCK • NETWORK_STATE • FINE_LOCATION • COARSE_LOCATION • WRITE_EXTERNAL_STORAGE • VIBRATE • READ_PHONE_STATE
	Framework	• No ha sido posible determinar el *framework*.
	Librerías	La librería encontrada es: • libFPhi.Extractor
	Vistas	• El archivo .apk cuenta con los archivos necesarios para generar las vistas de la aplicación.
	Ingreso de Datos	• Se ha encontrado que antes de realizar una transferencia, se validan los datos del destinatario. • Se utiliza un código de confirmación para completar la transferencia.
	Código para Autenticación	• Se encontró que la aplicación permite el acceso por reconocimiento facial y por PIN. • Se identificaron los métodos userPassword() y facialRecognition() para autenticación por usuario/*password* y reconocimiento facial respectivamente.

Fases	Actividades	Resultados Encontrados
Análisis Estático	Autenticación *Online/Offline*	• Esta aplicación realiza autenticación *online* a la URL del Código E.6. Esto se refleja tanto en los Logs generados como en el código de la aplicación.
	Información Adicional al Usuario/*Password*	• Se ha encontrado que esta aplicación no utiliza identificadores o información adicional durante el proceso de autenticación.
	Métodos de Autenticación	• La aplicación brinda dos opciones de autenticación: Usuario/*password*, y Reconocimiento Facial
	Bloqueo	• Ante un ingreso incorrecto del *password* por tres veces, la aplicación se bloquea por 24 horas.
	Autenticación Única	• La aplicación no solicita el ingreso de credenciales al regresar a primer plano. • Se solicita el ingreso de credenciales luego de 5 minutos de inactividad.
	Autenticación en dos Pasos	• La aplicación B no realiza autenticación en dos pasos.
	Análisis Estático Automatizado	• La aplicación ha sido firmada utilizando SHA1, cuyo algoritmo es conocido. • La aplicación utiliza un generador de números aleatorios inseguro. • El detalle del análisis se encuentra en el Anexo D.2.
Análisis Dinámico	Protocolo Utilizados	• Se encontró que la aplicación utiliza el puerto 443, correspondiente a HTTPS. • Mediante OWASP ZAP se identificó el usuario y contraseña en texto plano.
	Herramientas Existentes	• Mediante Manitree se encontró que existen dos servicios (*Firebase Messaging Service* y *Firebase Instance Id Service*), los cuales no están correctamente protegidos. • Mediante Pidcat se obtuvieron los logs generados sólo por la aplicación B.
	Proveedor de Contenido	• La aplicación B no contiene ningún proveedor de contenido exportado.
	Inyección SQL	• La aplicación no contiene URLs susceptibles a Inyección SQL.

4.5.3 RESULTADOS OBTENIDOS PARA LA APLICACIÓN C

Tabla 4.3 Resultados para la aplicación C

Fases	Actividades	Resultados Encontrados
Recolección de Información	Funcionalidad y Flujo de Trabajo	• La aplicación utiliza usuario*/password* siempre, para el proceso de autenticación. • No se envía SMS de verificación al autenticar. • Una vez autenticado, el usuario puede realizar transacciones con cuentas previamente registradas en la banca electrónica mediante un código de confirmación.
	Interfaces de Red y Componentes de Hardware	• La aplicación utiliza conexión mediante Wi-Fi, y datos móviles. Además, cuenta con acceso al GPS, almacenamiento del dispositivo y cámara.
	Transacciones Comerciales e Interacción con otras Aplicaciones	• Esta aplicación permite varias transacciones a cuentas previamente registradas en banca electrónica.
	Firmado de la Aplicación	• Se encuentra el nombre del desarrollador, el nombre del banco y la ciudad. • El certificado se encuentra dentro de la fecha de validez.
Análisis Estático	Obtención del Código Fuente	• El archivo .apk se obtuvo sin problema. • El contenido se extrajo sin problemas. • La conversión se realiza de manera exitosa, sin embargo, hay código que no se ha recuperado.
	Permisos	Los permisos identificados son: • INTERNET • ACCESS_NETWORK_STATE • ACCESS_WIFI_STATE • WRITE_EXTERNAL_STORAGE • ACCESS_COARSE_LOCATION • ACCESS_FINE_LOCATION • CAMERA • WAKE_LOCK • WRITE_SETTINGS
	Framework	• El *framework* Utilizado es Apache Cordova.
	Librerías	• La librería encontrada es: • libFPhi.Extractor
	Vistas	• El archivo .apk cuenta con los archivos necesarios para generar las vistas de la aplicación.

Fases	Actividades	Resultados Encontrados
Análisis Estático	Ingreso de Datos	• Se ha encontrado que antes de realizar una transferencia, se validan los datos del destinatario. • Se utiliza un código de confirmación para completar la transferencia.
	Código para Autenticación	• Se encontró que la aplicación permite el acceso por reconocimiento facial y por PIN. • Se identificaron los métodos usuario/Contraseña() y ReconocimientoFacial() para autenticación por usuario/contraseña y reconocimiento facial respectivamente. • La longitud máxima de la contraseña es de 14 caracteres.
	Autenticación *Online/Offline*	• Esta aplicación realiza autenticación *online* a la URL del Código E.13. Esto se refleja tanto en los Logs generados como en el código de la aplicación.
	Información Adicional al Usuario/*Password*	• Se ha encontrado que esta aplicación no utiliza identificadores o información adicional durante el proceso de autenticación.
	Métodos de Autenticación	• La aplicación brinda dos opciones de autenticación: Usuario/*password*, y Reconocimiento Facial
	Bloqueo	• Ante un ingreso incorrecto del *password* por tres veces, la aplicación se bloquea por 24 horas.
	Autenticación Única	• La aplicación no solicita el ingreso de credenciales al regresar a primer plano. • Se solicita el ingreso de credenciales luego de 5 minutos de inactividad.
	Autenticación en dos Pasos	• La aplicación B no realiza autenticación en dos pasos.
	Análisis Estático Automatizado	• La aplicación ha sido firmada utilizando SHA1, cuyo algoritmo es conocido. • La aplicación registra en logs información sensible. • El detalle del análisis se encuentra en el Anexo D.3.
Análisis Dinámico	Protocolo Utilizados	• Se encontró que la aplicación utiliza el puerto 443, correspondiente a HTTPS. • Mediante OWASP ZAP se identificó el usuario y contraseña en texto plano.
	Herramientas Existentes	• Mediante Manitree no se encontraron problemas con la aplicación.

Fases	Actividades	Resultados Encontrados
Análisis Dinámico	Herramientas Existentes	• Mediante Pidcat se obtuvieron los logs generados sólo por la aplicación C.
	Proveedor de Contenido	• La aplicación C no contiene ningún proveedor de contenido exportado.
	Inyección SQL	• La aplicación no contiene URLs susceptibles a Inyección SQL.

4.6 ANÁLISIS DE RESULTADOS

Con base en la ejecución del protocolo para las aplicaciones bancarias A, B y C, como se detalla en las secciones 4.2, 4.3 y 4.4, y cuyos resultados obtenidos se presentan en las secciones 4.5.1, 4.5.2 y 4.5.3 respectivamente; se realiza el análisis de los resultados obtenidos como se detalla a continuación:

4.6.1 ANÁLISIS DE RESULTADOS PARA LA APLICACIÓN A

4.6.1.1 Recolección de Información

4.6.1.1.1 Funcionalidad y Flujo de Trabajo

El uso de un PIN de cuatro dígitos como método de autenticación representa un punto débil de la aplicación, no solo por la longitud del PIN, sino porque basta con que se ingrese mediante este PIN a la aplicación, para que el usuario tenga total libertad de realizar cualquier tipo de transacción como por ejemplo: transferencias, pagos de servicios, etc.

Además, el envío de un código de activación al correo electrónico o número celular registrados sólo se da durante la primera vez que el usuario ingresa a la aplicación, mientras que para las siguientes veces que el usuario accede a la aplicación no se usa ningún código de confirmación. Este es otro punto débil, ya que el usuario es notificado sólo si el ingreso del PIN ha sido incorrecto. Si alguien consigue el PIN del usuario y el dispositivo móvil registrado, puede realizar cualquier movimiento y el usuario será notificado sólo cuando esta ya haya sido realizada con éxito.

4.6.1.1.2 Interfaces de Red y Componentes de Hardware

La aplicación A solicita permisos que no utiliza, como por ejemplo los permisos de acceso a SMS y teléfono, ya que no es recomendable que la aplicación solicite más permisos de los necesarios para su funcionamiento. Por otra parte, el permiso

de ubicación permite que la aplicación guarde la ubicación del dispositivo, desde la cual se accedió la última vez. Esta información puede ser de utilidad en caso de existir algún comportamiento sospechoso en los movimientos de la cuenta bancaria que maneja la aplicación, lo cual contribuye a mejorar el nivel de seguridad de la aplicación.

4.6.1.1.3 Transacciones Comerciales e Interacción con otras Aplicaciones

La aplicación bancaria A permite realizar varios tipos de transacciones como: transferencias directas e interbancarias, pago de servicios básicos, compra de recargas para telefonía móvil, etc. Sin embargo, estas transacciones no requieren de ningún código de verificación, lo cual pone en riesgo el dinero del cliente.

4.6.1.1.4 Firmado de la Aplicación

El certificado utilizado para firmar la aplicación A sólo muestra el nombre de la entidad bancaria a la que pertenece, y se encuentra vigente, por lo que el certificado digital maneja un adecuado nivel de seguridad.

4.6.1.2 Análisis Estático

4.6.1.2.1 Obtención del Código Fuente

Al realizar el proceso de ingeniería inversa de la aplicación A, no se logró recuperar completamente el código fuente que la compone, lo cual dificulta el proceso de análisis estático. Sin embargo, se ha encontrado el código correspondiente para todos los puntos que comprende el proceso de análisis estático de la aplicación A.

4.6.1.2.2 Permisos

En este punto se ha encontrado que la aplicación A utiliza permisos en el archivo `AndroidManifest.xml`, los cuales no se especifican al momento de instalar la aplicación. Estos permisos controlan procesos como: listar los procesos que se están corriendo en el dispositivo Android, estado del dispositivo, activar la pantalla, leer los SMS y leer los contactos. Estos permisos manejan información sensible de dispositivo, por lo que no ayudan a mantener un adecuado nivel de seguridad de la aplicación.

4.6.1.2.3 Framework

Para la aplicación bancaria A no se encuentra una carpeta `src`, tampoco se encuentra información referente al *Framework* en el cual fue desarrollada la

aplicación en la carpeta `assets`. Por lo tanto, en este aspecto, la aplicación A no expone información del *Framework* en el que fue desarrollada, lo cual es bueno ya que se oculta información como por ejemplo, el lenguaje que utiliza.

4.6.1.2.4 Librerías

Para las librerías que utiliza la aplicación A no existen vulnerabilidades reportadas, por lo que en este punto se maneja un adecuado nivel de seguridad.

4.6.1.2.5 Vistas

Para la aplicación A se ha encontrado que la carpeta res contiene los archivos necesarios para generar las vistas de la aplicación, por lo que estos podrían ser utilizados para crear una aplicación maliciosa de aspecto similar a la aplicación bancaria A.

4.6.1.2.6 Ingreso de Datos

Tomando como ejemplo el proceso para una transferencia electrónica interna, se ha encontrado que se realiza una verificación del ingreso de los datos del destinatario, de este modo se evita que se envíe la transferencia a un destinatario incorrecto. Sin embargo, este proceso no utiliza algún código de confirmación, lo cual pone en riesgo el dinero del cliente.

4.6.1.2.7 Código para Autenticación

Para brindar acceso a la aplicación, la autenticación del usuario se realiza mediante un PIN de cuatro dígitos, el cual es muy débil, ya que no incluye caracteres alfanuméricos sino que son solamente cuatro números para acceder a la aplicación y poder realizar todas las transacciones disponibles. En este caso se observa que el diseño de la aplicación está concebido de forma que el objetivo principal de la aplicación sea la usabilidad y no la seguridad.

4.6.1.2.8 Autenticación online/Offline

Se ha encontrado que la autenticación de la aplicación se realiza de manera *online*, lo cual es recomendable ya que de este modo se previene que se haga un *bypass* de la autenticación, en el caso que esta fuera realizada sólo por la aplicación (*offline*).

4.6.1.2.9 Información Adicional al Usuario/Password

La aplicación A utiliza identificadores adicionales durante el proceso de autenticación como: identificador de transacción, identificador de dispositivo e identificador único; esto contribuye a mejorar el nivel de seguridad de la misma ya que, se asegura que el ingreso a la aplicación se realice desde un dispositivo registrado.

4.6.1.2.10 Métodos de Autenticación

La aplicación maneja dos métodos de autenticación: mediante reconocimiento facial o mediante un PIN. Se recomienda que se mejoren estos métodos, utilizando factor de doble autenticación y otro elemento, por ejemplo un *Captcha*, de modo que ofrezca un mejor nivel de seguridad, al evitar ataques automatizados mediante *scripts*.

4.6.1.2.11 Bloqueo

La aplicación A permite un máximo de tres intentos para acceder, luego de lo cual la aplicación queda bloqueada por un día. Para cada ocasión en la que el usuario ingresa incorrectamente el PIN, se envía una notificación al correo electrónico registrado por el cliente. Se recomienda mejorar este punto enviando la notificación no solo al correo electrónico, sino también al número celular del cliente, ya que no muchas personas están pendientes al correo electrónico. Además, para liberar del bloqueo a la aplicación, se recomienda que sea necesario comunicarse con la entidad bancaria directamente para verificar los datos del cliente.

4.6.1.2.12 Autenticación Única

En este aspecto, la aplicación A maneja un adecuado nivel de seguridad, ya que cada vez que la aplicación pasa a segundo plano, para regresar a primer plano, solicita el ingreso del PIN nuevamente.

4.6.1.2.13 Autenticación en dos Pasos

Esta aplicación utiliza doble factor de autenticación solamente durante la primera vez que el usuario se autentica, lo cual es un punto débil de esta aplicación. Se recomienda utilizar no solo un doble factor de autenticación, sino que también otro elemento, por ejemplo un *Captcha*, el cual impida que la aplicación pueda ser atacada de forma automatizada utilizando scripts.

4.6.1.2.14 Análisis Estático Automatizado

En este punto se ha encontrado problemas en la aplicación A, ya que esta ha sido firmada utilizando el algoritmo SHA1 con RSA, el cual no se recomienda utilizar, ya que presenta problemas de colisiones según el informe presentado por la herramienta MobSF.

Entre otro de los puntos débiles detectados en esta sección se encuentra que la aplicación utiliza un generador de números aleatorios inseguro, así como la función `Java Hash Code`, la cual no debe utilizarse en implementaciones de seguridad.

Además, la aplicación registra en logs gran parte de los eventos que va generando. No es recomendable que la aplicación bancaria registre en logs información sensible del cliente como su número de cuenta, teléfono, etc.

4.6.1.3 Análisis Dinámico

4.6.1.3.1 Protocolos Utilizados

Se ha encontrado que la aplicación A maneja un adecuado nivel de seguridad para este punto, gracias a que emplea el protocolo HTTPS, mediante el puerto 443 para todas sus comunicaciones. Además, al capturar el tráfico generado por la aplicación A, no se encontró el PIN en texto plano, lo cual favorece a la seguridad del usuario.

4.6.1.3.2 Scripts Existentes

Los resultados obtenidos mediante el *script* Manitree muestran como punto de alta importancia, que la aplicación cuenta con un receptor de SMS configurado, el cual no utiliza. Como punto de media importancia se encuentra que la aplicación puede modificar la prioridad de los *intent*, los cuales sirven para modificar componentes como *activities*, *services*, *broadcast receivers*, etc. Este comportamiento es muy común en aplicaciones maliciosas, por lo que no se recomienda su uso en aplicaciones móviles en general.

4.6.1.3.3 Proveedor de Contenido

La aplicación A contiene un proveedor de contenido exportado (FacebookContentProvider), lo cual debe ser corregido modificando la propiedad `android:exported="true"` por `android:exported="false"` en el archivo `AndroidManifest.xml`. Sin embargo, al realizar las pruebas no se encontró que

este proveedor esté filtrando información de ningún tipo, ya que al realizar las peticiones a este proveedor de contenido no devuelve datos.

4.6.1.3.4 Inyección SQL

Para la aplicación bancaria A se ha encontrado que, dentro de las URLs que utiliza para conectarse, no existen problemas de seguridad que hagan a la aplicación vulnerable al ataque de inyección SQL.

4.6.2 ANÁLISIS DE RESULTADOS PARA LA APLICACIÓN B

4.6.2.1 Recolección de Información

4.6.2.1.1 Funcionalidad y Flujo de Trabajo

Desde la primera ocasión, la aplicación B solicita el nombre de usuario, así como la contraseña, para poder acceder a la aplicación, lo cual brinda mayor seguridad que un PIN de cuatro dígitos. En este caso, se recomienda el uso de autenticación en dos pasos, dado que actualmente esta no se utiliza.

Para realizar transacciones dentro de la aplicación se utiliza un código de confirmación, lo cual favorece a la seguridad de la aplicación dado que el usuario necesitará de este código cada vez que desee realizar una transacción.

4.6.2.1.2 Interfaces de Red y Componentes de Hardware

Se ha encontrado que la aplicación bancaria B, solicita permisos que no utiliza como el acceso al ID de dispositivo y datos de llamada. Además, utiliza el acceso al GPS del dispositivo, el cual sólo se utiliza al inicio de la aplicación para ubicar la agencia más cercana, pero esta información podría ser utilizada de mejor manera. Por ejemplo, para guardar la ubicación desde la cual se realizó la última transacción, de este modo se podría mejorar el nivel de seguridad de la aplicación.

4.6.2.1.3 Transacciones Comerciales e Interacción con otras Aplicaciones

La aplicación bancaria B permite realizar varios tipos de transacciones como: transferencias directas e interbancarias, pago de servicios básicos, compra de recargas para telefonía móvil, etc. Para realizar estas transacciones se requiere de un código de confirmación adicional, esto favorece al nivel de seguridad de la aplicación, dado que el usuario requerirá de este código cada vez que desee realizar una transacción.

4.6.2.1.4 Firmado de la Aplicación

En el certificado utilizado para firmar la aplicación se ha encontrado el nombre de la empresa que desarrolló la aplicación, esta información no se recomienda exponerla.

