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

EVALUACIÓN DE LA SEGURIDAD DE APLICACIONES MÓVILES BANCARIAS

TESIS PARA OPTAR AL GRADO DE
MAGISTER EN CIENCIAS, MENCIÓN COMPUTACIÓN

CRISTIÁN ANDRÉS ROJAS POBLETE

PROFESOR GUÍA:
ALEJANDRO HEVIA ANGULO

MIEMBROS DE LA COMISIÓN:
TOMÁS BARROS ARANCIBIA
LUIS MATEU BRULÉ
ROSA ALARCÓN CHOQUE

SANTIAGO DE CHILE
2016

Resumen

En tiempos recientes, las entidades financieras están ofreciendo cada día más a sus clientes la posibilidad de operar con ellos mediante la llamada *Banca Móvil*: El usuario baja una aplicación a su celular la cual le permite realizar las operaciones bancarias más comunes: Consulta de saldo, revisión de información de su tarjeta de crédito, transferencias de dinero a otras cuentas, etc. A estas operaciones agregan otras como contacto en caso de emergencia bancaria, consulta de cuáles son los cajeros automáticos y sucursales del banco más cercanas al usuario, etc.

Los bancos ofrecen estas aplicaciones a través de las tiendas de las diferentes plataformas móviles disponibles, bajo la premisa de permitir a sus usuarios el acceso a sus operaciones bancarias en forma fácil y segura. Dado que estas aplicaciones manejan información sensible, como la personal, geográfica y financiera, cabe hacerse la pregunta de cuál es el estado de su seguridad.

Para responder esta pregunta, se realizó un estudio el cual involucró a diez aplicaciones de banca móvil nacionales para la plataforma Android disponibles en la Google Play Store. A cada aplicación se le realizó un procedimiento automatizado de ingeniería reversa, para posteriormente analizar estáticamente el código fuente extraído como parte de ese proceso. Adicionalmente, se realizaron pruebas pasivas sobre la aplicación en funcionamiento con la cooperación de algún cliente del banco que ofrece la aplicación. Posterior a ello, se determinó qué vulnerabilidades de seguridad tiene cada aplicación, y en base a ello se creó una taxonomía de malas prácticas en el desarrollo de estas aplicaciones las cuales tienen como consecuencia las vulnerabilidades encontradas.

Esta tesis tiene como objetivo clasificar las aplicaciones bancarias móviles ofrecidas por los bancos nacionales dentro de la taxonomía recién mencionada, y dar lineamientos respecto de buenas prácticas de seguridad al momento de diseñar, implementar y lanzar al mercado aplicaciones móviles que manejan información sensible para sus usuarios.

“Este es un programa de entrenamiento, similar a la realidad programada de la Matrix. Tiene las mismas reglas básicas, como la gravedad. Lo que debes aprender es que estas reglas no son diferentes a las reglas de un sistema computacional. Algunas pueden ser torcidas, otras pueden ser rotas.”

— Laurence Fishburne en ”The Matrix”. Warner Bros., 1999.

Dedicatoria

A mi madre, Mónica, de quien obtuve las mayores lecciones de vida, la moral y la fuerza para seguir adelante y tirar a la basura prejuicios y estándares.

A mi padre, José Antonio, en quién me inspiré para seguir los caminos de la ciencia y la gestión.

A mi hermano Camilo, por mostrarme que con carácter y ética, se pueden lograr cosas muy importantes en la vida.

A mi hermano Nicolás, por mostrarme que el camino que hay que seguir siempre es el propio.

A mi hermana Valentina, por romper el esquema ingenieril de nuestra generación.

A mis sobrinos, Simón y Belén, y a mi ahijado Máximo, para quienes espero puedan hacer uso de tecnología más segura en el futuro.

A Alejandro Hevia, quien se atrevió a *prestarle ropa* a un tesista muy poco ortodoxo en sus planteamientos, contenidos, y motivación.

A Nancy Hitschfeld, Cecilia Bastarrica, Alejandro Hevia, Gonzalo Huerta-Canepa y Alejandra Beghelli por ofrecerme múltiples oportunidades de desarrollar una carrera como docente universitario, la cual se ha convertido en mi pasión. Destaco especialmente a Jaime Navón, de quien obtuve una de mis mayores lecciones como profesor: Deshacerme de cualquier prejuicio hacia mis estudiantes respecto de su origen o casa de estudios a la cual asisten.

A todos aquellos cuentacorrentistas anónimos que me apoyaron con los experimentos de la presente investigación. Sin Ustedes no hubiera podido conocer cómo los bancos hacen su trabajo en términos de seguridad. También a todos aquellos que me ayudaron a contactar a los tomadores de decisión tecnológicos de estos bancos. No es tarea fácil contactarlos, y sin Ustedes, la tarea hubiera sido imposible.

A todos aquellos que conocí en mi carrera como profesional de Seguridad de la Información: Francisco Flores, Manuel Moreno, Francisca Moreno, Aldo Tobar, Miguel Diaz, Juan Antonio Serrano, Patricia Almonacid, Rodrigo Cifuentes, Rodrigo Contreras, Ritna Roco, Paola Fuentes, Marcelo Castilla, Ricardo Cid, y otros más que quizás esté olvidando. Aún así, a todos ellos, va mi saludo y reconocimiento por aquellos intercambios de conocimiento que fueron de valor para ellos y para mí.

A todos mis amigos, por ser parte de mi vida, por alentarme en el momento más difícil de mi carrera durante el 2011, y sobre todo en mi carrera docente, y por las cervezas (buenas y malas) que hemos compartido juntos. Daniel Pizarro, Paulina Fernandez, Carolina Castro, Fernanda Llantén, María Paz Llantén, Estefanía Natalino, Carlos Castro, Alejandra Parra, Luis Felipe Salazar, Ignacio Rencoret, Natalia Pumarino, Nicole Furkert, Miguel Rojas, Ariadna Martinich, Adolfo Carrasco, Héctor Horta y Macarena Bueno, ¡SALUD!

A Karina Bunster. Como dice aquella canción de Fito Paez, yo no buscaba a nadie y te vi.

Y finalmente, a Naya. Tu partida nos dolió profundamente, y cada vez que te recuerdo, las lágrimas son obligatorias. Hasta siempre.

Tabla de contenido

1

Introducción

En los últimos años, los bancos chilenos han lanzado al mercado aplicaciones que permiten a sus clientes realizar transacciones desde sus teléfonos móviles y tablets. Estos lanzamientos vienen acompañados de fuertes campañas de marketing en pos del uso de tales aplicaciones. Sin embargo, ¿es la seguridad una parte primordial en el desarrollo de este tipo de aplicaciones? Desde hace un tiempo que existen cuestionamientos a la forma en que los bancos gestionan la seguridad de sus sistemas y aplicaciones web disponibles para los usuarios [1]. A eso se añaden los riesgos asociados al desarrollo de aplicaciones móviles, los cuales de no ser controlados podrían significar que sus usuarios sean víctimas de ataques a la confidencialidad de su información. La información a proteger en los dispositivos móviles típicamente pertenece a una de las siguientes tres áreas:

- **Información Bancaria**: Montos almacenados en cuentas corrientes/a la vista, transacciones realizadas por el cliente, números de tarjetas de crédito/débito, etc.
- **Información Personal**: Datos personales (nombre, edad, etc.), de contacto (e-mail, números telefónicos y otros), geográficos (progresión de la ubicación mediante GPS), fotos, archivos dentro del teléfono, etc.
- **Información del aparato**: Identificadores telefónicos (IMSI, IMEI, ANDROID_ID, etc.), llaves de encriptación de la aplicación (o de otras), etc.

Es por esto que es necesario estudiar la seguridad de las aplicaciones lanzadas al mercado por parte de instituciones financieras, ya que las consecuencias de un ataque pueden ser las siguientes:

- Caídas del servicio móvil durante transacciones.
- Filtración de datos personales del usuario de la aplicación.
- Robo de credenciales (ej. login/password) del usuario, las cuales podrían ser reutilizadas por los atacantes para acceder a su información bancaria, o peor aún, realizar transacciones monetarias en su nombre.

Motivación y aporte

Los clientes de los bancos pueden operar con la banca en línea de tres formas:

1. A través de un sitio web de escritorio.
2. A través de un sitio web móvil.
3. A través de una aplicación móvil.

En el primer caso, el banco desarrolla un sitio web para browsers de escritorio, generalmente con la mayor cantidad de servicios disponibles para el cliente. El caso del sitio web móvil es parecido al primero, pero típicamente con menor funcionalidad que el de escritorio, y con una interfaz de usuario adaptada a dispositivos móviles. El tercer caso es una aplicación desarrollada con las API's ofrecidas por la plataforma celular, funcionando en forma nativa dentro del aparato e interactuando con el banco mediante servicios web.

La motivación de la presente tesis se basa en los siguientes aspectos relacionados con la seguridad en aplicaciones móviles:

i. En el caso de las aplicaciones web móviles, el usuario puede ver en todo momento si el sitio usa el protocolo SSL/TLS para asegurar el transporte de datos mediante la aparición en la interfaz de usuario de un elemento indicador, por ejemplo un candado verde en la barra de direcciones, cosa la cual no ocurre en la aplicación móvil. Las API de las plataformas móviles no ofrecen la posibilidad de asegurar en forma estandarizada (como sí lo hacen los browsers móviles) de que el usuario se está conectando en forma encriptada al servidor con el cual interactúa.

ii. Aunque la aplicación se comunique vía SSL/TLS con el servidor, esa conexión podría no necesariamente estar bien configurada. El protocolo SSL/TLS es una tecnología que evoluciona muy rápido, debido a la rápida aparición de vulnerabilidades en ella, y dada también su aparente simplicidad de implementación, lo que causa una falsa sensación de seguridad [2].

iii. Otro problema observado es que las API de las plataformas móviles ofrecen la posibilidad de acceder a diversos recursos del aparato, como el disco duro, GPS, mensajes de texto (SMS), etc., privilegios los cuales muchas veces son abusados por los desarrolladores en el sentido de que piden más privilegios que los que realmente necesitan[1] [3]. Esto sin contar con el hecho de que los usuarios no leen los privilegios que solicita la aplicación, problema exacerbado por el modelo todo-o-nada[2] que ofrece a los desarrolladores la plataforma Android en particular.

iv. Muchas aplicaciones bancarias almacenan información en áreas del teléfono, las cuales durante su operación normal no debieran ser accesibles por el usuario ni por otras aplicaciones. Por ejemplo, en la plataforma Android, cada aplicación tiene un área de almacenamiento para sí, inaccesible por parte de otras aplicaciones y por parte de usuarios que accedan a la memoria interna del aparato (vía USB o exploradores de archivos), esto si el usuario no ha rooteado[3] su equipo. Si eso ocurre el usuario podría acceder a información la cual el banco supone que no debería poder ver o modificar.

El aporte que busca hacer el estudiante con esta tesis es realizar un catastro de la seguridad de las aplicaciones móviles bancarias chilenas. Todo esto en base a una evaluación con datos obtenidos mediante la investigación propuesta (ver "Metodología") para determinar la verosimilitud de la conjetura respecto de que los problemas recién mencionados son prevalentes.

[1] Lo que constituye una violación del *Principio de Privilegio Mínimo* (Ver la sección "Principios de Seguridad de Sistemas").

[2] En el caso de Android, los usuarios deben revisar la lista de privilegios solicitados por la aplicación, y aceptarlos todos, si no, la aplicación no será instalada.

[3] Al ser la plataforma Android basada en Linux, tiene un usuario `root` el cual tiene privilegios de superusuario. Al recibir el usuario su aparato no tiene acceso a ese usuario, a menos que, aprovechando alguna vulnerabilidad, obtenga ese privilegio. Ese proceso se conoce como *rooting*.

2

Estado del Arte y Antecedentes

Seguridad de software

La seguridad, en un sistema de información, se define como la capacidad de un sistema de funcionar adecuadamente, aún dentro de un entorno hostil [4]. Al momento de ser más específicos y hablar de seguridad de software, esta se define como la construcción de software que sea capaz de funcionar correctamente bajo ataque malicioso [5]. Dos conceptos que son base para cualquier análisis de seguridad de software como el propuesto en esta tesis, son la triada CIA y los conceptos de identificación, autentificación y autorización, los cuales se describen a continuación.

La triada CIA

La triada CIA[1] es un modelo que permite desarrollar políticas de seguridad, identificar áreas problemáticas y soluciones necesarias para la seguridad [6] mediante sus tres componentes fundamentales:

- Confidencialidad: Los activos de información deben estar disponibles para ser

[1] A pesar de la referencia a una agencia gubernamental que pudiera resultar a partir del conocimiento del lector, en este caso es una sigla que indica sus componentes fundamentales: Confidencialidad, Integridad y Disponibilidad (Availability).

vistos sólo por quienes corresponde y se deben impedir accesos no autorizados a ellos.

- Integridad: Los datos deben ser protegidos de ser modificados o borrados por partes no autorizadas, y en caso de que partes autorizadas hayan realizado modificaciones que no debieron, éstas deben ser deshechas.
- Disponibilidad: Los activos de información deben estar disponibles para aquellas partes autorizadas las cuales requieran hacer uso de ellos.

Identificación, autentificación y autorización

Existe otra triada la cual resulta importante para el análisis, que tiene que ver con cómo un sistema identifica, autentifica y autoriza el acceso de sujetos (usuarios, otras aplicaciones, servidores, etc) a un sistema de información [7]:

Identificación

Mediante la identificación, el sistema pregunta (o no) al sujeto por algún elemento que le permita determinar si es quien dice ser.

Autentificación

Aquí el sistema verifica las credenciales presentadas por el sujeto en la etapa de identificación para verificar si es quién dice ser.

Cabe destacar que para que ocurra el proceso de autentificación, el sujeto debe proveer alguna combinación de los siguientes tres factores de autentificación:

- **Conocimiento**: El sujeto provee algo que sabe, por ejemplo, una password o un PIN[2].
- **Propiedad**: El sujeto provee algo que posee, como una llave, un token o una tarjeta de acceso.
- **Característica**: El sujeto provee algo que *es*, o sea, alguna característica personal mediante biometría (ej., huella dactilar, scanner ocular o incluso la firma).

[2]Personal Identification Number. Usualmente es la clave que se usa en los cajeros automáticos o para desbloquear equipos móviles.

Considerando estos factores, se considera que un sistema es seguro si se usan 2 de ellos, esto debido al Principio de separación de privilegios (ver la sección "Principios de seguridad de sistemas"). Esto se conoce como autentificación de dos factores o 2FA[3].

Autorización

Posterior a la autentificación, el sistema verifica los privilegios y permisos de acceso que posee el sujeto sobre los objetos a los cuales accede, en base a su modelo de privilegios, quien es el que determina quién puede hacer qué sobre qué recurso, evitando y restringiendo así accesos no autorizados.

El ciclo de vida del desarrollo de software (SDLC)

También vale la pena mencionar que el desarrollo de software, sea éste web, de escritorio o móvil, se rige por un proceso conocido como el SDLC[4], el cual se compone de varias etapas, las cuales, independiente de la metodología de desarrollo utilizada (Cascada, Iterativa, Ágil, Espiral, etc.), son las siguientes [8]:

1. Requisitos: Al inicio del SDLC, se define lo que se desea que el software realice (y qué no). Usualmente los requisitos se dividen en:
    - Requisitos funcionales:
        - Definiciones de usuarios y roles (el *quién*),
        - Ambientes de despliegue (el *dónde*),
        - Objetos y datos (el *qué*),
        - Actividades y acciones (el *cómo*),
        - Secuenciación y temporización (el *cuándo*).
    - Requisitos operacionales:
        - Cómo el software será desplegado,
        - Cómo el software será operado,
        - Cómo el software será gestionado.
2. Diseño: En ésta etapa, las especificaciones definidas durante la etapa de requisitos se traducen en definiciones arquitecturales las cuales servirán de guía para la implementación posterior.

[3]Two-Factor Authentication. Usualmente combina autentificación por propiedad y conocimiento.
[4]Software Development LifeCycle.

3. Implementación: Aquí es cuando el software es desarrollado, codificado y configurado.
4. Pruebas: El software es probado en busca de errores de implementación, fallas de diseño, vulnerabilidades de seguridad. Usualmente a esta etapa se la conoce como Quality Assurance (QA).
5. Aceptación: Posterior a las pruebas, se recibe la documentación de requisitos y pruebas, y si el software cumple con los requisitos y alcanza un nivel adecuado de confiabilidad, es aceptado.
6. Operaciones: En este momento el software entra en operación, y tiene varias subetapas:
    - Despliegue: Instalación y configuración del software,
    - Operaciones: Funcionamiento en el día a día del software,
    - Mantenimiento: Corrección de errores encontrados post-instalación, adición de nueva funcionalidad,
    - Enajenación: Fin de la vida útil del software, por obsolescencia o aparición de una mejor versión, y gestión de los datos contenidos en éste.

Principios de seguridad de sistemas

En 1975, Saltzer y Schroeder publican el que probablemente es la principal referencia en seguridad de la información [9]. En su paper, definen ocho principios de seguridad de sistemas de información. Estos principios son los siguientes:

1. **Economía de mecanismos**: Se deberá mantener el sistema lo más simple posible, ya que la revisión y corrección de código vulnerable se torna más difícil en entornos complejos. Ejemplo de tal complejidad es el fenómeno de la *fragmentación*, el cual se observa en plataformas móviles, y se manifiesta a través de las diferencias entre tamaños de pantalla, hardware y configuraciones observadas dentro de una misma plataforma móvil, lo cual causa que el código creado dependa muchas veces de diferentes configuraciones de hardware, lo cual lo hace más complejo, por lo tanto propenso a errores, y finalmente a vulnerabilidades.
2. **Política por defecto a prueba de fallos**: Este principio define una política base para el desarrollo de cualquier sistema de información, estableciendo lo que se conoce como una *política de inclusión*, vale decir, parte de una base en la cual se niega el acceso a todos los recursos del sistema de información, para ir añadiendo,

a medida que se van requiriendo, los permisos de acceso a diferentes recursos del sistema. Ejemplo de esto en aplicaciones móviles es el hecho de que, previo a su instalación, éstas parten sin ningún privilegio de acceso a recursos del aparato, y, dependiendo del modelo de permisos de la plataforma, es el usuario el que le provee los permisos mediante una *lista blanca*.

3. **Completa mediación**: Aquí se establece que toda vez que un sujeto (sistema, usuario, programa, etc.) requiera el acceso sobre un objeto (recurso), los permisos del sujeto sobre el objeto deben ser verificados. Esto quiere decir que no basta con que, por ejemplo, un usuario esté autentificado ante un sistema para acceder a los recursos de éste, sino que también se debe establecer qué recursos está autorizado a acceder y qué puede hacer con ellos, además de verificar tales autorizaciones cada vez que el usuario quiera hacer uso de ellos.
4. **Diseño abierto**: El diseño de los sistemas no puede depender de secretos para reforzar su seguridad. Esto va en contra de la filosofía de la "Seguridad a través de la oscuridad"[5] que se basa en confiar la seguridad de un sistema de información en el secreto de su implementación. Un ejemplo clásico en este sentido es el Principio de Kerckhoffs [10], el cual establece que un algoritmo criptográfico no debe basar su seguridad en el secreto de su diseño, sino que en el largo de su(s) llave(s) de cifrado.
5. **Separación de privilegios**: La seguridad de un sistema se refuerza mediante la participación de múltiples actores para realizar acciones riesgosas sobre el sistema. Por ejemplo, antiguamente para lanzar un misil nuclear se requería el uso de dos llaves, cada una de ellas operada por una persona diferente[6]. Un ejemplo más moderno es el uso de 2FA para sistemas bancarios y algunas aplicaciones como GMail y Amazon Web Services.
6. **Privilegio mínimo**: Cualquier sujeto puede tener sobre un objeto sólo los permisos que éste requiere para realizar su labor, y nada más. Un ejemplo de violación de este principio es el abuso de permisos solicitados por aplicaciones, como el ocurrido en el caso de la aplicación "Brightest Flashlight", cuyo propósito era usar la luz de la cámara del teléfono como linterna, pero además del permiso para usar la linterna, solicitaba permisos como acceso a los contactos del teléfono, GPS, etc. [11].
7. **Mecanismo menos común**: Este principio establece que se deben reducir los mecanismos comunes entre sujetos con diferentes niveles de privilegios. A diferentes grupos de sujetos se les asignan diferentes grupos de permisos, lo

[5]Esto se conoce en inglés como *"Security through obscurity"*.
[6]Este estándar de procedimiento militar se conoce como *Two-Man Rule*.

cual causa que si se logra doblegar la seguridad de un grupo de sujetos (o un sujeto perteneciente a este grupo), los daños causados a sus recursos sean limitados y no extensibles a todo el sistema. Ejemplo de esto es el hecho de que las plataformas móviles impiden el acceso de usuarios a áreas de superusuario (root).

8. **Aceptabilidad psicológica**: La inclusión de seguridad en un sistema conlleva una penalización en su usabilidad. Se debe buscar el equilibrio entre estas dos propiedades. Es esencial que la interfaz de usuario sea lo más fácil de usar posible, ya que con ello los usuarios podrán aplicar los mecanismos de protección adecuadamente y los errores se minimizan.

Desarrollo de aplicaciones móviles

El desarrollo de aplicaciones para aparatos móviles (smartphones, tablets, etc.) es distinto al desarrollo de aplicaciones para equipos de escritorio. Hay que partir considerando la forma de interacción con éstos: Ya no se usa la pantalla para ver información, el teclado para ingresar información, y el mouse para apuntar a información. En los dispositivos móviles, es la pantalla la que sirve para las tres funciones mencionadas. Además, esta pantalla es considerablemente más pequeña que la de los equipos de escritorio, y el dispositivo para apuntar e ingresar información al dispositivo es el dedo del usuario, cuya precisión es mucho menor a la del puntero del mouse, ya que la superficie relativa del espacio de la pantalla es más pequeña.

Otro aspecto importante que diferencia a los aparatos móviles de los equipos de escritorio son los recursos disponibles como parte del sistema para aplicaciones. En los sistemas de escritorio, se tiene acceso a disco duro, Internet, periféricos mediante USB, Bluetooth, etc. En el caso de los sistemas operativos móviles se puede acceder también al GPS del teléfono, lista de direcciones, lista de llamadas, mensajes SMS/MMS, brújula u otros sensores integrados al equipo, etc.

Finalmente, está el tema de la plataforma. Cada aparato tiene un sistema operativo el cual es derivado de una plataforma móvil, la cual funciona mediante una API[7] a la cual acceden los desarrolladores mediante un SDK[8], el cual funciona en base a uno o varios lenguajes de programación. Hoy en día, existen cuatro formas de desarrollar

[7] Application Programming Interface. Es el conjunto de rutinas, protocolos y herramientas que permiten el desarrollo de aplicaciones.

[8] Software Development Kit: Es un conjunto de herramientas (lenguajes, compiladores, editores, etc.) que permite el desarrollo de aplicaciones para una determinada plataforma.

aplicaciones móviles para estas plataformas: Web, nativa, híbrida y web embebida. A continuación, una descripción de cada una de éstas [12].

Aplicaciones web móviles

Esta forma de desarrollar aplicaciones está orientada principalmente a crear una versión del sitio web principal de la organización, pero en una versión más pequeña y accesible para su uso en los browsers integrados a las plataformas móviles. Usualmente requieren de la combinación entre HTML, CSS y un fuerte uso de Javascript para su desarrollo.

Una importante limitación de este tipo de aplicaciones es que no requieren el uso de las API provistas por la plataforma móvil, si es que esto no es provisto por el browser. Por ejemplo, una aplicación web podría acceder a la información del GPS del aparato a través del browser, pero no podría acceder al almacén de mensajería SMS.

Aplicaciones nativas

Las aplicaciones nativas son aquellas que aprovechan todas las funciones provistas por las API incorporadas en el sistema operativo del teléfono. Los desarrolladores de cada plataforma ponen SDK's a disposición de los desarrolladores de aplicaciones, los cuales permiten que estos últimos puedan crear aplicaciones las cuales pueden acceder a recursos del aparato mediante un modelo de permisos definido.

Aplicaciones híbridas

En este caso, los desarrolladores tienen a disposición un SDK multiplataforma, en el cual desarrollan sus aplicaciones mediante una API ofrecida por el SDK, generalmente basada en HTML/CSS/Javascript. Estas aplicaciones se empaquetan dentro de un contenedor nativo, mediante el cual interactúan con la API del sistema operativo del móvil. Esto permite que la aplicación pueda ser desarrollada una sola vez, y desplegada en múltiples plataformas móviles. Sin embargo, tiene la desventaja que, al ser una aplicación web, su funcionalidad es limitada en comparación con una aplicación nativa. Además, existe una penalización en términos de performance por la interacción entre la aplicación, el contenedor, y la API del sistema operativo.

Aplicaciones web embebidas

Una aplicación embebida es una mezcla entre una aplicación web y una aplicación nativa. Esto se logra mediante el uso de un componente llamado WebView, el cual permite incrustar un browser dentro de la aplicación [13]. Las aplicaciones de éste tipo permiten al usuario interactuar con un sitio web móvil. La diferencia con las aplicaciones híbridas se basa en la localidad: Mientras que en las aplicaciones híbridas existe una biblioteca nativa que hace uso de código HTML/CSS/Javascript incluido dentro de ellas, la aplicación web embebida usa WebView para llamar a un sitio web y permitir al usuario interactuar con éste directamente a través de la aplicación, sin necesidad de usar un browser.

Privacidad de información personal y del equipo móvil

Hoy en día la penetración de la computación móvil es tal, que manejamos mucha información en nuestros equipos: Datos personales, contactos, fotos, ubicación geográfica, etc. También dejamos lo que se conoce como *traza digital*[9]: Los sitios que visitamos, las redes inalámbricas a las cuales nos conectamos, las acciones que realizamos en las aplicaciones que ocupamos. Esta traza digital permite que las organizaciones que desarrollan aplicaciones móviles puedan hacer *profiling* acerca de nosotros: Esto quiere decir que no sólo pueden obtener de nosotros nuestro género, dirección física, edad, etc., sino que también pueden inferir cosas como nuestra religión, nivel educativo, costumbres de compra, tendencia política, preferencia sexual, etc., la mayor parte del tiempo sin nosotros saberlo. Esto ocurre en particular en aplicaciones gratuitas que usan publicidad para obtener ganancias, ya que los proveedores de publicidad (ej. Google), al tener múltiples clientes, fuentes de información, y usuarios, pueden determinar todos aquellos datos, incluso sin nuestro consentimiento [14].

Una de las formas de obtener información acerca de las personas es mediante el cruce de datos: obtención de información desde múltiples bases de datos, relacionadas mediante uno o más datos en común. Esto se conoce como *linkability*, y es una técnica utilizada sobre todo cuando un equipo se conecta a varias aplicaciones que recolectan información [15]. Los datos que pueden ser utilizados para el linkability son generalmente identificadores como el RUT de la persona, su correo electrónico, el IMEI[10] o

[9] En la literatura en inglés ésto se conoce como "digital exhaust", o "digital footprint", cuyas traducciones directas al español son inadecuadas para los propósitos de ésta tesis.

[10] El International Mobile Station Equipment Identity (IMEI) es un número que funciona como identificador único de un aparato de telefonía celular.

ANDROID_ID[11] del teléfono, etc., y pueden ser usados uno a uno o agrupados como pseudo-identificadores [16].

La plataforma Android

La plataforma Android nace con la adquisición de la empresa Android Inc., por parte de Google en 2005. Dos años después se crea la Open Handset Alliance con el objetivo de crear un sistema operativo de código abierto para celulares, el cual ve la luz en 2008, con el T-Mobile G1 [17].

Con los años ha crecido para convertirse en una popular plataforma de computación móvil, debido principalmente a su adopción por parte, tanto de fabricantes de hardware grandes como LG, Motorola, Samsung y otros, como de empresas más pequeñas, lo cual también ha permitido su adopción por parte de múltiples grupos adquisitivos.

Versiones

Android es una plataforma la cual ha progresado desde su lanzamiento, hasta convertirse en una alternativa no sólo para smartphones, sino también para tablets, relojes, y más recientemente, automóviles y televisores. Cada versión tiene un nombre (basado en algún dulce), fecha de lanzamiento, versión y nivel de API. Esta última característica es importante, ya que permite a los desarrolladores identificar con qué versiones de la API van a trabajar, y a cuales apuntar al momento del deployment y a cuales no [18].

Versión	Nivel(es) API	Nombre	Lanzamiento
1.5	3	Cupcake	Abril 2009
1.6	4	Donut	Septiembre 2009
2.0, 2.0.1, 2.1	5, 6, 7	Éclair	Octubre 2009
2.2, 2.2.3	8	Froyo	Mayo 2010
2.3 - 2.3.2, 2.3.3 - 2.3.7	9, 10	Gingerbread	Diciembre 2010
3.0, 3.1, 3.2	11, 12, 13	Honeycomb	Febrero 2011
4.0 - 4.0.2, 4.0.3 - 4.0.4	14, 15	Ice Cream Sandwich	Octubre 2011
4.1, 4.2, 4.3	16, 17, 18	Jelly Bean	Junio 2012
4.4	19, 20[12]	KitKat	Octubre 2013

[11] El ANDROID_ID es un valor asignado al teléfono al momento de su activación con Google, el cual permanece en el teléfono hasta que es formateado.

[12] El nivel API 20 es el mismo que el 19, pero en esta ocasión se les agregaron las *wearable extensions*, las cuales dan pie al desarrollo de aplicaciones para dispositivos vestibles, como relojes y similares [19].

Versión	Nivel(es) API	Nombre	Lanzamiento
5.0, 5.1	21, 22	Lollipop	Noviembre 2014
6.0	23	Marshmallow	Octubre 2015

Algunas características relevantes de las diferentes versiones de Android son las siguientes [20]:

- Cupcake: Mejoras de interfaz de usuario, uso de acelerómetros, integración de teclado en pantalla.
- Donut: Búsqueda rápida en Google, soporte para VPN y WiFi, recolección de datos de uso de batería.
- Éclair: Soporte Bluetooth, manejo de múltiples cuentas Google y otras.
- Froyo: Tethering[13], soporte para Adobe Flash.
- Gingerbread: Soporte NFC, soporte para pantallas grandes, como las de tablets.
- Honeycomb: Versión sólo para tablets, la cual incluye soporte para dispositivos externos USB.
- Ice Cream Sandwich: Autentificación mediante reconocimiento facial, mejoras en gráficos y rotación de pantalla.
- Jelly Bean: Uso de Google Wallet (pagos con tarjeta de crédito mediante NFC), Google Now (Asistente personal), nuevos modos de cámara.
- KitKat: Integración de nueva máquina virtual ART en forma experimental (ver la sección "Runtime" para más información).
- Lollipop: Reemplazo definitivo de Dalvik por ART (ver "Runtime"), soporte para CPU's de 64 bits, mejoras de uso de batería, entrada y salida de audio mediante dispositivos USB.
- Marshmallow: Soporte para autentificación biométrica (impresión dactilar), modo "No Molestar", nuevo modelo de permisos basado en autorización de uso de recursos posterior a la instalación de aplicaciones.

Cabe recordar que, al ser un proyecto opensource, el código fuente de cada nueva versión de Android es lanzado a la red mediante el proyecto Android Open Source Project (AOSP), y es tomado por desarrolladores los cuales crean versiones alternativas como Cyanogenmod, los cuales proveen a los usuarios el acceso a root que las versiones distribuidas oficialmente no traen [21].

[13] El *tethering* permite al dispositivo compartir su conexión a Internet, ya sea mediante USB o funcionando como un pequeño Access Point WiFi.

Arquitectura

La arquitectura de la plataforma Android se basa en una pila (software stack) a través de la cual permite que las aplicaciones aprovechen los recursos del dispositivo móvil, como se observa en la figura siguiente:

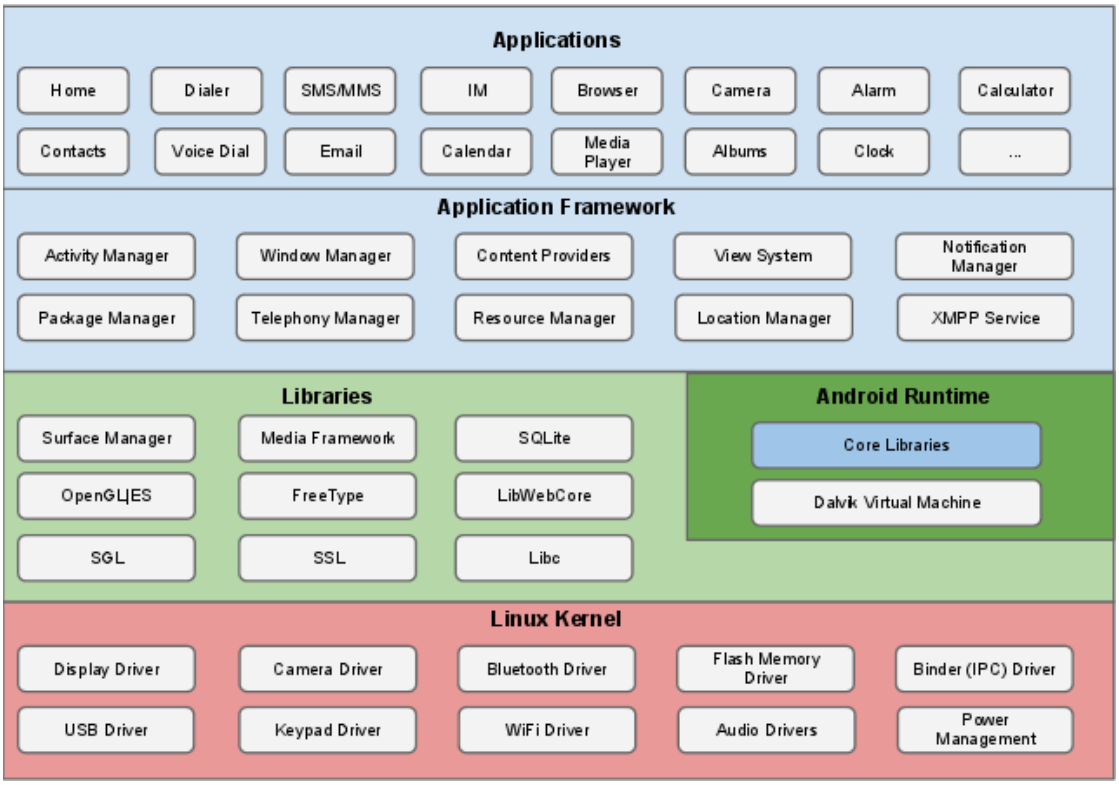


Figura 2.1: Arquitectura de la plataforma Android.