4.6.2.2 Análisis Estático

4.6.2.2.1 Obtención del Código Fuente

Al realizar el proceso de ingeniería inversa de la aplicación B, no se logró recuperar completamente el código fuente que la compone, lo cual dificulta el análisis estático. Por ejemplo, en el punto 4.6.2.2.12 no se ha podido encontrar el código encargado de solicitar nuevamente el ingreso de credenciales luego de 5 minutos de inactividad.

4.6.2.2.2 Permisos

En este punto se ha encontrado que la aplicación B utiliza permisos en el archivo `AndroidManifest.xml`, los cuales no se especifican al momento de instalar la aplicación. Estos permisos controlan procesos como: obtener datos de las cuentas del dispositivo, leer el estado del dispositivo y para activar la pantalla. Estos permisos manejan información sensible de dispositivo, por lo que no ayudan a mantener un adecuado nivel de seguridad de la aplicación.

4.6.2.2.3 Framework

Para la aplicación bancaria B no se encuentra una carpeta `src`, tampoco se encuentra información referente al *framework* en el cual fue desarrollada la aplicación en la carpeta `assets`. Por lo tanto, en este aspecto, la aplicación B no expone información del Framework en el que fue desarrollada, lo cual es bueno ya que se oculta información como por ejemplo, el lenguaje que utiliza.

4.6.2.2.4 Librerías

Para la librería que utiliza la aplicación B no existen vulnerabilidades reportadas, por lo que en este punto se maneja un adecuado nivel de seguridad.

4.6.2.2.5 Vistas

Para la aplicación B se ha encontrado que la carpeta `res` contiene los archivos necesarios para generar las vistas de la aplicación, por lo que estos podrían ser

utilizados para crear una aplicación maliciosa de aspecto similar a la aplicación bancaria B.

4.6.2.2.6 Ingreso de Datos

Tomando como ejemplo el proceso para una transferencia electrónica interna, se ha encontrado que se realiza una verificación del ingreso de los datos del destinatario, de este modo se evita que se envíe la transferencia a un destinatario incorrecto. Además, para este proceso se utiliza un código de confirmación, lo cual favorece al nivel de seguridad de la aplicación.

4.6.2.2.7 Código para Autenticación

La aplicación bancaria B maneja un nivel adecuado se seguridad en este aspecto, ya que, para el ingreso a la aplicación, siempre solicita el nombre de usuario y contraseña. Además, tanto el nombre de usuario como la contraseña pueden incluir caracteres alfanuméricos. Se podría mejorar este punto utilizando autenticación de dos pasos.

4.6.2.2.8 Autenticación online/Offline

Se ha encontrado que la autenticación de la aplicación se realiza de manera *online*, lo cual es recomendable ya que de este modo se previene que se haga un *bypass* de la autenticación, en el caso que esta fuera realizada sólo por la aplicación (*offline*).

4.6.2.2.9 Información Adicional al Usuario/Password

La aplicación B utiliza no utiliza identificadores adicionales como identificador de dispositivo, de transacción, etc., durante el proceso de autenticación. Se recomienda utilizar este tipo de identificadores, de modo que se pueda tener más detalle del dispositivo desde el cual se realiza cualquier transacción.

4.6.2.2.10 Métodos de Autenticación

La aplicación maneja dos métodos de autenticación: mediante reconocimiento facial o mediante usuario*/password*. Se recomienda que se mejoren estos métodos, utilizando factor de doble autenticación y otro elemento, por ejemplo un *Captcha*, de modo que ofrezca un mejor nivel de seguridad, al evitar ataques automatizados mediante *scripts*.

4.6.2.2.11 Bloqueo

La aplicación B permite un máximo de tres intentos para acceder, luego de lo cual la aplicación queda bloqueada por un día. Para cada ocasión en la que el usuario ingresa incorrectamente el PIN, se envía una notificación al correo electrónico registrado por el cliente. Se recomienda mejorar este punto enviando la notificación no solo al correo electrónico, sino también al número celular del cliente, ya que no muchas personas están pendientes al correo electrónico. Además, para liberar del bloqueo a la aplicación, se recomienda que sea necesario comunicarse con la entidad bancaria directamente para verificar los datos del cliente, ya que actualmente, la aplicación se desbloquea automáticamente luego de 24 horas.

4.6.2.2.12 Autenticación Única

Para este punto no ha sido posible identificar el código encargado de solicitar el ingreso de credenciales nuevamente, después de cierto tiempo o cuando la aplicación regresa a primer plano, ya que este código posiblemente se encuentre en el lado del servidor. Por esto, se ha determinado de forma experimental que la aplicación solicita nuevamente el ingreso de credenciales luego de cinco minutos de inactividad.

4.6.2.2.13 Autenticación en dos Pasos

La aplicación B no utiliza doble factor de autenticación, lo cual es un punto débil de esta aplicación. Se recomienda utilizar no solo un doble factor de autenticación, sino que también otro elemento, por ejemplo un *Captcha*, el cual impida que la aplicación pueda ser atacada de forma automatizada utilizando *scripts*.

4.6.2.2.14 Análisis Estático Automatizado

En este punto se ha encontrado problemas con el certificado utilizado para firmar la aplicación, ya que este utiliza el algoritmo SHA1 con RSA, el cual no se recomienda utilizar, ya que presenta problemas de colisiones según el informe presentado por la herramienta MobSF. Además, se encuentra que en el código de la aplicación se utiliza un generador de números aleatorios, el cual es inseguro y no se recomienda su uso. Además, la aplicación B registra en logs información sensible, lo cual tampoco es recomendable.

4.6.2.3 Análisis Dinámico

4.6.2.3.1 Protocolos Utilizados

Se ha encontrado que la aplicación B utiliza el protocolo HTTPS para comunicarse con el servidor, lo cual favorece a la seguridad de la aplicación. Sin embargo, mediante la herramienta OWASP ZAP se ha logrado capturar el tráfico generado por la aplicación B. Al momento de analizar los paquetes capturados se han encontrado el nombre de usuario y contraseña de acceso a la aplicación en texto plano, lo cual no es recomendable, dado que estas credenciales pueden ser utilizadas por otra persona para acceder a la aplicación.

4.6.2.3.2 Scripts Existentes

Para la aplicación bancaria B se ha encontrado que, como punto de media importancia, existen dos servicios (*Firebase Messaging Service* y *Firebase Instance Id Service*), los cuales, según el informe del *script* Manitree, no se encuentran protegidos adecuadamente. Esto no es recomendable ya que, al no requerir un permiso, estos servicios son accesibles a cualquier otra aplicación. Los servicios *Firebase Messaging y Firebase Instance Id* son utilizados para enviar notificaciones a la aplicación o indicarle que existen datos disponibles para la sincronización, esta información fue encontrada al buscar la palabra *Firebase* en Internet.

4.6.2.3.3 Proveedor de Contenido

La aplicación B no contiene proveedores de contenido exportado, por lo que maneja un adecuado nivel de seguridad en este punto.

4.6.2.3.4 Inyección SQL

Para la aplicación B no se han encontrado vulnerabilidades de inyección SQL.

4.6.3 ANÁLISIS DE RESULTADOS PARA LA APLICACIÓN C

4.6.3.1 Recolección de Información

4.6.3.1.1 Funcionalidad y Flujo de Trabajo

Desde la primera ocasión, la aplicación C solicita el nombre de usuario, así como la contraseña, para poder acceder a la aplicación, lo cual brinda mayor seguridad que un PIN de cuatro dígitos. En este caso, se recomienda el uso de autenticación en dos pasos, dado que actualmente esta no se utiliza.

Para realizar transacciones dentro de la aplicación se utiliza un código de confirmación, lo cual favorece a la seguridad de la aplicación dado que el usuario necesitará de este código cada vez que desee realizar una transacción.

4.6.3.1.2 Interfaces de Red y Componentes de Hardware

La aplicación bancaria C maneja un adecuado nivel de seguridad en este punto, ya que utiliza los permisos que solicita al momento de su instalación. La información de GPS se utiliza actualmente para encontrar la agencia más cercana. Sin embargo, esta información podría ser utilizada de mejor manera. Por ejemplo, para guardar la ubicación desde la cual se realizó la última transacción, de este modo se podría mejorar el nivel de seguridad de la aplicación.

4.6.3.1.3 Transacciones Comerciales e Interacción con otras Aplicaciones

La aplicación bancaria C permite realizar varios tipos de transacciones como: transferencias directas e interbancarias, pago de servicios básicos, compra de recargas para telefonía móvil, etc. Para realizar estas transacciones se requiere de un código de confinación adicional, esto favorece al nivel de seguridad de la aplicación, dado que el usuario requerirá de este código cada vez que desee realizar una transacción.

4.6.3.1.4 Firmado de la Aplicación

Para este punto, se ha identificado el nombre del desarrollador, así como el departamento de la empresa en la que fue desarrollada, además de la ciudad. Esta información no es recomendable que esté expuesta.

4.6.3.2 Análisis Estático

4.6.3.2.1 Obtención del Código Fuente

Utilizando el archivo .dex de la aplicación C no se ha logrado encontrar mayor información acerca su funcionamiento. Sin embargo, al explorar el contenido del archivo .apk, se ha encontrado en la ruta `assets/www` los archivos necesarios para autenticación, transferencias, etc. Estos archivos utilizan código HTML y JavaScript.

4.6.3.2.2 Permisos

En este punto se ha encontrado que la aplicación C sólo utiliza un permiso (*Wake lock*) en el archivo `AndroidManifest.xml`, el cual no se especifica al momento

de instalar la aplicación. Este permiso controla el proceso de activar la pantalla. Este permiso permite evitar que el dispositivo se bloquee, lo cual se recomienda eliminar, ya que si alguien obtiene acceso al dispositivo, con la aplicación abierta y el usuario autenticado, podría acceder a información sensible del cliente como su número de cuenta, saldo, etc.

4.6.3.2.3 Framework

Para la aplicación bancaria C no se encuentra una carpeta `src` como parte del archivo .apk. Sin embargo se encuentra información referente al *framework* en el cual fue desarrollada la aplicación en la carpeta `assets`. De este modo se encuentra que la aplicación fue creada utilizando Apache Cordova. Esto cual no es recomendable ya que, mediante la información del *Framework*, se puede conocer por ejemplo, el lenguaje en el que fue desarrollada la aplicación.

4.6.3.2.4 Librerías

Para la librería que utiliza la aplicación B no existen vulnerabilidades reportadas, por lo que en este punto se maneja un adecuado nivel de seguridad

4.6.3.2.5 Vistas

Para la aplicación C se ha encontrado que la carpeta `res` contiene los archivos necesarios para generar las vistas de la aplicación, por lo que estos podrían ser utilizados para crear una aplicación maliciosa de aspecto similar a la aplicación bancaria C.

4.6.3.2.6 Ingreso de Datos

Se ha encontrado que los procesos de la aplicación C no se pueden analizar mediante logs ya que estos no muestran información de las clases o variables que está utilizando la aplicación. Esto favorece a su seguridad, ya que, al desconocer el código para realizar las diferentes transacciones se dificulta el análisis y explotación de posibles vulnerabilidades. Por otra parte, si el código reside en el servidor con el cual se comunica la aplicación, disminuye la eficiencia de esta, ya que depende de la velocidad de conexión con el servidor.

4.6.3.2.7 Código para Autenticación

Se ha encontrado que la aplicación utiliza código HTML y JavaScript. Además, durante la exploración del código de la aplicación, se ha encontrado que este

contiene información sensible como números de cédula, nombres completos, direcciones e incluso se ha encontrado la imagen de un cheque. Todo esto no es recomendable ya que la aplicación C pone en riesgo la información personal de los propietarios de dichos datos. Además, se podría mejorar el nivel de seguridad de la aplicación C utilizando autenticación de dos pasos.

4.6.3.2.8 Autenticación online/Offline

Se ha encontrado que la autenticación de la aplicación se realiza de manera *online*, lo cual es recomendable ya que de este modo se previene que se haga un *bypass* de la autenticación, en el caso que esta fuera realizada sólo por la aplicación (*offline*).

4.6.3.2.9 Información Adicional al Usuario/Password

La aplicación C utiliza no utiliza identificadores adicionales como identificador de dispositivo, de transacción, etc., durante el proceso de autenticación. Se recomienda utilizar este tipo de identificadores, de modo que se pueda tener más detalle del dispositivo desde el cual se realiza cualquier transacción.

4.6.3.2.10 Métodos de Autenticación

La aplicación maneja dos métodos de autenticación: mediante reconocimiento facial o mediante usuario/*password*. Se recomienda que se mejoren estos métodos, utilizando factor de doble autenticación y otro elemento, por ejemplo un *Captcha*, de modo que ofrezca un mejor nivel de seguridad, al evitar ataques automatizados mediante *scripts*.

4.6.3.2.11 Bloqueo

La aplicación C permite un máximo de tres intentos para acceder, luego de lo cual la aplicación queda bloqueada. Para cada ocasión en la que el usuario ingresa incorrectamente el PIN, se envía una notificación al correo electrónico registrado por el cliente. Se recomienda mejorar este punto enviando la notificación no solo al correo electrónico, sino también al número celular del cliente, ya que no muchas personas están pendientes al correo electrónico. Además, para liberar del bloqueo a la aplicación, esta solicita comunicarse con la entidad bancaria directamente, lo cual provee un adecuado nivel de seguridad.

4.6.3.2.12 Autenticación Única

Para este punto no ha sido posible identificar el código encargado de solicitar el ingreso de credenciales nuevamente, después de cierto tiempo o cuando la aplicación regresa a primer plano, ya que este código posiblemente se encuentre en el lado del servidor. Por esto, se ha determinado de forma experimental que la aplicación solicita nuevamente el ingreso de credenciales luego de cinco minutos de inactividad.

4.6.3.2.13 Autenticación en dos Pasos

La aplicación C no utiliza doble factor de autenticación, lo cual es un punto débil de esta aplicación. Se recomienda utilizar no solo un doble factor de autenticación, sino que también otro elemento, por ejemplo un *Captcha*, el cual impida que la aplicación pueda ser atacada de forma automatizada utilizando *scripts*.

4.6.3.2.14 Análisis Estático Automatizado

En este punto se ha encontrado problemas con el certificado utilizado para firmar la aplicación, ya que este utiliza el algoritmo SHA1 con RSA, el cual no se recomienda utilizar, ya que presenta problemas de colisiones según el informe presentado por la herramienta MobSF. Se encuentra que la aplicación registra en logs información sensible, lo cual tampoco es recomendable.

4.6.3.3 Análisis Dinámico

4.6.3.3.1 Protocolos Utilizados

Se ha encontrado que la aplicación C utiliza el protocolo HTTPS para comunicarse con el servidor, lo cual favorece a la seguridad de la aplicación. Sin embargo, se ha encontrado que se puede identificar el nombre de usuario y contraseña al capturar el tráfico generado por la aplicación C, mediante la herramienta OWASP ZAP. Lo cual resulta perjudicial para la seguridad de los datos del usuario, dado que estas credenciales pueden ser utilizadas por otra persona para acceder a la aplicación.

4.6.3.3.2 Scripts Existentes

En este punto, se ha encontrado que la aplicación bancaria C maneja un adecuado nivel de seguridad ya que la herramienta Manitree no reporta puntos vulnerables.

4.6.3.3.3 Proveedor de Contenido

La aplicación C no contiene proveedores de contenido exportado, por lo que maneja un adecuado nivel de seguridad en este punto.

4.6.3.3.4 Inyección SQL

En este punto, no se han encontrado vulnerabilidades, por lo que la aplicación maneja un nivel de seguridad adecuado.

CAPÍTULO 5
CONCLUSIONES Y RECOMENDACIONES

5.1 CONCLUSIONES

- Mediante el análisis de las aplicaciones A, B y C, se observa que existe la necesidad de un mayor control sobre el proceso de desarrollo de las aplicaciones móviles, ya que actualmente existen falencias como la inclusión datos de clientes en el código de la aplicación, el firmado de la aplicación mediante el algoritmo SHA1 con RSA, el cual tiene problemas de colisión, el registro en logs de datos sensibles del usuario, etc. Todo esto puede comprometer la información personal de los clientes.
- Al analizar la fundamentación teórica, tanto de Android, como de OWASP *Mobile Security Project*, se ha encontrado que, se puede adaptar esta metodología general para la prueba de aplicaciones móviles basadas en Android, a casos específicos como el del análisis de aplicaciones móviles de entidades bancarias del Ecuador.
- Gracias a la selección de pasos específicos de la metodología OWASP *Mobile Security Project*, considerando que se trata de aplicaciones bancarias, se ha conseguido diseñar una primera versión de un protocolo de pruebas para verificar el estado de la seguridad en las mismas. El protocolo se compone de las fases de: recolección de información, análisis estático y análisis dinámico.
- Se ha logrado implementar el análisis de aplicaciones móviles en Android, para la banca electrónica del Ecuador, gracias al uso de un dispositivo virtual creado mediante *Android Virtual Device Manager*, así como de herramientas gratuitas como: OpenSSL, Dex2jar, JD-Gui, Logcat, OWASP ZAP y Drozer.
- Mediante el uso de herramientas para desarrollar cada una de las tres fases: recolección de información, análisis estático y análisis dinámico, así como de herramientas que realizan un análisis automatizado de los archivos .apk, se han analizado los resultados obtenidos, como por ejemplo: datos de clientes dentro del código de la aplicación, información del cliente registrada en los logs que genera la aplicación, datos de la empresa que desarrolló la

aplicación, etc. de modo que se pueda conocer el estado de la seguridad para cada aplicación.

- La aplicación bancaria A tiene un Proveedor de Contenido exportado, sin embargo, los permisos de acceso a este se encuentran configurados adecuadamente, de modo que otras aplicaciones no pueden acceder a los datos que contiene. En cambio, las aplicaciones B y C no contienen Proveedores de contenido exportados.
- La aplicación bancaria C incluye dentro de su código fuente, información personal de uno de sus clientes, así como la imagen de un cheque, lo cual atenta contra la privacidad de este usuario, resaltando la falta de controles de seguridad sobre el desarrollo de la aplicación.

5.2 RECOMENDACIONES

- Para el desarrollo de aplicaciones móviles basadas en Android, se debe considerar no sólo la funcionalidad de la misma, sino también que cuando la aplicación ya sea subida a la tienda oficial, esta no contenga información personal de clientes, empresa que desarrolló la aplicación, nombre y correo del desarrollador, etc.
- Al evaluar la seguridad de la aplicación bancaria A, se encuentra que el uso de un pin de cuatro dígitos no provee un adecuado nivel de seguridad, ya que esta no es muy extensa y el patrón puede ser fácilmente reconocido, por lo que se recomienda cada vez que se realice la autenticación, se envíe una notificación, mediante SMS o correo electrónico, tanto si la autenticación se ha realizado de manera exitosa como si ha ocurrido algún error.
- Si bien existen herramientas que realizan un análisis automatizado de los archivos .apk, se recomienda que también se realice el análisis de forma manual, esto permite detectar información contenida en el código fuente, que las herramientas de análisis automático no detectan.
- Para el análisis de los protocolos de comunicación que utilizan las aplicaciones bancarias, se recomienda utilizar un dispositivo Android físico, ya que con el dispositivo virtual no se consigue capturar el tráfico generado por las aplicaciones.

- Antes de la exploración del código fuente de las aplicaciones bancarias, se recomienda explorar el contenido del archivo .apk descomprimido, ya que se ha encontrado que algunas aplicaciones incluyen información importante como fotos de cheques, información de usuarios, etc.
- Durante la exploración del código fuente, se recomienda la búsqueda de palabras como: PIN, login, authentication, *password*, etc. Esto ayudará a encontrar con mayor facilidad el código relacionado al proceso de autenticación.
- Dado que algunas de las herramientas como *Android Virtual Device Manager* y OWASP ZAP funcionan mejor en una máquina física, se recomienda que todas las herramientas sean instaladas en una máquina física y no virtual, excepto para el caso de Drozer, la cual se recomienda tenerla en una máquina virtual ya que el *Firewall* de Windows puede bloquear su ejecución.

REFERENCIAS BIBLIOGRÁFICAS

[1] A. Becker and M. Pant, *Android*. 2009.

[2] “Platform Architecture | Android Developers.” [Online]. Available: https://developer.android.com/guide/platform/index.html. [Accessed: 01-Dec-2017].

[3] L. J. and Y. J., “Research on development of android applications,” *Proc. - 2011 4th Int. Conf. Intell. Networks Intell. Syst. ICINIS 2011*, pp. 69–72, 2011.

[4] G. Robinson and G. R. S. Weir, “Understanding android security,” in *Communications in Computer and Information Science*, 2015, vol. 534, pp. 189–199.

[5] “Mobile Security Project - OWASP.” [Online]. Available: https://www.owasp.org/index.php/Mobile_Security_Project_Archive#tab=M-Security_Testing. [Accessed: 16-Feb-2018].

[6] “Android Interfaces and Architecture | Android Open Source Project.” [Online]. Available: https://source.android.com/devices/#Hardware Abstraction Layer. [Accessed: 01-Dec-2017].

[7] “ART and Dalvik | Android Open Source Project.” [Online]. Available: https://source.android.com/devices/tech/dalvik/. [Accessed: 01-Dec-2017].

[8] E-Linux, “Android Architecture,” *eLinux.org*, pp. 4–7, 2011.

[9] “Gartner | Market Share Android vs iOS.,” 2017. [Online]. Available: https://www.gartner.com/newsroom/id/3609817. [Accessed: 17-Feb-2018].

[10] “Dashboards | Android Developers.” [Online]. Available: https://developer.android.com/about/dashboards/index.html. [Accessed: 01-Dec-2017].

[11] A. Gupta, *Learning Pentesting for Android Devices*, 1st ed. Birmingham: Packt Publishing, 2014.

[12] *Unlocking Android*, 1st ed. Greenwich: Manning Publications, 2009.

[13] A. Dubey and A. Misra, *Android Security: Attacks and Defenses*, 1st ed. New York: Taylor & Francis Group, 2013.

[14] A. Kumar and T. Vithani, “A comprehensive mobile application development and testing lifecycle,” *2014 IT Prof. Conf.*, vol. I, pp. 1–27, 2014.

[15] R. Meier, *Professional Android 2 Application Development*. 2010.

[16] “The Activity Lifecycle | Android Developers.” [Online]. Available: https://developer.android.com/guide/components/activities/activity-lifecycle.html. [Accessed: 01-Dec-2017].

[17] Google Android, “Introduction to Android | Android Developers,” *Android Developers*, 2015. [Online]. Available: http://developer.android.com/about/index.html.

[18] C. Dominic, E. Tyrone, C. Shaun, and W. Ollie, *The Mobile Application Hacker’s Handbook*, 1st ed. Indianapolis: John Wiley & Sons, Inc., 2015.

[19] “7-Zip.” [Online]. Available: http://www.7-zip.org/. [Accessed: 08-Feb-2018].

[20] “WinRAR.” [Online]. Available: https://www.winrar.es/descargas. [Accessed: 17-Mar-2018].

[21] W. Shin, S. Kiyomoto, K. Fukushima, and T. Tanaka, “Towards formal analysis of the permission-based security model for Android,” in *5th International Conference on Wireless and Mobile Communications, ICWMC 2009*, 2009, pp. 87–92.

[22] J. Burns, “Mobile Application Security on Android: Context on Android security,” *BlackHat US*, pp. 1–27, 2009.

[23] R. Meier, *Professional Android 4 Application Development*. 2012.

[24] M. L. Murphy, *Beginning android*. 2009.

[25] S. Komatineni and D. MacLean, *Pro android 4*, vol. 9781430240. 2012.

[26] L. Darcey, *Sams Teach Yourself Android Application Development in 24 Hours*. 2010.

[27] J. Simon, *Head First Android Development*. 2012.

[28] S. Mlot, “Google’s Bouncer Malware Tool Hacked.,” *PC Mag.*, p. 1, 2012.

[29] “jarsigner.” [Online]. Available: https://docs.oracle.com/javase/7/docs/technotes/tools/windows/jarsigner.html. [Accessed: 16-Feb-2018].

[30] “OWASP Mobile Security Testing Guide - OWASP.” [Online]. Available: https://www.owasp.org/index.php/OWASP_Mobile_Security_Testing_Guide. [Accessed: 27-Feb-2018].

[31] Ousterhout John, “HTTP and HTTPS,” 2010. [Online]. Available: https://web.stanford.edu/~ouster/cgi-bin/cs142-fall10/lecture.php?topic=http. [Accessed: 27-Feb-2018].

[32] “iOS - Apple.” [Online]. Available: https://www.apple.com/la/ios/ios-11/. [Accessed: 27-Feb-2018].

[33] “HTML5 Introduction.” [Online]. Available: https://www.w3schools.com/html/html5_intro.asp. [Accessed: 27-Feb-2018].

[34] I. S. CRUZ, “USOS Y TIPOS DE APLICACIONES MÓVILES.” Tecnológico Nacional de México Instituto tecnológico de Salina Cruz, 2015.

[35] A. Shabtai, Y. Fledel, U. Kanonov, Y. Elovici, S. Dolev, and C. Glezer, “Google android: A comprehensive security assessment,” *IEEE Secur. Priv.*, vol. 8, no. 2, pp. 35–44, 2010.

[36] M. Costa, M. Castro, L. Zhou, L. Zhang, and M. Peinado, “Bouncer: Securing Software by Blocking Bad Input,” *Proc. twenty-first ACM SIGOPS Symp. Oper. Syst. Princ. - SOSP ’07*, vol. 41, no. 6, p. 117, 2007.

[37] Xuxian Jiang, “An Evaluation of the Application Verification Service in Android 4.2,” *Google’s App Verif. Serv. (in Android 4.2) An Overv.*, 2012.

[38] “Android – Google Play Protect.” [Online]. Available: https://www.android.com/play-protect/. [Accessed: 01-Dec-2017].

[39] N. Elenkov, *Android Security Internals*, vol. 2015, no. 6. 2015.

[40] “Mobile Security Project Archive - OWASP,” 2016. [Online]. Available:

https://www.owasp.org/index.php/Mobile_Security_Project_Archive#tab=M-Security_Testing. [Accessed: 01-Dec-2017].

[41] “VM VirtualBox |.” [Online]. Available: https://www.virtualbox.org/. [Accessed: 20-Feb-2018].

[42] “Create and Manage Virtual Devices | Android Studio.” [Online]. Available: https://developer.android.com/studio/run/managing-avds.html. [Accessed: 09-Feb-2018].

[43] “Android.” [Online]. Available: https://www.android.com/. [Accessed: 19-Feb-2018].

[44] “Google Play.” [Online]. Available: https://play.google.com/store. [Accessed: 12-Feb-2018].

[45] “Porting Android to x86.” [Online]. Available: http://www.android-x86.org/download. [Accessed: 28-Jan-2018].

[46] “Android x86.” [Online]. Available: https://code.google.com/archive/p/android-x86/downloads. [Accessed: 20-Feb-2018].

[47] “VMware – Official Site.” [Online]. Available: https://www.vmware.com/. [Accessed: 09-Feb-2018].

[48] “Create and Manage Virtual Devices | Android Studio.” [Online]. Available: https://developer.android.com/studio/run/managing-avds.html. [Accessed: 28-Jan-2018].

[49] “Android Studio y SDK Tools.” [Online]. Available: https://developer.android.com/studio/index.html?hl=es-419. [Accessed: 12-Feb-2018].

[50] S. U. Wenliang Du, “Android Repackaging Attack Lab,” 2016.

[51] H. Darwish and M. Husain, “Security Analysis of Mobile Money Applications on Android.”

[52] “Android File Managers.” [Online]. Available:

https://blogs.umass.edu/Techbytes/2014/12/05/android-file-managers-2/. [Accessed: 21-Feb-2018].

[53] “Building and Running from the Command Line.” [Online]. Available: https://stuff.mit.edu/afs/sipb/project/android/docs/tools/building/building-cmdline.html. [Accessed: 21-Feb-2018].

[54] “APKPure.com.” [Online]. Available: https://apkpure.com/es/. [Accessed: 28-Jan-2018].

[55] “APK Extractor - Aplicaciones Android en Google Play.” [Online]. Available: https://play.google.com/store/apps/details?id=com.ext.ui&hl=es_419. [Accessed: 28-Jan-2018].

[56] “ES File Explorer File Manager - Aplicaciones Android en Google Play.” [Online]. Available: https://play.google.com/store/apps/details?id=com.estrongs.android.pop&hl=es_419. [Accessed: 28-Jan-2018].

[57] “Download Android Studio and SDK Tools | Android Studio.” [Online]. Available: https://developer.android.com/studio/index.html. [Accessed: 28-Jan-2018].

[58] “Android Debug Bridge | Android Studio.” [Online]. Available: https://developer.android.com/studio/command-line/adb.html?hl=es-419. [Accessed: 28-Jan-2018].

[59] “PeaZip | Free archiver, free RAR TAR ZIP files utility.” [Online]. Available: http://www.peazip.org/. [Accessed: 20-Feb-2018].

[60] “AXMLPrinter2.” [Online]. Available: https://code.google.com/archive/p/android4me/downloads. [Accessed: 08-Feb-2018].

[61] S. Colley, *The Mobile Application Hacker Handbook*. Wiley.

[62] “AXMLPrinter2.” [Online]. Available: https://code.google.com/archive/p/android4me/downloads. [Accessed: 28-Jan-2018].

[63] Y. Tian, “Android Analysis Tools.”

[64] “Android Development Environment.”

[65] “ApkAnalyser.” [Online]. Available: https://github.com/sonyxperiadev/ApkAnalyser. [Accessed: 28-Jan-2018].

[66] “Logcat Command-line Tool | Android Studio.” [Online]. Available: https://developer.android.com/studio/command-line/logcat.html. [Accessed: 13-Feb-2018].

[67] “Apktool - A tool for reverse engineering 3rd party, closed, binary Android apps.” [Online]. Available: https://ibotpeaches.github.io/Apktool/. [Accessed: 13-Feb-2018].

[68] “Dex2jar.” [Online]. Available: https://github.com/pxb1988/dex2jar/blob/2.x/README.md. [Accessed: 28-Jan-2018].

[69] “JD-Gui.” [Online]. Available: https://github.com/java-decompiler/jd-gui/releases. [Accessed: 28-Jan-2018].

[70] “Análisis de archivos APK con MobSF.” [Online]. Available: https://www.welivesecurity.com/la-es/2016/12/19/analizar-apk-con-mobsf/. [Accessed: 21-Feb-2018].

[71] “Análisis de APK con AppMon.” [Online]. Available: https://www.welivesecurity.com/la-es/2016/09/30/analisis-de-apk-appmon/. [Accessed: 21-Feb-2018].

[72] “Construyendo un Laboratorio de Análisis de Aplicaciones Android.,” 2015.

[73] “MobSF.” [Online]. Available: https://github.com/MobSF/Mobile-Security-Framework-MobSF. [Accessed: 28-Jan-2018].

[74] “XML External Entity (XXE) Processing - OWASP.” [Online]. Available: https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Processing. [Accessed: 21-Feb-2018].

[75] “Server Side Request Forgery - OWASP.” [Online]. Available:

https://www.owasp.org/index.php/Server_Side_Request_Forgery. [Accessed: 21-Feb-2018].

[76] “AppMon.” [Online]. Available: https://dpnishant.github.io/appmon/. [Accessed: 21-Feb-2018].

[77] “Androwarn.” [Online]. Available: https://github.com/maaaaz/androwarn. [Accessed: 28-Jan-2018].

[78] “Pentest Magazine The Hackers Mobile Application Penetration Testing Arsenal | Mohit Sahu - Academia.edu,” 2016. [Online]. Available: http://www.academia.edu/26958346/Pentest_Magazine_The_Hackers_Mobile_Application_Penetration_Testing_Arsenal. [Accessed: 21-Feb-2018].

[79] V. Javier Mozos Pérez, J. Friginal López, and J. Carlos Ruiz García DISCA-UPV, “A study of vulnerabilities on Android systems,” 2013.

[80] “AndroL4b.” [Online]. Available: https://github.com/sh4hin/Androl4b. [Accessed: 28-Jan-2018].

[81] “AppUse.” [Online]. Available: https://appsec-labs.com/appuse-pro/. [Accessed: 28-Jan-2018].

[82] “The Cobra Den | Mobile Security and Research.” [Online]. Available: https://thecobraden.com/projects/cobradroid/. [Accessed: 28-Jan-2018].

[83] “Drozer.” [Online]. Available: https://labs.mwrinfosecurity.com/tools/drozer/. [Accessed: 28-Jan-2018].

[84] “OWASP Zed Attack Proxy Project - OWASP.” [Online]. Available: https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project. [Accessed: 28-Jan-2018].

[85] “Sublime Text.” [Online]. Available: https://www.sublimetext.com/. [Accessed: 23-Feb-2018].

[86] “Permissions Overview | Android Developers.” [Online]. Available: https://developer.android.com/guide/topics/permissions/overview.html. [Accessed: 24-Feb-2018].

[87] “How to know if android app was programmed in a framework such as ionic, cordova or phonegap - Quora.” [Online]. Available: https://www.quora.com/How-do-I-know-if-android-app-was-programmed-in-a-framework-such-as-ionic-cordova-or-phonegap. [Accessed: 27-Feb-2018].

[88] P. J. C. N.-B. L. ; C. ; N. L. of S. ; O. ; Q. M. Deitel University of London ; Trinity College Dublin, *Android for programmers : an app-driven approach*. 2012.

[89] “Apache Cordova.” [Online]. Available: https://cordova.apache.org/. [Accessed: 28-Feb-2018].

[90] “Ionic Framework.” [Online]. Available: https://ionicframework.com/. [Accessed: 28-Feb-2018].

[91] Y. Zhauniarovich, A. Philippov, O. Gadyatskaya, B. Crispo, and F. Massacci, “Towards Black Box Testing of Android Apps.”

[92] “NVD - Search and Statistics.” [Online]. Available: https://nvd.nist.gov/vuln/search. [Accessed: 28-Feb-2018].

[93] “CVE security vulnerability database. Security vulnerabilities, exploits, references and more.” [Online]. Available: https://www.cvedetails.com/. [Accessed: 28-Feb-2018].

[94] “Exploits Database by Offensive Security.” [Online]. Available: https://www.exploit-db.com/. [Accessed: 28-Feb-2018].

[95] “US-CERT | United States Computer Emergency Readiness Team.” [Online]. Available: https://www.us-cert.gov/. [Accessed: 28-Feb-2018].

[96] N. Godfrey, *Decompiling Android*, 1st ed. New York: Apress, 2012.

[97] “URL independientes | Search | Google Developers.” [Online]. Available: https://developers.google.com/search/mobile-sites/mobile-seo/separate-urls. [Accessed: 01-Mar-2018].

[98] “SECURING THE MOBILE BANKING CHANNEL The mobile opportunity.”

[99] “Cómo evitar errores de desarrollo en Android con PidCat.” [Online].

Available: https://www.welivesecurity.com/la-es/2016/06/28/auditar-aplicaciones-android-pidcat/. [Accessed: 28-Feb-2018].

[100] “Risk Analysis of Android Based Appliance - Checkmate.” [Online]. Available: http://niiconsulting.com/checkmate/2011/11/risk-analysis-of-android-based-appliance/. [Accessed: 28-Feb-2018].

[101] “Java JDK - Oracle Technology Network | Oracle.” [Online]. Available: http://www.oracle.com/technetwork/java/javase/downloads/index.html. [Accessed: 17-Mar-2018].

[102] “Create and Manage Virtual Devices | Android Studio.” [Online]. Available: https://developer.android.com/studio/run/managing-avds.html. [Accessed: 17-Mar-2018].

[103] “Java SE Development Kit.” [Online]. Available: http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html. [Accessed: 08-Feb-2018].

[104] “OpenSSL for Windows.” [Online]. Available: http://gnuwin32.sourceforge.net/packages/openssl.htm. [Accessed: 08-Feb-2018].

[105] “JD-GUI.” [Online]. Available: http://jd.benow.ca/. [Accessed: 08-Feb-2018].

[106] “MobSF.” [Online]. Available: https://github.com/MobSF/Mobile-Security-Framework-MobSF/wiki/1.-Documentation. [Accessed: 08-Feb-2018].

[107] “OWASP Zed Attack Proxy Project - OWASP.” [Online]. Available: https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project. [Accessed: 08-Feb-2018].

[108] “Python.org.” [Online]. Available: https://www.python.org/. [Accessed: 08-Feb-2018].