Kernel

Todo parte en el Kernel[14]. Android está basado en el Kernel Linux 2.6 parchado de manera tal que pueda funcionar en teléfonos. El Kernel contiene una serie de drivers[15] los cuales permiten el uso de diferentes componentes del teléfono, como Bluetooth, GPS, etc. Además, el kernel actúa como una capa de abstracción entre el hardware y las demás capas de la arquitectura.

[14]Software fundamental de todo sistema operativo el cual permite la interacción de las aplicaciones con el hardware del sistema, mediante bibliotecas conocidas como *drivers*.

[15]Un driver es una biblioteca de software la cual permite al kernel realizar operaciones sobre un recurso específico de hardware. Por ejemplo, en el caso de los dispositivos móviles modernos, estos contienen un sistema Bluetooth. Para poder usar Bluetooth y realizar las operaciones que éste ofrece, se debe ocupar la API que ofrece un driver específico para ese chip Bluetooth.

Bibliotecas

Android incluye una serie de bibliotecas las cuales permiten el manejo de diferentes tipos de datos y son específicas para cada hardware incluido en el dispositivo. Por ejemplo, la biblioteca *Media Framework* concentra los codecs[16] que permiten la reproducción de audio y video. La biblioteca SQLite permite el almacenamiento de información en bases de datos basadas en un pequeño archivo, en vez del uso de servidores RBDMS[17]. La biblioteca OpenGL permite la generación de gráficos 2D y 3D. Y así sucesivamente.

Runtime

Dentro de la misma capa de las bibliotecas se encuentra el runtime[18], el cual está conformado por los siguientes componentes:

- Máquina Virtual (Virtual Machine): Este componente es una versión de la máquina virtual Java (JVM) optimizada para el uso en dispositivos móviles, cuya potencia de procesamiento y memoria son más bajas que las de sus contrapartes de escritorio. A diferencia de la JVM, ésta no ejecuta archivos `.class`, sino archivos `.dex`, compilados a partir de los primeros, los cuales ofrecen mayor optimización en entornos de bajos recursos como los de los dispositivos móviles. Hasta la versión 4.4 la máquina virtual era Dalvik, pero problemas legales y mejoras propuestas (como paso de tipo de compilación, de JIT[19] a AOT[20]) derivaron en el cambio de la máquina virtual a ART en la versión 5.0 de la plataforma.
- Bibliotecas núcleo (Core Libraries): Éstas permiten a los desarrolladores hacer aplicaciones mediante el uso del lenguaje de programación Java. En su mayor parte son réplicas de las bibliotecas Java Standard Edition (Java SE).

Marco de aplicaciones

El marco de aplicaciones (Application Framework) provee servicios de alto nivel a las aplicaciones mediante clases Java. Éstos manejan funciones básicas del teléfono como

[16] Un codec es un programa que puede codificar y decodificar medios como audio y video.

[17] Relational Database Management System. Ejemplos de tales sistemas son Oracle, MySQL, PostgreSQL, etc.

[18] El runtime es el ambiente que propicia la ejecución de programas. En el caso de lenguajes como Java, permite la compilación de programas multiplataforma a código específico para la plataforma objetivo.

[19] Just-In-Time Compilation: El software se compila al momento de su ejecución.

[20] Ahead-Of-Time Compilation: La aplicación se compila al momento de ser instalada en el dispositivo.

gestión de llamadas, alarmas, notificaciones, etc., las cuales son accesibles por las aplicaciones.

Aplicaciones

Las aplicaciones creadas por los desarrolladores, junto con otras aplicaciones base, se encuentran en esta capa. Las aplicaciones base son el browser, el marcador (aplicación que cumple funciones de llamada de voz), el almacén de contactos telefónicos, el gestor de mensajes de texto, etc.

Almacenamiento

Android define dos tipos de almacenamiento: Un área de almacenamiento interno, la cual agrupa los espacios de almacenamiento de datos por cada aplicación instalada en el sistema, los cuales inicialmente sólo pueden ser accesibles por la propia aplicación. El otro tipo de almacenamiento es externo: Es un área del sistema de archivos accesible por aplicaciones, y es en esta área donde se guardan los archivos descargados, fotos, video, etc. [22].

Seguridad y privacidad en Android

Permisos

La seguridad de la plataforma Android se basa en un modelo de acceso discrecional (DAC) caracterizado por su modelo de permisos y su sistema de aislamiento (sandbox). Cada aplicación instalada en el sistema es aislada usando funciones propias del kernel de Linux: Cada aplicación tiene su propio usuario Linux, lo cual permite que sus datos sean sólo accesibles por la aplicación, y cualquier instancia de ejecución (proceso) de ésta le pertenezcan a ese usuario, impidiendo el acceso de otros procesos a sus datos.

Las aplicaciones tampoco pueden acceder a recursos fuera de su sandbox por defecto. Para acceder a ellos, requieren obtener permisos de una de dos formas:

- Si son aplicaciones firmadas por los fabricantes del hardware, cuentan con los permisos por defecto.
- Si no lo son, el usuario le entrega los permisos solicitados al momento de instalar la aplicación.

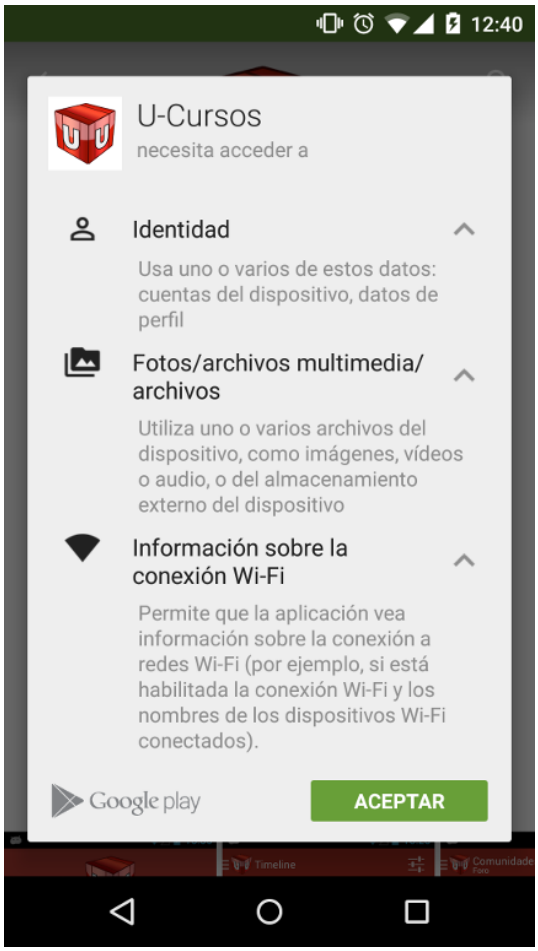


Figura 2.2: Solicitud de permisos al momento de la instalación de la aplicación.

Estos permisos son los que habilitan a la aplicación a usar recursos como GPS, archivos en el almacenamiento externo, etc. [23]. Android ofrece aproximadamente 130 permisos, de los cuales el desarrollador de la aplicación puede solicitar al usuario del aparato los que necesite o quiera a discreción, y el usuario debe aceptarlos todos, si no, no podrá utilizar la aplicación. Además, estos permisos se hacen valer en forma silenciosa cada vez que la aplicación lo requiera, vale decir, no se notifica de ninguna manera al usuario cada vez que la aplicación hace uso de algún recurso autorizado previamente. [24].

La situación antes descrita debería cambiar con la introducción de Android Marshmallow (6.0) a finales de 2015, versión en la cual el modelo de permisos de la plataforma cambió a un modelo *opt-in*, en el cual al momento de instalar la aplicación, no se le preguntará nada al usuario, sino que en el momento en que ésta requiera el acceso a algún recurso del sistema, preguntará al usuario si acepta o rechaza el uso de tal recurso. Este permiso lo podrá pedir una vez o solicitarlo para siempre (con la posibilidad de

revocarlo en forma futura) [25]. A pesar de ello, al momento de la publicación de esta tesis, sólo el 13.3 % de equipos Android tienen la versión 6.0, y en su gran mayoría son dispositivos tipo *high-end*, es decir, equipos de alta tecnología y alto precio [26].

Fragmentación

Al ser Android una plataforma opensource, esto significa que cualquier proveedor de hardware móvil puede crear equipos basados en la plataforma, con sus propias configuraciones, añadiendo características creadas por el mismo proveedor. Esto causa un fenómeno conocido como fragmentación, el cual se manifiesta en que no hay una forma estandarizada de Android, sino que existen múltiples configuraciones de los siguientes aspectos del equipo móvil [27]:

- Marcas de proveedores de hardware móvil
- Marcas de proveedores de telefonía celular (ISP)
- Tamaño y resolución de pantalla
- Capacidad de almacenamiento y procesamiento
- Versiones del sistema operativo y niveles de API
- Aplicaciones extras añadidas por el proveedor
- Sensores (movimiento, GPS, humedad, presión, temperatura, etc.)

Estos aspectos causan dificultades respecto de la distribución de aplicaciones, ya que al haber muchos equipos con diferentes niveles de API y configuraciones, el diseño, las pruebas y la implementación resultan complejas debido a que hay características que se encuentran en algunas versiones de Android y en otras no [28]. Y eso incluye las características de seguridad. Un ejemplo de ello es la Android Keystore, característica que permite almacenar en forma segura información como llaves de cifrado, la cual fue introducida en la API nivel 18. Versiones anteriores de Android no tienen esta característica.

Otro aspecto de seguridad que preocupa bastante es el hecho de que, por la fragmentación de proveedores de hardware móvil e ISP's, la propagación de nuevas características y actualizaciones de seguridad del sistema operativo pueden tomar meses, debido a que tales actualizaciones son integradas al código base y liberadas por Google al AOSP (o a otros canales definidos con ellos), dejando a discreción de los fabricantes de hardware móvil el integrar estas actualizaciones a sus respectivas versiones de Android para los equipos que ellos producen. A su vez, los ISP's que

venden directamente los equipos a los usuarios reciben las versiones de Android de los fabricantes de los equipos que venden, y ellos tienen discreción respecto de cuándo enviar (*push*) las actualizaciones del sistema operativo a los respectivos equipos de los usuarios. Esto causa que una vulnerabilidad grave de seguridad a nivel del sistema operativo sea parchada, no en días, sino en meses [29].

Instalación de aplicaciones maliciosas

Aquí hablaremos derechamente de malware. Google ofrece a los desarrolladores y usuarios la plataforma centralizada Play Store (para publicación e instalación de software, respectivamente) la cual se encarga de la distribución de aplicaciones para la plataforma. Si bien, la plataforma permite una adecuada distribución de software para equipos Android, la popularidad del sistema también atrae a creadores de malware [30]. A continuación analizaremos los problemas de este tipo que se producen con la distribución de malware, tanto en la Play Store, como fuera de ella.

Aplicaciones provenientes desde fuera de la Play Store

Además de poder instalar aplicaciones desde la Play Store, Android permite que el usuario instale aplicaciones desde otras fuentes (llamadas *fuentes desconocidas*), lo cual significa que el usuario podrá bajar directamente un paquete APK[21] e instalarlo en el equipo. Si el usuario decide permitir la instalación de aplicaciones desde fuentes desconocidas, el equipo advertirá al usuario de los riesgos de tal cambio de configuración:

[21] El paquete que contiene la aplicación a instalar en la plataforma Android se conoce como APK (Application Package File).

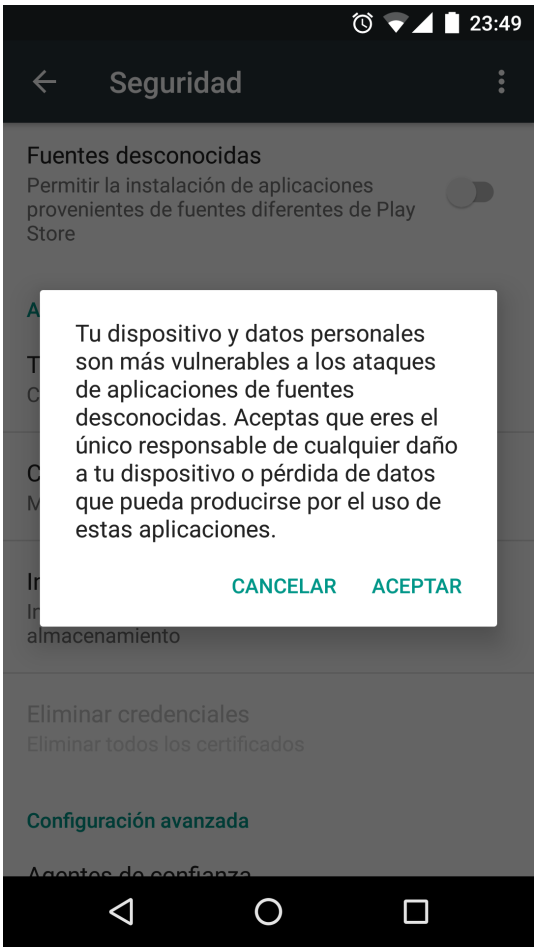


Figura 2.3: Advertencia al usuario antes de permitir la instalación de aplicaciones desde fuentes desconocidas (arriba al fondo se encuentra el selector que permite al usuario cambiar la configuración).

Tal advertencia tiene una razón de ser: Al no existir control sobre el software distribuido en fuentes desconocidas, existe el riesgo de que los usuarios instalen malware, muchas veces sin darse cuenta, el cual causa los siguientes efectos sobre el equipo [31]:

- Filtración de información personal o credenciales del usuario
- Generación de llamadas o SMS "Premium" (que implican costos adicionales de dinero al usuario)
- Envío de spam vía SMS
- Cambio de configuraciones automatizado y sin consentimiento del usuario
- Secuestro del teléfono o sus datos (Ransomware[22])

[22] El Ransomware es un tipo de malware el cual sus creadores *secuestran* el equipo (bloqueándolo) o los datos (cifrándolos con una llave pública, mientras los atacantes poseen la llave privada asociada), para posteriormente exigir un rescate (*ransom*).

Aplicaciones ofrecidas a través de la Play Store

El hecho de que exista un punto centralizado a través del cual se distribuya software, tampoco hace que éste sea más seguro. Para poder publicar una aplicación en la Play Store, se requiere pagar un *fee* de USD 25. La aplicación, antes de poder ser publicada, deberá ser firmada por el desarrollador y su publicación es casi inmediata [32]. Esto causa que, si bien, existe un monitoreo por parte de Google respecto de si los desarrolladores suben aplicaciones maliciosas a la Play Store, la remoción o bloqueo de tales aplicaciones es reactiva, y a veces tardía, como ocurrió con el caso de `DroidDream`, troyano diseñado para robar información personal y del teléfono, e instalar una puerta trasera en el equipo con el objeto de bajar más código malicioso e instalarlo en éste. En este caso la recomendación de Google es revisar bien los permisos antes de instalar aplicaciones en el teléfono [33].

Un caso más reciente es el de `Android.Xiny.19.origin`, reportado en enero de 2016, el cual es un troyano que afectó a más de 60 aplicaciones disponibles en la Play Store, principalmente juegos. El malware envía a sus servidores C2[23] información del teléfono, como el IMEI, información del operador móvil, versión del sistema operativo, etc., pero además instala código adicional en el equipo e instala y borra otros programas (si el usuario rooteó su equipo) [34].

Si bien, el la proporción de aplicaciones maliciosas en la Google Play Store es aún baja (menos del 1 % [31]), eso no significa que la tienda de aplicaciones esté libre de ser blanco de ser usada como proxy para atacar equipos que hacen uso de ella mediante malware.

Trabajo Relacionado

En esta sección se indicarán qué trabajos similares han habido relacionados con seguridad en aplicaciones móviles, y sus similitudes y diferencias con la presente tesis.

Antes de la revisión de tales trabajos, cabe señalar que una aplicación móvil opera en tres entornos, los cuales pueden ser utilizados como superficie de ataque[24] [36]:

[23] C2: También conocido como C&C, se refiere a **Command & Control**, esquema utilizado por los desarrolladores de malware para tomar control u obtener información de los equipos de sus víctimas en forma centralizada.

[24] La superficie de ataque es el conjunto de formas en las cuales un atacante puede causar daño a un sistema de información [35].

1. El aparato: En él se puede instalar malware nocivo, explotar alguna vulnerabilidad a nivel de sistema operativo, de sus aplicaciones pre-instaladas e instaladas, etc.
2. La red: El entorno de red al cual se conecta el equipo puede ser susceptible de ser monitoreado o penetrado, acción mediante la cual un atacante podría obtener información de transacciones o personal en tránsito, por ejemplo mediante un ataque Man-In-The-Middle[25], robo de sesiones, uso de redes Wi-Fi inseguras, etc.
3. El centro de datos: El servidor al cual se conectan las aplicaciones bancarias puede ser susceptible a ataques de negación de servicio, inyecciones de código, etc.

El análisis de Aranha y Cruz [37] es similar en su propuesta, sin embargo, la principal diferencia con la presente tesis es que se concentra principalmente en la parte de red, reportando varias vulnerabilidades relacionadas con la calidad de las conexión SSL/TLS de las aplicaciones analizadas en forma similar a como se hace aquí, y con el agregado de que ellos lograron exitosamente no sólo montar ataques MitM utilizando tales vulnerabilidades. No hay análisis de vulnerabilidades en el entorno del aparato (mal uso de cifrado en reposo, abusos de privilegios, abuso de logging, etc.) ni en el del centro de datos, mientras que la presente tesis analiza vulnerabilidades de los dos primeros entornos de la lista.

La investigación de IOActive [38] hace una revisión similar a la de esta tesis, esta vez para la plataforma iOS. El análisis es a 40 aplicaciones bancarias de alrededor del mundo, sin indicar cuales fueron. Las recomendaciones indicadas en sus conclusiones son similares a las entregadas en esta tesis, sin embargo, hay algunos temas que dan para discusión:

- Es destacable que el estudio recomiende el uso de cifrado de información sensible mediante las API criptográficas ofrecidas por la plataforma iOS. Esta tesis recomienda el uso de la Android Keystore, mecanismo similar para la plataforma impulsada por Google.
- Otra de las recomendaciones indicadas es la detección por parte de la aplicación de si el equipo ha sido *jailbreakeado*[26], pero no indica qué hacer con esa información: Si alertar al usuario de los riesgos del uso de un equipo jailbreakeado o bloquear la aplicación y restringir su uso en equipos jailbreakeados. En Chile, jailbreakear o rootear equipos está permitido por la ley [39], y una medida así establecería una postura adversarial del banco contra sus clientes.

[25] El ataque *Man-In-The-Middle* (también conocido como MitM) consiste en interponerse entre dos nodos de una conexión para leer, insertar o modificar los datos transferidos entre ambos.

[26] Proceso similar al de *rooting*, esta vez para la plataforma iOS.

- El estudio también recomienda el uso de obfuscación de código para retrasar a atacantes que busquen hacer ingeniería reversa de la aplicación. Cabe destacar que es un disuasivo bastante más efectivo que el de cifrado de código, sin embargo no es una medida en la cual confiar completamente en términos de seguridad, como lo indica el principio de Diseño Abierto (Ver la sección "Principios de Seguridad de Sistemas").

3

Definición del problema

Objetivo general

El objetivo general de ésta tesis es proponer una taxonomía de malas prácticas de seguridad y una clasificación asociada que permita categorizar las aplicaciones móviles bancarias de la plataforma Android desarrolladas por bancos chilenos. La taxonomía considera malas prácticas de seguridad típicamente observadas en procesos de desarrollo de software.

Ésta taxonomía se realizará en base a un proceso de ingeniería reversa a dichas aplicaciones bancarias. La clasificación mencionada ayudará a proveer evidencia cuantitativa (por ejemplo, qué porcentaje de las aplicaciones caen en una mala práctica determinada) respecto de las consecuencias de no seguir buenas prácticas de desarrollo de software seguro.

Objetivos específicos

Los objetivos específicos de la presente tesis son los siguientes:

1. Evaluar la seguridad de la plataforma Android en el contexto de la seguridad de aplicaciones desarrolladas para ésta, en aspectos tales como las herramientas

disponibles en su API, su modelo de privilegios de acceso a recursos, y los riesgos asociados con su fragmentación, mal uso de privilegios y desarrollo inseguro de aplicaciones.

2. Generar un procedimiento que busque potenciales falencias específicas de seguridad en aplicaciones Android a partir de la ingeniería reversa aplicada a sus respectivos paquetes APK, y al análisis de su contenido.
3. Clasificar las aplicaciones móviles bancarias examinadas en base a los tipos de malas prácticas encontradas y determinar en qué etapa del SDLC fueron introducidas, y usar esta clasificación para conjeturar datos estadísticos respecto de las malas prácticas que causan las vulnerabilidades encontradas en aplicaciones móviles.
4. Presentar de manera formal la taxonomía de malas prácticas y sus consecuencias en las aplicaciones analizadas, e indicar qué buenas prácticas (controles) permiten el tratamiento de estas vulnerabilidades.
5. Generar recomendaciones de seguridad para el público general respecto del uso de aplicaciones móviles, a la luz de la evidencia encontrada en el trabajo de tesis.

En este punto, se considera importante clarificar los aspectos no cubiertos por ésta tesis. El principal es la realización de esta tarea sobre Android y no sobre iOS (también popular entre desarrolladores y usuarios de equipos móviles). La razón principal por la cual no se trabajará en iOS es la dificultad de obtener paquetes de aplicaciones a ser instaladas en los terminales de esa plataforma, lo cual causa un problema de oscuridad respecto de las implementaciones y el diseño de aplicaciones, lo cual haría dificultoso su análisis. Adicionalmente, Android es una plataforma abierta, implementada por varios vendedores de hardware móvil para múltiples segmentos de precios, y con ello, más asequible por una mayor cantidad de personas. De hecho, la cuota de mercado de la plataforma Android supera ampliamente a la de iOS, poseyendo aproximadamente el 80 % de la cuota de mercado de sistemas móviles a nivel mundial [40].

En la siguiente sección se mostrará la metodología utilizada para la obtención de los objetivos mencionados.

4

Metodología

Para poder generar una taxonomía, primero hay que determinar qué vulnerabilidades tienen las aplicaciones móviles bancarias chilenas existentes en la Google Play Store. Para ello se siguieron los pasos detallados a continuación.

Selección de aplicaciones a analizar

Para esta investigación, el criterio de selección de aplicaciones es el siguiente:

- La aplicación debe pertenecer a un banco nacional, o banco extranjero con operaciones en Chile.
- La aplicación debe encontrarse disponible en la Google Play Store.
- La aplicación debe ser una aplicación de banca móvil. No se incluye en ésta categoría aquellas aplicaciones de pago móvil o billeteras móviles.

En base a este criterio, las aplicaciones seleccionadas son las siguientes:

1. Mi Banco de Chile (Banco de Chile)
2. BBVA
3. BCI Móvil (Banco de Crédito e Inversiones)

4. CorpBanca
5. Banco Falabella Chile
6. Itaú Chile (Banco Itaú Chile)
7. Banca Personas (Banco Santander Chile)
8. ScotiaBank Chile - Banca Móvil (Scotiabank)
9. Banco Security
10. Banco BICE

Respecto de esta selección, cabe consignar lo que corresponde a conglomerados bancarios. Por ejemplo, Banco de Chile pertenece a un conglomerado bancario el cual incluye al Banco Edwards, cuya aplicación también se encuentra disponible en la Google Play Store. Sin embargo el análisis de esa aplicación fue obviado, ya que ambas aplicaciones (Mi Banco de Chile y Mi Banco Edwards) sólo difieren en elementos gráficos y la dirección del servidor al cual se conectan. La misma situación ocurre con las aplicaciones BCI Móvil, TBanc Móvil y BCI Nova Móvil, en cuyo caso se obvió el análisis de las dos últimas aplicaciones.

Determinación de vulnerabilidades a buscar

Se desea buscar en las aplicaciones analizadas los siguientes aspectos:

1. Dependencias externas: bibliotecas (JAR, JNI, etc.) o frameworks híbridos que tengan vulnerabilidades cubiertas en la base de datos de vulnerabilidades de software CVE[1].
2. Uso de transporte vía texto plano: Si la aplicación usa HTTP para conectarse a alguno de los servidores y qué información transmite a y recibe de ellos.
3. Uso de transporte cifrado mediante HTTPS: Si el servidor es el mismo usado por la aplicación web del banco, Si las conexiones HTTPS con el servidor tienen vulnerabilidades importantes:
    - Protocolos obsoletos o inseguros, como SSL 2 y 3.
    - Algoritmos de cifrado inseguros, como RC4, DES, etc.
    - Firmas de certificados deprecadas, como SHA1[2].

[1]Common Vulnerabilities and Exposures. Es una base de datos que lleva un registro de vulnerabilidades descubiertas en paquetes de software (bibliotecas, software de escritorio/móvil/web). Está disponible para consulta y obtención en https://cve.mitre.org/

[2]Desde el año 2015 el uso de SHA1 es considerado inseguro por parte de los desarrolladores de browsers y clientes HTTP, quienes recomiendan que los certificados que usen ese algoritmo de firma sean renovados por certificados formados con SHA2 a partir del 2016 [2].

- Vulnerabilidades importantes en la implementación, como Heartbleed.
- Errores de configuración de certificados.

4. Tipo de conexión: Cómo se comunica la aplicación con el servidor: Mediante servicios web pesados (WSDL, XML-RPC, etc.) o livianos (JSON-REST), y si existe alguna vulnerabilidad asociada con el uso de tales tecnologías, como inyecciones, etc.
5. Almacenamiento de información sensible en el equipo: Tokens de identificación, cookies de sesión, etc., almacenados sin encriptar dentro del teléfono tanto en el log interno, base de datos de la aplicación, archivos de configuración, etc. También se busca saber, para el caso de que esa información sí sea almacenada en forma encriptada, si existen vulnerabilidades como mala gestión de llaves de cifrado, por ejemplo, uso de información del teléfono como llave de cifrado o uso de una llave de cifrado *hardcodeada* en el código fuente de la aplicación.
6. Permisos riesgosos: Uso de permisos inadecuados, como acceso a contactos, escritura en archivos fuera del área de almacenamiento de la aplicación, lectura del log interno del sistema, etc.
7. Servicios externos: Uso de servicios externos por parte de la aplicación, a los cuales ésta envía información, generalmente sin conocimiento por parte del usuario, como APM[3], MBaaS[4], o Analytics.
8. Timeout: El 62 % de los usuarios de plataformas móviles no bloquea sus equipos con una password, un PIN o un patrón [41]. Para este caso se desea saber si la aplicación implementa o no algún tipo de timeout que cierre la sesión después de un tiempo determinado.
9. Uso de autentificación por propiedad: Los bancos nacionales implementan sistemas de autentificación por propiedad, ya sea en la forma de un token OTP[5], o de una tarjeta de coordenadas. Originalmente este sistema fue pensado para la transferencia de dinero entre cuentas bancarias en la banca web, pero hay bancos que también lo usan como parte de un esquema 2FA para la autentificación de usuarios al momento de ingresar al sistema. Se busca determinar si se usa ese sistema en las aplicaciones analizadas.

[3]Application Performance Management. Este tipo de servicios permite monitorear la performance y disponibilidad de aplicaciones, lo cual implica el envío de datos desde la aplicación cada cierto tiempo mediante bibliotecas incluidas en el paquete instalable.

[4]Mobile Backend as a Service: Este servicio ofrece a los desarrolladores un servicio Cloud al cual conectar sus aplicaciones y despreocuparse de aspectos como la configuración de servidores, servicios push, etc.

[5]One-Time Password. Generalmente se conoce como *Digipass*, y es un dispositivo el cual genera cada cierto tiempo un número el cual está sincronizado con el banco y permite validar que el usuario está en posesión de éste al ser consultado por parte del sistema.

Ingeniería reversa de la aplicación y análisis de transporte

Las aplicaciones Android vienen empaquetadas en un archivo APK las cuales contienen lo siguiente:

- Manifiesto de la aplicación: En el archivo `AndroidManifest.xml` la aplicación declara la clase la cual es su punto de entrada, su nombre, los permisos que solicita, los componentes que ésta tiene (servicios, etc.), el nivel mínimo de API que requiere para funcionar, las bibliotecas internas que requiere para funcionar, etc.
- Clases de la aplicación: En el archivo `classes.dex` se encuentra el código ejecutable de la aplicación en formato DEX (Dalvik Executable Format), el cual es el resultado de la compilación de clases Java posteriormente optimizadas para la máquina virtual de Android (sea ésta Dalvik o ART).
- Recursos: Activos de la aplicación, como archivos de tipo de letra, imágenes, strings para múltiples idiomas, etc.
- Bibliotecas nativas: En el directorio `lib/` se almacenan las bibliotecas nativas compiladas para diferentes plataformas (ARM, x86, x86_64, MIPS) que pudieran ser requeridas por la aplicación.

El objetivo principal en esta etapa del análisis es la obtención del código fuente de la aplicación, sea éste extraído desde las clases o desde archivos Javascript ubicados en los recursos (en el caso de las aplicaciones híbridas). Esto se puede realizar mediante un proceso de decompilación. Para ello se creó una herramienta llamada **Sabar**, la cual es una recolección de varias herramientas:

1. **apk2gold**[6]: Ésta herramienta permite la automatización de la ingeniería reversa de un paquete APK, mediante los siguientes pasos:
    a. Descompresión del contenido del paquete, el cual se encuentra en formato ZIP.
    b. Decodificación de los archivos internos mediante el uso de la herramienta **apktool**[7].
    c. Conversión de las clases mediante el uso de la herramienta **dex2jar**, la cual convierte el contenido de `classes.dex` en clases Java (archivos `.class`).
    d. Decompilación de clases Java y conversión de éstas a código fuente mediante el uso de la herramienta **Java Decompiler**[8].

[6]https://github.com/injcristianrojas/apk2gold
[7]https://ibotpeaches.github.io/Apktool/
[8]http://jd.benow.ca/

2. Un analizador de permisos y sus riesgos asociados.
3. **jsbeautifier**: Usualmente, las aplicaciones híbridas obfuscan y comprimen el código Javascript incluido en ellas. JSBeautifier permite su desobfuscación y *embellecimiento* (formateado e indentación del código).
4. **Findbugs**: Analizador estático de clases Java.

Análisis estático del código fuente

Teniendo el código fuente y las clases a la mano, se pueden realizar dos análisis estáticos:

El primero es un análisis estático automatizado sobre las clases Java usando la herramienta Findbugs, combinada con un plugin para ésta llamado **Find Security Bugs**[9], el cual se especializa en buscar vulnerabilidades de seguridad y mejora las capacidades de Findbugs de encontrarlas.

La segunda es realizar un análisis estático manual del código fuente Java en busca de información como la siguiente:

- Uso de cifradores en reposo
- Envío de información a logs
- URL's de conexiones a servidores

Análisis de seguridad en el transporte de información

En la motivación de esta tesis se mencionó que un aspecto importante del uso de este tipo de aplicaciones, es si la conexión entre la aplicación y el servidor es segura. Sin embargo, al ser SSL/TLS una tecnología simple de implementar, el sólo uso de ella no es garantía de seguridad en el transporte [2], ya que se deben considerar dos aspectos fundamentales:

- El certificado debe ser manejado adecuadamente: No debe ser autofirmado, debe ser firmado con un algoritmo fuerte (SHA2 o superior), la cadena de certificación debe estar bien hecha, y la aplicación no debe bypassear su validación correspondiente.

[9] http://h3xstream.github.io/find-sec-bugs/

- La conexión debe estar bien configurada: No deben haber cifradores deprecados o inseguros, se deben usar versiones seguras de los protocolos TLS, se debe configurar el servidor para que use los mejores algoritmos de cifrado (ej. AES-GCM, ECDHE, etc.), se debe implementar Forward Secrecy (FS), etc.

Para poder probar si la conexión al servidor tiene un nivel adecuado de seguridad, lo primero es montar un proxy para interceptar todas las comunicaciones que se realizan hacia el servidor.

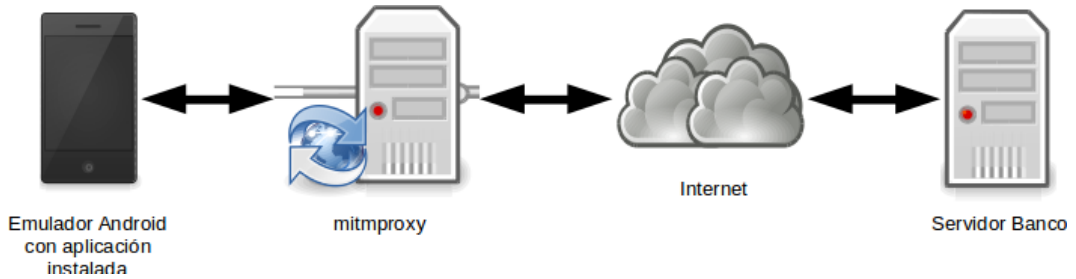


Figura 4.1: Configuración de proxy para pruebas de comunicación entre la aplicación y el servidor.