[109] “PidCat.” [Online]. Available: https://github.com/JakeWharton/pidcat. [Accessed: 08-Feb-2018].

[110] “Manitree.” [Online]. Available: https://github.com/antitree/manitree. [Accessed: 08-Feb-2018].

[111] “Drozer.” [Online]. Available: https://labs.mwrinfosecurity.com/tools/drozer/. [Accessed: 08-Feb-2018].

ANEXOS

ANEXO A
PROCESO DE REGISTRO EN BANCA ELECTRÓNICA

Figura A.1 Página web de ingreso para banca electrónica

Tomando como base que ya se dispone de una cuenta abierta en el Banco B, la misma que se encuentra habilitada para la realización de transacciones mediante el uso de banca electrónica, se procede a la habilitación de la misma mediante la página web oficial del banco. Para este punto es necesario que las aplicaciones bancarias A, B y C siguen un procedimiento similar

Figura A.2 Fase de verificación de cliente

Figura A.3 Fase de registro de usuario

En primer lugar, como se muestra en la Figura A.1, se presenta la pantalla de autenticación, en la cual se solicita el ingreso del nombre de usuario. Como aún no se ha registrado en el sistema, se selecciona la opción “Regístrese aquí”.

A continuación, como se muestra en la Figura A.2, se solicitan datos de la cuenta bancaria, así como la clave de la tarjeta. Para el ingreso de la clave de la tarjeta, se hace uso de un teclado en pantalla, con el objetivo de evitar que la clave sea capturada mediante algún *keylogger*.

Figura A.4 Fase de selección de imagen

Posteriormente, se elige un nombre de usuario, el cual está basado en el nombre del cliente o a su vez, se puede ingresar uno que el cliente decida. Esto se muestra en la Figura A.3.

A continuación, como medida adicional de seguridad se selecciona una imagen por parte del usuario, esta imagen será mostrada cada vez que el usario ingrese a banca electrónica para verificar que la página mostrada es auténtica. Esto brinda protección contra *phishing*. Esto se muestra en la Figura A.4.

Posteriormente, el cliente debe seleccionar al menos tres de las múltiples preguntas de seguridad que se ofrecen en la Figura A.5, con sus respectivas respuestas.

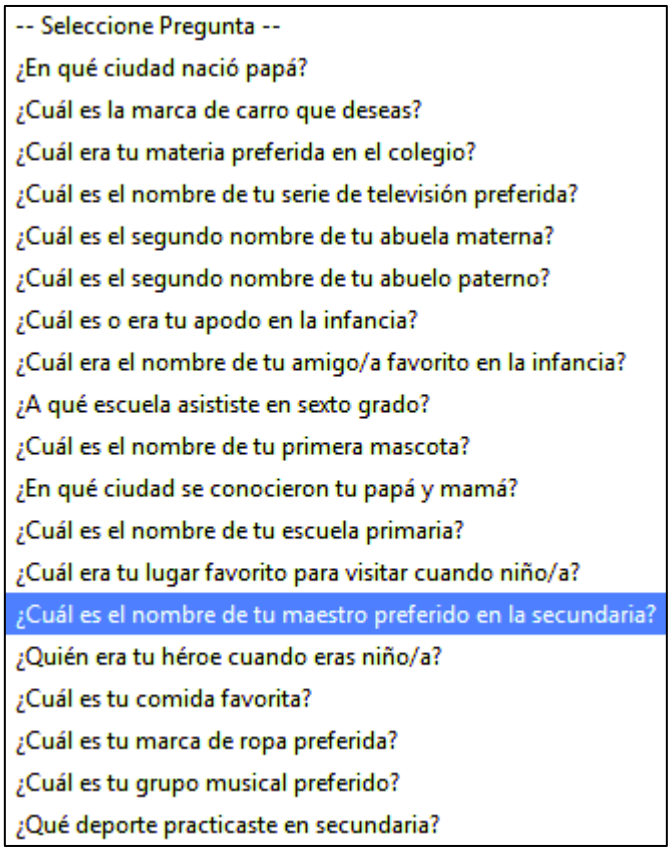


Figura A.5 Fase de selección de preguntas

Como se muestra en la Figura A.6, al ingresar el nombre de usuario se muestra la imagen de seguridad.

Finalmente, Para el inicio de sesión se realiza la verificación mediante la contraseña, el nombre de usuario y la verificación de que la imagen seleccionada durante el proceso de registro sea la que aparece, así como su respectivo nombre.

Una vez introducidas las credenciales se registra el equipo desde donde se está accediendo a la banca electrónica y se confirma el país, además una de las preguntas de seguridad. En la página siguiente se solicita la verificación del uso de la conexión segura mediante https cada vez que el usuario quiera acceder a banca electrónica, como se muestra en la Figura A.7. Hecho esto, el usuario ya está listo para hacer uso de la banca electrónica y para instalar las aplicaciones móviles.

Figura A.6 Inicio de sesión posterior al registro

Figura A.7 Autenticación de usuario

ANEXO B
RESULTADOS ESPERADOS PARA LA EJECUCIÓN DEL PROTOCOLO DE PRUEBAS

Tabla B.1 Resultados esperados

Fases	Actividades	Resultados Esperados	Herramientas	Comandos
Recolección de Información	Funcionalidad y Flujo de Trabajo	Detalle de la funcionalidad de la aplicación.	Android Virtual Device	No aplica
	Interfaces de Red y Componentes de Hardware	Lista de interfaces de red y componentes de hardware con los que interactúa la aplicación.	Android Virtual Device	No aplica
	Transacciones Comerciales e Interacción con otras Aplicaciones	Conjunto de transacciones comerciales que permite realizar la aplicación.	Android Virtual Device	No aplica
	Firmado de la Aplicación	Nombre del desarrollador de la aplicación	Jarsigner	>jarsigner –verify –certs –verbose AppX.apk
		Datos del certificado usado para firmar la aplicación.	OpenSSL	>openssl pkcs7 -in CERT.RSA -print_certs -inform DER -out out.cer
Análisis Estático	Obtención del Código Fuente	Archivo .apk de la aplicación a analizar.	Apk Extractor	No aplica
		Contenido del archivo .apk	PeaZip	No aplica
		Archivo .jar a partir del archivo .dex.	Dex2jar	>d2j-dex2jar.bat C:\Users\josep\Desktop\classes.dex
	Permisos en el Archivo AndroidManifest.xml	Archivo AndroidManifest.xml en formato legible.	AXMLPrinter2	>java -jar AXMLPrinter2.jar AndroidManifest.xml > AndroidManifest_decoded_.xml

Fases	Actividades	Resultados Esperados	Herramientas	Comandos
Análisis Estático	*Framework*	Tipo de aplicación	Explorador de Archivos	No aplica
	Librerías	Librerías utilizadas por la aplicación.	Explorador de Archivos	No aplica
	Vistas	Archivos necesarios para generar las vistas de la aplicación.	Explorador de Archivos	No aplica
	Ingreso de Datos	Código para validación de datos.	JD-Gui	No aplica
		Logs del ingreso de datos.	Logcat	adb logcat
	Código para Autenticación	Código para el proceso de autenticación.	JD-Gui	No aplica
		Logs del proceso de autenticación.	Logcat	adb logcat
	Autenticación *Online/Offline*	Código para el proceso de autenticación.	JD-Gui	No aplica
		Logs del proceso de autenticación.	Logcat	adb logcat
	Información Adicional al Usuario/ *Password*	Código que involucre información adicional para el proceso de autenticación.	JD-Gui	No aplica
		Logs con información adicional	Logcat	adb logcat
	Métodos de Autenticación	Tipos de autenticación que usa la aplicación.	JD-Gui	No aplica
	Bloqueo	Código para bloqueo de la aplicación.	JD-Gui	No aplica
		Logs del bloqueo de la aplicación.	Logcat	adb logcat

Fases	Actividades	Resultados Esperados	Herramientas	Comandos
	Autenticación Única	Código de autenticación única	JD-Gui	No aplica
Análisis Estático	Autenticación en Dos Pasos	Código para confirmación de la autenticación	JD-Gui	No aplica
		Logs de confirmación de autenticación.	Logcat	>adb logcat
	Análisis Estático Automatizado	Resultados de la evaluación automática de la aplicación.	MobSF	No aplica
Análisis Dinámico	Protocolos de Comunicación Utilizados	Protocolos que usa la aplicación.	Network Connections	No aplica
		Contenido de los paquetes generados.	OWASP ZAP	No aplica
	Scripts Existentes para la Búsqueda de Vulnerabilidades en Aplicaciones Android	Logs generados sólo por la aplicación analizada.	Pidcat	>C:\Python\python.exe pidcat.py com.nombre_del_paquete
		Puntos de posible vulnerabilidad en el archivo AndroidManifest.xml	Manitree	>C:\Python\python.exe manitree.py –f AndroidManifest.xml
	Proveedor de Contenido	Nombre completo del paquete.	Drozer	>run app.package.list –f nombre_aplicación
		Proveedor(es) de contenido exportado(s).		>run app.package.attacksurface nombre_del_paquete
		Configuración de los permisos del proveedor de contenido.		>run app.provider.info -a nombre_del_paquete
	Inyección desde el Lado del Cliente	URLs que contiene la aplicación.	Drozer	>run scanner.provider.finduris –a nombre_del_paquete

Fases	Actividades	Resultados Esperados	Herramientas	Comandos
Análisis Dinámico	Inyección desde el Lado del Cliente	Conocer si se pueden realizar peticiones a los URLs.	Drozer	>run app.provider.query uri
		Saber si los URLs son vulnerables al ataque SQL Injection.		>run scanner.provider.injection –a nombre_del_paquete

ANEXO C

ARCHIVO ANDROIDMANIFEST.XML EN FORMATO LEGIBLE

C.1 ARCHIVO ANDROIDMANIFEST.XML DE LA APLICACIÓN BANCARIA A, EN FORMATO LEGIBLE.

El documento se encuentra adjunto como anexo digital.

C.2 ARCHIVO ANDROIDMANIFEST.XML DE LA APLICACIÓN BANCARIA B, EN FORMATO LEGIBLE.

El documento se encuentra adjunto como anexo digital.

C.3 ARCHIVO ANDROIDMANIFEST.XML DE LA APLICACIÓN BANCARIA C, EN FORMATO LEGIBLE.

El documento se encuentra adjunto como anexo digital.

ANEXO D
ANÁLISIS ESTÁTICO AUTOMATIZADO

D.1 ANÁLISIS ESTÁTICO AUTOMATIZADO DE LA APLICACIÓN A

El documento se encuentra adjunto como anexo digital.

D.2 ANÁLISIS ESTÁTICO AUTOMATIZADO DE LA APLICACIÓN B

El documento se encuentra adjunto como anexo digital.

D.3 ANÁLISIS ESTÁTICO AUTOMATIZADO DE LA APLICACIÓN C

El documento se encuentra adjunto como anexo digital.

ANEXO E
EJECUCIÓN DEL PROTOCOLO PARA LAS APLICACIONES B Y C

E.1 EJECUCIÓN DEL PROTOCOLO PARA LA APLICACIÓN B

E.1.1 Recolección De Información

Para desarrollar la fase de recopilación de información de la aplicación B, se instala la aplicación en un dispositivo Android virtual. La aplicación se descarga desde la Play Store y los pasos necesarios para su análisis se presentan a continuación.

E.1.1.1 Funcionalidad y Flujo de Trabajo de la Aplicación

Figura E.1 Pantalla inicial de la aplicación móvil B

A diferencia de la aplicación A, la aplicación móvil B siempre realiza el mismo proceso de autenticación desde la primera vez. Para la autenticación del usuario se utilizan las mismas credenciales que fueron registradas anteriormente en la banca electrónica, es decir, un nombre de usuario y una contraseña.

Durante este proceso, tanto si se ha dado de manera exitosa como si ha ocurrido un problema, no se envía un mensaje de texto al número de teléfono del usuario. En cualquier caso, se envía un correo electrónico a la dirección registrada indicando que se intenta autenticarse. La Figura E.1 muestra la pantalla presentada al abrir la aplicación, esta muestra imágenes promocionales de la entidad bancaria, ubicación

de agencias y cajeros, información de contacto y la opción de compartir la aplicación.

Tanto para la opción de Agencias y Cajeros como para la de Promociones la aplicación utiliza la información de ubicación del dispositivo para ofrecer la información adecuada. Para la opción de Compartir, la aplicación permite enviar un enlace mediante redes sociales, para poder descargar la aplicación.

E.1.1.2 Interfaces de Red y Componentes de Hardware

La aplicación solicita varios permisos para su instalación, tanto el acceso a componentes de hardware como los demás permisos requeridos por la aplicación para su funcionamiento se muestran en la Figura E.2. Además, estos permisos también se encuentran detallados en el archivo `AndroidManifest.xml`.

Figura E.2 Permisos solicitados por la aplicación B

Esta aplicación, usa las siguientes interfaces de red:

- Comunicación móvil.
- Comunicación WLAN Wi-Fi.

Los componentes de hardware con los que la aplicación interactúa son:

- GPS.
- Almacenamiento del dispositivo.
- Cámara.

Además de esto, la aplicación utiliza los datos como la identidad y el identificador de dispositivo.

E.1.1.3 Transacciones Comerciales e Interacción con Otras Aplicaciones

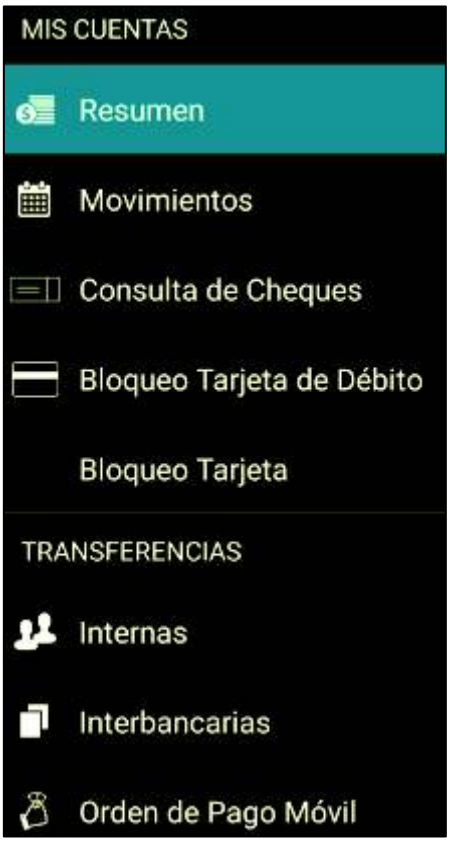


Figura E.3 Parte de las transacciones que permite realizar la aplicación B

La imagen E.3 muestra un extracto de las transacciones que permita realizar la aplicación B. Además de la información acerca del estado de cuenta, la aplicación permite la realización de transacciones comerciales como:

- Transferencias directas e interbancarias.
- Pago de tarjetas y servicios básicos.
- Consulta de préstamos.
- Bloqueo de tarjetas.
- Orden de pago móvil.
- Recargas.

Para realizar las transacciones, se requiere un código de confirmación el cual es enviado al correo electrónico del usuario. Por ejemplo, para el caso de una transferencia directa se requiere, además de los números de cuenta origen y destino, el código de confirmación de cuatro dígitos enviado al correo del usuario.

Figura E.4 Resultado de Jarsigner para la aplicación B

Mediante la exploración inicial de la aplicación móvil B, se ha encontrado que esta interactúa con las aplicaciones de redes sociales (Messenger, WhatsApp, Gmail, etc.), para enviar un enlace invitando a descargar la aplicación B desde la tienda oficial de Google.

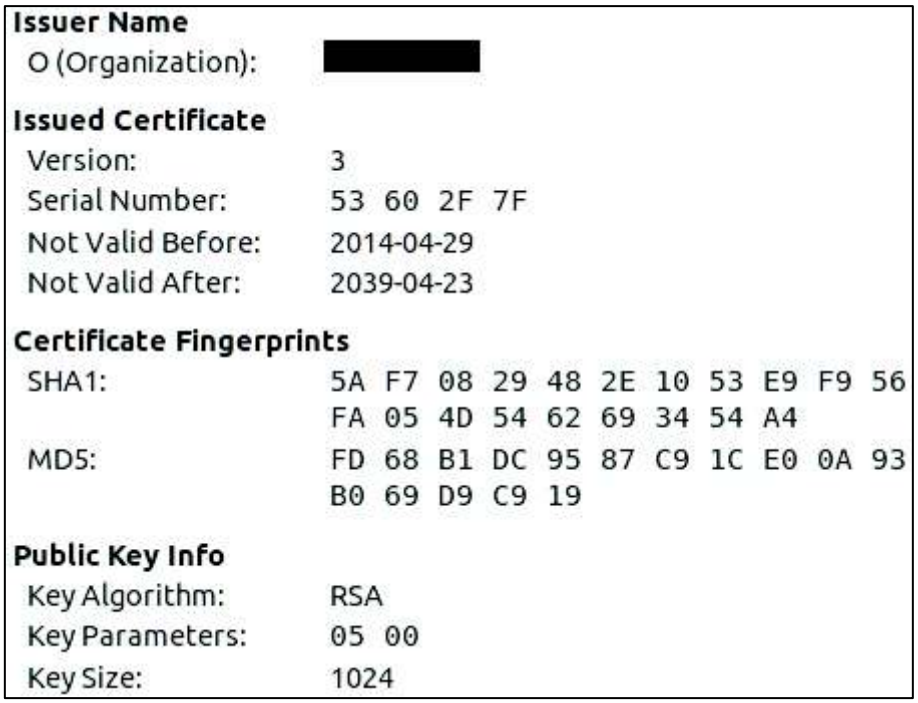


Figura E.5 Información del archivo .cer para la aplicación B

E.1.1.4 Firmado de la Aplicación

Gracias a la herramienta Jarsigner se obtiene información acerca del certificado que se utilizó para firmar la aplicación. Como se ve en la Figura E.4, para la aplicación B la información que se obtiene es la del campo OU (*Organizational Unit*). Para este caso el nombre de este campo corresponde al de la empresa que ha desarrollado la aplicación.

A continuación, se utiliza la herramienta OpenSSL para procesar el archivo .RSA contenido en la carpeta `META-INF`, con el objetivo de obtener información adicional. Como se muestra en la Figura E.5, la información obtenida incluye, adicionalmente a la conseguida con Jarsigner, el algoritmo utilizado (RSA), sumas de verificación en SHA1 y MD5, etc.

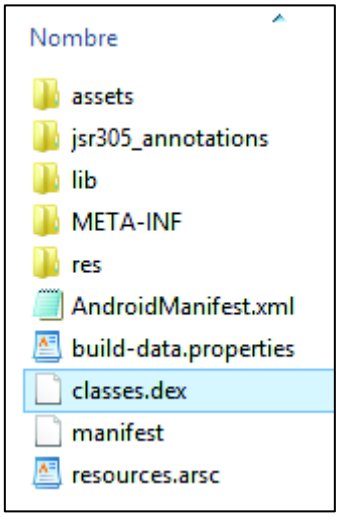


Figura E.6 Contenido del archivo .apk de la aplicación B

E.1.2 ANÁLISIS ESTÁTICO

A continuación se realiza la ejecución del protocolo para análisis estático de la aplicación B, de acuerdo con el procedimiento descrito en la sección 2.6. Para esto, se ha instalado la aplicación B en un dispositivo virtual Android.

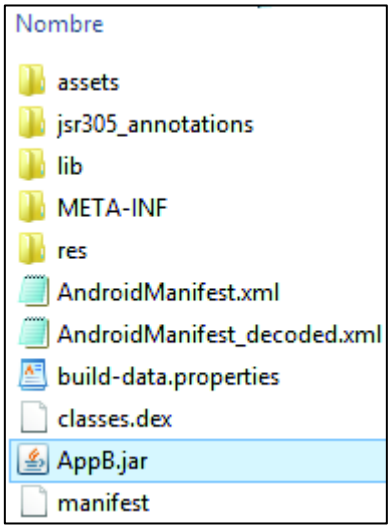


Figura E.7 Archivo .jar de la Aplicación B

E.1.2.1 Obtención del Código Fuente

Siguiendo el procedimiento descrito en la sección 2.6.1, mediante la herramienta Apk Extractor se obtiene el archivo .apk de la aplicación B. Como se muestra en la Figura E.6.

Luego de utilizar la herramienta Dex2jar sobre el archivo `classes.dex` , el cual se muestra en la Figura E.6 , se obtiene el archivo .jar como se muestra en la Figura E.7.

E.1.2.2 Permisos en el Archivo AndroidManifest.xml

Utilizando el procedimiento de la sección 2.6.2, en el cual se utiliza la herramienta AXMLPrinter2 para obtener el archivo `AndroidManifest.xml` en un formato legible, se pueden buscar dentro del archivo resultante, los permisos que utiliza la aplicación bancaria B. A continuación se lista un extracto de los permisos más relentes, encontrados a partir de la etiqueta *<uses-permission>*:

- CAMERA
- WRITE SETTINGS
- INTERNET
- GET_ACCOUNTS
- WAKE_LOCK
- NETWORK_STATE
- FINE_LOCATION
- COARSE_LOCATION
- WRITE_EXTERNAL_STORAGE
- VIBRATE
- READ_PHONE_STATE

El contenido completo del archivo `AndroidManifest.xml` para la aplicación bancaria C, en formato legible, se encuentra en el ANEXO C.2.

E.1.2.3 Framework

De acuerdo con el procedimiento de la sección 2.6.3, se explora el contenido de la carpeta `assets`. Como se muestra en la Figura E.8, esta carpeta no contiene

información acerca del *framework* en el cual fue desarrollado. Además, en la Figura E.6, en la cual se muestra el contenido del archivo .apk de la aplicación B, no se encuentra una carpeta `src`, que pueda brindar información acerca del *framework*. Por lo tanto, para esta aplicación no ha sido posible determinar el *framework*.

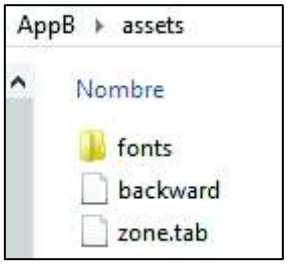


Figura E.8 Contenido de la carpeta assets de la aplicación B

E.1.2.4 Librerías

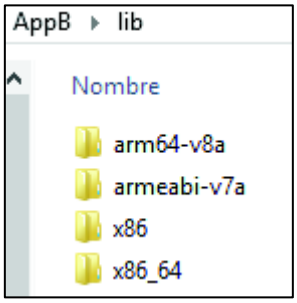


Figura E.9 Contenido de la carpeta lib de la aplicación B

De acuerdo con el procedimiento de la sección 2.6.4, se explora el contenido de la carpeta `lib`. Como se muestra en la Figura E.9, esta carpeta contiene varias carpetas dependiendo de la arquitectura del procesador.

Figura E.10 Resultados obtenidos para una de las librerías de la aplicación B

Al examinar el contenido de cada una de las carpetas dentro de la carpeta `lib`, se ha encontrado sólo una librería:

- libFPhi.Extractor.so

La Figura E.10 muestra los resultados de la búsqueda de esta librería en una de las fuentes recomendadas en la sección 2.6.4. Como se observa, no existen vulnerabilidades reportadas para esta librería.

E.1.2.5 Vistas

De acuerdo con el procedimiento de la sección 2.6.5, dado que la aplicación cuenta con un archivo .apk, se ha explorado el contenido de la carpeta `res`. Como se ve en la Figura E.11, la carpeta res contiene los archivos para generar las vistas de la aplicación. Por lo tanto, se podría crear una aplicación maliciosa de aspecto similar a la aplicación B.

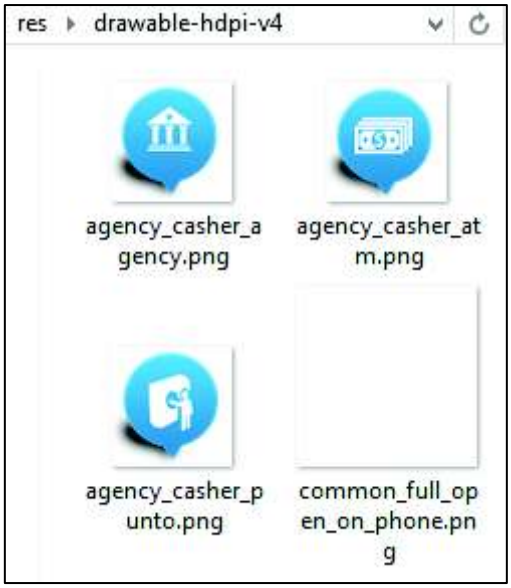


Figura E.11 Extracto del contenido de la carpeta res de la aplicación B

E.1.2.6 Ingreso de Datos

De acuerdo con la sección 2.6.6, se analiza el proceso de ingreso de datos utilizando las herramientas JD-Gui y Logcat mientras se ejecuta la aplicación en un dispositivo virtual Android. Considerando el proceso de una transferencia electrónica, se han obtenido los logs mostrados en la Figura E.12. Como se observa en la Figura E.12, esta aplicación utiliza la clase

`network.TransferRequest`. Además, durante el proceso de transferencia se observa que también se incluye el nombre de usuario que se utiliza para autenticarse, los números de cuenta origen y destino, y el monto de la transferencia.

```
1   04-23 21:03:23.632 D//RequestProcessor.java:66( 2586): 21:03:23.631 SpiceManagerThread
    0 Adding request to queue 119515429: CachedSpiceRequest [
    requestCacheKey=internal.transfer.nombre_usuario.#cuenta_origen.#cuenta_destino.monto,
    cacheDuration=30000, spiceRequest=com.appb.appb.network.TransferRequest@3a686fe1] size
    is 0
```

Figura E.12 Extracto de los logs generados por la aplicación B durante una transferencia directa

Para la aplicación B se ha encontrado que la validación de datos de una transferencia se realiza al momento de registrar la cuenta destino en la página web de la entidad bancaria. Ya que, para poder realizar una transferencia, sólo es posible realizarla a una cuenta previamente registrada.

Como se muestra en el Código E.1, en la clase `TransferRequest` se encuentra el método `createCacheKey`, el cual utiliza los datos detectados en los logs de la Figura E.12.

```
1   public String createCacheKey()
2   {
3     TransferBody.Transfer localTransfer = this.body.getTransfer();
4     return "internal.transfer." + localTransfer.getUsername() + "." +
      localTransfer.getDebitAccount().getNumber() + "." +
      localTransfer.getCreditAccount().getNumber() + "." +
      localTransfer.getDebitAmount();
5   }
```

Código E.1 Extracto de código de la clase TransferRequest

Como se indica en los logs de la Figura E.13, para completar la transferencia (línea 1), la aplicación B solicita un código de uso único (OTP), esto se evidencia en la Figura E.13 (línea 2).

```
1   04-23 20:59:25.619 I/ActivityManager( 1486): Displayed com.appb.appb/.TransferDetails:
    +475ms
2   04-23 20:59:25.643 D//SpiceServiceListenerNotifier.java:175( 2586): 20:59:25.643 main
    Processing request added: CachedSpiceRequest [requestCacheKey=null, cacheDuration=0,
    spiceRequest=com.appb.appb.network.OTPRequest@XXXXXXXX]
```

Figura E.13 Logs de la solicitud del OTP

Finalmente, como se muestra en el Código E.2, para la solicitud del OTP (*One Time Password*), se utiliza el código de la clase `OTPRequest`, en la cual se llama a su vez a la clase `Router` para completar la sintaxis de la solicitud (línea 3).

```
1  public OTPRequest(OTPBody paramOTPBody)
2  {
3    super(Transfer.OTP.class, Router.class);
4    this.body = paramOTPBody;
5  }
```

Código E.2 Método OTPRequest

E.1.2.7 Código Utilizado para Autenticación

De acuerdo a lo definido en la sección 2.6.7, se analiza el proceso de autenticación de la aplicación B mediante las herramientas Logcat y JD-Gui. La aplicación B realiza la autenticación mediante el uso de un nombre de usuario y un *password*, los cuales deben ser previamente registrados en la Banca Electrónica. A diferencia de la aplicación A, la cual solicita usuario y contraseña sólo la primera vez y luego solamente un PIN de cuatro dígitos, esta aplicación solicita siempre el usuario y contraseña.

```
1  protected void onCreate(Bundle paramBundle)
2  {
3    this.facialRecognition.setOnClickListener(new View.OnClickListener()
4    {
5      public void onClick(View paramAnonymousView)
6      {
7        LoginOptions.this.startAuthenticationActivity();
8      }
9    });
10   this.userPassword.setOnClickListener(new View.OnClickListener()
11   {
12     public void onClick(View paramAnonymousView)
13     {
14       LoginOptions.this.startLoginActivity();
15     }
16   });
17 }
```

Código E.3 Opciones de autenticación de la aplicación B

Como se muestra en el Código E.3, la aplicación B brinda la opción de autenticarse mediante usuario y contraseña (`userPassword`) o a su vez utilizando reconocimiento facial (`facialRecognition`). Las opciones disponibles son controladas mediante la clase `LoginOptions`.

```
1  04-23 20:35:06.967 I/ActivityManager( 1486): Displayed com.appb.appb/.Login: +224ms
2  04-23 20:36:43.412 W/json    ( 2452): https://www.appb.com.ec/detectrd/
   image.htm?id=XXXXXX
3  04-23 20:36:43.617 I/ActivityManager( 1486): Displayed com.appb.appb/.Step2: +234ms
```

Figura E.14 Clases y URL identificadas en los logs generados por la aplicación B

El ingreso mediante reconocimiento facial está controlado por el método en la línea 7 del Código E.3, como se observa, esta opción utiliza el nombre de `AuthenticationActivity`. Para el ingreso mediante usuario y contraseña se usa el método mostrado en la línea 14 del Código E.3, esta opción recibe el nombre de `LoginActivity`.

```
1  public void onRequestSuccess(SecurityImage.KeyPath paramKeyPath)
2  {
3    SecurityImage._instance(paramKeyPath.detectidImage);
4    Customer._instance().setUsername(Login.this.username.getText().toString());
5    Login.this.getDialog().hide();
6    paramKeyPath = new Intent(Login.this, Step2.class);
7  }
```

Código E.4 Código para obtener la imagen de seguridad

Como se muestra en la línea 1 de la Figura E.14, una vez que se ha seleccionado la opción de utilizar el método de usuario/contraseña, se llama a la clase `Login`.

En la clase `Login`, cuando se ha ingresado correctamente el nombre se usuario, se utiliza el método `onRequestSuccess`, mediante el cual se obtiene la imagen de seguridad presentada al usuario. En la línea 2 de la Figura E.14 se muestra el log correspondiente para obtener la imagen de seguridad.

```
1  public void onRequestSuccess(Customer.KeyPath paramKeyPath)
2  {
3    Login.instance.finish();
4    paramKeyPath = new Intent(Step2.this, Main.class);
5    Step2.this.finish();
6  }
```

Código E.5 Método onRequestSuccess

Como se muestra en la línea 3 del Código E.4, se utiliza la clase SecurityImage para obtener la imagen de seguridad, posteriormente, como se indica en la línea 6 del mismo código, se llama a la clase Step2, lo cual fue detectado en la línea 3 de la Figura E.14. Una vez que se ha llamado a la clase `Step2` como muestra el Código E.4, cuando el proceso se ha desarrollado con éxito se da por finalizada la

autenticación como se muestra en la línea 2 del Código E.5**.** Finalmente, se llama a la clase `Main`.

E.1.2.8 Autenticación Online/Offline

Como se evidencia durante la fase de autenticación, la aplicación trabaja de manera *online*. Además, dentro de la clase `ApiService` se encuentra la URL con la cual se comunica la aplicación, esto se muestra en la línea 5 del Código E.6.

```
1  public class ApiService
2    extends RetrofitGsonSpiceService
3  {
4    private static final String API_FOLDER = "/bpapp-api";
5    private static final String BASE_URL = "https://www.appb.com.ec/appbapp-api";
6    public static final String HOST_API = "https://www.appb.com.ec";
```

Código E.6 URL con la cual se comunica la aplicación B

E.1.2.9 Información Adicional al Nombre de Usuario/Password

Utilizando el procedimiento desarrollado en la sección E.1.2.7, se ha determinado que la aplicación bancaria B no utiliza ningún tipo de identificador adicional, como identificador de dispositivo, identificador, etc. Esto se evidencia en la Figura E.14 y en el Código E.4.

E.1.2.10 Métodos de Autenticación

Mediante el procedimiento de la sección 2.6.10 se ha analizado el proceso de autenticación de la aplicación bancaria B. Como se muestra en el Código E.3, la aplicación B dispone de dos opciones para autenticación: la primera utilizando reconocimiento facial (línea 3) y la segunda mediante usuario/contraseña (línea 10).

E.1.2.11 Bloqueo

Utilizando el procedimiento de la sección 2.6.11, se analizan los logs generados mientras se ingresa erróneamente la contraseña de acceso repetidamente. De esta forma, la aplicación B se ha bloqueado luego del tercer intento en el cual se ingresó la contraseña incorrecta. Como se muestra en la Figura E.15, los logs generados durante el bloqueo de la aplicación contienen el fragmento de código `Writer response`, el cual se busca posteriormente en el código que compone la aplicación B.

Además, en la misma figura muestra que se realiza un desbloqueo automático de la aplicación luego de 24 horas.

```
04-23 22:51:49.007 I/System.out( 2586): Writer response: {"message":"El Usuario se encuentra actualmente bloqueado. Ingrese a BancoB, o espere 24 horas para el desbloqueo automÃ¡tico.","code":"0001"}
```

Figura E.15 Extracto de los logs generados por la aplicación B durante el ingreso erróneo de credenciales

Utilizando el fragmento de código `Writer response`, se ha logrado identificar la clase a la que pertenece, en este caso: `SystemPreferences`. En el código E.7, en la línea 10 se identifica la frase `Writer response`, y se observa que esta pertenece a una excepción y que se encarga de mostrar en un mensaje en pantalla los mensajes de error como el identificado en los logs de la Figura E.15.

```
1  catch (JSONException paramSpiceException)
2  {
3    showPopUpMessage(paramContext, localActivity.getResources().getString(2131034412),
     localActivity, paramProgressDialog, new OnOKClick()
4    {
5      public void OnClick() {}
6    });
7    return;
8    localObject = new StringWriter();
9    IOUtils.copy(paramSpiceException.getBody().in(), (Writer)localObject);
10   System.out.println("Writer response: " + localObject);
11   showPopUpMessage(paramContext, new JSONObject(((StringWriter)
     localObject).toString()).getString("message"), localActivity, paramProgressDialog,
     paramOnOKClick);
12   return;
13 }
```

Código E.7 Método para mostrar los mensajes recibidos

E.1.2.12 Autenticación Única

```
1  protected void onResume()
2  {
3    super.onResume();
4    this.mHandler.sendEmptyMessage(2);
5    this.mResumed = true;
6    this.mFragments.execPendingActions();
7  }
```

Código E.8 Método onResume.

Utilizando el procedimiento de la sección 2.6.12, se han analizado los logs generados por la aplicación bancaria B durante el proceso de autenticación, lo cual se realizó en la sección E.1.2.7. De este modo, en la clase `Main` se ha encontrado que, cuando la aplicación regresa a primer plano (método onResume), no existe

ningún tipo de restricción ni tampoco se solicita nuevamente el ingreso de credenciales. Esto se evidencia en el Código E.8.

Además, durante la exploración inicial de la aplicación, se ha encontrado que la aplicación vuelve a solicitar el ingreso de credenciales en un tiempo aproximado de 5 minutos, sin embargo, el código asociado a este proceso no se ha encontrado.