La aplicación se instala sobre una versión emulada de Android KitKat (versión 4.4, nivel de API 19)[ˆkitkat_noother] utilizando las herramientas incluidas en el Kit de Desarrollo para la plataforma (Android SDK). Posteriormente, en el mismo equipo donde se encuentra el emulador, se instala el software proxy Mitmproxy[10], el cual permite el análisis de tráfico HTTP que pase a través de él. Mitmproxy, además, permite el análisis de tráfico HTTPS mediante la emisión de un certificado raíz, el cual se instala en el ambiente Android emulado mediante un modo de la aplicación llamado "regular proxy" [42].

Cabe mencionar que se utilizó KitKat para las pruebas por dos razones: Primera, desde noviembre de 2014 hasta noviembre de 2016, ha sido la versión de Android con mayor porcentaje de uso entre los usuarios de la plataforma[11]. La segunda razón es que, a partir de Lollipop, no se puede configurar en Mitmproxy el modo regular proxy recién mencionado, y se requiere el uso de otros modos más complejos de configurar y operar.[12]

[10]https://mitmproxy.org/
[11]Ver http://www.droid-life.com/tag/distribution/
[12]Ver http://docs.mitmproxy.org/en/latest/modes.html

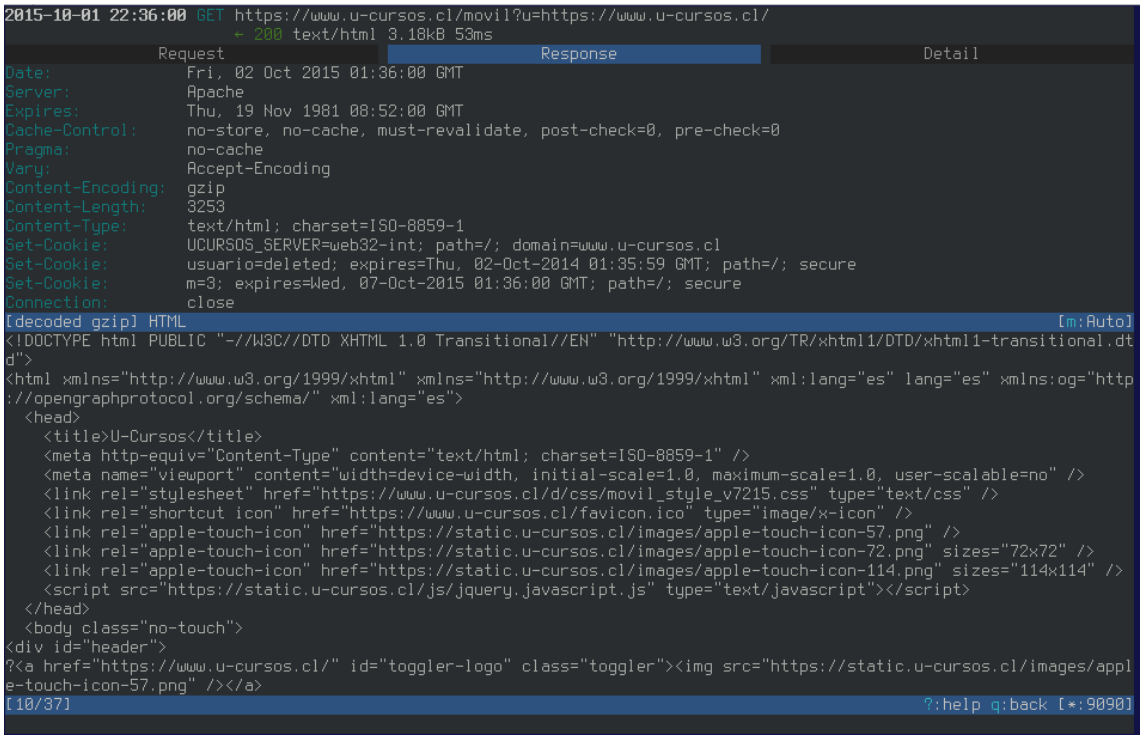


Figura 4.2: Mitmproxy interceptando tráfico HTTPS.

A través de la interceptación del tráfico de la aplicación con el servidor, es posible determinar la URL a la cual se conecta la aplicación. Teniendo esa información, se hace un análisis con la herramienta Qualys SSL Test[13] , la cual entrega un reporte completo del estado de los aspectos mencionados anteriormente, y finalmente *le pone nota* al servidor HTTPS.

Análisis pasivo del comportamiento de la aplicación

En esta etapa se monta un ambiente el cual permita analizar los siguientes aspectos de la aplicación en funcionamiento:

- Información almacenada en el área de almacenamiento de la aplicación.
- Información almacenada por la aplicación fuera de su área de almacenamiento.
- Información enviada al log del aparato.
- Información enviada y recibida al servidor mediante HTTP(S).

[13]https://www.ssllabs.com/ssltest/

Este ambiente se detalla en la siguiente figura:

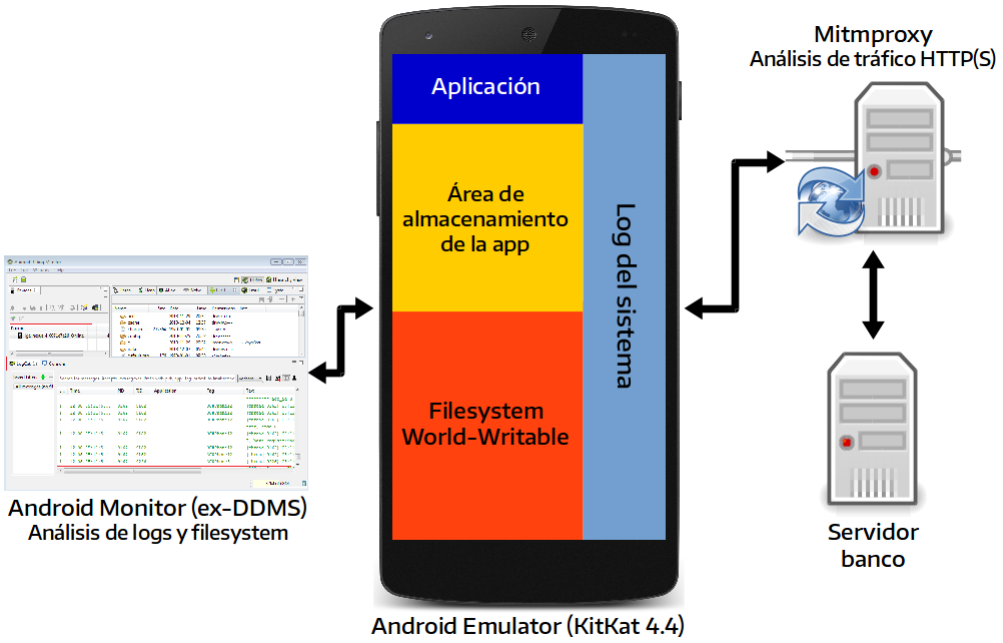


Figura 4.3: Configuración para análisis pasivo de aplicaciones móviles

Este ambiente es una extensión del ambiente mencionado en la sección anterior, ya que, además del uso de Mitmproxy para monitorear el tráfico de red hacia el servidor, se usa la herramienta Android Monitor, la cual permite acceder al sistema emulado para extraer y monitorear los demás aspectos antes mencionados.

Montado el ambiente, se requiere la cooperación de algún cliente del banco cuya aplicación se está analizando. Después de que el cliente acepta participar del experimiento, la aplicación se instala en el ambiente emulado y el cliente realiza las siguientes acciones en su cuenta mediante la aplicación:

1. Ingresar a su cuenta.
2. Revisar su cartola de cuenta corriente.
3. Revisar su estado de cuenta de tarjeta de crédito.
4. Realizar una transferencia de dinero a un tercero.
5. Mantener la aplicación abierta sin realizar acción alguna sobre ella durante un tiempo.

Confección de informes y de tesis

Vulnerabilidades y malas prácticas

Las vulnerabilidades encontradas en la sección se dividieron en las siguientes 25 categorías:

1. Acceso a contactos del usuario
2. Acceso a información acerca de otras aplicaciones en uso
3. Acceso al estado del teléfono
4. Escritura en el almacenamiento externo del teléfono
5. Lectura de logs del sistema
6. Uso de georreferenciación
7. Cadena de certificación HTTPS mal hecha
8. Carencia de Forward Secrecy
9. CRIME
10. Uso del cifrador RC4
11. POODLE y Uso del protocolo SSL3
12. Falta de protección ante downgrades de protocolo
13. No uso del protocolo TLS1.2
14. Abuso de logging (código de pruebas olvidado)
15. Actualización insegura de información (mediante HTTP)
16. Almacenamiento de información sensible en base de datos SQLite
17. Almacenamiento de información sensible en cache
18. Envío de información sensible a través de canal inseguro (HTTP)
19. Exceso de confianza en medidas anti-ingeniería reversa
20. Falta de timeout
21. Timeout mal implementado
22. Mal uso de cifrado en reposo
23. Mal uso de cifrado redundante
24. Potencial Tampering de datos
25. Uso innecesario del PAN de Tarjeta de Crédito en tránsito

Y en base a las vulnerabilidades encontradas, se infirió una taxonomía de malas prácticas incurridas por parte de los desarrolladores de aplicaciones móviles bancarias, la cual es la siguiente:

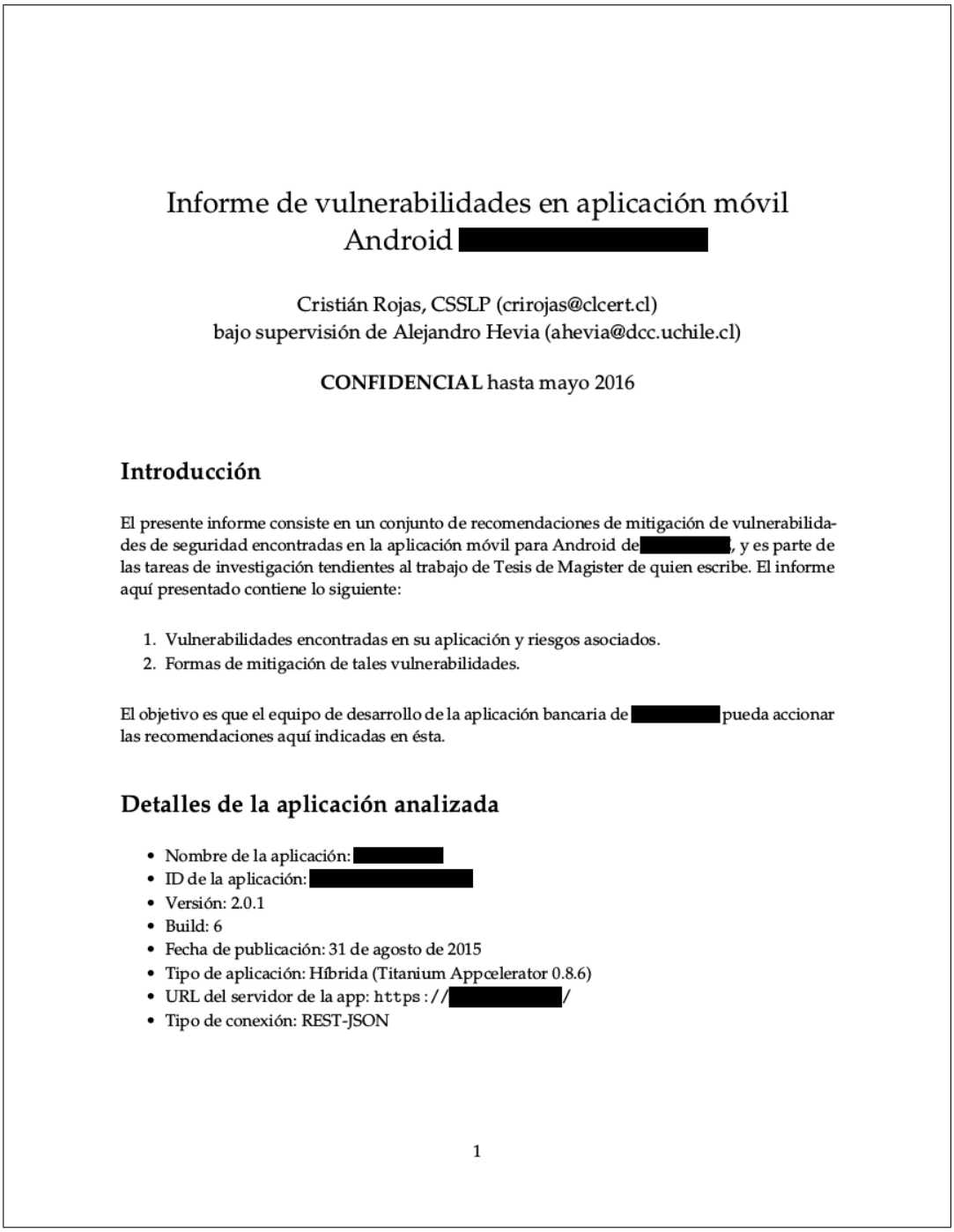
1. Abuso en el uso de información personal de los usuarios
2. Mal uso de cifrado
3. Falta de protección de información sensible almacenada en el teléfono
4. Falsa sensación de seguridad en el uso de SSL/TLS
5. Código abandonado
6. Uso de medidas anti-ingeniería reversa como medida de seguridad
7. Mala programación de medidas de seguridad
8. Abuso de privilegios

Los detalles de la categorización de vulnerabilidades y taxonomía de malas prácticas, los cuales incluyen información más detallada y formas de mitigación de vulnerabilidades y buenas prácticas para contrarrestar a las malas, se detalla en "Vulnerabilidades encontradas" y "Taxonomía de malas prácticas"

Generación de tesis e informe de vulnerabilidades a bancos

Mediante el uso de código Markdown y la herramienta Pandoc, se generó la presente tesis, y un informe personalizado para cada banco, el cual contiene:

- Una introducción.
- Algunos detalles de la aplicación analizada (ID de la aplicación, versión, fecha de publicación, tipo de aplicación, dirección del servidor al cual se conecta, tipo de conexión, etc.)
- Vulnerabilidades encontradas en la aplicación y su procedimiento de mitigación correspondiente.
- Recursos interesantes: Páginas con manuales de buenas prácticas, herramientas y bibliotecas API para mejorar la seguridad, etc.



Informe de vulnerabilidades en aplicación móvil Android [redacted]

Cristián Rojas, CSSLP (crirojas@clcert.cl)
bajo supervisión de Alejandro Hevia (ahevia@dcc.uchile.cl)

CONFIDENCIAL hasta mayo 2016

Introducción

El presente informe consiste en un conjunto de recomendaciones de mitigación de vulnerabilidades de seguridad encontradas en la aplicación móvil para Android de [redacted], y es parte de las tareas de investigación tendientes al trabajo de Tesis de Magister de quien escribe. El informe aquí presentado contiene lo siguiente:

1. Vulnerabilidades encontradas en su aplicación y riesgos asociados.
2. Formas de mitigación de tales vulnerabilidades.

El objetivo es que el equipo de desarrollo de la aplicación bancaria de [redacted] pueda accionar las recomendaciones aquí indicadas en ésta.

Detalles de la aplicación analizada

- Nombre de la aplicación: [redacted]
- ID de la aplicación: [redacted]
- Versión: 2.0.1
- Build: 6
- Fecha de publicación: 31 de agosto de 2015
- Tipo de aplicación: Híbrida (Titanium Appcelerator 0.8.6)
- URL del servidor de la app: `https://`[redacted]/
- Tipo de conexión: REST-JSON

1

Figura 4.4: Ejemplo de informe entregado a un banco cuya app fue analizada.

Entrega de informe de vulnerabilidades a bancos

Bajo los preceptos de la divulgación responsable [43], y teniendo en consideración que las vulnerabilidades aquí presentadas podrían ser utilizadas por inescrupulosos para

atacar bancos o a sus clientes, se diseñó el siguiente procedimiento de divulgación:

1. A cada banco se le entrega en forma confidencial su respectivo informe.
2. El contacto inicial y entrega se realizará a tomadores de decisiones, sean éstos:
    - Jefes de arquitectura
    - Jefes de desarrollo de la aplicación móvil
    - Encargados de seguridad de la información del banco (CISOs, etc.)
3. La entrega se realizará en forma personal, no a través de intermediarios, como departamentos de servicio al cliente, los cuales no dan ninguna garantía de que el informe llegará a las manos adecuadas.

En la siguiente sección se mostrarán los resultados obtenidos en esta investigación.

5

Resultados del análisis de aplicaciones móviles bancarias

En nuestra opinión, el estado general de la seguridad de las aplicaciones bancarias en Chile es bastante pobre. Se las promociona a sus clientes como aplicaciones seguras, sin embargo se han encontrado múltiples vulnerabilidades que hablan de una desprolijidad en cuanto a la preocupación por la seguridad de los usuarios de estas aplicaciones se refiere [44].

Uno de los principales problemas observados es el acceso a varios privilegios innecesarios en los teléfonos de los usuarios (listas de contactos, acceso al sistema de archivos del equipo, etc.). También se observan casos en los que los timeouts de cierre de sesión, o no son implementados o son mal implementados. Otro aspecto interesante es que los desarrolladores de estas aplicaciones probablemente ven a los equipos móviles como verdaderas cajas fuertes de las cuales nadie podría extraer absolutamente ningún tipo de información. Por eso se observan problemas como almacenamiento en texto plano de información, ya sea en la cache o en archivos SQLite.

La tabla 5.1 muestra en forma anonimizada qué vulnerabilidades fueron encontradas, y cuáles aplicaciones móviles incurren en tales vulnerabilidades.

Cuadro 5.1: Vulnerabilidades encontradas y bancos en los que fueron encontradas

ID	Vulnerabilidad	Banco 1	Banco 2	Banco 3	Banco 4	Banco 5	Banco 6	Banco 7	Banco 8	Banco 9	Banco 10	Total
V1	Acceso a contactos del usuario			Si		Si	Si		Si			4
V2	Acceso a información acerca de otras aplicaciones en uso	Si	Si				Si					3
V3	Acceso al estado del teléfono			Si								1
V4	Escritura en el almacenamiento externo del teléfono	Si	Si			Si	Si	Si	Si	Si	Si	8
V5	Lectura de logs del sistema		Si									1
V6	Uso de georreferenciación	Si	Si	Si	Si	Si	Si	Si		Si	Si	9
V7	Cadena de certificación HTTPS mal hecha	Si	Si	Si		Si	Si	Si		Si	Si	8
V8	Carencia de Forward Secrecy	Si	Si	Si	Si		Si	Si	Si	Si		8
V9	CRIME								Si			1
V10	Uso del cifrador RC4	Si	Si	Si	Si		Si		Si	Si	Si	8
V11	POODLE y Uso del protocolo SSL3				Si				Si	Si		3
V12	Falta de protección ante downgrades de protocolo	Si	Si									2
V13	No uso del protocolo TLS1.2				Si							1
V14	Abuso de logging (código de pruebas olvidado)	Si	Si						Si		Si	4
V15	Actualización insegura de información (mediante HTTP)					Si						1
V16	Almacenamiento de información sensible en base de datos SQLite	Si	Si		Si			Si	Si			5
V17	Almacenamiento de información sensible en cache				Si			Si				2
V18	Envío de información sensible a través de canal inseguro (HTTP)										Si	1
V19	Exceso de confianza en medidas anti-ingeniería reversa		Si				Si					2
V20	Falta de timeout							Si				1
V21	Timeout mal implementado	Si		Si		Si				Si		4
V22	Mal uso de cifrado en reposo			Si					Si			2
V23	Mal uso de cifrado redundante						Si					1
V24	Potencial Tampering de datos	Si	Si	Si	Si	Si	Si	Si	Si	Si	Si	10
V25	Uso innecesario del PAN de Tarjeta de Crédito en tránsito								Si			1

Otro aspecto preocupante es que se confía en demasía en la seguridad en el transporte (HTTPS), pero se ignora que una configuración HTTPS de baja calidad puede ser tan insegura como el no tenerla del todo. Al igual que como ocurre muchas veces con las páginas web, las conexiones HTTPS a los centros de datos de las aplicaciones móviles bancarias chilenas adolecen de serios problemas de calidad, lo cual abre la posibilidad de ataques tipo Man-In-The-Middle. Esto se observa en las notas obtenidas a través del análisis externo mediante la herramienta Qualys SSL Test. Hoy en día, nada menor a una nota A es aceptable, considerando el volumen de conexiones que se generan día a día y los ambientes a través de los cuales los usuarios se conectan a sus bancos (ej. cafés, hogares, empresas, puntos de acceso Wi-Fi abiertos…). La tabla 5.2 indica los niveles de calidad de las conexiones de los bancos analizados.

Cuadro 5.2: Tabla de calidad de conexiones HTTPS por aplicación bancaria

	Banco 1	Banco 2	Banco 3	Banco 4	Banco 5	Banco 6	Banco 7	Banco 8	Banco 9	Banco 10
Nota Qualys SSL Test	F	F	C	C	A	C	B	C	F	B

Estas vulnerabilidades son producto de una serie de malas prácticas derivadas principalmente por ignorancia de parte de los desarrolladores y encargados de tales proyectos, quienes no obtienen la educación adecuada en seguridad, ya sea en cursos, en sus instituciones educativas o en la documentación de los lenguajes y frameworks utilizados. Esto se traduce en problemas como mal uso de cifrado, abuso de privilegios, falsa sensación de seguridad al utilizar herramientas de protección, abuso en el uso de información personal de los usuarios, etc. En el cuadro siguiente se observa la incidencia de cada mala práctica por banco.

Cuadro 5.3: Malas prácticas por aplicación bancaria

Mala práctica	Banco 1	Banco 2	Banco 3	Banco 4	Banco 5	Banco 6	Banco 7	Banco 8	Banco 9	Banco 10	Total
Abuso en el uso de información personal de los usuarios	Si	Si	Si		Si	Si	Si	Si	Si	Si	9
Mal uso de cifrado			Si			Si		Si			3
Falta de protección de información sensible almacenada en el teléfono	Si	Si	Si	Si	Si	Si	Si	Si	Si	Si	10
Falsa sensación de seguridad en el uso de SSL/TLS	Si	Si	Si	Si	Si	Si	Si	Si	Si	Si	10
Código abandonado	Si	Si						Si		Si	4
Uso de medidas anti-ingeniería reversa como medida de seguridad		Si				Si					2
Abuso de privilegios	Si	Si	Si		Si	Si	Si	Si	Si	Si	9

En las secciones siguientes se expondrá el detalle, tanto de las vulnerabilidades encontradas, las malas prácticas incurridas, y recomendaciones de seguridad para paliar tales malas prácticas y vulnerabilidades.

6

Vulnerabilidades encontradas

Como parte del análisis realizado, se encontró una serie de vulnerabilidades, las cuales están categorizadas en esta sección. Primero se explicará cada vulnerabilidad, y luego se indicará en forma anonimizada qué aplicación la tiene.

En el cuadro 5.1, se muestran las vulnerabilidades encontradas y cuales aplicaciones bancarias, en forma anonimizada, las tienen.

Vulnerabilidades asociadas a abuso de permisos

Se observaron algunas vulnerabilidades asociadas al abuso del modelo de permisos expuesto en la sección "Permisos" en "Seguridad y privacidad en Android". Los permisos mal utilizados son los siguientes:

V1: Acceso a contactos del usuario

En este caso, la aplicación hace uso de cualquiera de los siguientes permisos:

- `android.permission.READ_CONTACTS`: Permite que la aplicación lea la información de contactos del teléfono, mediante objetos llamados proveedores, los

cuales permiten acceder a diferentes tipos de almacenes de contactos, como la tarjeta SIM o servicios online sincronizables (ej. Google, Outlook, etc.)
- `android.permission.WRITE_CONTACTS`: Permite que la aplicación agregue o modifique contactos usando los proveedores antes mencionados.

El uso de este permiso es peligroso, ya que supone un problema de privacidad para el usuario al acceder a a información de sus contactos personales, sin indicar explícitamente para qué requiere acceder a, o modificar tal información [45].

V2: Acceso a información acerca de otras aplicaciones en uso

El permiso `android.permission.GET_TASKS` permite a la aplicación acceder a información de otras aplicaciones que estén ejecutándose en el equipo. Hay aplicaciones las cuales requieren de este permiso, por ejemplo, las llamadas *task killers*, que son aplicaciones que permiten "limpiar" la memoria del teléfono cerrando aplicaciones que lleven un buen tiempo sin funcionar. Además, una aplicación maliciosa podría obtener información acerca de qué aplicaciones están instaladas en el teléfono y enviarla a servidores externos.

Estos potenciales problemas causaron de que Google retirara este permiso en la API versión 21 (Lollipop), y además, a partir de esa versión de la plataforma, toda instrucción que hace uso de tal permiso no funciona.

V3: Acceso al estado del teléfono

El permiso `android.permission.READ_PHONE_STATE` permite a la aplicación acceder a información identificable del teléfono, como el IMEI. Las aplicaciones utilizan esa información como llave para cifrado simétrico (ver más acerca de esto en la vulnerabilidad V22), pero también la envían a proveedores APM para breadcrumbing[1].

V4: Escritura en el almacenamiento externo del teléfono

El teléfono incluye un área de almacenamiento externo, el cual no tiene las protecciones incluidas en el área de almacenamiento correspondiente a la aplicación, y por lo tanto,

[1] Application Performance Management (APM) es un servicio remoto que ofrece a los desarrolladores la posibilidad de analizar el rendimiento del funcionamiento de una aplicación y reportar posibles errores en su funcionamiento durante el uso de ésta por parte de usuarios. Una de sus funciones es la de *breadcrumbing*, que permite monitorear en qué pantallas de la aplicación ha estado el usuario y qué acciones ha realizado sobre ellas.

cualquier archivo almacenado en esta área externa podría ser accesible por otras aplicaciones y a través de conexiones USB al teléfono. Este acceso se realiza mediante el permiso `android.permission.WRITE_EXTERNAL_STORAGE`.

V5: Lectura de logs del sistema

Android hace uso de un log interno para temas de performance o errores en aplicaciones. En la vulnerabilidad V14 se explica cómo muchas veces por olvido, se envía información sensible de los usuarios a este log.

Este abuso de permisos se basa en leer información enviada por otras aplicaciones o por el sistema operativo a este log interno, el cual no tiene cifrado ni mayor protección que el uso de este privilegio.

V6: Uso de georreferenciación

Una función ofrecida por las aplicaciones bancarias nacionales es la que permite al usuario obtener una lista o mapa de los cajeros Redbanc o sucursales del banco más cercanas al lugar en el que se encuentra. Para ello se requiere acceso a los sistemas de georreferenciación del teléfono, que son principalmente de dos tipos:

- Georreferenciación fina: Éste tipo de georreferenciación se basa en el uso de satélites GPS y se puede acceder a ello mediante el permiso `android.permission.ACCESS_FINE_LOCATION`. Su precisión es de entre 2 y 20 metros.
- Georreferenciación gruesa: Se basa en el uso de torres celulares y hotspots Wifi para entregar una ubicación aproximada del aparato. También se conoce como A-GPS (Assisted GPS) o Network Location Provider, y su precisión es de entre 100 y 200 metros. Se puede acceder a este tipo de georreferenciación con el permiso `android.permission.ACCESS_COARSE_LOCATION`[46].

El uso de georreferenciación fina supone dos problemas para los usuarios:

1. Un potencial compromiso de su privacidad. Al tener alta precisión, el permiso de acceso a este tipo de georreferenciación podría significar que una aplicación realice el rastreo de una persona y sus movimientos.

2. El GPS utiliza una cantidad considerable de energía de batería del móvil en comparación con el A-GPS.

Dado que la función de localización de cajeros y sucursales no requiere mayor precisión, la recomendación en este caso es quitar el permiso `ACCESS_FINE_LOCATION` y dejar el `ACCESS_COARSE_LOCATION`. Además del beneficio para la privacidad del usuario, la georreferenciación gruesa consume menos energía de la batería del aparato y es mucho más rápida en la obtención de información geográfica de éste.

Vulnerabilidades asociadas a mal uso de conexiones HTTPS con el servidor

V7: Cadena de certificación HTTPS mal hecha

Esta vulnerabilidad tiene que ver principalmente con performance y completitud. Los clientes HTTPS contienen un almacén de certificados confiables de autoridades certificadoras (CA[2]), sin embargo no son éstas quienes emiten directamente los certificados para los sitios que los solicitan, sino que son CA's intermedias, dependientes de las principales (o raíz). Es por ello que los servidores deben ofrecer a los clientes, además del certificado correspondiente a sus sitios, los certificados asociados a las CA's intermedias que los emitieron.

Se han observado dos problemas que afectan a los servidores a los cuales las aplicaciones móviles se conectan:

- **Falta de certificados de CA's intermedias**: Los servidores no incluyen los certificados de la CA intermedia que los emite. Ante esto, los clientes buscan reconstruir la cadena completa de certificación, ya sea teniendo en caché la CA intermedia producto de alguna conexión anterior con un sitio que tiene un certificado emitido por ésta, o utilizando la información incluida en el certificado de la CA raíz. Sin embargo, ninguno de estos métodos es confiable, y además ralentiza la negociación.
- **Exceso de certificados ofrecidos**: También se observa que los servidores incluyen el certificado de la CA raíz en la cadena enviada al cliente, lo cual ralentiza la conexión [47].

[2] Autoridad Certificadora (Certificate Authority).

Adicionalmente se observó que varios de los certificados, tanto de servidor como de CA's intermedias, están firmados con el algoritmo SHA1. Ya se está volviendo factible la obtención de colisiones para tal algoritmo [48], por lo tanto, los fabricantes de browsers y clientes HTTP están deprecándolo para 2016, lo cual significará problemas de conexión desde las aplicaciones que hagan uso de HTTPS [49].

V8: Carencia de Forward Secrecy

Forward Secrecy[3] es una propiedad ofrecible por protocolos de intercambio de llaves en negociaciones de conexiones HTTPS, en la cual, si la llave privada del servidor es expuesta a futuro, esto no significará compromiso alguno para las llaves de sesión utilizadas en las sesiones del servidor con todos los clientes con los cuales éste se haya conectado. Para obtener esta propiedad, usualmente se utiliza el algoritmo Diffie-Hellman Efímero (DHE)[50].

La vulnerabilidad en este caso corresponde a la falta absoluta de inclusión del algoritmo DHE en los servidores. Esto por un tema de performance, ya que DHE es más lento que RSA a la hora de la negociación. Sin embargo, el uso de DHE con Curva Elíptica (ECDHE) permite superar tales problemas y mejorar la seguridad de las conexiones [51].

V9: CRIME

CRIME (Compression Ratio Info-leak Made Easy) es una vulnerabilidad que afecta a cookies transmitidas vía HTTPS con la función de compresión TLS[4]. Hoy en día, la mayoría de los servidores modernos tienen desactivada la compresión TLS, pero aún quedan algunos que la soportan [52].

V10: Uso del cifrador RC4

EL cifrador RC4 es utilizado en conexiones TLS, el cual tiene múltiples vulnerabilidades, y debido a que éstas vulnerabilidades potencialmente permiten recoger fragmentos de información sensible desde conexiones TLS que hacen uso de tal cifrador [47], la IETF prohibió su uso [53].

[3]También conocido en la literatura como Perfect Forward Secrecy.

[4]Cabe mencionar que existen 2 formas de compresión de transmisión en conexiones HTTPS: La compresión propia de HTTP y la de TLS. CRIME afecta a la compresión TLS.

V11: POODLE y uso del protocolo SSL3

POODLE (Padding Oracle On Downgraded Legacy Encryption) es una vulnerabilidad tipo Man-In-The-Middle la cual permite extraer información de cookies de sesión mediante un código Javascript malicioso [54]. Ésta vulnerabilidad, y el hecho de que SSL3 es un protocolo que tiene casi 20 años en funcionamiento hacen que su eliminación de la lista de protocolos ofrecidos por los servidores HTTPS sea fuertemente encarecida[5].

V12: Falta de protección ante downgrades de protocolo

Un downgrade de protocolo se produce cuando un adversario en modo Man-In-The-Middle intenta interferir con la negociación de una conexión HTTPS con el objetivo de influenciar en sus parámetros, cosa de poder forzar el uso de un cifrador o protocolo más débil. Uno de los usos de este ataque es el de forzar una conexión TLS1.0 a SSL3 y así aprovechar la vulnerabilidad V11.

Para impedir este tipo de ataques, los servidores y clientes están en proceso de implementar una señal, `TLS_FALLBACK_SCSV`, la cual es enviada por el cliente al servidor si éste detecta un downgrade de protocolo. Si el servidor tiene implementada la recepción de tal señal, corta la conexión para prevenir el ataque [55]. Se sabe que la biblioteca OpenSSL versión 1.0.1j y superiores implementa esta señal, pero se desconoce si otros servidores con soporte HTTPS la implementan.

V13: No uso del protocolo TLS1.2

Cuando se trata de información sensible, siempre es recomendable el uso de los mejores algoritmos, cifradores y protocolos. El hecho de que un servidor no utilice el mejor protocolo de comunicación HTTPS es considerado como una vulnerabilidad en la conexión, hasta el punto que el PCI Counsil dio a quienes tengan o busquen la certificación PCI-DSS 3.1 plazo hasta julio de 2016 para eliminar SSL2, SSL3 y TLS1.0 de sus listas de protocolos [47].

[5]Ver la recomendación de la IETF: https://tools.ietf.org/html/rfc7568

Otras vulnerabilidades

V14: Abuso de logging (código de pruebas olvidado)

Android, como todo sistema Linux, hace uso de un log interno el cual puede ser utilizado para reportar eventos ocurridos dentro del sistema. Generalmente, las aplicaciones reportan automáticamente al log eventos relacionados con uso de recursos, stacktrace de errores, recolección de basura, etc.

De la misma forma, los desarrolladores de aplicaciones pueden utilizar este log para realizar tareas de corrección de errores (debugging). Por ejemplo, verificar valores de variables y su progresión durante la ejecución de la aplicación. Lamentablemente, muchas veces ese código de prueba es olvidado en la aplicación y pasa a producción, provocando filtración de información aprovechable por cualquiera que pudiera leer el log, por ejemplo:

- Otras aplicaciones que tengan permisos para leerlo (mediante el permiso `android.permission.READ_LOGS`).
- Malware instalado en el equipo.
- Alguien que active el modo de desarrollador en el teléfono y pueda leer el log mediante la conexión USB desde un computador.

Se ha observado en algunas aplicaciones que éstas envían al log interno datos sensibles como los siguientes:

- Información geográfica (extraída desde GPS).
- Cookies de sesión.
- RUT del usuario.
- Límite de monto a transferir a terceros.
- Password del usuario
- Desafío y respuesta a desafío de tarjeta de coordenadas.
- Datos del destinatario de transferencia bancaria (nombre, número de cuenta, dirección de correo electrónico)
- Información de cuenta corriente del usuario (saldos, operaciones)
- Información de tarjeta de crédito del usuario (PAN[6], etc.)

[6]El PAN es el número de tarjeta de crédito (Primary Account Number). El PAN mostrado en los ejemplos es un número de prueba para tarjetas VISA obtenido desde https://www.auricsystems.com/support-center/sample-credit-card-numbers/

V15: Actualización insegura de información (mediante HTTP)

También se ha observado que hay aplicaciones que realizan actualizaciones de información sensible a través de canales no encriptados (HTTP). En particular, se observó que hay actualizaciones de la URL del endpoint al cual la aplicación se conecta mediante HTTP por texto plano, información que podría ser modificada vía MitM[7]. Cabe destacar que este riesgo no sólo afecta a funcionalidades de actualización de información, ya que un estudio reveló que el 35 % de las conexiones de aplicaciones móviles son no cifradas [40].

V16: Almacenamiento de información sensible en base de datos SQLite o preferencias

Android ofrece a los desarrolladores una API que les permite almacenar información dentro de una base de datos local SQLite (mediante el uso del paquete `android.database.sqlite`). Esta base de datos queda almacenada en el área de almacenamiento de la aplicación, y no es difícil de obtener, ya sea por parte de otra aplicación (en particular en un equipo rooteado) o por alguien conectado al equipo mediante USB.

Se ha observado que se almacena la siguiente información en estas bases de datos:

- Nombre de usuario (usualmente el RUT).
- Cookies de sesión (en particular en aplicaciones híbridas y web embebida).
- Identificadores del teléfono (ej. IMEI o ANDROID_ID). Estos datos
- Datos de transferencias bancarias (nombre, e-mail del destinatario, etc.)

Cabe señalar que, el RUT, los identificadores del teléfono y los datos asociados a transferencias bancarias son datos sensibles ya que permiten que una persona sea trazable mediante linkability[8] cruzando esos datos con los contenidos en otras bases de datos. Adicionalmente, las cookies de sesión, al ser capturadas, podrían ser utilizados para secuestrar sesiones[9].

[7]Otro ejemplo de esta vulnerabilidad es el del teclado para celulares Samsung, el cual se autoactualizaba mediante una conexión HTTP: https://nakedsecurity.sophos.com/2015/06/17/samsung-keyboard-app-could-let-a-crook-crack-your-phone/

[8]Ver la sección "Privacidad de información personal y del equipo móvil" para la definición de linkability.

[9]El secuestro de sesión es un ataque mediante el cual se toma el identificador de sesión de una cookie y se copia a otra, estando ésta última bajo control de un atacante. Mediante esto, tal atacante podría hacerse pasar por el usuario autentificado con el identificador de sesión, sin siquiera tener control de sus credenciales.

Adicionalmente, esta información no es cifrada de ninguna manera. También se han observado casos en los que esta información se almacena en el archivo `SharedPreferences.xml`, el cual permite almacenar datos de preferencias.

V17: Almacenamiento de información sensible en cache

Las aplicaciones de tipo Web Embebida, al ser basadas en el uso del sitio web móvil del banco mediante el componente WebView (ver la sección "Aplicaciones web embebidas") almacenan un cache de datos web de la misma forma que un browser. Cuando se trata de información sensible como es la bancaria, el hecho que el cache web esté habilitado constituye una vulnerabilidad, ya que el cache queda guardado en el disco duro sin cifrado alguno, y esto puede dar pie para filtraciones de información[10].

V18: Envío de información sensible a través de canal inseguro (HTTP)

De la misma forma que en V15, hay aplicaciones que envían información a través de canal HTTP en texto plano, por ejemplo la información de la ubicación actual del usuario.

V19: Exceso de confianza en medidas anti-ingeniería reversa

Los frameworks para aplicaciones híbridas ofrecen a los desarrolladores la posibilidad de cifrar su código Javascript para prevenir procesos de ingeniería reversa como sí se pueden dar en aplicaciones nativas. Uno de ellos en particular, y que es utilizado por algunas aplicaciones bancarias nacionales, es el Titanium Appcelerator, el cual toma todos los archivos Javascript en los que está implementada la aplicación, los encripta, los concatena todos en forma secuencial y los deja en un archivo binario. Sin embargo, la llave de cifrado es la misma siempre y es fácil de obtener. De hecho, un proceso de ingeniería reversa para estas aplicaciones es automatizable [56].

V20: Falta de timeout

La plataforma Android soporta, en sus versiones base, tres tipos de autentificación para hacer uso del teléfono:

[10]https://www.owasp.org/index.php/Testing_for_Browser_cache_weakness_(OTG-AUTHN-006)

- PIN: Uso de un número similar al de los cajeros automáticos, pero con mayor libertad respecto del largo de éste (no se limita a 4 dígitos).
- Password.
- Patrón: También conocido como password gráfica, es una clave que se basa en una figura dibujada por el usuario sobre una grilla con 9 puntos de contacto.

De acuerdo con el estudio de Berkeley y Google, el 62 % de los usuarios de teléfonos móviles no utiliza ningún método de bloqueo de sus equipos [41]. Es por esto que la implementación de un timeout de cierre en una aplicación móvil es de extrema importancia, ya que un teléfono desbloqueado puede ser fuente, al menos, de filtración de información bancaria.

Varias aplicaciones, en particular las de tipo web embebida, no implementan un timeout del todo, vale decir, no hay un cierre o logout automático pasado un cierto tiempo de inactividad. Si la aplicación es reactivada en forma posterior, existe el riesgo de que información sensible sea visible, y ese riesgo aumenta en equipos sin timeout.

V21: Timeout mal implementado

Algunas aplicaciones, sobre todo las híbridas y web móviles, si bien implementan timeouts, lo implementan de manera inadecuada. Los objetivos del timeout pasan por los aspectos de:

- Integridad, al evitar de que se realicen operaciones indebidas en la cuenta bancaria del usuario.
- Confidencialidad, al evitar que información sensible sea vista después de un tiempo determinado.

Si bien los casos analizados atacan bien el tema de la integridad, no lo hacen adecuadamente en el caso de la confidencialidad, ya que tales timeouts funcionan si y solo si el usuario intenta pasar de una página a otra dentro de la aplicación cuando el timeout haya finalizado. Si un usuario deja la aplicación en espera, por ejemplo, en la pantalla de saldos, y el timeout expira, aún se podrá ver esa información, hasta que el usuario trate de ir a otra pantalla dentro de la aplicación.

V22: Mal uso de cifrado en reposo

En su afán por proteger información de los usuarios en reposo dentro de la aplicación, los desarrolladores de varias de las aplicaciones analizadas integraron el uso de algoritmos de cifrado simétrico para almacenar datos sensibles dentro de la aplicación. Se observaron los siguientes problemas en la implementación de tal funcionalidad:

1. Uso de información del teléfono como llave de cifrado: Se usa información del teléfono, como el IMEI o el ANDROID_ID, como llave de cifrado. Tal información es relativamente simple de obtener, ya que existen APIs las cuales permiten el acceso a tal información sin cifrar.
2. Uso de llaves de cifrado *hardcodeadas*[11]: Se genera una llave de cifrado aleatoria, la cual sin embargo es incluida en el código fuente, como un string más. Tal información podría ser obtenida mediante ingeniería reversa.
3. Supuesto enmascaramiento de llaves en algoritmos de hash: En este caso, se toma una llave basada en información interna simple de obtener (por ejemplo el IMEI o el ANDROID_ID), una llave hardcodeada, o la concatenación de ambas, y se pasa por un algoritmo de hash (en los casos observados se usó MD5), para usar el digest resultante como llave de cifrado.
4. Mala generación de vectores de inicialización: Hubo casos en los que se requirió un vector de inicialización (IV) como parte del uso del algoritmo de cifrado para tales casos. Tales IV's deben ser aleatorios y los desarrolladores usaron IV's fijos y hardcodeados.

Para corregir esta vulnerabilidad, se recomienda la generación de una llave de cifrado aleatoria y que ésta sea almacenada, no en el área de almacenamiento de la aplicación, sino que en un sistema llamado Android Keystore[12], el cual almacena claves en forma segura, cifrándolos con una llave basada en el método de bloqueo del sistema elegido por el usuario, sea éste un PIN, password o patrón con el cual se autentifica ante el teléfono. Esto ayuda también a forzar a los usuarios a bloquear sus equipos, como se indicó en V20.

[11] Hardcoding es una práctica mediante la cual, datos de configuración de una aplicación son incrustados en el código fuente y posteriormente son traspasados mediante el proceso de compilación al binario de la aplicación.

[12] Android Keystore es un sistema de almacenamiento seguro de llaves o valores que se desea que sean secretos. Funciona como una tabla de datos encriptada mediante el PIN, password o patrón del usuario del equipo, por lo tanto no es fácil de obtener. Puede ver la documentación oficial de Android al respecto en http://developer.android.com/intl/es/training/articles/keystore.html y un ejemplo de su uso en http://www.androidauthority.com/use-android-keystore-store-passwords-sensitive-information-623779

V23: Mal uso de cifrado redundante

Relacionado con la vulnerabilidad anterior, es el hecho de que hay aplicaciones las cuales cifran la información usando un algoritmo de cifrado simétrico antes de su envío a través de cifrado en tránsito (HTTPS) hasta el servidor. Esto, si bien no debería ser un mayor problema, resulta ser redundante, debido a que la información ya es cifrada durante su transmisión hacia el servidor, y si es el objetivo de los desarrolladores el que la información sea almacenada en forma segura en sus servidores, tal cifrado debe ser realizado por el servidor posterior a la recepción de tales datos desde el móvil, considerando que el servidor tiene una mayor capacidad que el móvil en cuanto a recursos computacionales se trata.

V24: Potencial Tampering de datos

La aplicación, al momento de realizar una transferencia de dinero, pudiera ser vulnerable a que un usuario malicioso, o algún malware instalado en el teléfono modifique los datos de transacciones de transferencias de dinero entre cuentas, antes de que esa información sea enviada al servidor vía HTTPS. Éste ataque funcionaría de la forma que se muestra en la figura:

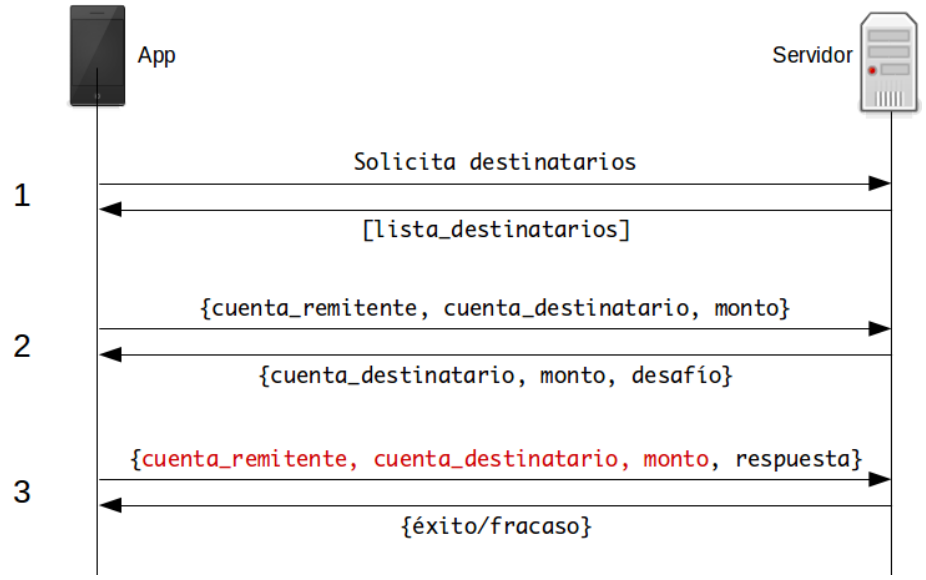


Figura 6.1: Modificación maliciosa de datos de transferencias

1. La aplicación solicita al servidor la lista de destinatarios registrados por el usuario en su cuenta, ante lo cual éste se la envía.

2. El usuario selecciona un destinatario y el monto a transferir en la aplicación, la cual envía esa información al servidor junto con la cuenta remitente. El servidor responde enviándole a la aplicación los datos de la cuenta destinatario, el monto a transferir y un desafío, que puede ser algunas coordenadas de la tarjeta o el número que aparezca en el token, dependiendo del esquema 2FA utilizado por el banco.
3. El usuario ingresa la respuesta al desafío solicitado, y la aplicación envía ese dato, junto a la cuenta remitente, la cuenta de destinatario y el monto a transferir. Estos datos pueden ser falseados de tal forma que la cuenta remitente no sea la correcta, la cuenta destino no sea la correcta o el monto no sea el que el usuario deseó transferir. Si el servidor no valida esta información, hay riesgo serio de robo de dinero mediante transferencias a cuentas no deseadas.

Para mitigar este problema, se propone el uso de un ID de transacción, como se muestra en la siguiente figura:

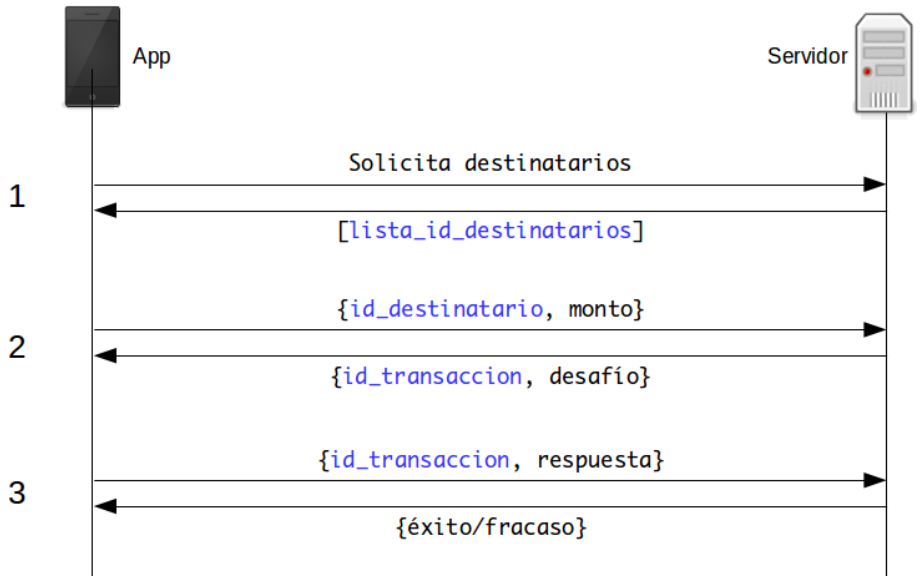


Figura 6.2: Mitigación ante potencial tampering de datos de transferencias

En este caso, en vez de dejar información validable en manos del cliente, la cual podría ser modificada maliciosamente, se deja esa información en el servidor y ésta se asocia con un ID de transacción, el cual es enviado a la aplicación junto con el desafío (por ejemplo, mediante una cookie o un token), o asociar tal ID de transacción con la sesión del usuario en el servidor. El usuario ingresa la respuesta, a la cual la aplicación adosa este ID de transacción y la envía al servidor, el cual realiza los chequeos usuales (existencia de la cuenta de destinatario, el cliente tiene fondos suficientes, etc.), realiza la

transacción, y responde al usuario si ésta fue exitosa o fallida.

Adicionalmente, antes de la validación, el servidor puede enviar en vez de una lista de destinatarios con mucha información, sólo enviar una lista con pares `{id_destinatario, nombre_destinatario}` en el paso 1, y cuando el usuario seleccione un destinatario, sólo enviar su id_destinatario en el paso 2.

V25: Uso innecesario del PAN de Tarjeta de Crédito en tránsito

La aplicación, al momento del usuario consultar su estado de tarjeta de crédito, muestra al usuario una versión anonimizada del PAN de ésta, de la forma `XXXX-XXXX-XXXX-4448`. Sin embargo, el servidor envía a la aplicación el PAN completo (en la forma `4444-4444-4444-4448`) y es la aplicación la que la anonimiza. Además de ser innecesario enviar el PAN completo desde el servidor a la aplicación, es potencialmente inseguro, ya que mediante un proxy con soporte SSL se podría extraer esa información encriptada en tránsito. La recomendación es anonimizar tal valor antes de ser enviado desde el servidor, en conformidad con el Requisito 3 de la versión 3.1 del estándar PCI-DSS[13].

En la próxima sección se indica la taxonomía de malas prácticas que se derivan de las vulnerabilidades observadas en esta sección.

[13]https://www.pcisecuritystandards.org/documents/PCI_DSS_v3-1.pdf

7

Taxonomía de malas prácticas

Basándose en las vulnerabilidades observadas, se puede inferir un conjunto de malas prácticas en las cuales los desarrolladores de las aplicaciones móviles bancarias incurren. Ésta taxonomía puede ser extensible también a aplicaciones móviles que no sean bancarias pero que manejen información sensible de sus usuarios.

Adicionalmente, en esta sección se indica qué buenas prácticas pueden ser implementadas para tratar las vulnerabilidades derivadas de tales malas prácticas. En el cuadro 5.3, se indica qué aplicaciones bancarias, en forma anonimizada, incurren en estas malas prácticas.

A continuación, la taxonomía en detalle.

Abuso en el uso de información personal de los usuarios

Vulnerabilidades relacionadas: V1, V2, V3, V5, V6, V25.

Etapa del SDLC en la que es introducida: Diseño.

Los equipos móviles llevan mucha información del usuario consigo. El acceso a tal información por parte de las aplicaciones móviles a menudo acarrea riesgos asociados con la privacidad. Para citar un ejemplo, el caso de la aplicación *La Nueva Forma de Marcar*, encargada por la Subsecretaría de Telecomunicaciones [15], puso en evidencia los

abusos a los cuales se ven sometidos muchas veces los usuarios de aplicaciones móviles. Estos abusos se ven reflejados principalmente en el envío a servidores controlados por los desarrolladores de la aplicación de información personal, de ubicación, identificadores del teléfono, etc., sin el conocimiento y menos la autorización del usuario.

Buenas prácticas asociadas

Se recomienda realizar un catastro de la información a la cual la aplicación tiene acceso, ya sea mediante los permisos solicitados, input del usuario, fuentes externas, etc., y ser transparente con el usuario respecto de:

- Qué información recolectará la aplicación,
- Dónde será enviada la información,
- Para qué será utilizada,
- Por cuánto tiempo será retenida.

También se recomienda implementar una política de retención de datos. Una forma básica de hacerlo es seguir estas reglas básicas:

1. Si no se necesita un dato, no recolectarlo.
2. Si el dato es necesario, obtenerlo para procesamiento inmediato, pero no almacenarlo.
3. Si se requiere almacenar el dato, indicar un tiempo de retención, posterior al cual, el dato será removido (e informar de ello al usuario).

Mal uso de cifrado

Vulnerabilidades relacionadas: V22, V23.

Etapas del SDLC en las que es introducida: Diseño, implementación.

El uso de cifrado crea una falsa sensación de seguridad, basada en el hecho de que se cree que la criptografía es infalible y quien la utiliza no debe preocuparse de nada más. Sin embargo, nada está más lejos de la realidad. Si no se usan bien las llaves de cifrado, o no se usan adecuadamente los vectores de inicialización, las ventajas del uso de criptografía se neutralizan.

Buenas prácticas asociadas

Se recomienda lo siguiente:

1. No basar la llave en información simple de obtener, como el IMEI o el ANDROID_ID.
2. No hardcodear la llave en el código. Esa información es obtenible en forma relativamente fácil mediante el uso de ingeniería reversa.
3. De almacenar la llave de cifrado, utilizar mecanismos seguros de almacenamiento. Idealmente se podría utilizar un sistema HSM[1], sin embargo, su precio lo hace inviable para su uso en equipos móviles. En Android existe el sistema llamado Android Keystore, mencionado anteriormente, el cual permite guardar llaves en un sistema cifrado con información proveniente del usuario.
4. Al usar algoritmos de cifrado que requieran el uso de un vector de inicialización, que este vector sea siempre aleatorio.

Falta de protección de información sensible almacenada en el teléfono

Vulnerabilidades relacionadas: V4, V14, V16, V17, V18, V21.

Etapa del SDLC en la que es introducida: Diseño.

En este caso, no se protege adecuadamente la información almacenada en el equipo por parte del usuario o de la aplicación, sea ésta de los siguientes tipos:

- Información financiera
- Información de contactos
- Identificadores (ID de usuario, RUT…)

Esto se produce principalmente debido a la falsa asunción de que el área de almacenamiento es inviolable y las medidas de cifrado tomadas fueron adecuadas. Estas asunciones son falsas ya que existen formas de acceder al área de almacenamiento, tanto de la aplicación como del teléfono, y muchas veces la criptografía usada es mal utilizada (por ejemplo usando llaves de cifrado basadas en información fácilmente obtenible). Ésto trae como consecuencia potenciales ataques y robo de información desde dentro del equipo, en particular en equipos rooteados.

[1] El *Hardware Security Module* permite la gestión y almacenamiento seguro de claves de cifrado para autentificación fuerte y provee mecanismos de procesamiento criptográfico.

Buenas prácticas asociadas

Se recomienda:

1. Evitar en lo posible el almacenamiento de información en el teléfono, ya sea en el área de la aplicación o en el almacenamiento externo.
2. Si es absolutamente necesario, Utilizar criptografía simétrica y utilizar llaves de cifrado basadas en información externa al teléfono, por ejemplo mediante el uso de Android Keystore, mencionado anteriormente.

Falsa sensación de seguridad en el uso de SSL/TLS

Vulnerabilidades relacionadas: V7, V8, V9, V10, V11, V12, V13.

Etapa del SDLC en la que es introducida: Implementación.

Existe una confusión respecto del uso correcto de SSL/TLS en servidores web HTTPS. Se piensa que sólo por tener un certificado instalado sobre un servidor con esas capacidades se está seguro. Sin embargo, muchas tecnologías ofrecidas por tales servidores son obsoletas o inseguras, principalmente por un tema de compatibilidad hacia atrás. Esto combinado con su facilidad de configuración causa una falsa sensación de seguridad, lo que hace que haya una diferencia abismal entre configurar un servidor HTTPS y configurarlo apropiadamente [2].

Estos problemas son fácilmente visibles con el uso de herramientas automatizadas, como Qualys SSL Test o Cipherscan, y se manifiestan en los siguientes problemas con la conexión:

1. Certificados mal configurados (apuntan a un host distinto de aquel en el que se encuentran instalados), con algoritmos de firma antiguos (ej. SHA1), sin una cadena de certificación completa, etc.
2. Uso de cifradores y algoritmos de hash inseguros, como RC4, RSA con llaves cortas, DES, EXPORT, SHA1, MD5, etc.
3. No implementación de Forward Secrecy mediante el uso de curva elíptica.
4. Configuraciones expuestas a vulnerabilidades tipo VUWACONA[2], como FREAK, CRIME, BEAST, POODLE, Heartbleed.

[2]VUlnerability With A COol NAme. Acrónimo creado para referirse a vulnerabilidades que hacen uso de acrónimos o nombres atractivos [57].

5. Uso de protocolos inseguros (SSL2, SSL3) y no uso del protocolo más seguro que existe en la actualidad: TLS1.2.

Sólo uno de los bancos cuya aplicación fue analizada tiene buena seguridad (nota A en el Qualys SSL Test). De los demás, 6 tienen nota mediocre (B, C) y 3 tienen mala seguridad (nota F), como se observa en el cuadro 5.2. La mala configuración de estos servidores tiene como consecuencia que las conexiones podrían ser monitoreadas e información sensible podría ser extraída de tales conexiones principalmente mediante ataques MitM[3].

Buenas prácticas asociadas

Lo primero que hay que tomar en cuenta son las buenas prácticas para la configuración de SSL/TLS. Existen varias fuentes de información al respecto:

1. La Fundación Mozilla preparó una guía para la adecuada configuración de servidores. Ésta guía define 3 perfiles de configuración recomendados (modern, intermediate, old) que permiten a los administradores de servidores configurarlos de acuerdo al perfil de los clientes que deseen que tengan soporte al momento de conectarse a su servidor: https://wiki.mozilla.org/Security/Server_Side_TLS.
2. Adicionalmente, la misma fundación puso disponible al público una herramienta que permite generar la configuración SSL/TLS para ser utilizada en servidores, basándose en el perfil de configuración deseado, el servidor que se utiliza (Apache, NginX, Lighttpd, HAProxy, Amazon Web Services Elastic Load Balancer), y la versión de la biblioteca OpenSSL instalada: https://mozilla.github.io/server-side-tls/ssl-config-generator/.
3. El libro más completo que existe sobre configuración SSL/TLS contiene información detallada respecto de buenas prácticas de configuración, detalles técnicos y análisis de vulnerabilidades. El libro se titula “Bulletproof SSL and TLS” y se encuentra disponible en https://www.feistyduck.com/books/bulletproof-ssl-and-tls/

Para asegurarse que la configuración sea adecuada, ésta debe ser probada. Existen dos herramientas interesantes en este sentido, ambas para diferentes propósitos:

[3]El ataque *Man-In-The-Middle* (también conocido como MitM) consiste en interponerse entre dos nodos de una conexión para leer, insertar o modificar los datos transferidos entre ambos.

1. Cipherscan: Esta herramienta se vale de la biblioteca OpenSSL y permite realizar dos tareas. La primera es entregar un informe respecto de la configuración del servidor analizado. La segunda realiza el mismo informe y entrega un informe de si el servidor tiene problemas con SSL/TLS y pasos para adaptarlo a alguno de los perfiles de configuración establecidos por la Fundación Mozilla anteriormente mencionados. Ésta herramienta puede ser usada indistintamente en servidores expuestos a Internet como servidores de prueba, staging o pre-producción. Está disponible en https://github.com/jvehent/cipherscan.
2. Qualys SSL Test: Ésta herramienta permite realizar un análisis de un sitio web HTTPS ya expuesto a Internet y entregar una nota (similar a las estadounidenses, de la A a la F) para calificar el estado de la configuración HTTPS de un sitio web determinado. Sus análisis son más profundos que los de Cipherscan, ya que chequea las rutas de certificación, hace simulaciones de conexiones de diferentes clientes (browsers, equipos móviles, etc.) y revisa aspectos como el uso del header HSTS (HTTP Strict Transport Security), si se usa el sistema de verificación de no vencimiento de certificado OCSP, etc. La herramienta es de uso gratuito, y se encuentra en https://www.ssllabs.com/ssltest/.

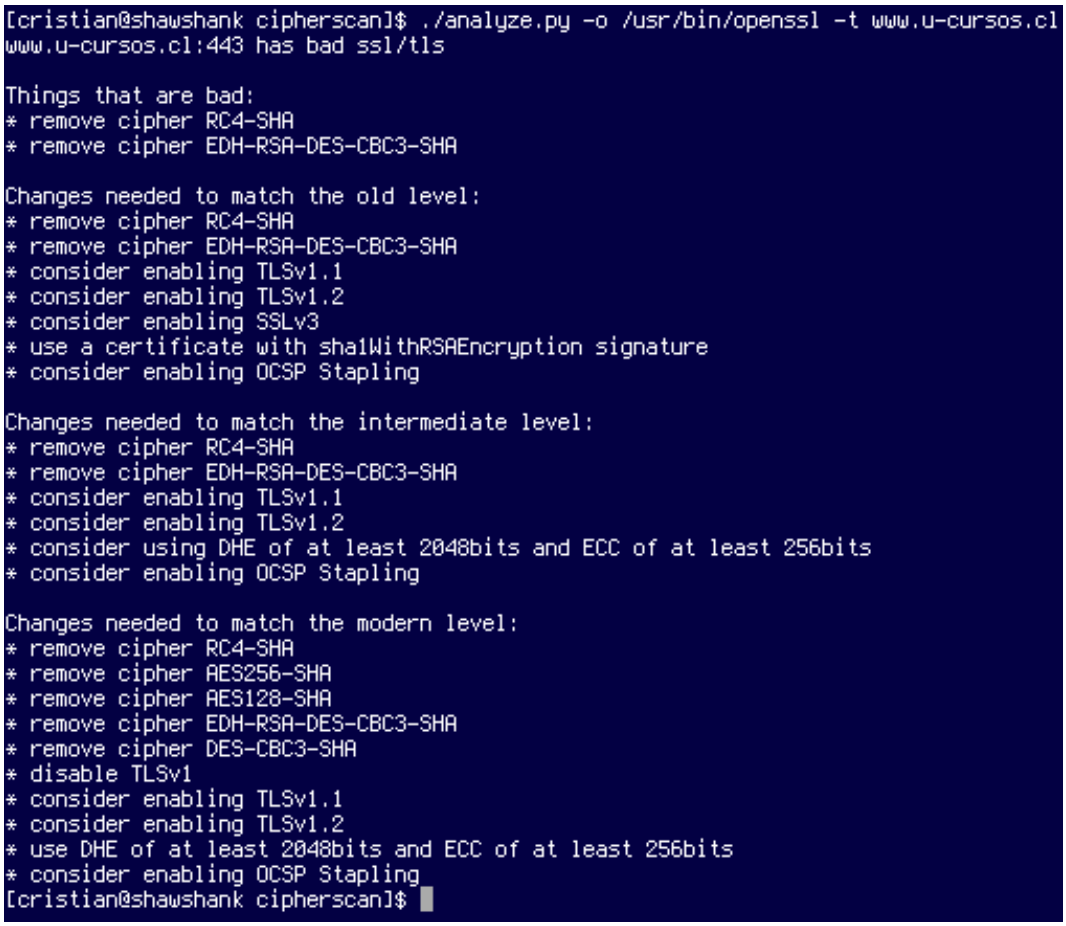


Figura 7.1: Output del analizador de Cipherscan.

Código abandonado

Vulnerabilidades asociadas: V14.

Etapas del SDLC en las que es introducida: Implementación, pruebas.

Un problema de calidad y seguridad se produce al utilizar, durante la etapa de pruebas y debugging de la aplicación, elementos como:

- Configuraciones de debugging al momento de compilar, las cuales causan que los binarios generados incluyan símbolos relacionados con código fuente u configuración de la aplicación.
- Uso de función de logging incorporada a la API del framework, como `android.util.Log`.

- Uso de código *TRACE*: Código incluido por los desarrolladores como una forma de apoyo a su proceso de debugging, que tiene como objetivo que la aplicación indique en alguna parte, por ejemplo, la progresión del valor de una variable en algún momento de la ejecución.

Si bien estos elementos pueden ser útiles, generalmente son considerados como una mala práctica de desarrollo, ya que su remoción generalmente es engorrosa y trae riesgo de pasar a llevar código que sí funciona (regresión). Además es un problema de seguridad dado que ese código sobrante puede filtrar información del usuario y de la aplicación [5].

Buenas prácticas asociadas

1. Controlar el proceso de compilación, de tal forma que no se puedan filtrar simbolos ni datos del código fuente.
2. Crear alguna clase intermedia a la función de logging de la API, que permita la deshabilitación del logging de manera simple antes del lanzamiento de la aplicación a producción.
3. No usar codigo *TRACE*. Preferir las funciones de logging del framework.

Uso de medidas anti-ingeniería reversa como medida de seguridad

Vulnerabilidades asociadas: V19.

Etapa del SDLC en la que es introducida: Implementación.

Confiar en el secretismo de la implementación no es una buena idea, como lo demuestra el Principio de Diseño Abierto. Intentar cifrar u ofuscar el código de la aplicación no es 100 % efectivo para proteger secretos de implementación. Esto conlleva a una falsa sensación de seguridad, y a la creencia de que se está protegiendo adecuadamente la aplicación y sus datos, aún cuando no es así.

Buenas prácticas asociadas

Lo principal en este caso es hacer de cuenta que un adversario siempre tendrá acceso al código de la aplicación, y asegurarla tomando en consideración tal asunción.

Abuso de privilegios

Vulnerabilidades asociadas: V1, V2, V3, V4, V5.

Etapa del SDLC en la que es introducida: Diseño.

Se ha observado que las aplicaciones analizadas solicitan acceso a la siguiente información:

- Contactos del usuario
- Otras aplicaciones en ejecución en el equipo
- Estado del teléfono
- Logs internos del sistemas
- Información geográfica (GPS)

También se ha observado que en algunos casos, estos privilegios son solicitados, ya sea por frameworks híbridos, como por bibliotecas APM o de Analytics que algunas aplicaciones usan, como condición para su uso.

Ésta mala práctica es un problema de privacidad para los usuarios, que ven (o más bien dicho, no ven) cómo su información es exfiltrada desde el teléfono hacia servidores que ellos mismos desconocen. Las estadísticas indican que las aplicaciones Android en promedio envían información potencialmente sensible a 3,1 dominios externos sin conocimiento ni consentimiento de sus usuarios [58]

Buenas prácticas asociadas

Se recomienda evitar lo más posible el acceso a privilegios que sean potencialmente peligrosos en términos de privacidad. Si se requiere el uso de georreferenciación para búsqueda de cajeros cercanos, utilizar sólo la gruesa (coarse). No acceder ni a contactos ni otros datos del teléfono. En el caso de que algún framework o biblioteca de analytics o APM requieran de algún privilegio, auditarlos y averiguar si se puede configurar la biblioteca para deshabilitar su uso, y si no se puede, desechar la biblioteca por completo y buscar alternativas.

Como una forma de agrupar en un conjunto de buenas prácticas lo expuesto en esta sección, en la siguiente se darán algunas recomendaciones de seguridad, aplicables en particular para aplicaciones móviles financieras y también en general para cualquier aplicación móvil que de desarrolle para la plataforma Android.