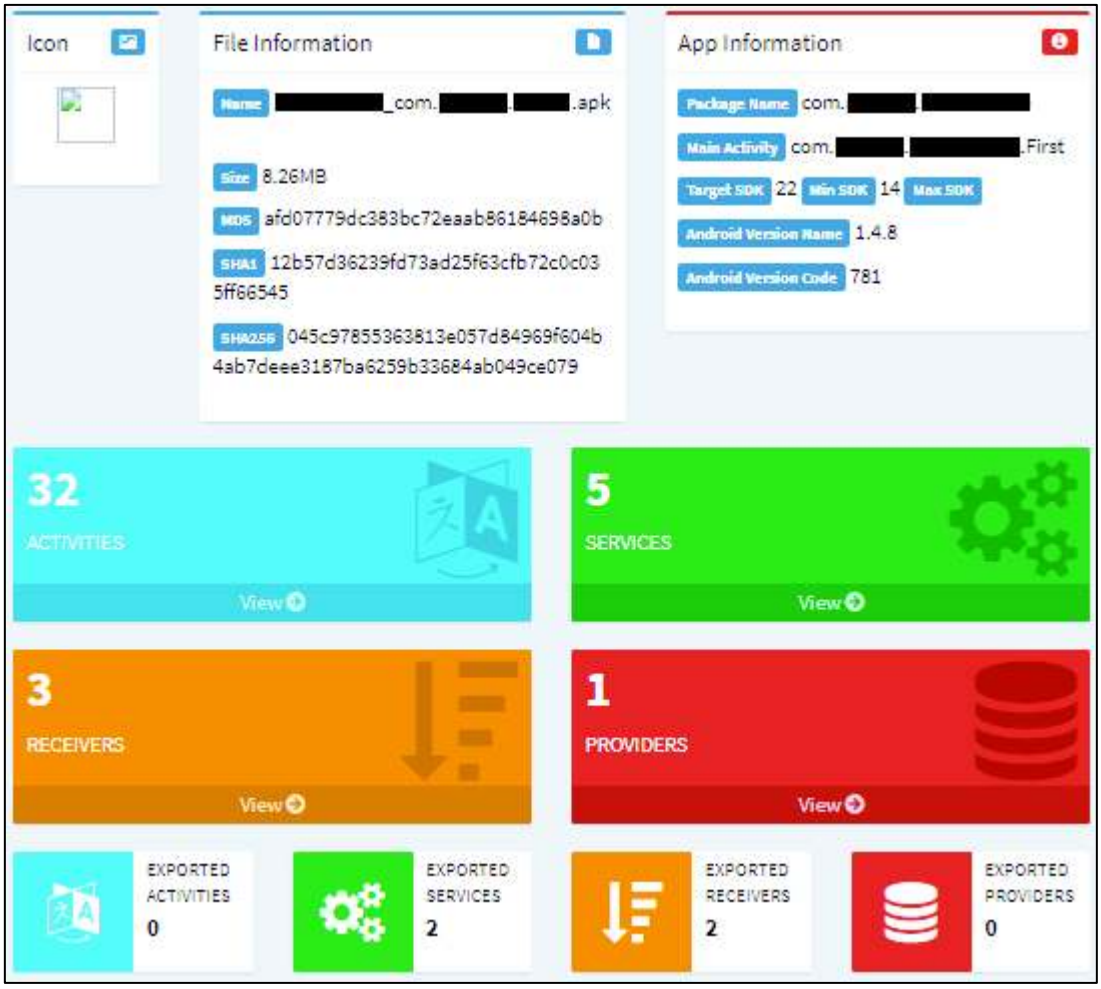


Figura E.16 Resultado de la herramienta MobSF para la aplicación B

E.1.2.13 Autenticación en dos Pasos

Mediante el análisis realizado durante el proceso de autenticación, en la sección E.1.2.7, se encuentra que la aplicación no envía mensajes de texto con algún tipo de código de confirmación como parte de la fase de autenticación. Excepto para el caso de las transferencias, cuyo procedimiento fue revisado a detalle en la sección E.1.2.7.

E.1.2.14 Análisis Estático Automatizado

Con base en el punto 2.6.14, se realiza el análisis estático automatizado de la aplicación bancaria B, para lo cual se utiliza la herramienta MobSF.

El extracto de los resultados obtenidos se muestra en la Figura E.16. Mediante la herramienta MobSF se ha encontrado que la aplicación bancaria B está diseñada para ejecutarse en dispositivos con una versión de Android a partir de *Ice Cream Sandwich* (API 14). Esta aplicación no utiliza proveedores de contenido que se encuentren exportados, lo cual se verifica a detalle en la sección E.1.3.3.

Al igual que la aplicación bancaria A, la aplicación bancaria B se encuentra firmada utilizando el algoritmo SHA1 con RSA, el cual tiene problemas de colisión.

Como extracto de los permisos que la herramienta MobSF califica como peligrosos se encuentran:

- Ubicación del dispositivo.
- Acceso al almacenamiento externo del dispositivo.
- Escritura de las configuraciones del dispositivo.
- Cámara.
- Estado del teléfono, etc.

En la sección *Code Analysis* del análisis realizado por la herramienta MobSF, se encuentra que la aplicación bancaria B utiliza la función Java Hash Code, la cual se considera como una función *Hash* débil y que no se recomienda su uso en implementaciones de seguridad.

Además, la aplicación B registra en logs información sensible, la cual no se recomienda que sea registrada en logs. Entre otro de los puntos cuya severidad se considera como alta se encuentra el uso de MD5 como función de Hash, de la cual se conoce que existen problemas de colisión.

El resultado detallado del análisis automatizado de la aplicación B se encuentra en el Anexo D.2.

E.1.3 ANÁLISIS DINÁMICO

Para el desarrollo de esta sección se utiliza el procedimiento de la sección 2.7. Para esto se utiliza un dispositivo físico Android físico para el punto E.1.3.1, mientras que para los otros tres puntos se utiliza un dispositivo Android virtual.

E.1.3.1 Protocolos de Comunicación Utilizados

Utilizando el procedimiento de la sección 2.7.1 se ejecuta la aplicación en el dispositivo físico, mientras se utilizan las herramientas Network Connections y OWASP ZAP.

- **Network Connections**

Figura E.17 Resultado de la herramienta Network Connections para la aplicación B

Realizando la captura del tráfico generado por la aplicación B se obtuvieron los resultados de la Figura E.18. En el resultado obtenido se encuentra la dirección IP con la que se comunica la aplicación bancaria B, así como su ubicación. Además, se encuentra que la aplicación B utiliza el puerto 443, correspondiente a HTTPS.

- **OWASP ZAP**

ID	Req. Tim...	Mét...	URL	C...	Reas...	...	Size Res...	Highe...
6	16/04/18 ...	GET	https://www.[redacted].co...	200	OK	4...	333 bytes	Bajo
21	16/04/18 ...	GET	https://www.[redacted].co...	404	Not F...	1...	104 bytes	
1	16/04/18 ...	GET	https://www.[redacted].co...	404	Not F...	4...	57 bytes	
17	16/04/18 ...	GET	https://www.[redacted].co...	200	OK	7...	125 bytes	Bajo
20	16/04/18 ...	GET	https://www.[redacted].co...	200	OK	6...	530 bytes	Bajo
23	16/04/18 ...	GET	https://www.[redacted].co...	200	OK	5...	269 bytes	Bajo

Figura E.18 Extracto del resultado de OWASP ZAP para la aplicación B

Utilizando el procedimiento de la sección 2.7.1 y mediante la herramienta OWASP ZAP, se realiza el análisis del tráfico generado por la aplicación bancaria B. Para esto se ha utilizado un dispositivo Android físico. En la Figura E.18 se muestran las URLs con las cuales se comunica la aplicación B. Como se observa en esta figura, se utiliza el protocolo HTTPS para todas las URLs.

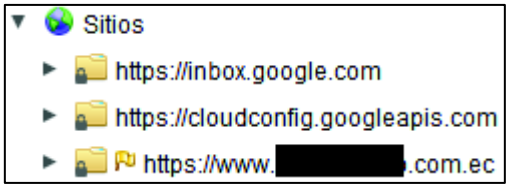


Figura E.19 URLs con las que se comunica la aplicación B

Como se muestra en la Figura E.19, la herramienta OWASP ZAP ha permitido identificar las URLs con las cuales se comunica la aplicación B.

Figura E.20 Extracto del contenido de los paquetes capturados, generados por la aplicación B

Analizando el contenido de los paquetes para cada una de las URLs de la Figura E.19, se encuentra en texto plano tanto el usuario como la contraseña de acceso. Esto se observa en la Figura E.20.

E.1.3.2 Scripts Existentes para la Búsqueda de Vulnerabilidades en Aplicaciones Android

De acuerdo a lo descrito en la sección 2.7.2, se utilizan los *scripts* Pidcat y Manitree como herramientas para el análisis de vulnerabilidades de la aplicación B. Utilizando estos *scripts*, se consiguió identificar URLs con las cuales se comunica la aplicación B, así como permisos potencialmente peligrosos.

- Pidcat

Mediante el *script* Pidcat, se ha conseguido filtrar los logs que va generando el dispositivo Android, de modo que solamente se visualicen aquellos generados por la aplicación B. Como se muestra en la Figura E.21, se ha conseguido identificar una de las URLs con la cual se comunica la aplicación B.

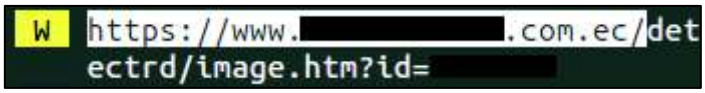


Figura E.21 Extracto de los resultados de la herramienta Pidcat para la aplicación B

- Manitree

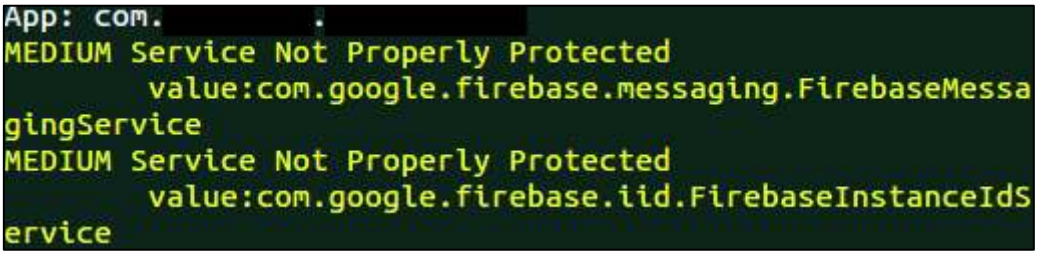


Figura E.22 Extracto de los resultados de la herramienta Manitree para la aplicación B

Utilizando el *script* Manitree para el análisis del archivo `AndroidManifest.xml` de la aplicación B, se han obtenido los resultados, cuyo extracto se muestra en la Figura E.22. Como se observa, existen dos puntos considerados como de media vulnerabilidad, los cuales mencionan servicios no protegidos adecuadamente.

E.1.3.3 Proveedor de Contenido

Utilizando el procedimiento de la sección 2.7.3, mediante la herramienta Drozer, se encuentra que la aplicación bancaria B no contiene ningún proveedor de contenido exportado. Como se muestra en la Figura E.23, Drozer indica que el número de proveedores de contenido exportados es cero.

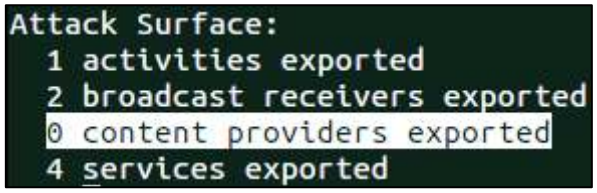


Figura E.23 Resultados de Drozer para la aplicación móvil B

E.1.3.4 Inyección desde el Lado del Cliente

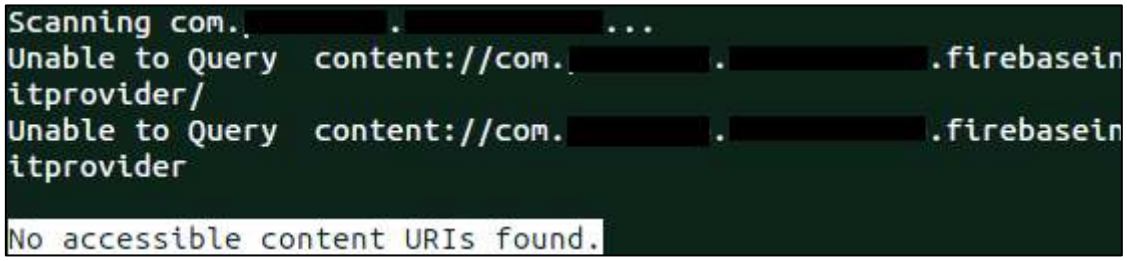


Figura E.24 Consulta a los URIs encontradas en la aplicación B

Mediante el procedimiento de la sección 2.7.4, utilizando la herramienta Drozer, se encuentra que la aplicación B no contiene URIs accesibles, como se muestra en la Figura E.24.

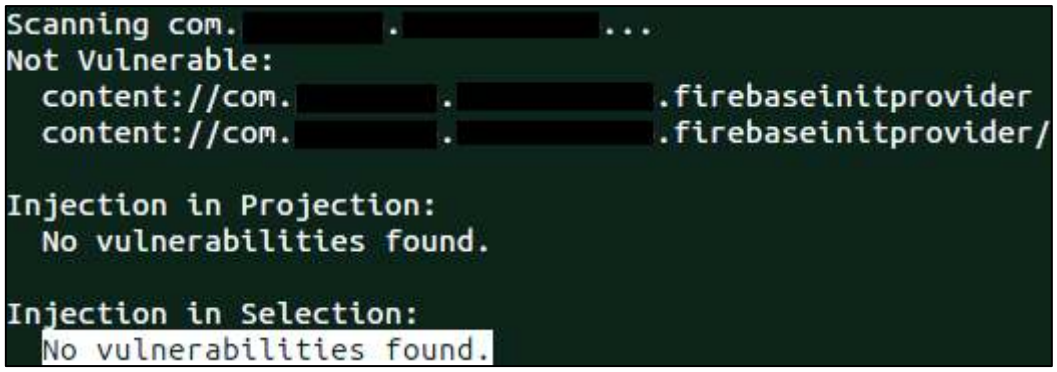


Figura E.25 Prueba de Inyección SQL para la aplicación B

Posteriormente, utilizando la herramienta Drozer, se comprueba si los URIs encontrados en la aplicación B son susceptibles al ataque de inyección SQL. Como se muestra en la Figura E.25, no existen vulnerabilidades de este tipo.

E.2 EJECUCIÓN DEL PROTOCOLO PARA LA APLICACIÓN C

E.2.1 RECOLECCIÓN DE INFORMACIÓN

Al igual que con las aplicaciones anteriores, la aplicación C se instala en un ambiente virtual Android. Esta aplicación también se ha descargado desde la tienda oficial y el procedimiento necesario para su análisis detalla a continuación.

E.2.1.1 Funcionalidad y Flujo de Trabajo de la Aplicación

Figura E.26 Pantalla inicial de la aplicación móvil C

Al igual que la aplicación B, la aplicación bancaria C realiza siempre el mismo proceso de autenticación desde la primera vez que se utiliza, es decir, emplea un nombre de usuario y una contraseña registrados previamente en la banca virtual.

Cada vez que el usuario accede a la aplicación se envía un correo electrónico a la dirección registrada por el usuario para indicar que el acceso se ha dado de manera exitosa o si ha ocurrido un error en el proceso. Como se muestra en la Figura E.26, al abrir la aplicación se tiene acceso a los puntos de atención, promociones, información de contacto, recomendar la aplicación y preguntas frecuentes.

Para obtener los puntos de atención la aplicación usa la ubicación del dispositivo, mientras que para las demás opciones solamente se despliega la información directamente o a su vez, se abre el navegador web para desplegar la información adicional (Promociones).

Además, esta aplicación permite compartir un mensaje mediante redes sociales, en el cual se invita a descargar la aplicación. A diferencia de la aplicación B, esta aplicación no comparte un enlace a la página, sólo el mensaje.

E.2.1.2 Interfaces de Red y Componentes de Hardware

Para su instalación, la aplicación C solicita permisos de acceso a los componentes de hardware del dispositivo, ubicación, además de otros, esto se muestra en la

Figura E.27. Los permisos están también detallados en el archivo `AndroidManifest.xml`.

Figura E.27 Permisos solicitados por la aplicación C

Esta aplicación, utiliza las siguientes interfaces de red:

- Comunicación WLAN Wi-Fi.

Los componentes de hardware con los que la aplicación interactúa son:

- GPS
- Almacenamiento del dispositivo.
- Cámara

E.2.1.3 Transacciones Comerciales e Interacción con Otras Aplicaciones

Adicional a la información acerca del estado de cuenta, la aplicación permite realizar transacciones como:

- Transferencias directas e interbancarias.
- Pago de servicios básicos.
- Recargas.
- Depósito de cheques.

Para realizar estas transacciones se requiere de un código de confirmación, el cual es enviado al correo electrónico del usuario. Además, para el caso de transferencias, se requiere confirmar previamente mediante otro código, el(los)

número(s) de cuenta destino. Mediante la exploración de la aplicación C, se ha encontrado que esta no interactúa con otras aplicaciones.

```
X.509, CN=Juan Carlos           , OU=Sistemas, O=Banco
, ST=Guayas, C=593
[certificate is valid from 2/25/13 4:03 PM to 2/19/38 4:03 PM]
```

Figura E.28 Resultado de Jarsigner para la aplicación C

E.2.1.4 Firmado de la Aplicación

Issuer Name

C (Country):	#1303353933
ST (State):	Guayas
L (Locality):	[redacted]
O (Organization):	Banco[redacted]
OU (Organizational Unit):	Sistemas
CN (Common Name):	Juan Carlos[redacted]

Issued Certificate

Version:	3
Serial Number:	51 2B D1 BC
Not Valid Before:	2013-02-25
Not Valid After:	2038-02-19

Certificate Fingerprints

SHA1:	71 EE 03 78 F3 58 FE E9 24 C2 5F 54 F7 4F 96 92 CE E4 7B 45
MD5:	A2 F6 DA 22 E5 1F B5 88 36 7E 9A 90 F6 09 4D 35

Public Key Info

Key Algorithm:	RSA
Key Parameters:	05 00
Key Size:	1024

Figura E.29 Información del archivo .cer para la aplicación C

Utilizando el procedimiento de la sección 2.5.4, se obtiene la información del certificado utilizado para firmar la aplicación. Utilizando la herramienta Jarsigner se han obtenido los resultados mostrados en la Figura E.28.

Como se observa en la Figura E.28, la aplicación bancaria C brinda información del nombre del desarrollador, el departamento, ciudad, y código de país. Posteriormente, se utiliza la herramienta OpenSSL para procesar el archivo .RSA de la carpeta `META-INF`, para obtener información adicional. Como se muestra en la Figura E.29, además de los datos encontrados con Jarsigner, se obtiene

información del algoritmo utilizado (RSA), sumas de verificación en SHA1 y MD5, etc.

E.2.2 ANÁLISIS ESTÁTICO

A continuación se realiza la ejecución del protocolo para el análisis estático de la aplicación C, de acuerdo con el procedimiento definido en la sección 2.6. Para esto, se utiliza un dispositivo virtual Android.