8

Recomendaciones de seguridad para el desarrollo de aplicaciones móviles

Quizás el mayor aporte público de esta tesis, basado en la experiencia de quien la escribe y en los resultados de la presente investigación, es un conjunto de recomendaciones de seguridad en el desarrollo de aplicaciones móviles, las cuales están pensadas específicamente para los bancos analizados en esta tesis, pero también son útiles para cualquier equipo de desarrollo que esté a cargo del desarrollo de aplicaciones móviles. Tales recomendaciones se detallan a continuación.

Considerar el ambiente de operación de la aplicación

Como se vio anteriormente, una aplicación móvil opera en tres entornos [36]:

1. El aparato: Como se vio anteriormente, el equipo puede ser susceptible a ataques vía malware, mal uso, rooting, acceso no autorizado vía USB, etc. Lo que hay que tener en cuenta es que el aparato no es una caja fuerte, y por lo mismo, la información puede ser extraída o modificada en forma maliciosa.
2. La red: El entorno de red al cual se conecta el equipo puede ser susceptible de ser monitoreado o penetrado, acción mediante la cual un atacante podría obtener información de transacciones o personal en tránsito.

3. El centro de datos: El servidor al cual se conectan las aplicaciones bancarias puede ser susceptible a ataques de negación de servicio, inyecciones de código, etc.

La seguridad 100 % es imposible, por lo tanto, ninguno de estos tres entornos es infalible. Es importante que los desarrolladores de aplicaciones sean cuidadosos, analicen las potenciales vulnerabilidades que tiene cada uno de estos entornos, y mitiguen estas vulnerabilidades. Si bien ésta tesis se concentra en los dos primeros entornos, no se puede dejar de considerar la seguridad en el centro de datos. Poner atención a aspectos como:

- Configuración insegura del servidor: Configuración/usuarios/servicios por defecto o mal uso de opciones de seguridad.
- Vulnerabilidades de la plataforma: Servidor o sistema operativo sin parchar o con puertos abiertos innecesarios.
- Validación inexistente: La mayoría de los ataques de inyecciones (SQL, shell, etc.) se producen porque se asume que todo input del usuario es legítimo y no se valida que no existan caracteres que causen filtraciones de datos o violación de integridad de tales. Se recomienda desconfiar de todo input, incluso de los provenientes de las aplicaciones instaladas en los equipos de los usuarios, y validar tales inputs.
- Filtración de datos: Si el servidor tiene puertos abiertos innecesarios, usuarios con passwords por defecto, o es susceptible ante inyecciones, es posible que un atacante pueda exfiltrar información sensible de los usuarios desde allí.

Tener cuidado respecto del almacenamiento de información en el aparato

Se recomienda a los desarrolladores el ser cuidadosos a la hora del almacenamiento de datos financieros o personales en el aparato. Como se indicó recién, el equipo es susceptible a ataques que causan exfiltración de información. Dicho esto, se recomienda:

a. No almacenar información de ningún tipo dentro del teléfono, ni en el disco, ni en el cache, ni en el área de almacenamiento de la aplicación, ni en el área de almacenamiento externa.
b. Si es absolutamente necesario e inevitable el almacenamiento de datos en el equipo, cifrarlos en forma adecuada, utilizando algoritmos fuertes de cifrado

y almacenando sus llaves de cifrado en mecanismos que permitan aislar en un área cifrada información sensible, como la Android Keystore[1].

Remover código de debugging o logging

Antes de pasar la aplicación a producción, remover cualquier código de logging o debugging de la aplicación. Esto incluye:

- Código TRACE: Código incluido por los desarrolladores como una forma de apoyo a su proceso de debugging, que tiene como objetivo que la aplicación indique en alguna parte, por ejemplo, la progresión del valor de una variable en algún momento de la ejecución.
- Información enviada en forma voluntaria al log, mediante el uso de las funciones provistas por `android.util.Log` o similares.

Usar mecanismos que permitan cambiar desde un ambiente de pruebas/debugging o de desarrollo a uno de producción en forma simple y sin afectar la funcionalidad de la aplicación.

Preocuparse de la calidad de la conexión

La calidad de la conexión entre la aplicación y su servidor es de suma importancia, sobre todo porque muchas veces los usuarios usan las aplicaciones en entornos inseguros como redes inalámbricas con clave compartida. Se recomienda:

- Jamás usar HTTP sin cifrado. Con la disponibilidad de certificados económicos (incluso gratuitos, como los ofrecidos por Let's Encrypt[2]) ya no hay excusas para no utilizar HTTPS en las conexiones al backend de la aplicación.

[1] Android Keystore es un sistema de almacenamiento seguro de llaves o valores que se desea que sean secretos. Funciona como una tabla de datos encriptada mediante el PIN, password o patrón del usuario del equipo, por lo tanto no es fácil de obtener. Puede ver la documentación oficial de Android al respecto en http://developer.android.com/intl/es/training/articles/keystore.html y un ejemplo de su uso en http://www.androidauthority.com/use-android-keystore-store-passwords-sensitive-information-623779

[2] Let's Encrypt es una autoridad certificadora abierta, automatizada y gratuita [59]. Ver https://letsencrypt.org/.

- Usar certificados validados por una autoridad certificadora, firmados utilizando SHA256 y no evitar (bypassear) la verificación de tales certificados en la aplicación.
- Verificar la calidad de la conexión de las conexiones SSL/TLS. Revisar que no haya problemas como algoritmos débiles o protocolos inseguros mediante el uso de herramientas como Cipherscan[3] o Qualys SSL Test[4]. Poner atención a aspectos como:
    - Uso de algoritmos de intercambio de llaves fuertes: Se recomienda encarecidamente priorizar el uso de Diffie-Hellman con curva elíptica (ECDHE) por sobre RSA.
    - Uso de algoritmos de cifrado: Abandonar algoritmos antiguos (como RC4) y utilizar AES256.
    - Uso de algoritmos de validación de mensajes: Preferir SHA256 como mínimo en la conexión.
    - Uso de protocolos: Abandonar SSL2 y SSL3. Utilizar como mínimo TLS1.0.
    - Dar preferencia a las suites de cifrado por parte del servidor.
    - Activar el header HSTS (HTTP Strict Transport Security) en el servidor.
    - Utilizar la tecnología OCSP Stapling, la cual es más eficiente y menos propensa a problemas de privacidad que el chequeo continuo de listas de revocación de certificados (CRL's). [2][47]
- HTTPS es una tecnología la cual es sometida continuamente a pruebas e investigación. Por lo tanto, cada cierto tiempo aparecen nuevas vulnerabilidades. Mantenerse al tanto de tales novedades.

Implementar Timeouts

Como se mencionó anteriormente, mucha gente no bloquea sus equipos, ni con password, PIN o patrón. Si las aplicaciones no se cierran solas, éstas terminarán por exponer a inescrupulosos información financiera sensible.

[3]https://github.com/jvehent/cipherscan.
[4]https://www.ssllabs.com/ssltest/

Analizar la seguridad de las dependencias (bibliotecas, frameworks)

Verificar la seguridad y la privacidad de frameworks, bibliotecas y otros servicios de los cuales los desarrolladores dependen para que la aplicación pueda cumplir con su cometido. Estar atentos a vulnerabilidades que aparezcan en los paquetes de bibliotecas y frameworks, y auditar la privacidad de servicios de analytics y APM antes de integrarlos a la aplicación.

Existen herramientas automatizadas que permiten auditar tales dependencias, por ejemplo:

- OWASP Dependency Check: Esta herramienta permite analizar las dependencias en proyectos Java en general, y Android en particular. Se encuentra disponible en https://www.owasp.org/index.php/OWASP_Dependency_Check.
- Muchos frameworks híbridos se basan en funcionalidad Javascript. Retire.js es un buen proyecto el cual analiza las dependencias Javascript que pueda tener la aplicación y reporta cuales son vulnerables. Está disponible en https://retirejs.github.io/retire.js/.

Cabe destacar que estas herramientas extraen bases de datos desde el sitio del CVE[5], la cual es el registro central de toda vulnerabilidad en software liberado públicamente, y eso incluye bibliotecas y dependencias.

Analizar la seguridad en el código propio

Si bien es bastante difícil o caro, se recomienda usar alguna herramienta que realice análisis estático sobre el código desarrollado. Este tipo de herramientas revisa el código fuente (o en el caso de Java, sus clases en bytecode) en busca de vulnerabilidades. Existen varias soluciones en el mercado que pueden hacer este tipo de análisis, sin embargo si se desea partir con una solución opensource, se recomienda el uso de la herramienta Find Security Bugs, disponible en https://find-sec-bugs.github.io/

[5]Common Vulnerabilities and Exposures. Es una base de datos que lleva un registro de vulnerabilidades descubiertas en paquetes de software (bibliotecas, software de escritorio/móvil/web). Está disponible para consulta y obtención en https://cve.mitre.org/

Tener en cuenta la privacidad en el uso de georreferenciación

Como se indicó antes, los equipos cuentan con dos tipos de georreferenciación: Fina (GPS) y gruesa (A-GPS). Cuando se trata de asistir a los usuarios en la búsqueda de lugares cercanos (como cajeros automáticos o sucursales del banco) el uso de georreferenciación fina es un problema de privacidad y de energía para los usuarios de la aplicación. Se recomienda el uso exclusivo de georreferenciación gruesa (A-GPS).

Evitar el abuso de privilegios

No abusar de los privilegios. Si la aplicación no necesita acceso a los contactos del usuario para funcionar, no deben ser solicitados. Lo mismo aplica para el estado del teléfono, la información acerca de otras aplicaciones en uso, etc. Se recomienda guiarse siempre por el Principio de Privilegio Mínimo cuando se requiera el uso de algún recurso del equipo, y auditar privilegios.

Principio de Diseño Abierto

Es importante en el desarrollo de aplicaciones el no poner la seguridad de éstas en manos del secretismo de su implementación. Esto quiere decir que no es conveniente el confiar ciegamente en el uso de mecanismos anti-ingeniería reversa para prevenir la extracción del código fuente de la aplicación por parte de investigadores o inescrupulosos. Además de ser inefectivos ante terceros motivados o con buenas herramientas, tales mecanismos no protegen la aplicación del análisis de comportamiento de ésta en busca de vulnerabilidades. Ver la sección “Análisis de comportamiento” al respecto.

9

Futuras líneas de investigación

El trabajo realizado en esta tesis puede servir de punto de partida para investigaciones futuras, de las cuales se pueden destacar las siguientes:

Análisis de aplicaciones bancarias para la plataforma iOS: Tomando como base lo analizado en esta tesis, y principalmente las vulnerabilidades y taxonomía de malas prácticas, se puede realizar un trabajo similar para analizar la seguridad de aplicaciones bancarias para la plataforma iOS de Apple. Dentro de las ventajas en el desarrollo de una investigación así está, por ejemplo, el hecho de que al ser una plataforma estandarizada, casi no existe fragmentación, lo cual permite estudiar más rápido posibles vulnerabilidades que pudieran tener tales aplicaciones. Sin embargo, una investigación de tal calaña, requiere superar principalmente el problema de la ingeniería reversa de la aplicación, la cual, a diferencia de en el caso de Android, se basa en análisis dinámico, es decir, la extracción del binario ejecutándose en la memoria de un equipo jailbreakeado[1] [60].

Análisis de aplicaciones de tipo Digital Wallet: Existen aplicaciones que permiten a sus usuarios realizar pagos con éstas, los cuales son cargables a sus tarjetas de crédito bancarias emitidas por el banco. Estas aplicaciones, conocidas como Digital Wallet, permiten realizar pagos mediante el uso, principalmente, de tecnologías como NFC en el punto de venta. Un ejemplo de ello es la aplicación BBVA Wallet, ofrecida por el banco

[1] Proceso similar al de *rooting*, esta vez para la plataforma iOS.

del mismo nombre. Éstas aplicaciones manejan el PIN de la tarjeta de crédito e historial de transacciones, razón por la cual, es una investigación interesante en términos de seguridad.

10

Conclusiones

En este trabajo de tesis se pueden destacar varios aspectos relacionados a su aporte y rescatar algunas conclusiones, las cuales se indican a continuación:

1. **Ignorancia respecto de conceptos de seguridad en el desarrollo de software**: A excepción del curso de Seguridad de Software dictado por el tesista en el DCC, y de algunas pocas iniciativas privadas, no hay una integración de conceptos de seguridad en los conocimientos de desarrollo de aplicaciones, tanto web, como de escritorio, como móviles. Adicionalmente, la documentación disponible de lenguajes y frameworks de desarrollo de aplicaciones, de incluir conceptos de seguridad, los incluyen en forma aparte, en documentación complementaria y no como parte del *core* de la herramienta. Ésto causa un desconocimiento, por parte de desarrolladores, diseñadores, encargados de QA[1], jefes de equipo de desarrollo, etc., respecto de vulnerabilidades a las cuales podrían estar expuestas las aplicaciones que éstos desarrollan, y los datos que ellas almacenan, procesan y transmiten. No hay educación respecto de a qué riesgos las aplicaciones están expuestas, no hay pruebas de seguridad sobre éstas, y a pesar de que existen herramientas comerciales que permiten análisis automatizado de vulnerabilidades en aplicaciones móviles, se desconoce si se utilizan o no. Esto causa que, de todas las aplicaciones analizadas, no hay ninguna en la cual su equipo de desarrollo

[1] Aseguramiento de la calidad de un software (Quality Assurance).

(interno o externo) no caiga en alguna de las malas prácticas mencionadas en la taxonomía indicada en la presente tesis.

2. **Baja calidad de las conexiones HTTPS**: Un problema endémico que afecta a muchos servidores que utilizan SSL/TLS es la baja calidad y falsa sensación de seguridad. De acuerdo con SSL Pulse, sólo un 40 % de los sitios web tienen buena seguridad (nota A en el Qualys SSL Test) [61], y los servidores a los cuales se conectan las aplicaciones bancarias analizadas no tienen un mejor pasar: Sólo uno de ellos tiene una calidad aceptable de su conexión. Los demás, o tienen mediocre conexión o derechamente mala.
3. **Falta de puntos de contacto para reporte de vulnerabilidades**: Un problema fundamental observado durante la etapa de entrega de los informes de vulnerabilidades, y además a diario en sitios, es la completa falta de un punto de contacto formal al cual reportar vulnerabilidades o riesgos de seguridad. Generalmente, para poder reportarlos, el primer punto de contacto es el departamento de servicio al cliente, el cual no tiene el conocimiento adecuado de a quién redirigir la información, además de no contar con un procedimiento de tratamiento de la información reportada (no se utilizan tecnologías de cifrado como PGP, etc.), razón por la cual se recurrió al método de contactar a los tomadores de decisiones mediante contactos cercanos de conocimiento del tesista (contactos profesionales, clientes, otros investigadores) que conozcan a tales tomadores de decisiones. Incluso éste método tampoco fue totalmente efectivo: Sólo se logró contactar a 7 de los 10 bancos cuyas aplicaciones fueron analizadas. A los restantes se decidió, por un tema de tiempo, enviarles sus reportes junto con una carta al representante legal de cada banco, mediante correo certificado.
4. **La taxonomía y recomendaciones como aporte**: Tanto la taxonomía de malas prácticas como las recomendaciones indicadas anteriormente pueden ser utilizadas como un punto de partida para mejorar la seguridad en los procesos de desarrollo de aplicaciones móviles. Combinadas con herramientas automatizadas y herramientas a nivel de diseño e implementación integrables al proceso de desarrollo de aplicaciones móviles, pueden ser un activo definitorio en la mejora de la seguridad de los datos de los usuarios de aplicaciones móviles.

Bibliografía

[1] F. Lagos, "Bancos en Chile: La precaria seguridad de la banca en linea," 2012. [Online]. Disponible en http://blog.zerial.org/seguridad/precaria-seguridad-bancos. [Accedido 26-07-2016]

[2] I. Ristić, "SSL/TLS deployment best practices," 2014.

[3] R. Johnson, Z. Wang, C. Gagnon, A. Stavrou, "Analysis of android applications' permissions," en *Software Security and Reliability Companion (SERE-C), 2012 IEEE Sixth International Conference on*, 2012, pp. 45–46.

[4] A. Hevia, "CC51D - Seguridad de Datos," *Departamento de Ciencias de la Computación, Universidad de Chile*, 2009.

[5] G. McGraw, *Software security: Building security in*, vol. 1. Addison-Wesley Professional, 2006.

[6] C. Perrin, "The CIA triad," *TechRepublic Security Blog*, 2008 [Online]. Disponible en http://www.techrepublic.com/blog/it-security/the-cia-triad/. [Accedido 26-07-2016]

[7] C. Rojas, "Incorporación Sistemática de Requisitos de Seguridad de Software," 2008.

[8] M. Paul, *Official (ISC)2 guide to the CSSLP CBK*. CRC Press, 2013.

[9] J. H. Saltzer, M. D. Schroeder, "The protection of information in computer systems," *Proceedings of the IEEE*, vol. 63, no. 9, pp. 1278–1308, 1975.

[10] A. Kerckhoffs, "La cryptographie militaire, ou, des chiffres usités en temps de guerre: Avec un nouveau procédé de déchiffrement applicable aux systèmes à double clef," 1883.

[11] R. Scotka, "Shining a flashlight on mobile application permissions," *Veracode Blog*, 2014. [Online]. Disponible en https://www.veracode.com/blog/2014/04/shining-a-

flashlight-on-mobile-application-permissions. [Accedido 26-07-2016]

[12] M. Korf, E. Oksman, "Native, HTML5, or hybrid: Understanding your mobile application development options," *Developerforce Technical Library*, 2012. [Online]. Disponible en https://developer.salesforce.com/page/Native,_HTML5,_or_Hybrid:_Understanding_Your_Mobile_Application_Development_Options. [Accedido 26-07-2016]

[13] T. Luo, H. Hao, W. Du, Y. Wang, H. Yin, "Attacks on webView in the android system," en *Proceedings of the 27th annual computer security applications conference*, 2011, pp. 343–352.

[14] R. Ding, "The price of free: Privacy leakage in personalized mobile in-app ads," 2016.

[15] C. Rojas, "La nueva forma de desarrollar apps móviles," 2016. [Online]. Disponible en http://injcristianrojas.github.io/analisis/2016/01/29/la-nueva-forma-de-marcar/. [Accedido 26-07-2016]

[16] L. Sweeney, "K-anonymity: A model for protecting privacy," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 10, no. 05, pp. 557–570, 2002.

[17] N. B. Thakkar, "Google Android: An Emerging Innovative Software Platform For Mobile Devices," *International Journal for Scientific Research and Development*, vol. 1, no. 6, pp. 272–278, 2014.

[18] "Android, the world's most popular mobile platform," *Android Developers*, 2015. [Online]. Disponible en https://developer.android.com/about/index.html. [Accedido 26-07-2016]

[19] O. Cinar, *Android quick APIs reference*. Apress, 2015.

[20] P. Gilski, J. Stefanski, "Android OS: A Review," *TEM Journal*, vol. 4, no. 1, pp. 116–120, 2015.

[21] Y. Shao, X. Luo, C. Qian, "Rootguard: Protecting rooted android phones," *Computer*, vol. 47, no. 6, pp. 32–40, 2014.

[22] M. S. Ahmad, N. E. Musa, R. Nadarajah, R. Hassan, N. E. Othman, "Comparison between android and iOS operating system in terms of security," en *Information technology in asia (cITA), 2013 8th international conference on*, 2013, pp. 1–4.

[23] S. Demetriou, X. Zhou, M. Naveed, Y. Lee, K. Yuan, X. Wang, C. A. Gunter, "What's in your dongle and bank account? Mandatory and discretionary protection of android

external resources," 2015.

[24] T. Vidas, N. Christin, L. Cranor, "Curbing android permission creep," en *Proceedings of the web*, 2011, vol. 2.

[25] "Requesting Permissions at Run Time," *Android Developers*, 2015. [Online]. Disponible en https://developer.android.com/training/permissions/requesting.html. [Accedido 26-07-2016]

[26] "Dashboards," *Android Developers*, 2016. [Online]. Disponible en https://developer.android.com/about/dashboards/index.html. [Accedido 26-07-2016]

[27] OpenSignal, "Android fragmentation visualized, august 2015," 2015. [Online]. Disponible en https://opensignal.com/reports/2015/08/android-fragmentation/. [Accedido 26-07-2016]

[28] H. K. Ham, Y. B. Park, "Mobile application compatibility test system design for android fragmentation," en *Software engineering, business continuity, and education*, Springer, 2011, pp. 314–320.

[29] R. Amadeo, "Waiting for Android's inevitable security Armageddon," *Ars Technica*, 2015. [Online]. Disponible en http://arstechnica.com/gadgets/2015/08/waiting-for-androids-inevitable-security-armageddon/. [Accedido 26-07-2016]

[30] Y. Zhou, Z. Wang, W. Zhou, X. Jiang, "Hey, you, get off of my market: Detecting malicious apps in official and alternative android markets." en *NDSS*, 2012, vol. 25, pp. 50–52.

[31] A. P. Felt, M. Finifter, E. Chin, S. Hanna, D. Wagner, "A survey of mobile malware in the wild," en *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, 2011, pp. 3–14.

[32] K. W. Tracy, "Mobile Application Development Experiences on Apple's iOS and Android OS," *Potentials, IEEE*, vol. 31, no. 4, pp. 30–34, 2012.

[33] R. Cannings, "An update on android market security," *Google Mobile Blog*, 2011. [Online]. Disponible en https://googlemobile.blogspot.cl/2011/03/update-on-android-market-security.html. [Accedido 26-07-2016]

[34] "Trojan targeted dozens of games on Google Play," *Dr. Web*, 2016. [Online]. Disponible en https://news.drweb.com/show/?i=9803&lng=en&c=5. [Accedido

26-07-2016]

[35] P. K. Manadhata, J. M. Wing, "An attack surface metric," *IEEE Transactions on Software Engineering*, vol. 37, no. 3, pp. 371–386, 2011.

[36] NowSecure, "Mobile security primer," 2014. [Online]. Disponible en https://books.nowsecure.com/secure-mobile-development/en/primer/mobile-security.html. [Accedido 30-11-2016]

[37] R. Cruz, D. Aranha, "Análise de segurança em aplicativos bancários na plataforma android," *Workshop de Trabalhos de Iniciação Científica e de Graduação (WTICG) do SBSEG*, pp. 377–387, 2015.

[38] A. Sanchez, "Personal banking apps leak info through phone," 2014. [Online]. Disponible en http://blog.ioactive.com/2014/01/personal-banking-apps-leak-info-through.html. [Accedido 30-11-2016]

[39] P. Huichalaf, "¿Está en Chile permitido modificar el software o desbloquear un celular?" 2010. [Online]. Disponible en http://old.elmostrador.cl/seleccion/2010/08/10/esta-permitido-en-chile-modificar-el-software-o-desbloquear-un-celular/. [Accedido 02-12-2016]

[40] "2016 NowSecure Mobile Security Report," *NowSecure*, 2016. [Online]. Disponible en https://info.nowsecure.com/rs/201-XEW-873/images/2016-NowSecure-mobile-security-report.pdf. [Accedido 26-07-2016]

[41] S. Egelman, S. Jain, R. S. Portnoff, K. Liao, S. Consolvo, D. Wagner, "Are you ready to lock?" en *Proceedings of the 2014 aCM sIGSAC conference on computer and communications security*, 2014, pp. 750–761.

[42] M. Terpstra, "WhatsApp & privacy," *Radboud University Nijmegen, Netherlands*, 2013.

[43] S. Shepherd, "Vulnerability Disclosure: How Do We Define Responsible Disclosure?" *GIAC SEC Practical Repository, SANS Inst*, vol. 9, 2003.

[44] Alexis Ibarra, "Hackers se reúnen en Chile para revelar las brechas de seguridad." [Online]. Disponible en http://www.economiaynegocios.cl/noticias/noticias.asp?id=193987. [Accedido 27-07-2016]

[45] A. P. Felt, E. Chin, S. Hanna, D. Song, D. Wagner, "Android permissions demystified," en *Proceedings of the 18th aCM conference on computer and communications security*, 2011, pp. 627–638.

[46] M. Fasel, "A Good Look at Android Location Data," 2011. [Online]. Disponible en https://blog.shinetech.com/2011/10/14/a-good-look-at-android-location-data/.

[Accedido 26-07-2016]

[47] I. Ristić, "Bulletproof SSL and TLS," *Feisty Duck*, 2014.

[48] B. Schneier, "When Will We See Collisions for SHA-1," *Schneier on Security*, 2012. [Online]. Disponible en https://www.schneier.com/blog/archives/2012/10/when_will_we_se.html. [Accedido 26-07-2016]

[49] I. Ristić, "SHA1 deprecation: What you need to know," 2014. [Online]. Disponible en https://community.qualys.com/blogs/securitylabs/2014/09/09/sha1-deprecation-what-you-need-to-know. [Accedido 26-07-2016]

[50] H. Krawczyk, "Perfect forward secrecy," en *Encyclopedia of cryptography and security*, Springer, 2005, pp. 457–458.

[51] J. Hoffman-Andrews, "Forward Secrecy at Twitter," *The Twitter Engineering Blog*, 2013. [Online]. Disponible en https://blog.twitter.com/2013/forward-secrecy-at-twitter-0. [Accedido 26-07-2016]

[52] I. Ristić, "CRIME: Information Leakage Attack against SSL/TLS," 2012. [Online]. Disponible en https://community.qualys.com/blogs/securitylabs/2012/09/14/crime-information-leakage-attack-against-ssltls. [Accedido 26-07-2016]

[53] A. Popov, "Prohibiting RC4 Cipher Suites," *Computer Science*, vol. 2355, pp. 152–164, 2015.

[54] B. Möller, T. Duong, K. Kotowicz, "This POODLE bites: exploiting the SSL 3.0 fallback." Google, 2014.

[55] B. Moeller, A. Langley, "TLS Fallback Signaling Cipher Suite Value (SCSV) for Preventing Protocol Downgrade Attacks," 2015.

[56] M. Grassi, S. Guerrero, "The Nightmare Behind the Cross-Platform Apps Dream," en *BlackHat Asia 2015*.

[57] P. Ducklin, "The SLOTH attacks: why laziness about cryptography puts security at risk," *Sophos Naked Security*, 2016. [Online]. Disponible en https://nakedsecurity.sophos.com/2016/01/08/the-sloth-attacks-why-laziness-about-cryptography-puts-security-at-risk/. [Accedido 26-07-2016]

[58] D. Goodin, "User data plundering by Android and iOS apps is as rampant as you suspected," *Ars Technica*, 2015. [Online]. Disponible en http://arstechnica.com/security/2015/11/user-data-plundering-by-android-and-ios-apps-is-as-rampant-as-

you-suspected/. [Accedido 26-07-2016]

[59] Computer Science and Engineering - University of Michigan, "Computer science researchers aim to securely encrypt every website," 2014. [Online]. Disponible en https://www.eecs.umich.edu/eecs/about/articles/2014/Lets-Encrypt.html. [Accedido 02-12-2016]

[60] M. E. Joorabchi, A. Mesbah, "Reverse engineering iOS mobile applications," en *Reverse engineering (wCRE), 2012 19th working conference on*, 2012, pp. 177–186.

[61] Trustworthy Internet Movement, "SSL Pulse." [Online]. Disponible en https://www.trustworthyinternet.org/ssl-pulse/. [Accedido 26-07-2016]

Anexo: Informes entregados a los bancos

A continuación, se muestran los informes confeccionados y entregados a los bancos, en versión anonimizada. Cada uno de estos informes contiene los siguientes items:

- Introducción.
- Detalles de la aplicación analizada: tipo de aplicación y tipo de conexión.
- Vulnerabilidades encontradas en la aplicación y su procedimiento de mitigación correspondiente.
- Recursos interesantes: Páginas con manuales de buenas prácticas, herramientas y bibliotecas API para mejorar la seguridad, etc.

Informe Banco 1

Introducción

El presente informe consiste en un conjunto de recomendaciones de mitigación de vulnerabilidades de seguridad encontradas en la aplicación móvil para Android de Banco 1, y es parte de las tareas de investigación tendientes al trabajo de Tesis de Magister de quien escribe. El informe aquí presentado contiene lo siguiente:

1. Vulnerabilidades encontradas en su aplicación y riesgos asociados.
2. Formas de mitigación de tales vulnerabilidades.

El objetivo es que el equipo de desarrollo de la aplicación bancaria de Banco 1 pueda accionar las recomendaciones aquí indicadas en ésta.

Detalles de la aplicación analizada

- Tipo de aplicación: Nativa
- Tipo de conexión: REST-JSON

Vulnerabilidades encontradas y su mitigación

Potencial Tampering de datos

La aplicación, al momento de realizar una transferencia de dinero, pudiera ser vulnerable a que un usuario malicioso, o algún malware instalado en el teléfono modifique los datos de transacciones de transferencias de dinero entre cuentas, antes de que esa información sea enviada al servidor vía HTTPS. Éste ataque funcionaría de la forma que se muestra en la figura:

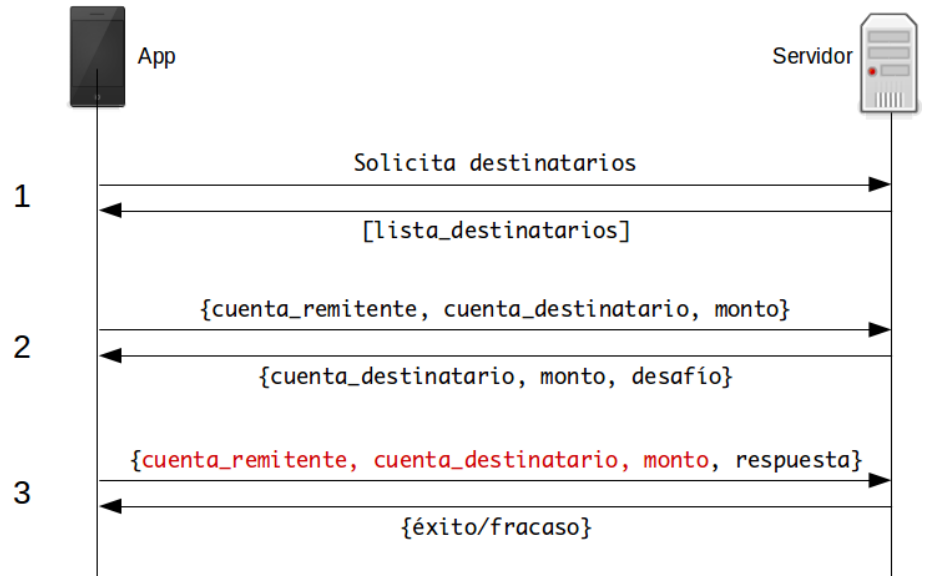


Figura 1: Modificación maliciosa de datos de transferencias

1. La aplicación solicita al servidor la lista de destinatarios registrados por el usuario en su cuenta, ante lo cual éste se la envía.
2. El usuario selecciona un destinatario y el monto a transferir en la app, la cual envía esa información al servidor junto con la cuenta remitente. El servidor responde enviándole a la app los datos de la cuenta destinario, el monto a transferir y un desafío, el cual en el caso del Banco 1 son tres coordenadas de la *Tarjeta de Clave Dinámica* del usuario.

3. El usuario ingresa la respuesta al desafío solicitado, y la aplicación envía ese dato, junto a la cuenta remitente, la cuenta de destinatario y el monto a transferir. Estos datos pueden ser falseados de tal forma que la cuenta remitente no sea la correcta, la cuenta destino no sea la correcta o el monto no sea el que el usuario deseó transferir. Si el servidor no valida esta información, hay riesgo serio de robo de dinero mediante transferencias a cuentas no deseadas.

Para mitigar este problema, se propone el uso de un ID de transacción, como se muestra en la siguiente figura:

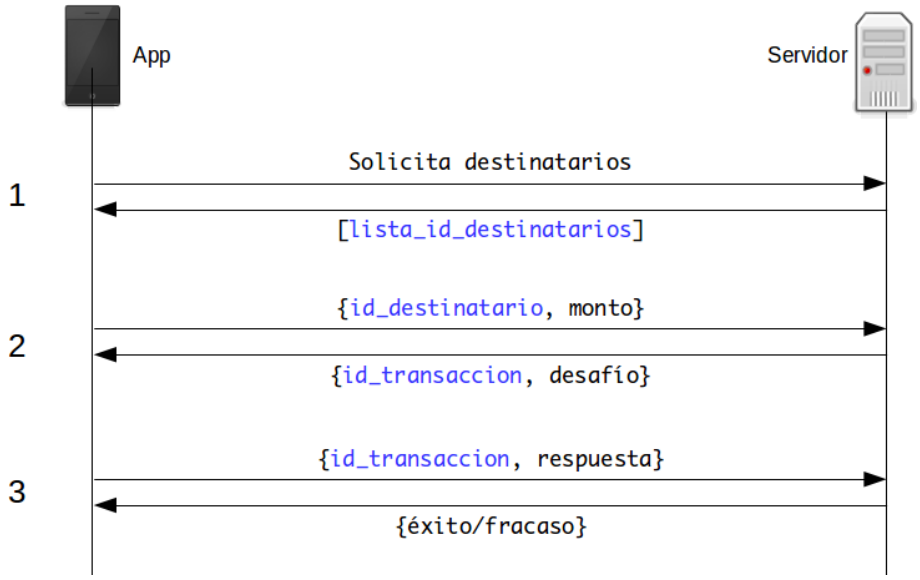


Figura 2: Mitigación ante potencial tampering de datos de transferencias

En este caso, en vez de dejar información validable en manos del cliente, la cual podría ser modificada maliciosamente, se deja esa información en el servidor y ésta se asocia con un ID de transacción, el cual es enviado a la app junto con el desafío. El usuario ingresa la respuesta, a la cual la app adosa este ID de transacción y la envía al servidor, el cual realiza los chequeos usuales (existencia de la cuenta de destinatario, el cliente tiene fondos suficientes, etc.), realiza la transacción, y responde al usuario si ésta fue exitosa o fallida.