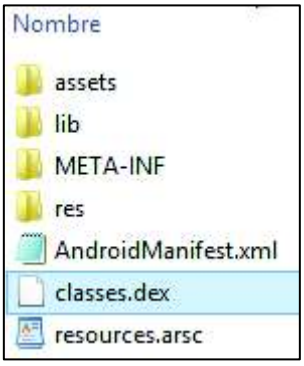


Figura E.30 Contenido del archivo .apk de la aplicación C

E.2.2.1 Obtención del Código Fuente

Utilizando el procedimiento de la sección 2.6.1, con la herramienta Apk Extractor se obtiene el archivo .apk de la aplicación bancaria C. En la Figura E.30 se muestra el contenido del archivo .apk para la aplicación C.

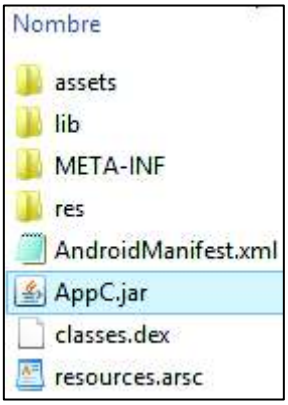


Figura E.31 Archivo .jar de la aplicación C

Posteriormente, se utiliza la herramienta Dex2jar con el archivo classes.dex mostrado en la Figura E.30 para obtener el archivo .jar como se muestra en la Figura E.31.

E.2.2.2 Permisos en el Archivo AndroidManifest.xml

Con base en el procedimiento de la sección 2.6.2, se analiza el archivo `AndroidManifest.xml`, correspondiente a la aplicación bancaria C, y luego de ser procesado por la herramienta AXMLPrinter2 para hacerlo legible.

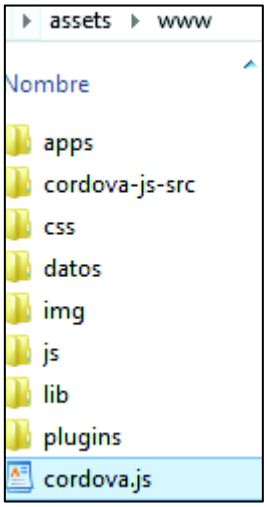


Figura E.32 Contenido de la carpeta assets de la aplicación C

Para identificar los permisos en el archivo `AndroidManifest.xml` se lo hace a partir de la etiqueta *<uses-permission>*, a continuación se listan los permisos encontrados:

- INTERNET
- ACCESS_NETWORK_STATE
- ACCESS_WIFI_STATE
- WRITE_EXTERNAL_STORAGE
- ACCESS_COARSE_LOCATION
- ACCESS_FINE_LOCATION
- CAMERA
- WAKE_LOCK
- WRITE_SETTINGS

El contenido completo del archivo `AndroidManifest.xml` para la aplicación bancaria C, en formato legible, se encuentra en el ANEXO C.3.

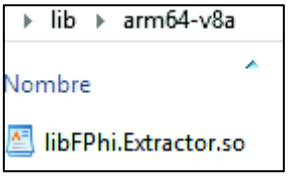


Figura E.33 Contenido de una de las subcarpetas de la carpeta lib, para la aplicación C

E.2.2.3 Framework

Utilizando el procedimiento de la sección 2.6.3, se ha analizado el contenido del archivo .apk de la aplicación bancaria C. Una vez que se ha revisado tanto la carpeta `assets`, como la carpeta `src`, se ha encontrado que la carpeta `assets` contiene una carpeta `www`, dentro de la cual se identifica el archivo `cordova.js`, como se muestra en la Figura E.32.

Al buscar la palabra `cordova`, en el contexto de Android, en Internet, se encuentra que Apache Cordova es un *framework* para desarrollo de aplicaciones móviles.

E.2.2.4 Librerías

De acuerdo con el procedimiento de la sección 2.6.4, se ha explorado el contenido de la carpeta `lib`, del archivo .apk descomprimido de la aplicación bancaria C.

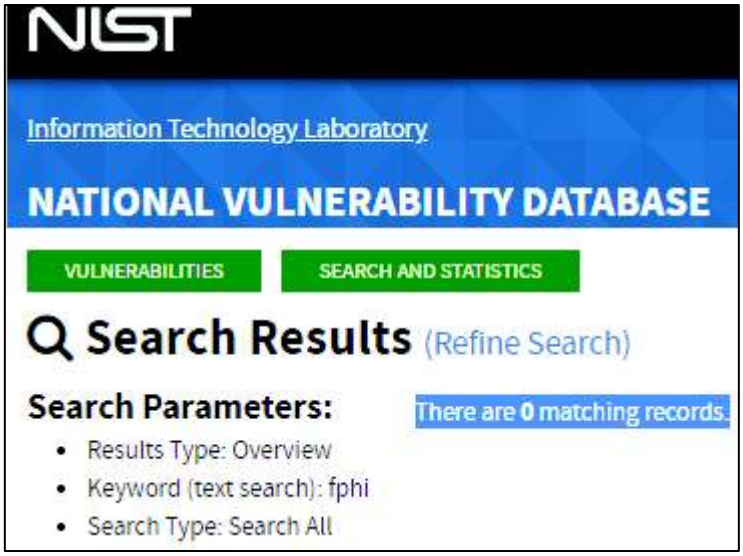


Figura E.34 Resultados obtenidos para una de las librerías de la aplicación C

Dentro de la carpeta `lib` se han explorado todas las subcarpetas, y se ha encontrado la presencia de la librería `libFPhi`, la cual es utilizada para

reconocimiento facial. Un ejemplo del contenido de las subcarpetas de la carpeta `lib` se muestra en la Figura E.33.

A continuación, se busca esta librería en las fuentes recomendadas en la sección 2.6.4. Como se observa en la Figura E.34, no existen vulnerabilidades reportadas para esta librería.

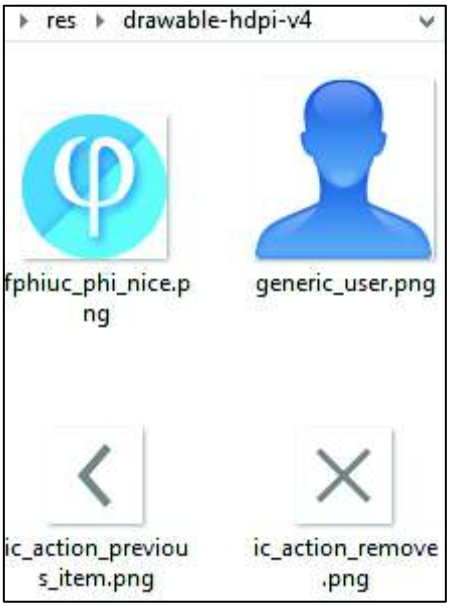


Figura E.35 Extracto del contenido de la carpeta res de la aplicación C

E.2.2.5 Vistas

Mediante el procedimiento especificado en la sección 2.6.5, se explora el contenido de la carpeta `res`, contenida en el archivo .apk de la aplicación bancaria C. Como se muestra en la Figura E.35, la carpeta res contiene los archivos necesarios para generar las vistas de la aplicación C. Por lo tanto, se podría crear una aplicación maliciosa que sea visualmente igual a la aplicación C.

E.2.2.6 Ingreso de Datos

De acuerdo con el procedimiento de la sección 2.6.6, se analizan los logs generados por la aplicación bancaria C, utilizando JD-Gui y Logcat. Para este punto se ha utilizado el proceso de una transferencia electrónica. En la Figura E.36 se observa la única porción de los generados, que puede dar una pista del funcionamiento de la aplicación C. El fragmento resaltado indica una de las carpetas que contiene el archivo .apk de la aplicación C.

```
4-25 00:21:19.070 I/chromium( 2386): [INFO:CONSOLE(2)] "Enviando Emit", source: file:///android_asset/www/lib/raven-js/dist/raven.min.js (2)
```

Figura E.36 Extracto de los logs generados por la aplicación C durante una transferencia directa

El fragmento resaltado en la Figura E.36 se busca también utilizando JD-Gui, y el resultado se observa en el Código E.9. Se encuentra que el fragmento pertenece a la clase `Config.class`, dentro del método `getStartUrl`. Como se indica en el Código E.9, la aplicación C utiliza el archivo `index.html`, contenido en la ruta `assets/www`.

```
public static String getStartUrl()
{
  if (parser == null) {
    return "file:///android_asset/www/index.html";
  }
  return parser.getLaunchUrl();
}
```

Código E.9 URL encontrada en el código de la aplicación C

Explorando el contenido de la carpeta `assets/www`, se ha encontrado el archivo TransferenciaInterna.html en la ruta `assets / www/ apps/ BancaPersonas/ view/Transferencia/mobile`.

```
ng-show=isGroupShown(etiquetas.lblOrigen)><label class="item item-input item-stacked-label"><select data-ng-required=true name=NumeroCuentaOrigen data-ng-model=NumeroCuentaOrigen data-ng-focus=isGroupShown(etiquetas.lblOrigen) data-ng-options="itemDebito.texto for itemDebito in catalogos.catalogoDebito.datos"></select></label></ion-item><ion-item class=item-stable ng-click=toggleGroup(etiquetas.lblDestino) ng-class="{active: isGroupShown(etiquetas.lblDestino)}"><div class=row><div class=col><i class=icon ng-class="isGroupShown(etiquetas.lblDestino)><label class="item item-input item-stacked-label"><select data-ng-required=true name=NumeroCuentaDestino data-ng-model=NumeroCuentaDestino data-ng-focus=isGroupShown(etiquetas.lblDestino) data-ng-options="itemCredito.texto for itemCredito in catalogos.catalogoCredito.datos" data-ng-change=actualizarEmailBeneficiario()></select></label></ion-item><ion-item class=item-stable ng-click=toggleGroup(etiquetas.lblConcepto) ng-class="{active: iv></div></ion-item><ion-item class=item-accordion ng-show=isGroupShown(etiquetas.lblConcepto)><label class="item item-input"><i class="icon ion-edit placeholder-icon"></i> <input type=text name=Concepto data-ng-model=Concepto
```

Código E.10 Extracto del código encontrado para el proceso de transferencia interna en la aplicación C

El Código E.10, es un extracto del archivo `TransferenciaInterna.html`, como se observa, para el proceso de transferencia interna se utilizan los números de cuenta origen y destino, así como una breve descripción del concepto de la

transferencia. La validación de los datos se da a partir del número de cuenta destino ingresado.

E.2.2.7 Código Utilizado para Autenticación

```
type=text id=username name=username placeholder={{lbl.phUser}} autocomplete=off
maxlength=50 data-ng-model=usuario.id></label></div><div class=padding data-ng-hide=
mostrarPassword><button type=submit class="button button-block button-positive
icon-right ion-log-in" data-ng-bind=lbl.btnLogin></button><div class=row><div class=col
></div><div class="col text-center"><img data-ng-src={{AppConfig.Assets.verisign}}><
form name=frmClave method=POST data-ng-submit=AutenticarUsuario() accept-charset=UTF-8
autocomplete=off data-ng-show="mostrarPassword && frmModo != 'clave'"><div class="list
list-inset"><label class="item item-input"><i class="icon ion-asterisk placeholder-icon
"></i> <input type=password id=password name=password autocomplete=off data-ng-model=
usuario.pass maxlength=14 placeholder={{lbl.pass}}></label></div><div class="padding
```

Código E.11 Extracto del código utilizado en el proceso de autenticación de la aplicación C

Mediante la exploración del contenido de la carpeta `assets/www`, se ha encontrado el archivo sign-in.html en la ruta `assets/www/apps/Login/views`. Al explorar el archivo sign-in.html, se ha encontrado que para el proceso de autenticación se utiliza sólo el nombre de usuario y contraseña, esto se muestra en el Código E.11. Además, la propiedad `autocomplete=off` tanto para el usuario como para la contraseña, impide que el sistema operativo Android recuerde estos datos, mientras que la propiedad `maxlength=14` limita la longitud de la contraseña a 14 caracteres.

Durante la exploración del código utilizado para la autenticación, se ha encontrado que este contiene datos reales de usuarios como nombres, números de cédula, direcciones y hasta la imagen de un cheque. Para recuperar la imagen del cheque se utilizó un convertidor de Base64 a imagen. La imagen obtenida del cheque se encuentra en el Anexo G.

E.2.2.8 Autenticación Online/Offline

El proceso de autenticación de la aplicación bancaria C se realiza de manera *online*. Para determinar esto se ha buscado la frase “https://” y se ha encontrado que en el archivo `general.min.js`, se encuentra la URL con la cual se comunica la aplicación C. En el Código E.13 se presenta un extracto del contenido del archivo `general.min.js`.

```
ebapitellerv2",urls:{IpMulticanal:"https://appc.bancoc.com/GYEMCANAL/",MultiChannelUrl:
s",ObtenerBalanceCliente:"ObtenerBalanceCliente",TellerObtenerBancos:"TellerObtenerBanc
ONDEADA"}}}]),angular.module("AppConfigs").factory("AppConfigSettingsComp",["AppConfigS
```

Código E.13 URL con la cual se comunica la aplicación C

E.2.2.9 Información Adicional al Nombre de Usuario/Password

Como se analizó en la sección E.2.2.7, la aplicación bancaria C no utiliza ningún tipo de identificador adicional, solamente emplea un nombre de usuario y contraseña.

E.2.2.10 Métodos de Autenticación

Dado que la información mostrada en los logs de la aplicación, contribuyó sólo para conocer la ubicación de los archivos que utiliza la aplicación bancaria C para los diferentes procesos (autenticación, transferencias, consultas, etc.), se exploró el contenido de la carpeta `assets/www`.

```
<ion-modal-view><ion-header-bar class="bar-header bar-positive"><h1 class=title>Acceso
</h1><a class="button button-clear" ng-click=closeModal()><i class="ion ion-close"></i>
</a></ion-header-bar><ion-content><div class="list list-inset positive"><div class="
item item-divider text-center">Escoge tu opción de ingreso</div><a class="item item-sel
" data-ng-click=procesoAutenticacionFacial()><img src=img/bg/iconos/reconocimiento.png>
 <span>Reconocimiento Facial</span></a> <a class="item item-sel" data-ng-click=
procesoLogin()><img src=img/bg/iconos/llave.png> <span>Usuario / Contraseña</span></a>
</div></ion-content></ion-modal-view>
```

Código E.14 Extracto del código encontrado para el proceso de autenticación

En la ruta `assets/www/apps/Login/views`, se encontró el archivo seleccionarAutenticacion.html. Como se muestra en el Código E.14, al explorar este archivo, se encuentra que la aplicación brinda dos opciones de autenticación: Reconocimiento Facial y Usuario/Contraseña.

E.2.2.11 Bloqueo

```
en el menú Mi Perfil.","0006":"Usuario Bloqueado (SFA), comuníquese con el Banco
```

Figura E.37 Extracto de los logs generados por la aplicación C durante el ingreso erróneo de credenciales

Siguiendo el procedimiento de la sección 2.6.11, se ha ingresado de manera repetida la contraseña de forma incorrecta durante el proceso de autenticación. Con

esto, se ha determinado que la aplicación bancaria C se bloquea al tercer intento de ingreso de una contraseña incorrecta.

Cuando esto sucede, la aplicación muestra el mensaje “Usuario Bloqueado, por favor comunicarse con el banco”. Utilizando esta frase se ha explorado el contenido de la carpeta `assets/www`.

Con esto, se ha encontrado la frase en el archivo `app-bloqueo.min.js`. En la Figura E.37 se muestra un extracto del contenido de este archivo, en el cual se observa la frase buscada.

E.2.2.12 Autenticación Única

Dado que el proceso de autenticación se realiza desde el lado del servidor, para la aplicación bancaria C no se ha podido encontrar el código correspondiente a esta sección en la carpeta `assets/www`.

Sin embargo, en la exploración inicial de la aplicación C se ha encontrado que esta solicita el ingreso de credenciales nuevamente luego de aproximadamente 5 minutos de inactividad.

E.2.2.13 Autenticación en Dos Pasos

Mediante la exploración del código ubicado en la carpeta `assets/www`, no se ha encontrado código que indique que esta aplicación utiliza autenticación en dos pasos. Lo cual se corrobora durante la exploración inicial de la aplicación bancaria C.

E.2.2.14 Análisis Estático Automatizado

De acuerdo con el procedimiento de la sección 2.6.14, se realiza el análisis automatizado de la aplicación bancaria C, mediante la herramienta MobSF. Un extracto de los resultados obtenidos se muestra en la Figura E.38.

Gracias a la herramienta MobSF se ha encontrado que la aplicación bancaria C se encuentra diseñada para ejecutarse en dispositivos Android con una versión a partir de *Lollipop* (API 21).

Para la aplicación C no se encuentran proveedores de contenido exportados, esto se verifica a profundidad en la sección E.2.3.3.

Junto con las aplicaciones A y B, la aplicación bancaria C ha sido firmada utilizando el algoritmo SHA1 con RSA, el cual es conocido por tener problemas de colisión.

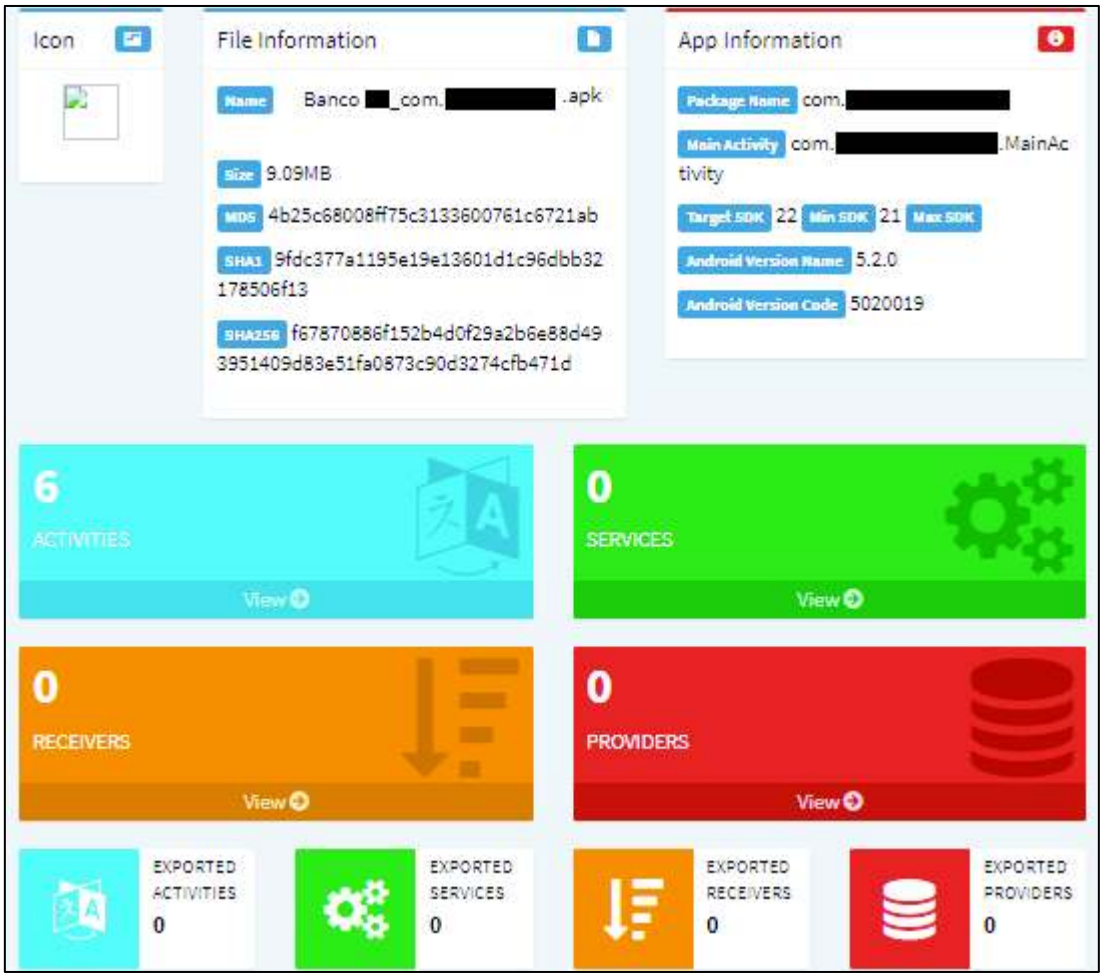


Figura E.38 Resultado de la herramienta MobSF para la aplicación C

Dentro del análisis de los permisos que solicita la aplicación C, se lista a continuación un extracto de aquellos calificados por MobSF como peligrosos:

- Ubicación del dispositivo.
- Acceso al almacenamiento externo del dispositivo.
- Escritura de las configuraciones del dispositivo.
- Cámara.
- Control de bloqueo de pantalla, etc.

En la sección *Code Analysis* de los resultados obtenidos se encuentra que la aplicación C registra en logs información sensible, la cual no se recomienda que sea registrada nunca en logs. Además, esta aplicación puede leer y escribir información en el almacenamiento externo del dispositivo Android. El resultado detallado del análisis realizado para la aplicación C se encuentra en el Anexo D.3.

E.2.3 ANÁLISIS DINÁMICO

Mediante el procedimiento de la sección 2.7 se realiza el análisis dinámico de la aplicación bancaria C. Para el punto E.2.3.1 se utiliza un dispositivo Android físico, mientras que las secciones restantes se utiliza un dispositivo Android virtual.

E.2.3.1 Protocolos de Comunicación Utilizados

Utilizando el procedimiento de la sección 2.7.1 se observan los protocolos que utiliza la aplicación C con el servidor, para lo cual se utilizan las herramientas Network Connections y OWASP ZAP.

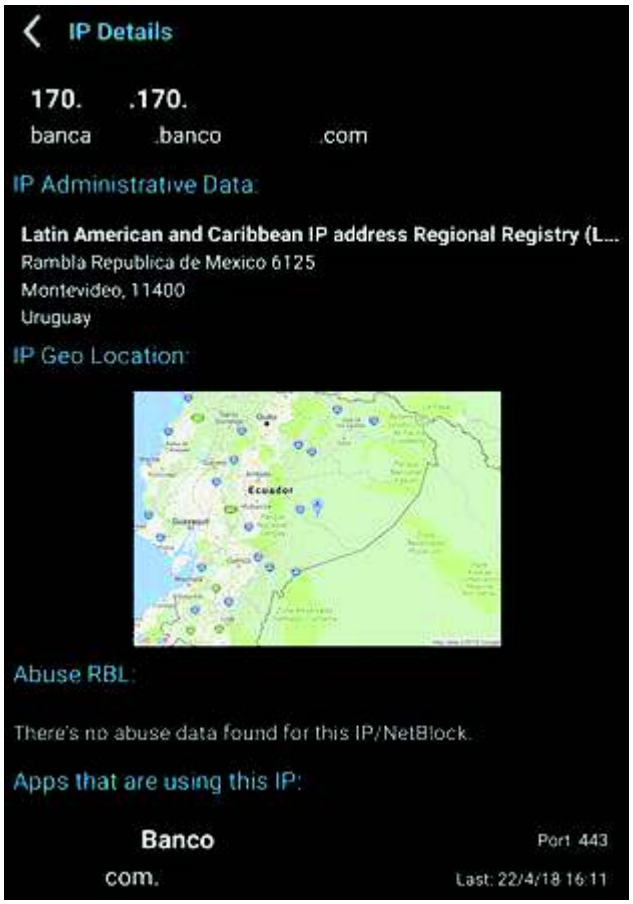


Figura E.39 Resultado de la herramienta Network Connections para la aplicación C

- Network Connections

Mediante la aplicación Network Connections se han obtenido los resultados de la Figura E.39. En los resultados se encuentra la dirección IP con la cual se comunica la aplicación B así como su ubicación. Adicionalmente, se encuentra que para la comunicación con el servidor, la aplicación A utiliza el puerto 44, el cual corresponde a HTTPS.

- OWASP ZAP

Utilizando la herramienta OWASP ZAP se ha realizado el análisis del tráfico generado por la aplicación bancaria C. Para esto se ha utilizado un dispositivo Android físico. Como se muestra en la Figura E.40, todas las URLs con las cuales se comunica la aplicación C utilizan el protocolo HTTPS.

ID	Req. Tim...	Mét...	URL	C...	Reas...	...	Size Res...	Highe...
5	3/04/18 2...	PO...	https://banca[redacted].banco[redacted]...	200	OK	8...	1.510 byt...	Bajo
13	3/04/18 2...	PO...	https://banca[redacted].banco[redacted]...	200	OK	5...	274 bytes	Bajo
14	3/04/18 2...	PO...	https://banca[redacted].banco[redacted]...	200	OK	1...	54.352 b...	Bajo
15	3/04/18 2...	PO...	https://banca[redacted].banco[redacted]...	200	OK	2...	1.605 byt...	Bajo
16	3/04/18 2...	PO...	https://banca[redacted].banco[redacted]...	200	OK	2...	458 bytes	Bajo
17	3/04/18 2...	PO...	https://banca[redacted].banco[redacted]...	200	OK	2...	219 bytes	Bajo
18	3/04/18 2...	PO...	https://banca[redacted].banco[redacted]...	200	OK	1...	212 bytes	Bajo
19	3/04/18 2...	PO...	https://banca[redacted].banco[redacted]...	200	HTT...	6...	1.522 byt...	Bajo
20	3/04/18 2...	PO...	https://banca[redacted].banco[redacted]...	200	HTT...	2...	292 bytes	Bajo

Figura E.40 Extracto del resultado de OWASP ZAP para la aplicación C

En la Figura E.41 se muestran todas las URLs con las cuales se comunica la aplicación bancaria C, las cuales se ha identificado utilizando la herramienta OWASP ZAP.

Figura E.41 URLs con las que se comunica la aplicación C

Mediante el análisis de los paquetes capturados durante el proceso de autenticación de la aplicación C, se ha identificado tanto el nombre de usuario, así como la contraseña en texto plano, lo cual se muestra en la Figura E.42.

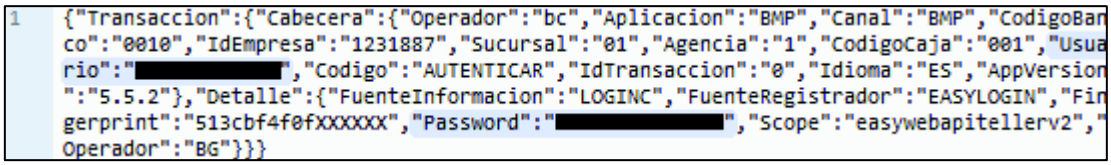


Figura E.42 Extracto del contenido de los paquetes capturados, generados por la aplicación C

E.2.3.2 Scripts Existentes para la Búsqueda de Vulnerabilidades en Aplicaciones Android

Utilizando el procedimiento definido en la sección 2.7.2, se utilizan los *scripts* Pidcat y Manitree como herramientas para el análisis de vulnerabilidades de la aplicación C. Mediante estos *scripts*, se ha conseguido identificar las carpetas que utiliza la aplicación C, por otra parte no se han detectado permisos potencialmente peligrosos.

- Pidcat

Utilizando el *script* Pidcat, se ha conseguido obtener los logs genrados por la aplicación bancaria C mientras esta realiza el proceso de autenticación. Como se observa en la Figura E.43, se ha encontrado uno de los directorios que utiliza la aplicación C durante su ejecución.

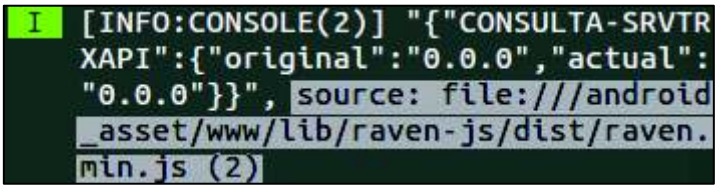


Figura E.43 Extracto de los resultados de la herramienta Pidcat para la aplicación C

- Manitree

Gracias al *script* Manitree se ha analizado el archivo `AndroidManifest.xml` de la aplicación C.

En la Figura E.44 se muestra un extracto del resultado obtenido, como se observa, la aplicación C no contiene permisos potencialmente peligrosos.

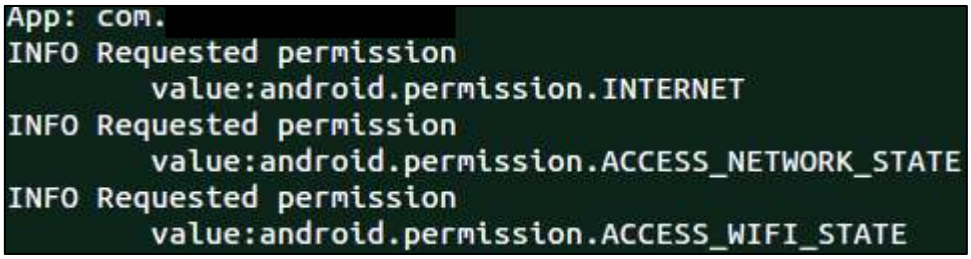


Figura E.44 Extracto de los resultados de la herramienta Manitree para la aplicación C

E.2.3.3 Proveedor de Contenido

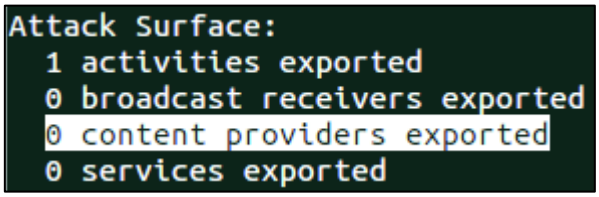


Figura E.45 Resultados de Drozer para la aplicación móvil C

De acuerdo con el procedimiento de la sección 2.7.3, utilizando la herramienta Drozer, se analiza la aplicación bancaria C. Como se muestra en la Figura E.45, el número de proveedores de contenido exportados es cero.

Figura E.46 Consulta a los URIs encontradas en la aplicación C

E.2.3.4 Inyección desde el Lado del Cliente

Utilizando el procedimiento de la sección 2.7.4, mediante la herramienta Drozer se analiza la aplicación C. Como se muestra en la Figura E.46, la aplicación C no contiene URIs accesibles.

A continuación, utilizando la herramienta Drozer, se comprueba si las URLs detectadas en la aplicación C son susceptibles al ataque de inyección SQL. Como se observa en la Figura E.47, no existen vulnerabilidades de este tipo.

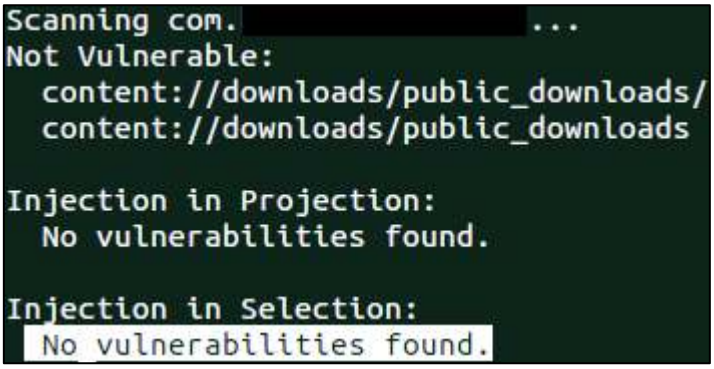


Figura E.47 Prueba de Inyección SQL para la aplicación C

ANEXO F

RESULTADOS DE LA HERRAMIENTA MANITREE

F.1 RESULTADOS DE LA HERRAMIENTA MANITREE PARA LA APLICACIÓN A

El documento se encuentra adjunto como anexo digital.

F.2 RESULTADOS DE LA HERRAMIENTA MANITREE PARA LA APLICACIÓN B

El documento se encuentra adjunto como anexo digital.

F.3 RESULTADOS DE LA HERRAMIENTA MANITREE PARA LA APLICACIÓN C

El documento se encuentra adjunto como anexo digital.

ANEXO G
IMAGEN DE UN CHEQUE CONTENIDA EN EL CÓDIGO DE LA APLICACIÓN BANCARIA C

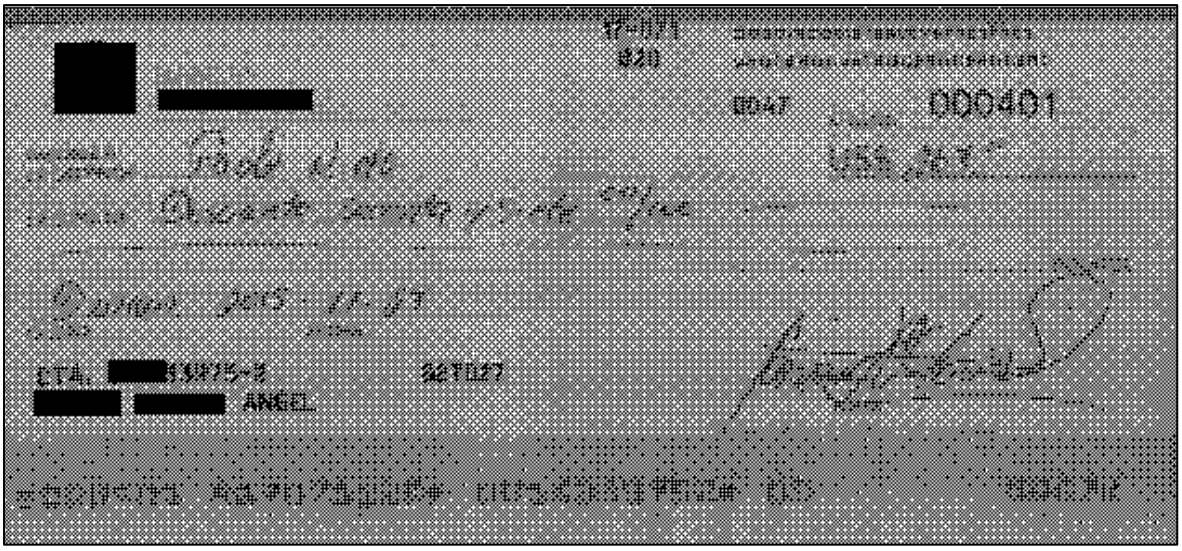


Figura G.1. Imagen frontal recuperada del código de la aplicación bancaria C

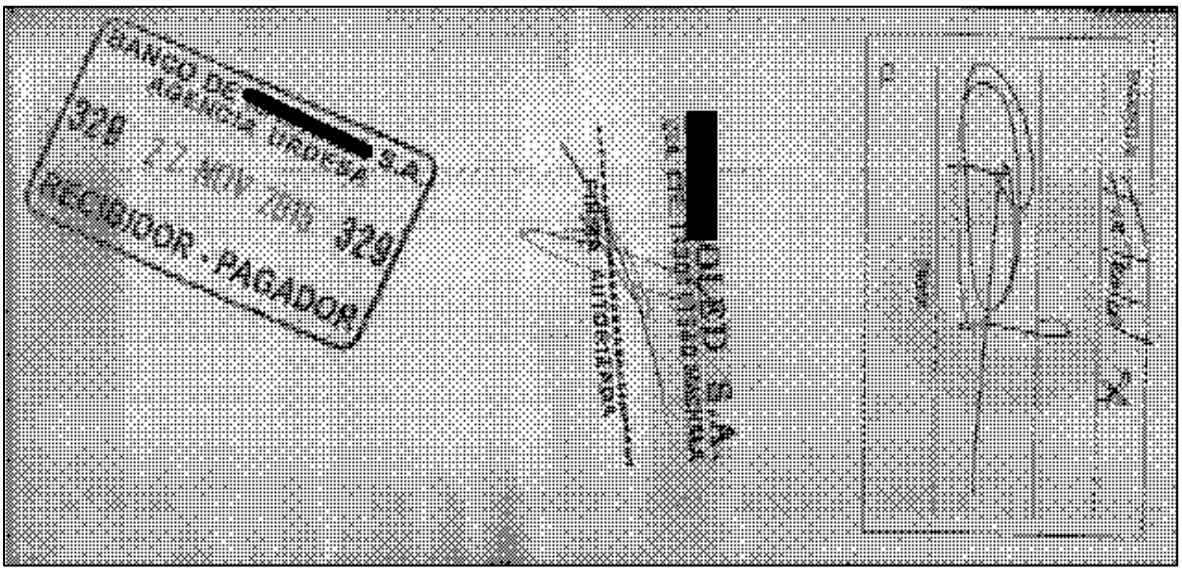


Figura G.2. Imagen posterior recuperada del código de la aplicación bancaria C

ORDEN DE EMPASTADO