Adicionalmente, antes de la validación, el servidor puede enviar en vez de una lista de destinatarios con mucha información, sólo enviar una lista con pares `{id_destinatario, nombre_destinatario}` en el paso 1, y cuando el usuario seleccione un destinatario, sólo enviar su id_destinatario en el paso 2.

Abuso de logging (código de pruebas olvidado)

Android, como todo sistema Linux, contiene un log interno el cual puede ser utilizado para reportar eventos ocurridos dentro del sistema. Generalmente, las aplicaciones reportan automáticamente al log eventos relacionados con uso de recursos.

Asimismo, los mismos desarrolladores pueden utilizar el log para realizar tareas de corrección de errores (debugging). Por ejemplo, verificar valores de variables y su progresión durante la ejecución de la aplicación. Lamentablemente, muchas veces ese código de prueba es olvidado en la aplicación y pasa a producción, provocando filtración de información aprovechable por cualquiera que pudiera leer el log, por ejemplo:

- Otras aplicaciones que tengan permisos para leerlo (mediante el permiso `android.permission.READ_LOGS`).
- Malware instalado en el equipo.
- Alguien que active el modo de desarrollador en el teléfono y pueda leer el log mediante la conexión USB desde un computador.

La aplicación Banco 1, envía al log interno de Android los siguientes datos sensibles:

- Información de cuenta corriente del usuario.
- Saldos.
- Operaciones sobre cuenta corriente.
- Datos de tarjeta de crédito.
- Desafío y respuesta a desafío de tarjeta de coordenadas.
- Datos del dastinatario de transferencia bancaria.

Se recomienda encarecidamente eliminar tal código de la versión de producción de la aplicación.

Almacenamiento de información sensible en base de datos SQLite

Como parte de su funcionamiento, muchas aplicaciones almacenan información en bases de datos SQLite, sin cifrado alguno. En el caso de la aplicación Banco 1, se almacena el nombre del usuario y su RUT. Se recomienda almacenar tal información en forma cifrada. Para ello se debe generar una llave de cifrado aleatoria y que ésta sea almacenada, no en el área de almacenamiento de la aplicación, sino que en un sistema llamado Android Keystore, el cual almacena claves en forma segura, cifrándolos con

una llave basada en el método de bloqueo del sistema elegido por el usuario, sea éste un PIN, password o patrón con el cual se autentifica ante el teléfono. Con esta llave se puede encriptar la información antes de guardarla, por ejemplo en la base de datos SQLite o en Shared Preferences, y luego usar esta misma llave para descifrar la información. Para mayor información respecto de Android Keystore, ir a la sección "Recursos Interesantes".

Problemas con la implementación del Timeout

De acuerdo con un estudio desarrollado por la Universidad de Berkeley y Google, el 62 % de los usuarios de teléfonos móviles no utiliza ningún método de bloqueo de sus equipos, sea éste el uso de un PIN, password o patrón[2]. Es por esto que la implementación de un timeout de cierre en una aplicación móvil es de extrema importancia, ya que un teléfono desbloqueado puede ser fuente, al menos, de filtración de información bancaria.

Si bien la aplicación Banco 1 implementa un timeout, éste debe mejorar su implementación, ya que su funcionamiento consiste en que al finalizar el tiempo de la sesión, aparece una ventana alertando al usuario del cierre automático de ésta. Tal ventana no cubre toda la pantalla del aparato, y deja ver parte del contenido. En vez de eso, la aplicación debería volver a la pantalla de login automáticamente.

Problemas con la conexión HTTPS

Problemas con la cadena de certificación De acuerdo con el análisis realizado, se determinó que la cadena de certificación del servidor tiene 2 problemas: Un certificado de más y los demás certificados corren riesgo de deprecación.

El servidor provee al momento de la negociación de conexión un certificado de más: el VeriSign Class 3 Public Primary Certification Authority - G5 (fingerprint `4eb6d578499b1ccf5f581ead56be3d9b6744a5e5`), el cual debe ser removido, ya que al ser de una autoridad certificadora confiable, ya está instalado en los clientes y su inclusión por parte del servidor ralentiza la negociación HTTPS.

Los otros dos certificados son los siguientes:

- www.banco1.cl (fingerprint anonimizado)

[2]https://www.eecs.berkeley.edu/~daw/papers/lock-ccs14.pdf

- VeriSign Class 3 Extended Validation SSL SGC CA (fingerprint `4a8a2a0e276ff33b5dd88a362146010f2a8b6aee`)

Ambos certificados son firmados con el algoritmo SHA1, lo cual causa que serán deprecados a partir del 1 de enero de 2016[3] y los clientes Android podrían arrojar errores de validación de certificados posterior a esa fecha. Se recomienda contactar a la autoridad certificadora y solicitar la renovación de ambos certificados.

Falta de protección ante downgrades de protocolo Se recomienda que el servidor utilice la biblioteca OpenSSL versión 1.0.1j o superior, la cual soporta la propiedad `TLS_FALLBACK_SCSV` y permite una mayor protección ante potenciales *downgrades* de protocolo. Un downgrade de protocolo es es un ataque el cual es montado mediante la técnica de *hombre en el medio*[4] por algún inescrupuloso que pudiera estar monitoreando redes Wi-Fi, por ejemplo, y en el cual causa que una conexión que utiliza una versión del protocolo de comunicación segura (por ejemplo TLSv1.1), realice un downgrade hacia una versión más insegura (por ejemplo TLSv1.0).

Uso del cifrador RC4 EL cifrador RC4 es vulnerable ante múltiples ataques criptoanalíticos y su uso está prohibido en configuraciones modernas[5]. La recomendación es eliminarlo de la lista de cifradores, lo cual se tratará en la sección "Mitigación".

Carencia de Forward Secrecy Forward Secrecy (secreto hacia el futuro) es una propiedad que causa que, si un atacante obtiene la llave secreta del servidor HTTPS, no podrá obtener las llaves de todas las sesiones que alguna vez se hayan generado con esa llave, y por lo tanto no podrá desencriptar el contenido de conversaciones que haya tenido el servidor con sus clientes.

El algoritmo DH (Diffie-Hellman) permite implementar Forward Secrecy. Una de las razones por las cuales los servidores no acostumbran implementarlo es por el hecho de que consume más recursos computacionales que el algoritmo RSA, usado profusamente hoy en día. Sin embargo, el uso de Diffie-Hellman con Curva Elíptica reduce significativamente el consumo de recursos. Generalmente este algoritmo se denomina ECDHE[6].

[3]Qualys SSL Labs, "SHA1 Deprecation: What You Need to Know" https://community.qualys.com/blogs/securitylabs/2014/09/09/sha1-deprecation-what-you-need-to-know

[4]Man-In-The-Middle: https://en.wikipedia.org/wiki/Man_in_the_middle_attack

[5]https://tools.ietf.org/html/rfc7465

[6]Qualys SSL Labs, "Deploying Forward Secrecy": https://community.qualys.com/blogs/securitylabs/2013/06/25/ssl-labs-deploying-forward-secrecy

Mitigación Las vulnerabilidades mencionadas en esta sección pueden ser corregidas cambiando la configuración SSL del servidor web al cual se conecta la aplicación. El analisis no arrojó resultados concluyentes respecto de qué servidor usa www.banco1.cl, por lo tanto no se pueden dar recomendaciones específicas para la configuración SSL del servidor. Sin embargo, existe una guía que puede ser útil para tal tarea, desarrollada por la Fundación Mozilla: https://wiki.mozilla.org/Security/Server_Side_TLS. Posterior a la configuración SSL del servidor, ésta se puede probar con las herramientas Cipherscan y Qualys SSL Test, indicadas en la sección "Recursos Interesantes".

Permisos

En el archivo `AndroidManifest.xml` se definen los permisos que la aplicación solicita al usuario para hacer uso de sus recursos al momento de instalar la aplicación. Algunos de estos permisos son riesgosos, y para el caso de Banco 1, son los siguientes:

Acceso a información acerca de otras aplicaciones en uso El permiso `android.permission.GET_TASKS` permite a la aplicación acceder a información de otras aplicaciones que estén ejecutándose en el equipo. Por problemas de privacidad fue deprecado en la API versión 21 (Lollipop). Se recomienda su eliminación de la lista de permisos.

Escritura en el almacenamiento externo del teléfono El teléfono incluye un área de almacenamiento externo, el cual no tiene las protecciones incluidas en el área de almacenamiento correspondiente a la aplicación, y por lo tanto, cualquier archivo almacenado en esta área externa podría ser accesible por otras aplicaciones y a través de conexiones USB al teléfono. Se recomienda remover el permiso `android.permission.WRITE_EXTERNAL_STORAGE`.

Uso de georreferenciación Una función ofrecida por las aplicaciones bancarias nacionales es la que permite al usuario obtener una lista o mapa de los cajeros Redbanc o sucursales del banco más cercanas al lugar en el que se encuentra. Para ello se requiere acceso a los sistemas de georreferenciación del teléfono, que son principalmente de dos tipos:

- Georreferenciación fina: Éste tipo de georreferenciación se basa en el uso de satélites GPS y se puede acceder a ello mediante el permiso

`android.permission.ACCESS_FINE_LOCATION`. Su precisión es de entre 2 y 20 metros.

- Georreferenciación gruesa: Se basa en el uso de torres celulares y hotspots Wifi para entregar una ubicación aproximada del aparato. También se conoce como A-GPS (Assisted GPS) o Network Location Provider, y su precisión es de entre 100 y 200 metros. Se puede acceder a este tipo de georreferenciación con el permiso `android.permission.ACCESS_COARSE_LOCATION`[7].

Dado que la función de localización de cajeros y sucursales no requiere mayor precisión, la recomendación en este caso es quitar el permiso `ACCESS_FINE_LOCATION` y dejar el `ACCESS_COARSE_LOCATION`. Además del beneficio para la privacidad del usuario, la geolocalización gruesa consume menos energía de la batería del aparato y es mucho más rápida en la obtención de información geográfica de éste.

Recursos interesantes

- **NowSecure: 42+ Best Practices for Secure iOS and Android Development**. Este manual incluye una serie de consejos y buenas prácticas para desarrollar aplicaciones seguras, tanto en iOS como en Android: https://goo.gl/ivJD4M
- **Cipherscan**: Herramienta via linea de comandos que analiza la configuración de servidores HTTPS y entrega reportes. Puede ser usada para analizar servidores en estado de pre-producción: https://github.com/jvehent/cipherscan.
- **Qualys SSL Test**: Esta herramienta web gratuita fue utilizada para analizar la configuración actual del servidor en producción, y es utilizable ya para cuando el servidor con su nueva configuración ya se encuentre expuesto a Internet. Entrega información sobre configuración de certificados, protocolos, cifradores, etc., y le *pone nota* al servidor: https://www.ssllabs.com/ssltest/.
- **Android Keystore**: Es un sistema de almacenamiento seguro de llaves o valores que se desea que sean secretos. Funciona como una tabla de datos encriptada mediante el PIN, password o patrón del usuario del equipo, por lo tanto no es fácil de obtener. Puede ver la documentación oficial de Android al respecto en http://developer.android.com/intl/es/training/articles/keystore.html y un ejemplo de su uso en http://www.androidauthority.com/use-android-keystore-store-passwords-sensitive-information-623779/.

[7]The Shine Blog, "A Good Look at Android Location Data": http://blog.shinetech.com/2011/10/14/a-good-look-at-android-location-data/

Informe Banco 2

Introducción

El presente informe consiste en un conjunto de recomendaciones de mitigación de vulnerabilidades de seguridad encontradas en la aplicación Banco 2 para Android, y es parte de las tareas de investigación tendientes al trabajo de Tesis de Magister de quien escribe. El informe aquí presentado contiene lo siguiente:

1. Vulnerabilidades encontradas en su aplicación y riesgos asociados.
2. Formas de mitigación de tales vulnerabilidades.

El objetivo es que el equipo de desarrollo de la aplicación bancaria del Banco 2 pueda accionar las recomendaciones aquí indicadas en ésta.

Detalles de la aplicación analizada

- Tipo de aplicación: Híbrida (Titanium Appcelerator 0.8.6)
- Tipo de conexión: REST-JSON

Vulnerabilidades encontradas y su mitigación

Potencial Tampering de datos

La aplicación, al momento de realizar una transferencia de dinero, pudiera ser vulnerable a que un usuario malicioso, o algún malware instalado en el teléfono modifique los datos de transacciones de transferencias de dinero entre cuentas, antes de que esa información sea enviada al servidor vía HTTPS. Éste ataque funcionaría de la forma que se muestra en la figura:

1. La aplicación solicita al servidor la lista de destinatarios registrados por el usuario en su cuenta, ante lo cual éste se la envía.
2. El usuario selecciona un destinatario y el monto a transferir en la app, la cual envía esa información al servidor junto con la cuenta remitente. El servidor responde enviándole a la app los datos de la cuenta destinario, el monto a transferir y un desafío, el cual en el caso del Banco 2 es la solicitud de un número generado por un token llamado *Multipass*.

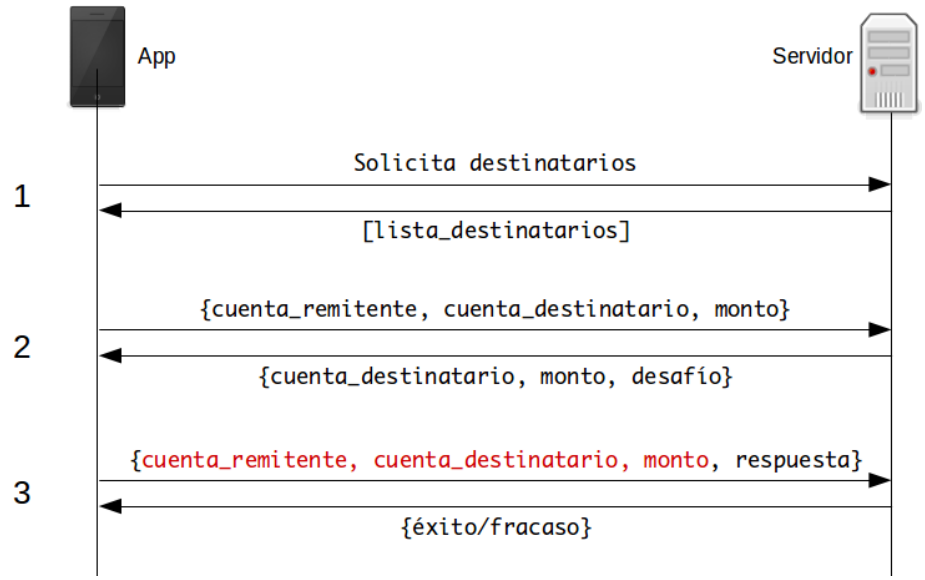


Figura 3: Modificación maliciosa de datos de transferencias

3. El usuario ingresa la respuesta al desafío solicitado, y la aplicación envía ese dato, junto a la cuenta remitente, la cuenta de destinatario y el monto a transferir. Estos datos pueden ser falseados de tal forma que la cuenta remitente no sea la correcta, la cuenta destino no sea la correcta o el monto no sea el que el usuario deseó transferir. Si el servidor no valida esta información, hay riesgo serio de robo de dinero mediante transferencias a cuentas no deseadas.

Para mitigar este problema, se propone el uso de un ID de transacción, como se muestra en la siguiente figura:

En este caso, en vez de dejar información validable en manos del cliente, la cual podría ser modificada maliciosamente, se deja esa información en el servidor y ésta se asocia con un ID de transacción, el cual es enviado a la app junto con el desafío. El usuario ingresa la respuesta, a la cual la app adosa este ID de transacción y la envía al servidor, el cual realiza los chequeos usuales (existencia de la cuenta de destinatario, el cliente tiene fondos suficientes, etc.), realiza la transacción, y responde al usuario si ésta fue exitosa o fallida.

Adicionalmente, antes de la validación, el servidor puede enviar en vez de una lista de destinatarios con mucha información, sólo enviar una lista con pares `{id_destinatario, nombre_destinatario}` en el paso 1, y cuando el usuario seleccione un destinatario, sólo enviar su id_destinatario en el paso 2.

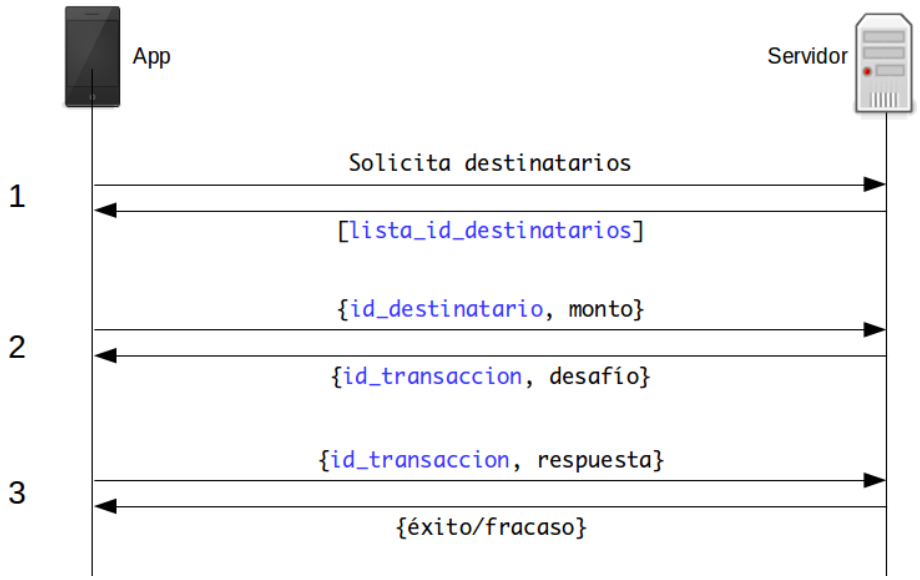


Figura 4: Mitigación ante potencial tampering de datos de transferencias

Almacenamiento de información sensible en base de datos SQLite

Como parte de su funcionamiento, muchas aplicaciones almacenan información en bases de datos SQLite, sin cifrado alguno. En el caso de la aplicación Banco 2, almacena el nombre del usuario, su RUT y el ANDROID_ID[8]. Se recomienda almacenar tal información en forma cifrada. Para ello se debe generar una llave de cifrado aleatoria y que ésta sea almacenada, no en el área de almacenamiento de la aplicación, sino que en un sistema llamado Android Keystore, el cual almacena claves en forma segura, cifrándolos con una llave basada en el método de bloqueo del sistema elegido por el usuario, sea éste un PIN, password o patrón con el cual se autentifica ante el teléfono. Con esta llave se puede encriptar la información antes de guardarla, por ejemplo en la base de datos SQLite o en Shared Preferences, y luego usar esta misma llave para descifrar la información. Para mayor información respecto de Android Keystore, ir a la sección "Recursos Interesantes".

Abuso de logging (código de pruebas olvidado)

Android, como todo sistema Linux, contiene un log interno el cual puede ser utilizado para reportar eventos ocurridos dentro del sistema. Generalmente, las aplicaciones reportan automáticamente al log eventos relacionados con uso de recursos.

[8]El ANDROID_ID es un valor asignado al teléfono al momento de su activación con Google, el cual permanece en el teléfono hasta que es formateado.

Asimismo, los mismos desarrolladores pueden utilizar el log para realizar tareas de corrección de errores (debugging). Por ejemplo, verificar valores de variables y su progresión durante la ejecución de la aplicación. Lamentablemente, muchas veces ese código de prueba es olvidado en la aplicación y pasa a producción, provocando filtración de información aprovechable por cualquiera que pudiera leer el log, por ejemplo:

- Otras aplicaciones que tengan permisos para leerlo (mediante el permiso `android.permission.READ_LOGS`).
- Malware instalado en el equipo.
- Alguien que active el modo de desarrollador en el teléfono y pueda leer el log mediante la conexión USB desde un computador.

La aplicación Banco 2, envía al log interno de Android el RUT del usuario y el mail del destinatario al cuál se le realiza una transferencia de dinero. Se recomienda encarecidamente eliminar tal código de la versión de producción de la aplicación.

Problemas de privacidad

Como parte de su implementación, la aplicación Banco 2 hace uso del servicio APM[9] llamado Crittercism. Si bien, este servicio puede ser muy útil para el monitoreo de performance de la aplicación, de la forma en que está configurado para ésta versión de la aplicación, conlleva el problema de que está enviando a una tercera parte información que permite identificar a un usuario de la aplicación, sin su conocimiento, por ejemplo su RUT. Se recomienda no enviar este tipo de información a Crittercism, menos sin el conocimiento y consentimiento del usuario, en base a la política de seguridad establecida en el sitio web del Banco 2.

Exceso de confianza en medidas anti-ingeniería reversa

La aplicación usa un sistema que busca evitar que un analista (o algún atacante) obtenga su código fuente mediante ingeniería reversa. Cabe indicar aquí que, aparte de un trabajo de investigación basado en información ya publicada, no es demasiado dificil obtener el código fuente de la aplicación, incluso de manera automatizada[10].

[9] Application Performance Management.

[10] Ver el trabajo de Marco Grassi y Sebastián Guerrero en la conferencia de seguridad BlackHat Asia 2015. Lo indicado entre las slides 28 y 32 es particularmente interesante para el framework Titanium Appcelerator, usado en algunas de las aplicaciones analizadas: https://www.nowsecure.com/blog/2015/04/13/the-nightmare-behind-the-cross-platform-apps-dream/

La recomendación es basarse en el Principio de Diseño Abierto[11]: No confiar la seguridad de una aplicación en secretos supuestamente imposibles de obtener. Siempre habrá alguna forma de obtener información acerca de la aplicación, y de utilizarla con malos fines.

Problemas con la conexión HTTPS

Cadena de certificación mal hecha De acuerdo con el análisis realizado, se determinó que la cadena de certificación del servidor provee un certificado de más al momento de realizarse la conexión. El certificado es el GlobalSign Root CA (fingerprint `b1bc968bd4f49d622aa89a81f2150152a41d829c`), que es el certificado raíz de la autoridad certificadora, el cual es redundante ya que los clientes (browsers, smartphones, etc.) ya lo incluyen como parte de su instalación por defecto. La recomendación es eliminar ese certificado y dejar los demás.

Falta de protección ante downgrades de protocolo Se recomienda que el servidor utilice la biblioteca OpenSSL versión 1.0.1j o superior, la cual soporta la propiedad `TLS_FALLBACK_SCSV` y permite una mayor protección ante potenciales *downgrades* de protocolo. Un downgrade de protocolo es es un ataque el cual es montado mediante la técnica de *hombre en el medio*[12] por algún inescrupuloso que pudiera estar monitoreando redes Wi-Fi, por ejemplo, y en el cual causa que una conexión que utiliza una versión del protocolo de comunicación segura (por ejemplo TLSv1.1), realice un downgrade hacia una versión más insegura (por ejemplo TLSv1.0).

Uso del cifrador RC4 EL cifrador RC4 es vulnerable ante múltiples ataques criptoanalíticos y su uso está prohibido en configuraciones modernas[13]. La recomendación es eliminarlo de la lista de cifradores, lo cual se tratará en la sección "Mitigación".

Carencia de Forward Secrecy Forward Secrecy (secreto hacia el futuro) es una propiedad que causa que, si un atacante obtiene la llave secreta del servidor HTTPS, no podrá obtener las llaves de todas las sesiones que alguna vez se hayan generado con esa llave, y por lo tanto no podrá desencriptar el contenido de conversaciones que haya tenido el servidor con sus clientes.

[11] https://www.owasp.org/index.php/Avoid_security_by_obscurity
[12] Man-In-The-Middle: https://en.wikipedia.org/wiki/Man_in_the_middle_attack
[13] https://tools.ietf.org/html/rfc7465

El algoritmo DH (Diffie-Hellman) permite implementar Forward Secrecy. Una de las razones por las cuales los servidores no acostumbran implementarlo es por el hecho de que consume más recursos computacionales que el algoritmo RSA, usado profusamente hoy en día. Sin embargo, el uso de Diffie-Hellman con Curva Elíptica reduce significativamente el consumo de recursos. Generalmente este algoritmo se denomina ECDHE[14].

Mitigación Las vulnerabilidades mencionadas en esta sección pueden ser corregidas cambiando la configuración SSL del servidor web al cual se conecta la aplicación. De acuerdo con el análisis, el servidor HTTP utilizado por la aplicación es un servidor Apache. La Fundación Mozilla dispuso en línea una herramienta para generar automáticamente configuraciones SSL para servidores Apache. Se puede encontrar aquí: https://mozilla.github.io/server-side-tls/ssl-config-generator/. Posterior a su configuración, ésta se puede probar con las herramientas Cipherscan y Qualys SSL Test, indicadas en la sección "Recursos Interesantes".

Abuso de permisos

En el archivo `AndroidManifest.xml` se definen los permisos que la aplicación solicita al usuario para hacer uso de sus recursos al momento de instalar la aplicación. Algunos de estos permisos son riesgosos, y para el caso del Banco 2, son los siguientes:

Escritura en el almacenamiento externo del teléfono El teléfono incluye un área de almacenamiento externo, el cual no tiene las protecciones incluidas en el área de almacenamiento correspondiente a la aplicación, y por lo tanto, cualquier archivo almacenado en esta área externa podría ser accesible por otras aplicaciones y a través de conexiones USB al teléfono. Se recomienda remover el permiso `android.permission.WRITE_EXTERNAL_STORAGE`.

Uso de georreferenciación Una función ofrecida por las aplicaciones bancarias nacionales es la que permite al usuario obtener una lista o mapa de los cajeros Redbanc o sucursales del banco más cercanas al lugar en el que se encuentra. Para ello se requiere acceso a los sistemas de georreferenciación del teléfono, que son principalmente de dos tipos:

[14]Qualys SSL Labs, "Deploying Forward Secrecy": https://community.qualys.com/blogs/securitylabs/2013/06/25/ssl-labs-deploying-forward-secrecy

- Georreferenciación fina: Éste tipo de georreferenciación se basa en el uso de satélites GPS y se puede acceder a ello mediante el permiso `android.permission.ACCESS_FINE_LOCATION`. Su precisión es de entre 2 y 20 metros.
- Georreferenciación gruesa: Se basa en el uso de torres celulares y hotspots Wifi para entregar una ubicación aproximada del aparato. También se conoce como A-GPS (Assisted GPS) o Network Location Provider, y su precisión es de entre 100 y 200 metros. Se puede acceder a este tipo de georreferenciación con el permiso `android.permission.ACCESS_COARSE_LOCATION`[15].

Dado que la función de localización de cajeros y sucursales no requiere mayor precisión, la recomendación en este caso es quitar el permiso `ACCESS_FINE_LOCATION` y dejar el `ACCESS_COARSE_LOCATION`. Además del beneficio para la privacidad del usuario, la geolocalización gruesa consume menos energía de la batería del aparato y es mucho más rápida en la obtención de información geográfica de éste.

Lectura de logs del sistema En la sección "Abuso de logging", se mencionó que la aplicación filtraba información al log del aparato, el cual no es cifrado, y puede ser accesible por cualquier otra aplicación. Para ese acceso se requiere el permiso `android.permission.READ_LOGS`, y ésta aplicación hace uso de este permiso. Esto constituye un potencial problema de privacidad, ya que al log podría filtrarse información sensible de otras aplicaciones.

Acceso a información acerca de otras aplicaciones en uso El permiso `android.permission.GET_TASKS` permite a la aplicación acceder a información de otras aplicaciones que estén ejecutándose en el equipo. Por problemas de privacidad fue deprecado en la API versión 21 (Lollipop). Se recomienda su eliminación de la lista de permisos.

Recursos interesantes

- **NowSecure: 42+ Best Practices for Secure iOS and Android Development**. Este manual incluye una serie de consejos y buenas prácticas para desarrollar aplicaciones seguras, tanto en iOS como en Android: https://goo.gl/ivJD4M

[15]The Shine Blog, "A Good Look at Android Location Data": http://blog.shinetech.com/2011/10/14/a-good-look-at-android-location-data/

- **Cipherscan**: Herramienta via linea de comandos que analiza la configuración de servidores HTTPS y entrega reportes. Puede ser usada para analizar servidores en estado de pre-producción: https://github.com/jvehent/cipherscan.
- **Qualys SSL Test**: Esta herramienta web gratuita fue utilizada para analizar la configuración actual del servidor en producción, y es utilizable ya para cuando el servidor con su nueva configuración ya se encuentre expuesto a Internet. Entrega información sobre configuración de certificados, protocolos, cifradores, etc., y le *pone nota* al servidor: https://www.ssllabs.com/ssltest/
- **Android Keystore**: Es un sistema de almacenamiento seguro de llaves o valores que se desea que sean secretos. Funciona como una tabla de datos encriptada mediante el PIN, password o patrón del usuario del equipo, por lo tanto no es fácil de obtener. Puede ver la documentación oficial de Android al respecto en http://developer.android.com/intl/es/training/articles/keystore.html y un ejemplo de su uso en http://www.androidauthority.com/use-android-keystore-store-passwords-sensitive-information-623779/

Informe Banco 3

Introducción

El presente informe consiste en un conjunto de recomendaciones de mitigación de vulnerabilidades de seguridad encontradas en la aplicación móvil para Android de Banco 3, y es parte de las tareas de investigación tendientes al trabajo de Tesis de Magister de quien escribe. El informe aquí presentado contiene lo siguiente:

1. Vulnerabilidades encontradas en su aplicación y riesgos asociados.
2. Formas de mitigación de tales vulnerabilidades.

El objetivo es que el equipo de desarrollo de la aplicación bancaria de Banco 3 pueda accionar las recomendaciones aquí indicadas en ésta.

Detalles de la aplicación analizada

- Tipo de aplicación: Nativa
- Tipo de conexión: HTTP-XML

Vulnerabilidades encontradas y su mitigación

Potencial Tampering de datos

La aplicación, al momento de realizar una transferencia de dinero, pudiera ser vulnerable a que un usuario malicioso, o algún malware instalado en el teléfono modifique los datos de transacciones de transferencias de dinero entre cuentas, antes de que esa información sea enviada al servidor vía HTTPS. Éste ataque funcionaría de la forma que se muestra en la figura:

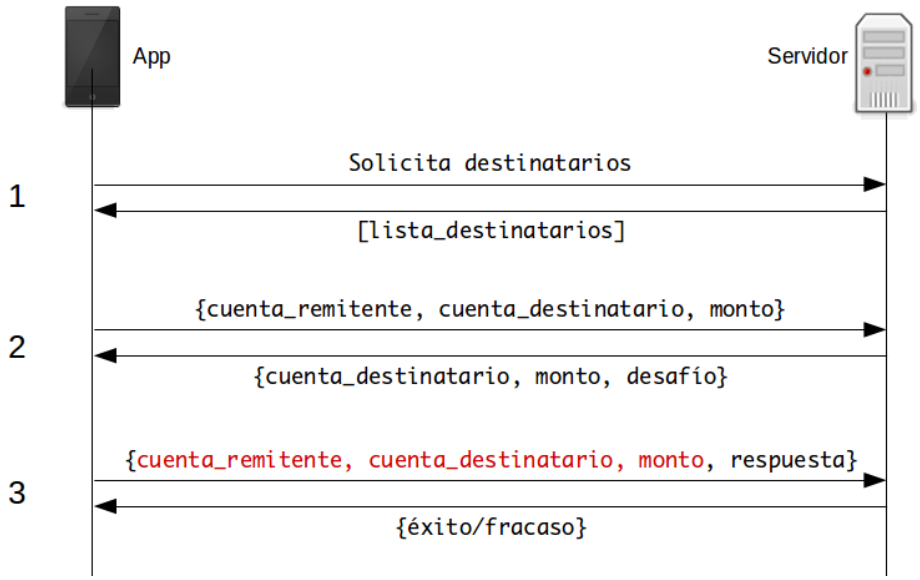


Figura 5: Modificación maliciosa de datos de transferencias

1. La aplicación solicita al servidor la lista de destinatarios registrados por el usuario en su cuenta, ante lo cual éste se la envía.
2. El usuario selecciona un destinatario y el monto a transferir en la app, la cual envía esa información al servidor junto con la cuenta remitente. El servidor responde enviándole a la app los datos de la cuenta destinario, el monto a transferir y un desafío, el cual en el caso del Banco 3 es la solicitud de un número generado por un token.
3. El usuario ingresa la respuesta al desafío solicitado, y la aplicación envía ese dato, junto a la cuenta remitente, la cuenta de destinatario y el monto a transferir. Estos datos pueden ser falseados de tal forma que la cuenta remitente no sea la correcta, la cuenta destino no sea la correcta o el monto no sea el que el usuario deseó transferir. Si el servidor no valida esta información, hay riesgo serio de robo de dinero mediante transferencias a cuentas no deseadas.

Para mitigar este problema, se propone el uso de un ID de transacción, como se muestra en la siguiente figura:

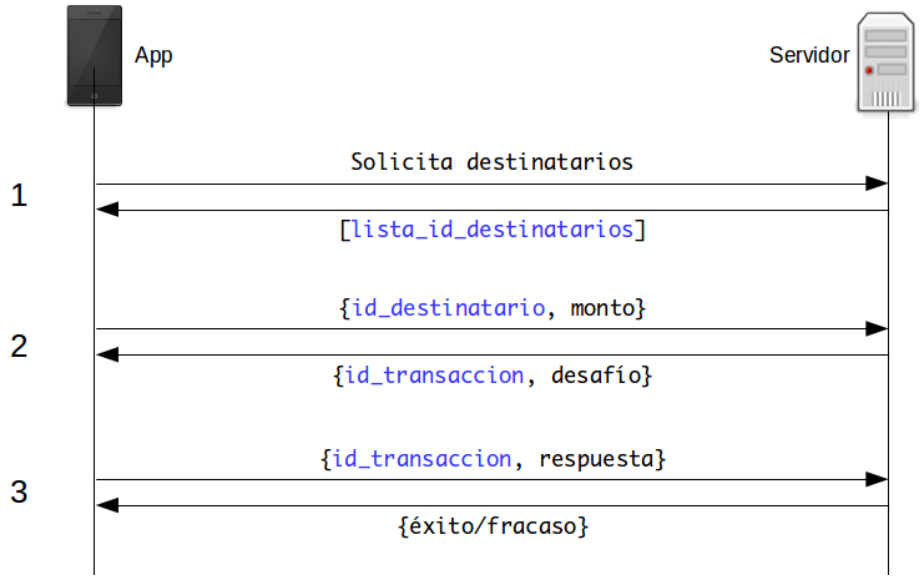


Figura 6: Mitigación ante potencial tampering de datos de transferencias

En este caso, en vez de dejar información validable en manos del cliente, la cual podría ser modificada maliciosamente, se deja esa información en el servidor y ésta se asocia con un ID de transacción, el cual es enviado a la app junto con el desafío. El usuario ingresa la respuesta, a la cual la app adosa este ID de transacción y la envía al servidor, el cual realiza los chequeos usuales (existencia de la cuenta de destinatario, el cliente tiene fondos suficientes, etc.), realiza la transacción, y responde al usuario si ésta fue exitosa o fallida.

Adicionalmente, antes de la validación, el servidor puede enviar en vez de una lista de destinatarios con mucha información, sólo enviar una lista con pares `{id_destinatario, nombre_destinatario}` en el paso 1, y cuando el usuario seleccione un destinatario, sólo enviar su id_destinatario en el paso 2.

Mal uso de cifrado en reposo

La aplicación del Banco 3 almacena en sus Shared Preferences (un archivo XML ubicado en su área de almacenamiento) el nombre del usuario cifrado con una biblioteca desarrollada específicamente para este propósito (`cl.banco3.crypto.Crypto`). Sin embargo, existe un problema en cómo realiza el proceso de cifrado. La biblioteca usa como llave de cifrado la siguiente combinación:

`md5(IMEI + '67510c5340276195')`

la cual consiste en la generación de un hash MD5 a partir de la concatenación del IMEI[16] y el string `'67510c5340276195'`, *hardcodeado*[17] en la aplicación (ver la función `getInnerCryptoKey()`). Ambos valores son simples de obtener: El IMEI es obtenible marcando un código especial en el aparato, y el string es obtenible haciendo ingeniería reversa de la aplicación. Se recomienda la generación de una llave de cifrado aleatoria y que ésta sea almacenada, no en el área de almacenamiento de la aplicación, sino que en un sistema llamado Android Keystore, el cual almacena claves en forma segura, cifrándolos con una llave basada en el método de bloqueo del sistema elegido por el usuario, sea éste un PIN, password o patrón con el cual se autentifica ante el teléfono. Para mayor información al respecto, ir a la sección "Recursos Interesantes".

Una recomendación importante es el uso de buenos algoritmos de cifrado. En la función `tripleDESCBCEncryption()` usada por `encryptUsername()` se observa lo que se utiliza el cifrador 3DES (Triple Data Encryption Standard) en modo CBC (Cipher Block Chaining): `Cipher.getInstance("DESede/CBC/NoPadding");`. La recomendación es usar el cifrador AES, más moderno y seguro[18].

Almacenamiento inseguro de información sensible

Durante las pruebas sobre la aplicación, se observó que información sensible fue almacenada en el archivo Shared Preferences (un archivo XML ubicado en el área de almacenamiento de la aplicación). La información sensible almacenada corresponde al RUT del usuario (reutilizado como login) y el nombre de éste.

Timeout mal implementado

De acuerdo con un estudio desarrollado por la Universidad de Berkeley y Google, el 62 % de los usuarios de teléfonos móviles no utiliza ningún método de bloqueo de sus equipos, sea éste el uso de un PIN, password o patrón[19]. Es por esto que la implementación de un timeout de cierre en una aplicación móvil es de extrema importancia, ya que un teléfono desbloqueado puede ser fuente, al menos, de filtración de información bancaria.

[16] El International Mobile Station Equipment Identity es un número que funciona como identificador único de un aparato de telefonía celular.

[17] *Hardcoding* es una práctica mediante la cual, datos de configuración son incrustados en el código fuente y posteriormente en los binarios de una aplicación.

[18] American Encryption Standard (Federal Information Processing Standard (FIPS) 197).

[19] https://www.eecs.berkeley.edu/~daw/papers/lock-ccs14.pdf

Si bien la aplicación del Banco 3 implementa un timeout, éste debe mejorar su implementación. Su funcionamiento consiste en que al finalizar el tiempo de la sesión, aparece una ventana alertando al usuario del cierre automático de ésta. Tal ventana no cubre toda la pantalla del aparato, y deja ver parte del contenido. En vez de eso, la aplicación debería volver a la pantalla de login automáticamente.

Problemas con la conexión HTTPS

Cadena de certificación mal hecha De acuerdo con el análisis realizado, se determinó que la cadena de certificación del servidor provee un certificado de más al momento de realizarse la conexión. El certificado es "VeriSign Class 3 Public Primary Certification Authority - G5"
(fingerprint `32f30882622b87cf8856c63db873df0853b4dd27`). Además de ser innecesario, es un certificado firmado con SHA1, lo cual lleva a deprecación a partir del 1 de enero de 2016[20] y los clientes Android podrían arrojar errores de validación de certificados posterior a esa fecha.

La recomendación es sólo entregar al cliente al momento de la negociación los siguientes certificados ya existentes en el servidor:

- www.banco3.cl (fingerprint anonimizado)
- Symantec Class 3 Secure Server CA - G4
  (fingerprint `ff67367c5cd4de4ae18bcce1d70fdabd7c866135`)

Uso del cifrador RC4 EL cifrador RC4 es vulnerable ante múltiples ataques criptoanalíticos y su uso está prohibido en configuraciones modernas[21]. La recomendación es eliminarlo de la lista de cifradores, lo cual se tratará en la sección "Mitigación".

Carencia de Forward Secrecy Forward Secrecy (secreto hacia el futuro) es una propiedad que causa que, si un atacante obtiene la llave secreta del servidor HTTPS, no podrá obtener las llaves de todas las sesiones que alguna vez se hayan generado con esa llave, y por lo tanto no podrá desencriptar el contenido de conversaciones que haya tenido el servidor con sus clientes.

El algoritmo DH (Diffie-Hellman) permite implementar Forward Secrecy. Una de las razones por las cuales los servidores no acostumbran implementarlo es por el hecho

[20] Qualys SSL Labs, "SHA1 Deprecation: What You Need to Know" https://community.qualys.com/blogs/securitylabs/2014/09/09/sha1-deprecation-what-you-need-to-know
[21] https://tools.ietf.org/html/rfc7465

de que consume más recursos computacionales que el algoritmo RSA, usado profusamente hoy en día. Sin embargo, el uso de Diffie-Hellman con Curva Elíptica reduce significativamente el consumo de recursos. Generalmente este algoritmo se denomina ECDHE[22].

Mitigación Las vulnerabilidades mencionadas en esta sección pueden ser corregidas cambiando la configuración SSL del servidor web al cual se conecta la aplicación. El analisis no arrojó resultados concluyentes respecto de qué servidor usa www.banco3.cl, por lo tanto no se pueden dar recomendaciones específicas para la configuración SSL del servidor. Sin embargo, existe una guía que puede ser útil para tal tarea, desarrollada por la Fundación Mozilla: https://wiki.mozilla.org/Security/Server_Side_TLS. Posterior a la configuración SSL del servidor, ésta se puede probar con las herramientas Cipherscan y Qualys SSL Test, indicadas en la sección "Recursos Interesantes".

Abuso de permisos

En el archivo `AndroidManifest.xml` se definen los permisos que la aplicación solicita al usuario para hacer uso de sus recursos al momento de instalar la aplicación. Algunos de estos permisos son riesgosos, y para el caso del Banco 3, son los siguientes:

Acceso a contactos del usuario La aplicación hace uso del permiso `android.permission.READ_CONTACTS`. Esto puede dar pie a abusos de privacidad en contra del usuario. Se recomienda remover este permiso de la aplicación.

Acceso al estado del teléfono El permiso `android.permission.READ_PHONE_STATE` permite a la aplicación acceder a información identificable del teléfono, como el IMEI. Posterior a la mitigación de los problemas criptográficos indicados en la sección "Mal uso de cifrado en reposo", se recomienda remover este permiso.

Uso de georreferenciación Una función ofrecida por las aplicaciones bancarias nacionales es la que permite al usuario obtener una lista o mapa de los cajeros Redbanc o sucursales del banco más cercanas al lugar en el que se encuentra. Para ello se requiere acceso a los sistemas de georreferenciación del teléfono, que son principalmente de dos tipos:

[22]Qualys SSL Labs, "Deploying Forward Secrecy": https://community.qualys.com/blogs/securitylabs/2013/06/25/ssl-labs-deploying-forward-secrecy

- Georreferenciación fina: Éste tipo de georreferenciación se basa en el uso de satélites GPS y se puede acceder a ello mediante el permiso `android.permission.ACCESS_FINE_LOCATION`. Su precisión es de entre 2 y 20 metros.
- Georreferenciación gruesa: Se basa en el uso de torres celulares y hotspots Wifi para entregar una ubicación aproximada del aparato. También se conoce como A-GPS (Assisted GPS) o Network Location Provider, y su precisión es de entre 100 y 200 metros. Se puede acceder a este tipo de georreferenciación con el permiso `android.permission.ACCESS_COARSE_LOCATION`[23].

Dado que la función de localización de cajeros y sucursales no requiere mayor precisión, la recomendación en este caso es quitar el permiso `ACCESS_FINE_LOCATION` y dejar el `ACCESS_COARSE_LOCATION`. Además del beneficio para la privacidad del usuario, la geolocalización gruesa consume menos energía de la batería del aparato y es mucho más rápida en la obtención de información geográfica de éste.

Recursos interesantes

- **NowSecure: 42+ Best Practices for Secure iOS and Android Development**. Este manual incluye una serie de consejos y buenas prácticas para desarrollar aplicaciones seguras, tanto en iOS como en Android: https://goo.gl/ivJD4M
- **Cipherscan**: Herramienta via linea de comandos que analiza la configuración de servidores HTTPS y entrega reportes. Puede ser usada para analizar servidores en estado de pre-producción: https://github.com/jvehent/cipherscan.
- **Qualys SSL Test**: Esta herramienta web gratuita fue utilizada para analizar la configuración actual del servidor en producción, y es utilizable ya para cuando el servidor con su nueva configuración ya se encuentre expuesto a Internet. Entrega información sobre configuración de certificados, protocolos, cifradores, etc., y le *pone nota* al servidor: https://www.ssllabs.com/ssltest/
- **Android Keystore**: Es un sistema de almacenamiento seguro de llaves o valores que se desea que sean secretos. Funciona como una tabla de datos encriptada mediante el PIN, password o patrón del usuario del equipo, por lo tanto no es fácil de obtener. Puede ver la documentación oficial de Android al respecto en http://developer.android.com/intl/es/training/articles/keystore.html y

[23]The Shine Blog, "A Good Look at Android Location Data": http://blog.shinetech.com/2011/10/14/a-good-look-at-android-location-data/

un ejemplo de su uso en http://www.androidauthority.com/use-android-keystore-store-passwords-sensitive-information-623779/

Informe Banco 4

Introducción

El presente informe consiste en un conjunto de recomendaciones de mitigación de vulnerabilidades de seguridad encontradas en la aplicación móvil para Android de Banco 4, y es parte de las tareas de investigación tendientes al trabajo de Tesis de Magister de quien escribe. El informe aquí presentado contiene lo siguiente:

1. Vulnerabilidades encontradas en su aplicación y riesgos asociados.
2. Formas de mitigación de tales vulnerabilidades.

El objetivo es que el equipo de desarrollo de la aplicación bancaria de Banco 4 pueda accionar las recomendaciones aquí indicadas en ésta.

Detalles de la aplicación analizada

- Tipo de aplicación: Web Embebida
- Tipo de conexión: HTTP-HTML

Vulnerabilidades encontradas y su mitigación

Potencial Tampering de datos

La aplicación, al momento de realizar una transferencia de dinero, pudiera ser vulnerable a que un usuario malicioso, o algún malware instalado en el teléfono modifique los datos de transacciones de transferencias de dinero entre cuentas, antes de que esa información sea enviada al servidor vía HTTPS. Éste ataque funcionaría de la forma que se muestra en la figura:

1. La aplicación solicita al servidor la lista de destinatarios registrados por el usuario en su cuenta, ante lo cual éste se la envía.

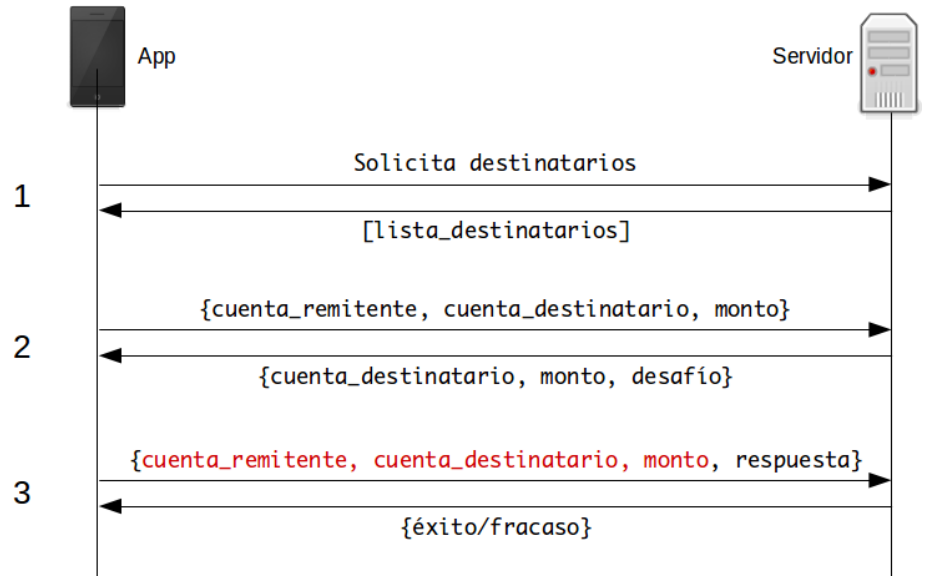


Figura 7: Modificación maliciosa de datos de transferencias

2. El usuario selecciona un destinatario y el monto a transferir en la app, la cual envía esa información al servidor junto con la cuenta remitente. El servidor responde enviándole a la app los datos de la cuenta destinario, el monto a transferir y un desafío, el cual en el caso de Banco 4 son 3 coordenadas de la *Tarjeta de clave On Line* del usuario.
3. El usuario ingresa la respuesta al desafío solicitado, y la aplicación envía ese dato, junto a la cuenta remitente, la cuenta de destinatario y el monto a transferir. Estos datos pueden ser falseados de tal forma que la cuenta remitente no sea la correcta, la cuenta destino no sea la correcta o el monto no sea el que el usuario deseó transferir. Si el servidor no valida esta información, hay riesgo serio de robo de dinero mediante transferencias a cuentas no deseadas.

Para mitigar este problema, se propone el uso de un ID de transacción, como se muestra en la siguiente figura:

En este caso, en vez de dejar información validable en manos del cliente, la cual podría ser modificada maliciosamente, se deja esa información en el servidor y ésta se asocia con un ID de transacción, el cual es enviado a la app junto con el desafío. El usuario ingresa la respuesta, a la cual la app adosa este ID de transacción y la envía al servidor, el cual realiza los chequeos usuales (existencia de la cuenta de destinatario, el cliente tiene fondos suficientes, etc.), realiza la transacción, y responde al usuario si ésta fue exitosa o fallida.

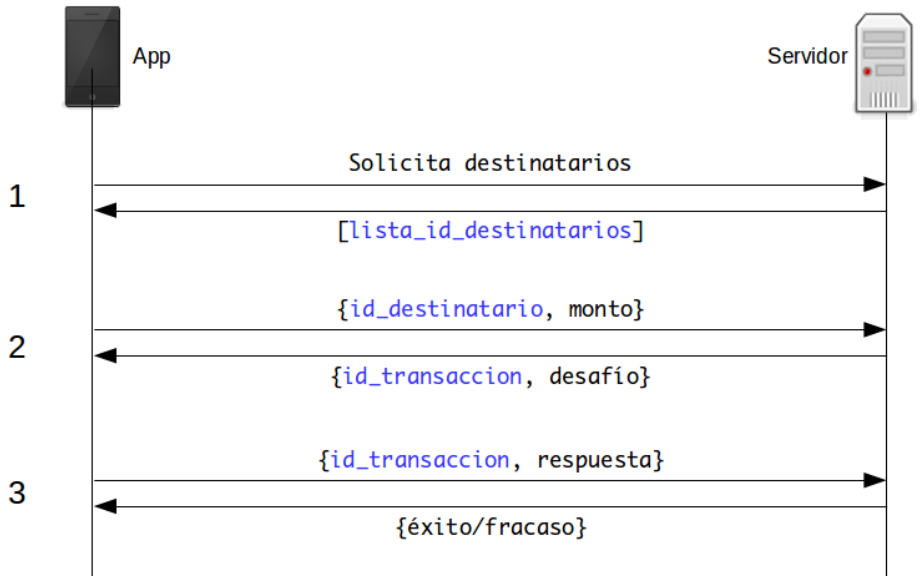


Figura 8: Mitigación ante potencial tampering de datos de transferencias

Adicionalmente, antes de la validación, el servidor puede enviar en vez de una lista de destinatarios con mucha información, sólo enviar una lista con pares `{id_destinatario, nombre_destinatario}` en el paso 1, y cuando el usuario seleccione un destinatario, sólo enviar su id_destinatario en el paso 2.

Almacenamiento de información sensible en cache

La aplicación Banco 4 es una aplicación tipo Web Embebida, vale decir, es un contenedor el cual usa el componente interno de browser del sistema operativo, y se conecta al sitio web móvil del banco. Cabe recordar que el browser tiene una función interna de caché de datos, y si bien la transmisión de datos se realiza en forma cifrada usando HTTPS, la caché no es cifrada, y al no serlo, información sensible puede quedar almacenada en el teléfono y accesible, por ejemplo, por otras aplicaciones en equipos rooteados[24].

Se recomienda que en el sitio web Banco 4 (el cual es usado en la aplicación) toda caché de datos sea deshabilitada. Ésto se logra mediante la aplicación de headers en la transmisión HTTP del servidor[25].

[24]OWASP, "Dangers of Jailbreaking and Rooting Mobile Devices": https://www.owasp.org/index.php/Projects/OWASP_Mobile_Security_Project_-_Dangers_of_Jailbreaking_and_Rooting_Mobile_Devices

[25]Stack Overflow, "Making sure a web page is not cached, across all browsers": http://stackoverflow.com/questions/49547/making-sure-a-web-page-is-not-cached-across-all-browsers

Almacenamiento de información sensible en base de datos SQLite

Como parte de su funcionamiento, muchas aplicaciones almacenan información en bases de datos SQLite, sin cifrado alguno. En el caso de la aplicación Banco 4, almacena las cookies de sesión. Se recomienda almacenar tal información en forma cifrada. Para ello se debe generar una llave de cifrado aleatoria y que ésta sea almacenada, no en el área de almacenamiento de la aplicación, sino que en un sistema llamado Android Keystore, el cual almacena claves en forma segura, cifrándolos con una llave basada en el método de bloqueo del sistema elegido por el usuario, sea éste un PIN, password o patrón con el cual se autentifica ante el teléfono. Con esta llave se puede encriptar la información antes de guardarla, por ejemplo en la base de datos SQLite o en Shared Preferences, y luego usar esta misma llave para descifrar la información. Para mayor información respecto de Android Keystore, ir a la sección "Recursos Interesantes".

Problemas con la conexión HTTPS

Uso del protocolo SSL3 El protocolo SSL en su tercera versión (SSL3) ya está a punto de cumplir 20 años en funcionamiento y ya ha sido deprecado por ser inseguro[26]. Se recomienda su remoción de la lista de protocolos, y sólo dejar TLSv1.0, TLSv1.1 y TLSv1.2.

No uso del protocolo TLS 1.2 El protocolo TLS en su versión 1.2 (TLS1.2) es el más avanzado existente en la actualidad. A pesar de que su adopción por parte de clientes ha sido lenta, es la opción más segura. Se recomienda encarecidamente su activación[27].

Uso del cifrador RC4 EL cifrador RC4 es vulnerable ante múltiples ataques criptoanalíticos y su uso está prohibido en configuraciones modernas[28]. La recomendación es eliminarlo de la lista de cifradores, lo cual se tratará en la sección "Mitigación".

Carencia de Forward Secrecy Forward Secrecy (secreto hacia el futuro) es una propiedad que causa que, si un atacante obtiene la llave secreta del servidor HTTPS, no podrá obtener las llaves de todas las sesiones que alguna vez se hayan generado

[26]https://tools.ietf.org/html/rfc7568

[27]Qualys SSL Labs, "Increased Penalty When TLS 1.2 Is Not Supported": https://community.qualys.com/blogs/securitylabs/2015/05/22/ssl-labs-increased-penalty-when-tls-12-is-not-supported

[28]https://tools.ietf.org/html/rfc7465

con esa llave, y por lo tanto no podrá desencriptar el contenido de conversaciones que haya tenido el servidor con sus clientes.

El algoritmo DH (Diffie-Hellman) permite implementar Forward Secrecy. Una de las razones por las cuales los servidores no acostumbran implementarlo es por el hecho de que consume más recursos computacionales que el algoritmo RSA, usado profusamente hoy en día. Sin embargo, el uso de Diffie-Hellman con Curva Elíptica reduce significativamente el consumo de recursos. Generalmente este algoritmo se denomina ECDHE[29].

Mitigación Las vulnerabilidades mencionadas en esta sección pueden ser corregidas cambiando la configuración SSL del servidor web al cual se conecta la aplicación. La Fundación Mozilla tiene disponible una guía que puede ser útil para ésta tarea: https://wiki.mozilla.org/Security/Server_Side_TLS.

Adicionalmente, se determinó que el servidor de la aplicación es un Microsoft Internet Information Services (IIS) versión 7.5. Para configurar éste servidor en forma fácil, existe una herramienta gratuita disponible, llamada IISCrypto, disponible en https://www.nartac.com/Products/IISCrypto.

Posterior a la configuración SSL del servidor, ésta se puede probar con las herramientas Cipherscan y Qualys SSL Test, indicadas en la sección "Recursos Interesantes".

Uso de georreferenciación

Una función ofrecida por las aplicaciones bancarias nacionales es la que permite al usuario obtener una lista o mapa de los cajeros Redbanc o sucursales del banco más cercanas al lugar en el que se encuentra. Para ello se requiere acceso a los sistemas de georreferenciación del teléfono, que son principalmente de dos tipos:

- Georreferenciación fina: Éste tipo de georreferenciación se basa en el uso de satélites GPS y se puede acceder a ello mediante el permiso `android.permission.ACCESS_FINE_LOCATION`. Su precisión es de entre 2 y 20 metros.
- Georreferenciación gruesa: Se basa en el uso de torres celulares y hotspots Wifi para entregar una ubicación aproximada del aparato. También se conoce como A-GPS (Assisted GPS) o Network Location Provider, y su precisión es de entre

[29]Qualys SSL Labs, "Deploying Forward Secrecy": https://community.qualys.com/blogs/securitylabs/2013/06/25/ssl-labs-deploying-forward-secrecy

100 y 200 metros. Se puede acceder a este tipo de georreferenciación con el permiso `android.permission.ACCESS_COARSE_LOCATION`[30].

Dado que la función de localización de cajeros y sucursales no requiere mayor precisión, la recomendación en este caso es quitar el permiso `ACCESS_FINE_LOCATION` y dejar el `ACCESS_COARSE_LOCATION`. Además del beneficio para la privacidad del usuario, la geolocalización gruesa consume menos energía de la batería del aparato y es mucho más rápida en la obtención de información geográfica de éste.

Recursos interesantes

- **NowSecure: 42+ Best Practices for Secure iOS and Android Development**. Este manual incluye una serie de consejos y buenas prácticas para desarrollar aplicaciones seguras, tanto en iOS como en Android: https://goo.gl/ivJD4M
- **Cipherscan**: Herramienta via linea de comandos que analiza la configuración de servidores HTTPS y entrega reportes. Puede ser usada para analizar servidores en estado de pre-producción: https://github.com/jvehent/cipherscan.
- **Qualys SSL Test**: Esta herramienta web gratuita fue utilizada para analizar la configuración actual del servidor en producción, y es utilizable ya para cuando el servidor con su nueva configuración ya se encuentre expuesto a Internet. Entrega información sobre configuración de certificados, protocolos, cifradores, etc., y le *pone nota* al servidor: https://www.ssllabs.com/ssltest/
- **Android Keystore**: Es un sistema de almacenamiento seguro de llaves o valores que se desea que sean secretos. Funciona como una tabla de datos encriptada mediante el PIN, password o patrón del usuario del equipo, por lo tanto no es fácil de obtener. Puede ver la documentación oficial de Android al respecto en http://developer.android.com/intl/es/training/articles/keystore.html y un ejemplo de su uso en http://www.androidauthority.com/use-android-keystore-store-passwords-sensitive-information-623779/

[30]The Shine Blog, "A Good Look at Android Location Data": http://blog.shinetech.com/2011/10/14/a-good-look-at-android-location-data/

Informe Banco 5

Introducción

El presente informe consiste en un conjunto de recomendaciones de mitigación de vulnerabilidades de seguridad encontradas en la aplicación móvil para Android del Banco 5, y es parte de las tareas de investigación tendientes al trabajo de Tesis de Magister de quien escribe. El informe aquí presentado contiene lo siguiente:

1. Vulnerabilidades encontradas en su aplicación y riesgos asociados.
2. Formas de mitigación de tales vulnerabilidades.

El objetivo es que el equipo de desarrollo de la aplicación bancaria del Banco 5 pueda accionar las recomendaciones aquí indicadas en ésta.

Detalles de la aplicación analizada

- Tipo de aplicación: Web Embebida
- Tipo de conexión: HTTP-HTML

Vulnerabilidades encontradas y su mitigación

Potencial Tampering de datos

La aplicación, al momento de realizar una transferencia de dinero, pudiera ser vulnerable a que un usuario malicioso, o algún malware instalado en el teléfono modifique los datos de transacciones de transferencias de dinero entre cuentas, antes de que esa información sea enviada al servidor vía HTTPS. Éste ataque funcionaría de la forma que se muestra en la figura:

1. La aplicación solicita al servidor la lista de destinatarios registrados por el usuario en su cuenta, ante lo cual éste se la envía.
2. El usuario selecciona un destinatario y el monto a transferir en la app, la cual envía esa información al servidor junto con la cuenta remitente. El servidor responde enviándole a la app los datos de la cuenta destinario, el monto a transferir y un desafío, el cual en el caso del Banco 5 es la solicitud de un número generado por un token.

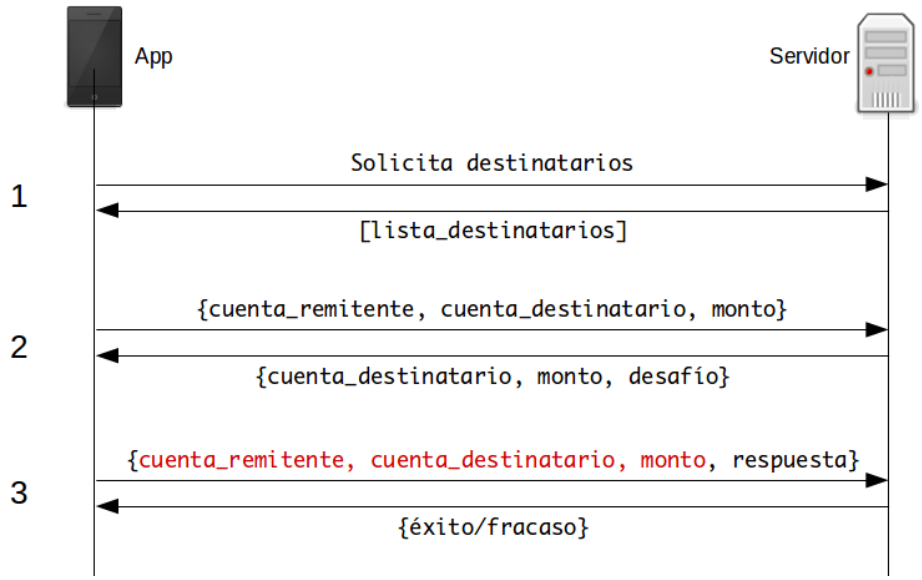


Figura 9: Modificación maliciosa de datos de transferencias

3. El usuario ingresa la respuesta al desafío solicitado, y la aplicación envía ese dato, junto a la cuenta remitente, la cuenta de destinatario y el monto a transferir. Estos datos pueden ser falseados de tal forma que la cuenta remitente no sea la correcta, la cuenta destino no sea la correcta o el monto no sea el que el usuario deseó transferir. Si el servidor no valida esta información, hay riesgo serio de robo de dinero mediante transferencias a cuentas no deseadas.

Para mitigar este problema, se propone el uso de un ID de transacción, como se muestra en la siguiente figura:

En este caso, en vez de dejar información validable en manos del cliente, la cual podría ser modificada maliciosamente, se deja esa información en el servidor y ésta se asocia con un ID de transacción, el cual es enviado a la app junto con el desafío. El usuario ingresa la respuesta, a la cual la app adosa este ID de transacción y la envía al servidor, el cual realiza los chequeos usuales (existencia de la cuenta de destinatario, el cliente tiene fondos suficientes, etc.), realiza la transacción, y responde al usuario si ésta fue exitosa o fallida.

Adicionalmente, antes de la validación, el servidor puede enviar en vez de una lista de destinatarios con mucha información, sólo enviar una lista con pares `{id_destinatario, nombre_destinatario}` en el paso 1, y cuando el usuario seleccione un destinatario, sólo enviar su id_destinatario en el paso 2.

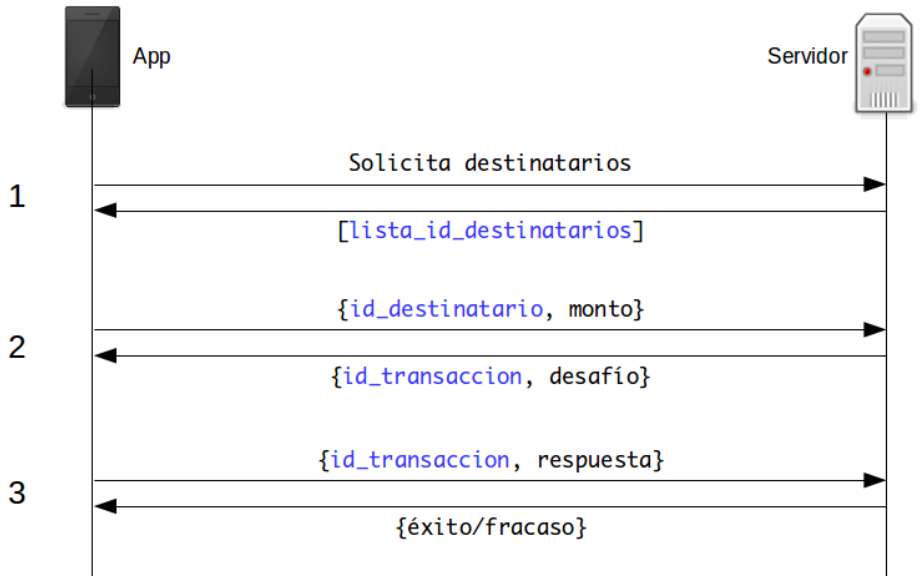


Figura 10: Mitigación ante potencial tampering de datos de transferencias

Adicionalmente, se observó que, para otras tareas distintas a las de transferencia, la aplicación envía demasiada información al servidor en la URL, como por ejemplo el PAN[31] de Tarjeta de Crédito. Se recomienda revisar lo enviado por la aplicación al servidor, y si no es información que se requiera por parte del usuario, asociarla con la sesión.

Cadena de certificación HTTPS mal hecha

De acuerdo con el análisis realizado, se determinó que la cadena de certificación del servidor provee un certificado de más al momento de realizarse la conexión. El certificado es el Entrust Root Certification Authority (fingerprint `b31eb1b740e36c8402dadc37d44df5d4674952f9`), que es el certificado raíz de la autoridad certificadora, el cual es redundante ya que los clientes (browsers, smartphones, etc.) ya lo incluyen como parte de su instalación por defecto. La recomendación es eliminar ese certificado y dejar los demás.

Timeout mal implementado

De acuerdo con un estudio desarrollado por la Universidad de Berkeley y Google, el 62 % de los usuarios de teléfonos móviles no utiliza ningún método de bloqueo de sus

[31] El PAN es el número de tarjeta de crédito (Primary Account Number).

equipos, sea éste el uso de un PIN, password o patrón[32]. Es por esto que la implementación de un timeout de cierre en una aplicación móvil es de extrema importancia, ya que un teléfono desbloqueado puede ser fuente, al menos, de filtración de información bancaria.

Si bien la aplicación Banco 5 implementa un timeout, éste debe mejorar su implementación. Si ésta se deja sin funcionar unos minutos, y luego se intenta operar sobre ésta, ésta vuelve a la página de login, sin embargo eso no evita filtración de información, ya que si, por ejemplo, el usuario deja de usar la aplicación con ésta mostrando el saldo de su cuenta corriente, esa información seguirá estando ahí hasta que intente realizar otra acción, lo que da pie a filtración de información.

Actualización insegura de información (mediante HTTP)

Al abrir la aplicación, ésta se conecta servidor de la aplicación y actualiza información sensible como la URL a la cual ésta debe conectarse, esto mediante una llamada a `http://www.banco5.cl/mobileapp/getUrls`, mediante una conexión HTTP sin cifrar, lo cual es muy peligroso ya que esa conexión podría ser interceptada y sus contenidos modificados para que la aplicación se conecte, por ejemplo, a un servidor de phishing[33]. Se recomienda que tal actualización se realice mediante HTTPS.

Abuso de permisos

En el archivo `AndroidManifest.xml` se definen los permisos que la aplicación solicita al usuario para hacer uso de sus recursos al momento de instalar la aplicación. Algunos de estos permisos son riesgosos, y para el caso del Banco 5, son los siguientes:

Uso de georreferenciación Una función ofrecida por las aplicaciones bancarias nacionales es la que permite al usuario obtener una lista o mapa de los cajeros Redbanc o sucursales del banco más cercanas al lugar en el que se encuentra. Para ello se requiere acceso a los sistemas de georreferenciación del teléfono, que son principalmente de dos tipos:

[32]https://www.eecs.berkeley.edu/~daw/papers/lock-ccs14.pdf

[33]Una muestra de los riesgos de actualizar información sensible por canales no cifrados se puede ver en el artículo de SOPHOS Naked Security, "Samsung keyboard app could let a crook crack your phone": https://nakedsecurity.sophos.com/2015/06/17/samsung-keyboard-app-could-let-a-crook-crack-your-phone/

- Georreferenciación fina: Éste tipo de georreferenciación se basa en el uso de satélites GPS y se puede acceder a ello mediante el permiso `android.permission.ACCESS_FINE_LOCATION`. Su precisión es de entre 2 y 20 metros.
- Georreferenciación gruesa: Se basa en el uso de torres celulares y hotspots Wifi para entregar una ubicación aproximada del aparato. También se conoce como A-GPS (Assisted GPS) o Network Location Provider, y su precisión es de entre 100 y 200 metros. Se puede acceder a este tipo de georreferenciación con el permiso `android.permission.ACCESS_COARSE_LOCATION`[34].

Dado que la función de localización de cajeros y sucursales no requiere mayor precisión, la recomendación en este caso es quitar el permiso `ACCESS_FINE_LOCATION` y dejar el `ACCESS_COARSE_LOCATION`. Además del beneficio para la privacidad del usuario, la geolocalización gruesa consume menos energía de la batería del aparato y es mucho más rápida en la obtención de información geográfica de éste.

Acceso a contactos del usuario La aplicación hace uso de los permisos `android.permission.READ_CONTACTS` y `android.permission.WRITE_CONTACTS`. Esto puede dar pie a abusos de privacidad en contra del usuario. Se recomienda remover estos permisos de la aplicación.

Escritura en el almacenamiento externo del teléfono El teléfono incluye un área de almacenamiento externo, el cual no tiene las protecciones incluidas en el área de almacenamiento correspondiente a la aplicación, y por lo tanto, cualquier archivo almacenado en esta área externa podría ser accesible por otras aplicaciones y a través de conexiones USB al teléfono. Se recomienda remover el permiso `android.permission.WRITE_EXTERNAL_STORAGE`.

Recursos interesantes

- **NowSecure: 42+ Best Practices for Secure iOS and Android Development**. Este manual incluye una serie de consejos y buenas prácticas para desarrollar aplicaciones seguras, tanto en iOS como en Android: https://goo.gl/ivJD4M

[34]The Shine Blog, "A Good Look at Android Location Data": http://blog.shinetech.com/2011/10/14/a-good-look-at-android-location-data/

Informe Banco 6

Introducción

El presente informe consiste en un conjunto de recomendaciones de mitigación de vulnerabilidades de seguridad encontradas en la aplicación móvil para Android de Banco 6, y es parte de las tareas de investigación tendientes al trabajo de Tesis de Magister de quien escribe. El informe aquí presentado contiene lo siguiente:

1. Vulnerabilidades encontradas en su aplicación y riesgos asociados.
2. Formas de mitigación de tales vulnerabilidades.

El objetivo es que el equipo de desarrollo de la aplicación bancaria de Banco 6 pueda accionar las recomendaciones aquí indicadas en ésta.

Detalles de la aplicación analizada

- Tipo de aplicación: Híbrida (Titanium Appcelerator 0.8.6)
- Tipo de conexión: REST-JSON

Vulnerabilidades encontradas y su mitigación

Potencial Tampering de datos

La aplicación, al momento de realizar una transferencia de dinero, pudiera ser vulnerable a que un usuario malicioso, o algún malware instalado en el teléfono modifique los datos de transacciones de transferencias de dinero entre cuentas, antes de que esa información sea enviada al servidor vía HTTPS. Éste ataque funcionaría de la forma que se muestra en la figura:

1. La aplicación solicita al servidor la lista de destinatarios registrados por el usuario en su cuenta, ante lo cual éste se la envía.
2. El usuario selecciona un destinatario y el monto a transferir en la app, la cual envía esa información al servidor junto con la cuenta remitente. El servidor responde enviándole a la app los datos de la cuenta destinario, el monto a transferir y un desafío, el cual en el caso del Banco 6 es la solicitud de un número generado por un token o enviado vía SMS al usuario.

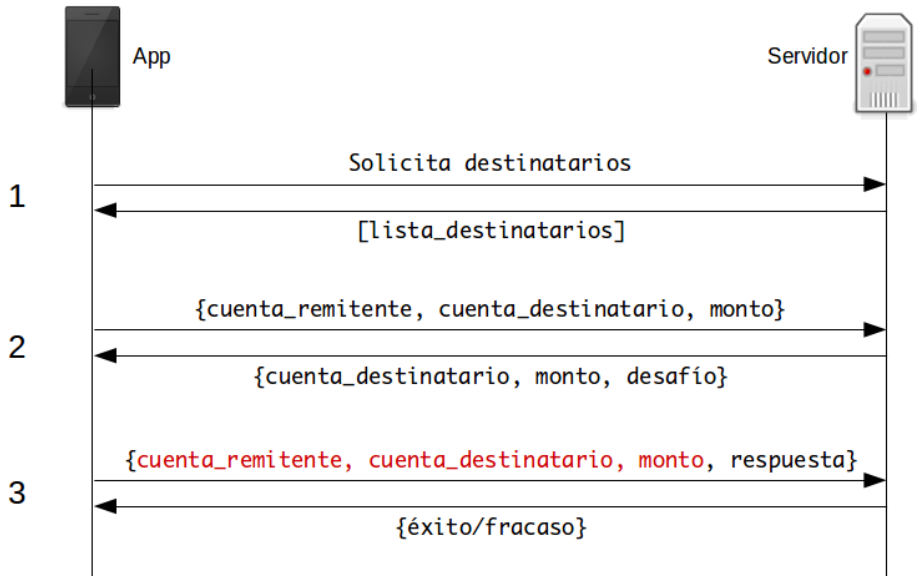


Figura 11: Modificación maliciosa de datos de transferencias

3. El usuario ingresa la respuesta al desafío solicitado, y la aplicación envía ese dato, junto a la cuenta remitente, la cuenta de destinatario y el monto a transferir. Estos datos pueden ser falseados de tal forma que la cuenta remitente no sea la correcta, la cuenta destino no sea la correcta o el monto no sea el que el usuario deseó transferir. Si el servidor no valida esta información, hay riesgo serio de robo de dinero mediante transferencias a cuentas no deseadas.

Para mitigar este problema, se propone el uso de un ID de transacción, como se muestra en la siguiente figura:

En este caso, en vez de dejar información validable en manos del cliente, la cual podría ser modificada maliciosamente, se deja esa información en el servidor y ésta se asocia con un ID de transacción, el cual es enviado a la app junto con el desafío. El usuario ingresa la respuesta, a la cual la app adosa este ID de transacción y la envía al servidor, el cual realiza los chequeos usuales (existencia de la cuenta de destinatario, el cliente tiene fondos suficientes, etc.), realiza la transacción, y responde al usuario si ésta fue exitosa o fallida.

Adicionalmente, antes de la validación, el servidor puede enviar en vez de una lista de destinatarios con mucha información, sólo enviar una lista con pares `{id_destinatario, nombre_destinatario}` en el paso 1, y cuando el usuario seleccione un destinatario, sólo enviar su id_destinatario en el paso 2.

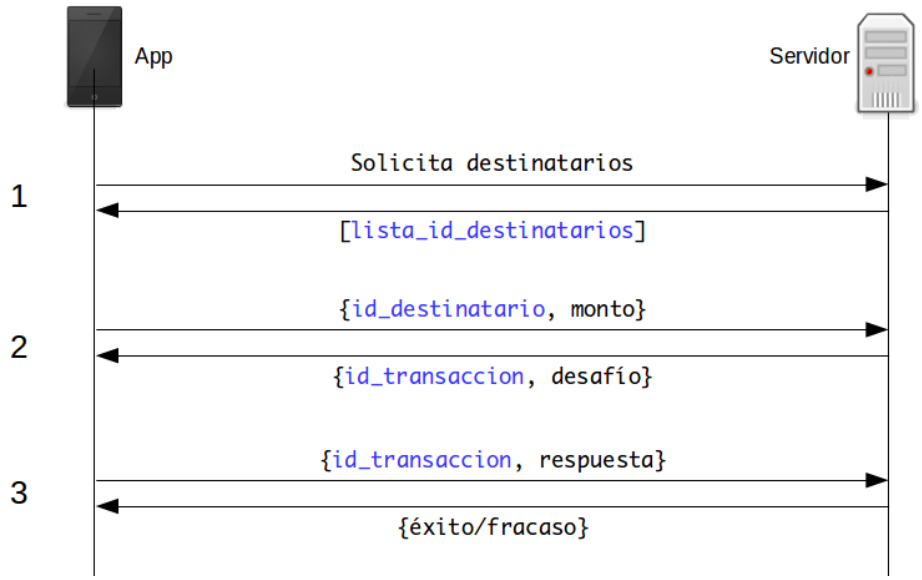


Figura 12: Mitigación ante potencial tampering de datos de transferencias

Problemas con la conexión HTTPS

Cadena de certificación mal hecha De acuerdo con el análisis realizado, se determinó que la cadena de certificación del servidor provee dos certificados de más al momento de realizarse la conexión. Estos certificados son los siguientes:

- Globalsign (fingerprint `9563f9a74ea68df514c9053d6af8cee72bd6cb88`)
- GlobalSign Root CA (fingerprint `b1bc968bd4f49d622aa89a81f2150152a41d829c`)

El primero es un certificado firmado con SHA1, lo cual lleva a deprecación a partir del 1 de enero de 2016[35] y los clientes Android podrían arrojar errores de validación de certificados posterior a esa fecha. El segundo es el certificado raíz de la autoridad certificadora, el cual es redundante ya que los clientes (browsers, smartphones, etc.) ya lo incluyen como parte de su instalación por defecto.

La recomendación es sólo entregar al cliente al momento de la negociación los siguientes certificados ya existentes en el servidor:

- mov.banco6.cl (fingerprint anonimizado)
- GlobalSign Extended Validation CA - SHA256 - G2 (fingerprint `65be102be26928650e0ef54dc8f4f15af5f98e8b`)

[35]Qualys SSL Labs, "SHA1 Deprecation: What You Need to Know" https://community.qualys.com/blogs/securitylabs/2014/09/09/sha1-deprecation-what-you-need-to-know

Uso del cifrador RC4 EL cifrador RC4 es vulnerable ante múltiples ataques criptoanalíticos y su uso está prohibido en configuraciones modernas[36]. La recomendación es eliminarlo de la lista de cifradores, lo cual se tratará en la sección "Mitigación".

Carencia de Forward Secrecy Forward Secrecy (secreto hacia el futuro) es una propiedad que causa que, si un atacante obtiene la llave secreta del servidor HTTPS, no podrá obtener las llaves de todas las sesiones que alguna vez se hayan generado con esa llave, y por lo tanto no podrá desencriptar el contenido de conversaciones que haya tenido el servidor con sus clientes.

El algoritmo DH (Diffie-Hellman) permite implementar Forward Secrecy. Una de las razones por las cuales los servidores no acostumbran implementarlo es por el hecho de que consume más recursos computacionales que el algoritmo RSA, usado profusamente hoy en día. Sin embargo, el uso de Diffie-Hellman con Curva Elíptica reduce significativamente el consumo de recursos. Generalmente este algoritmo se denomina ECDHE[37].

Mitigación Las vulnerabilidades mencionadas en esta sección pueden ser corregidas cambiando la configuración SSL del servidor web al cual se conecta la aplicación. De acuerdo con el análisis, el servidor HTTP utilizado por la aplicación es un Oracle HTTP Server (OHS), el cual al ser basado en Apache, permite que los mismos parámetros de configuración puedan ser utilizados. La documentación sobre SSL para el OHS está en https://docs.oracle.com/cd/B14099_19/web,1012/b14009/ssl.htm.

Adicionalmente, la Fundación Mozilla dispuso en línea una herramienta para generar automáticamente configuraciones SSL para servidores Apache, la cual se puede aplicar para OHS. Se puede encontrar aquí: https://mozilla.github.io/server-side-tls/ssl-config-generator/. Posterior a su configuración, ésta se puede probar con las herramientas Cipherscan y Qualys SSL Test, indicadas en la sección "Recursos Interesantes".

Exceso de confianza en medidas anti-ingeniería reversa

La aplicación usa un sistema que busca evitar que un analista (o algún atacante) obtenga su código fuente mediante ingeniería reversa. Cabe indicar aquí que, aparte de un

[36]https://tools.ietf.org/html/rfc7465

[37]Qualys SSL Labs, "Deploying Forward Secrecy": https://community.qualys.com/blogs/securitylabs/2013/06/25/ssl-labs-deploying-forward-secrecy

trabajo de investigación basado en información ya publicada, no es demasiado dificil obtener el código fuente de la aplicación, incluso de manera automatizada[38].

La recomendación es basarse en el Principio de Diseño Abierto[39]: No confiar la seguridad de una aplicación en secretos supuestamente imposibles de obtener. Siempre habrá alguna forma de obtener información acerca de la aplicación, y de utilizarla con malos fines.

Mal uso de cifrado redundante

Un aspecto descubierto después de la extracción del código indicado en el punto anterior es el hecho de que la aplicación encripta la password del usuario con una llave generada de la siguiente forma:

```
base64(md5(Ti.Platform.id + Ti.App.id))
```

Ésta forma de encriptar información acarrea los siguientes problemas:

1. El hecho de que se use la concatenación entre `Ti.Platform.id` y `Ti.App.id` como clave es un problema. `Ti.Platform.id` es el identificador conocido como ANDROID_ID, el cual es generado la primera vez que el teléfono se activa y se conecta a Google para registrarse, y es muy fácil de obtener. Lo mismo pasa con `Ti.App.id`, que es el identificador de la aplicación (en el caso de la app del Banco 6, es `cl.banco6.banco6movil`)
2. Además, se utilizó el cifrador AES con modo de operación ECB (AES-ECB). Se recomienda el uso del sistema AES-CBC o similar.

Android, a partir de su versión 4.0, implementa lo que se conoce como Android Keystore, componente el cual permite guardar llaves encriptadas en una base de datos encriptada con el pin, password o patrón que usa el usuario para bloquear el teléfono. Lamentablemente, Titanium Appcelerator no tiene biblioteca alguna que permita aprovechar esta herramienta y almacenar en ella claves generadas aleatoriamente. Sin embargo, de acuerdo con lo observado, la aplicación Banco 6 encripta sólo la password del usuario para luego enviarla al servidor al momento de conectarse con la aplicación,

[38]Ver el trabajo de Marco Grassi y Sebastián Guerrero en la conferencia de seguridad BlackHat Asia 2015. Lo indicado entre las slides 28 y 32 es particularmente interesante para el framework Titanium Appcelerator, usado en algunas de las aplicaciones analizadas: https://www.nowsecure.com/blog/2015/04/13/the-nightmare-behind-the-cross-platform-apps-dream/

[39]https://www.owasp.org/index.php/Avoid_security_by_obscurity

lo cual es redundante, ya que la aplicación ya encripta esa información usando HTTPS. Este tipo de operaciones es recomendable sólo cuando se almacena información sensible en el teléfono, lo cual no ocurre en la aplicación Banco 6.

Abuso de permisos

En el archivo `AndroidManifest.xml` se definen los permisos que la aplicación solicita al usuario para hacer uso de sus recursos al momento de instalar la aplicación. Algunos de estos permisos son riesgosos, y para el caso del Banco 6, son los siguientes:

Acceso a información acerca de otras aplicaciones en uso El permiso `android.permission.GET_TASKS` permite a la aplicación acceder a información de otras aplicaciones que estén ejecutándose en el equipo. Por problemas de privacidad fue deprecado en la API versión 21 (Lollipop). Se recomienda su eliminación de la lista de permisos.

Escritura en el almacenamiento externo del teléfono El teléfono incluye un área de almacenamiento externo, el cual no tiene las protecciones incluidas en el área de almacenamiento correspondiente a la aplicación, y por lo tanto, cualquier archivo almacenado en esta área externa podría ser accesible por otras aplicaciones y a través de conexiones USB al teléfono. Se recomienda remover el permiso `android.permission.WRITE_EXTERNAL_STORAGE`.

Uso de georreferenciación Una función ofrecida por las aplicaciones bancarias nacionales es la que permite al usuario obtener una lista o mapa de los cajeros Redbanc o sucursales del banco más cercanas al lugar en el que se encuentra. Para ello se requiere acceso a los sistemas de georreferenciación del teléfono, que son principalmente de dos tipos:

- Georreferenciación fina: Éste tipo de georreferenciación se basa en el uso de satélites GPS y se puede acceder a ello mediante el permiso `android.permission.ACCESS_FINE_LOCATION`. Su precisión es de entre 2 y 20 metros.
- Georreferenciación gruesa: Se basa en el uso de torres celulares y hotspots Wifi para entregar una ubicación aproximada del aparato. También se conoce como A-GPS (Assisted GPS) o Network Location Provider, y su precisión es de entre

100 y 200 metros. Se puede acceder a este tipo de georreferenciación con el permiso `android.permission.ACCESS_COARSE_LOCATION`[40].

Dado que la función de localización de cajeros y sucursales no requiere mayor precisión, la recomendación en este caso es quitar el permiso `ACCESS_FINE_LOCATION` y dejar el `ACCESS_COARSE_LOCATION`. Además del beneficio para la privacidad del usuario, la geolocalización gruesa consume menos energía de la batería del aparato y es mucho más rápida en la obtención de información geográfica de éste.

Acceso a contactos del usuario La aplicación hace uso de los permisos `android.permission.READ_CONTACTS` y `android.permission.WRITE_CONTACTS`. Esto puede dar pie a abusos de privacidad en contra del usuario. Se recomienda remover estos permisos de la aplicación.

Recursos interesantes

- **NowSecure: 42+ Best Practices for Secure iOS and Android Development**. Este manual incluye una serie de consejos y buenas prácticas para desarrollar aplicaciones seguras, tanto en iOS como en Android: https://goo.gl/ivJD4M
- **Cipherscan**: Herramienta via linea de comandos que analiza la configuración de servidores HTTPS y entrega reportes. Puede ser usada para analizar servidores en estado de pre-producción: https://github.com/jvehent/cipherscan.
- **Qualys SSL Test**: Esta herramienta web gratuita fue utilizada para analizar la configuración actual del servidor en producción, y es utilizable ya para cuando el servidor con su nueva configuración ya se encuentre expuesto a Internet. Entrega información sobre configuración de certificados, protocolos, cifradores, etc., y le *pone nota* al servidor: https://www.ssllabs.com/ssltest/

Informe Banco 7

Introducción

El presente informe consiste en un conjunto de recomendaciones de mitigación de vulnerabilidades de seguridad encontradas en la aplicación móvil para Android del Banco

[40]The Shine Blog, "A Good Look at Android Location Data": http://blog.shinetech.com/2011/10/14/a-good-look-at-android-location-data/

7, y es parte de las tareas de investigación tendientes al trabajo de Tesis de Magister de quien escribe. El informe aquí presentado contiene lo siguiente:

1. Vulnerabilidades encontradas en su aplicación y riesgos asociados.
2. Formas de mitigación de tales vulnerabilidades.

El objetivo es que el equipo de desarrollo de la aplicación bancaria del Banco 7 pueda accionar las recomendaciones aquí indicadas en ésta.

Detalles de la aplicación analizada

- Tipo de aplicación: Web Embebida
- Tipo de conexión: HTTP-HTML

Vulnerabilidades encontradas y su mitigación

Potencial Tampering de datos

La aplicación, al momento de realizar una transferencia de dinero, pudiera ser vulnerable a que un usuario malicioso, o algún malware instalado en el teléfono modifique los datos de transacciones de transferencias de dinero entre cuentas, antes de que esa información sea enviada al servidor vía HTTPS. Éste ataque funcionaría de la forma que se muestra en la figura:

1. La aplicación solicita al servidor la lista de destinatarios registrados por el usuario en su cuenta, ante lo cual éste se la envía.
2. El usuario selecciona un destinatario y el monto a transferir en la app, la cual envía esa información al servidor junto con la cuenta remitente. El servidor responde enviándole a la app los datos de la cuenta destinario, el monto a transferir y un desafío, el cual en el caso del Banco 7 puede ser las coordenadas de una tarjeta, el número de un token o un código enviado vía telefónica.
3. El usuario ingresa la respuesta al desafío solicitado, y la aplicación envía ese dato, junto a la cuenta remitente, la cuenta de destinatario y el monto a transferir. Estos datos pueden ser falseados de tal forma que la cuenta remitente no sea la correcta, la cuenta destino no sea la correcta o el monto no sea el que el usuario deseó transferir. Si el servidor no valida esta información, hay riesgo serio de robo de dinero mediante transferencias a cuentas no deseadas.

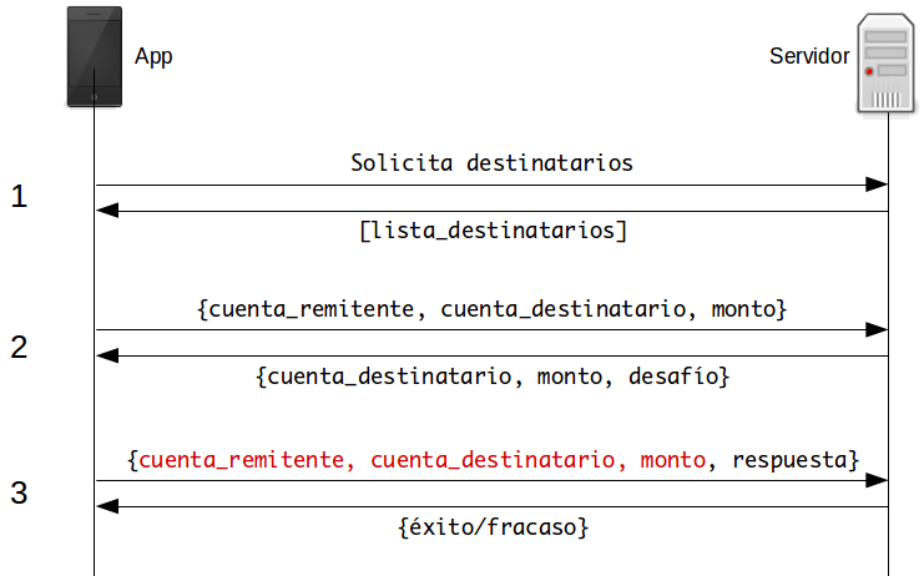


Figura 13: Modificación maliciosa de datos de transferencias

Para mitigar este problema, se propone el uso de un ID de transacción, como se muestra en la siguiente figura:

En este caso, en vez de dejar información validable en manos del cliente, la cual podría ser modificada maliciosamente, se deja esa información en el servidor y ésta se asocia con un ID de transacción, el cual es enviado a la app junto con el desafío. El usuario ingresa la respuesta, a la cual la app adosa este ID de transacción y la envía al servidor, el cual realiza los chequeos usuales (existencia de la cuenta de destinatario, el cliente tiene fondos suficientes, etc.), realiza la transacción, y responde al usuario si ésta fue exitosa o fallida.

Adicionalmente, antes de la validación, el servidor puede enviar en vez de una lista de destinatarios con mucha información, sólo enviar una lista con pares `{id_destinatario, nombre_destinatario}` en el paso 1, y cuando el usuario seleccione un destinatario, sólo enviar su id_destinatario en el paso 2.

Almacenamiento de información sensible en cache

La aplicación del Banco 7 es una aplicación tipo Web Embebida, vale decir, es un contenedor el cual usa el componente interno de browser del sistema operativo, y se conecta al sitio web móvil del banco. Cabe recordar que el browser tiene una función interna de caché de datos, y si bien la transmisión de datos se realiza en forma cifrada usando HTTPS, la caché no es cifrada, y al no serlo, información sensible puede quedar

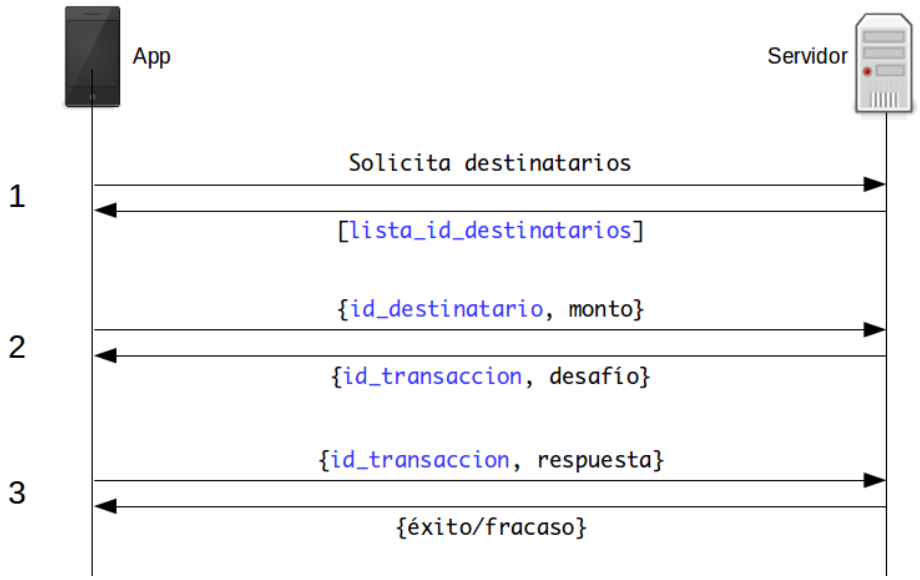


Figura 14: Mitigación ante potencial tampering de datos de transferencias

almacenada en el teléfono y accesible, por ejemplo, por otras aplicaciones en equipos rooteados[41].

Se recomienda que en el sitio web Banco 7 (el cual es usado en la aplicación) toda caché de datos sea deshabilitada. Ésto se logra mediante la aplicación de headers en la transmisión HTTP del servidor[42].

Almacenamiento de información sensible en base de datos SQLite

Como parte de su funcionamiento, muchas aplicaciones almacenan información en bases de datos SQLite, sin cifrado alguno. En el caso de la aplicación del Banco 7, almacena los datos de transferencias bancarias (nombre y correo electrónico del destinatario, etc.). Se recomienda no almacenar tal información del todo, ya que podría ser extraíble de la misma forma que se indicó en el punto anterior.

Falta de timeout

De acuerdo con un estudio desarrollado por la Universidad de Berkeley y Google, el 62 % de los usuarios de teléfonos móviles no utiliza ningún método de bloqueo de sus

[41] OWASP, "Dangers of Jailbreaking and Rooting Mobile Devices": https://www.owasp.org/index.php/Projects/OWASP_Mobile_Security_Project_-_Dangers_of_Jailbreaking_and_Rooting_Mobile_Devices

[42] Stack Overflow, "Making sure a web page is not cached, across all browsers": http://stackoverflow.com/questions/49547/making-sure-a-web-page-is-not-cached-across-all-browsers

equipos, sea éste el uso de un PIN, password o patrón[43]. Es por esto que la implementación de un timeout de cierre en una aplicación móvil es de extrema importancia, ya que un teléfono desbloqueado puede ser fuente, al menos, de filtración de información bancaria.

La aplicación Banco 7 implementa un timeout mayor a 10 minutos. Se recomienda implementar un timeout de al menos 5-10 minutos, y que éste cause que la aplicación vuelva automáticamente a la pantalla de login terminado ese período de tiempo.

Problemas con la conexión HTTPS

Cadena de certificación mal hecha De acuerdo con el análisis realizado, se determinó que la cadena de certificación del servidor contiene dos certificados que corren riesgo de deprecación. Estos certificados son los siguientes:

- www.banco7.cl (fingerprint anonimizado)
- GlobalSign Extended Validation CA - G2 (fingerprint `06456b2c4c26f37c95266793bbedff61e6373dc2`)

Ambos certificados son firmados con el algoritmo SHA1, lo cual causa que serán deprecados a partir del 1 de enero de 2016[44] y los clientes Android podrían arrojar errores de validación de certificados posterior a esa fecha. Se recomienda contactar a la autoridad certificadora y solicitar la renovación de ambos certificados.

Carencia de Forward Secrecy en algunos clientes y otros problemas Forward Secrecy (secreto hacia el futuro) es una propiedad que causa que, si un atacante obtiene la llave secreta del servidor HTTPS, no podrá obtener las llaves de todas las sesiones que alguna vez se hayan generado con esa llave, y por lo tanto no podrá desencriptar el contenido de conversaciones que haya tenido el servidor con sus clientes.

El algoritmo DH (Diffie-Hellman) permite implementar Forward Secrecy. Una de las razones por las cuales los servidores no acostumbran implementarlo es por el hecho de que consume más recursos computacionales que el algoritmo RSA, usado profusamente hoy en día. Sin embargo, el uso de Diffie-Hellman con Curva Elíptica reduce

[43] https://www.eecs.berkeley.edu/~daw/papers/lock-ccs14.pdf

[44] Qualys SSL Labs, "SHA1 Deprecation: What You Need to Know" https://community.qualys.com/blogs/securitylabs/2014/09/09/sha1-deprecation-what-you-need-to-know

significativamente el consumo de recursos. Generalmente este algoritmo se denomina ECDHE[45].

A pesar de que se encuentra implementado en el servidor, está implementado sólo para algunos clientes, y su implementación es débil. Se recomienda implementarlo para todos, como se muestra en la sección "Mitigación".

Mitigación Las vulnerabilidades mencionadas en esta sección pueden ser corregidas cambiando la configuración SSL del servidor web al cual se conecta la aplicación. El analisis no arrojó resultados concluyentes respecto de qué servidor usa mobile.banco7.cl, por lo tanto no se pueden dar recomendaciones específicas para la configuración SSL del servidor. Sin embargo, existe una guía que puede ser útil para tal tarea, desarrollada por la Fundación Mozilla: https://wiki.mozilla.org/Security/Server_Side_TLS. Posterior a la configuración SSL del servidor, ésta se puede probar con las herramientas Cipherscan y Qualys SSL Test, indicadas en la sección "Recursos Interesantes".

Abuso de permisos

En el archivo `AndroidManifest.xml` se definen los permisos que la aplicación solicita al usuario para hacer uso de sus recursos al momento de instalar la aplicación. Algunos de estos permisos son riesgosos, y para el caso del Banco 7, son los siguientes:

Escritura en el almacenamiento externo del teléfono El teléfono incluye un área de almacenamiento externo, el cual no tiene las protecciones incluidas en el área de almacenamiento correspondiente a la aplicación, y por lo tanto, cualquier archivo almacenado en esta área externa podría ser accesible por otras aplicaciones y a través de conexiones USB al teléfono. Se recomienda remover el permiso `android.permission.WRITE_EXTERNAL_STORAGE`.

Uso de georreferenciación Una función ofrecida por las aplicaciones bancarias nacionales es la que permite al usuario obtener una lista o mapa de los cajeros Redbanc o sucursales del banco más cercanas al lugar en el que se encuentra. Para ello se requiere acceso a los sistemas de georreferenciación del teléfono, que son principalmente de dos tipos:

[45]Qualys SSL Labs, "Deploying Forward Secrecy": https://community.qualys.com/blogs/securitylabs/2013/06/25/ssl-labs-deploying-forward-secrecy

- Georreferenciación fina: Éste tipo de georreferenciación se basa en el uso de satélites GPS y se puede acceder a ello mediante el permiso `android.permission.ACCESS_FINE_LOCATION`. Su precisión es de entre 2 y 20 metros.
- Georreferenciación gruesa: Se basa en el uso de torres celulares y hotspots Wifi para entregar una ubicación aproximada del aparato. También se conoce como A-GPS (Assisted GPS) o Network Location Provider, y su precisión es de entre 100 y 200 metros. Se puede acceder a este tipo de georreferenciación con el permiso `android.permission.ACCESS_COARSE_LOCATION`[46].

Dado que la función de localización de cajeros y sucursales no requiere mayor precisión, la recomendación en este caso es quitar el permiso `ACCESS_FINE_LOCATION` y dejar el `ACCESS_COARSE_LOCATION`. Además del beneficio para la privacidad del usuario, la geolocalización gruesa consume menos energía de la batería del aparato y es mucho más rápida en la obtención de información geográfica de éste.

Recursos interesantes

- **NowSecure: 42+ Best Practices for Secure iOS and Android Development**. Este manual incluye una serie de consejos y buenas prácticas para desarrollar aplicaciones seguras, tanto en iOS como en Android: https://goo.gl/ivJD4M
- **Cipherscan**: Herramienta via linea de comandos que analiza la configuración de servidores HTTPS y entrega reportes. Puede ser usada para analizar servidores en estado de pre-producción: https://github.com/jvehent/cipherscan.
- **Qualys SSL Test**: Esta herramienta web gratuita fue utilizada para analizar la configuración actual del servidor en producción, y es utilizable ya para cuando el servidor con su nueva configuración ya se encuentre expuesto a Internet. Entrega información sobre configuración de certificados, protocolos, cifradores, etc., y le *pone nota* al servidor: https://www.ssllabs.com/ssltest/

[46]The Shine Blog, "A Good Look at Android Location Data": http://blog.shinetech.com/2011/10/14/a-good-look-at-android-location-data/

Informe Banco 8

Introducción

El presente informe consiste en un conjunto de recomendaciones de mitigación de vulnerabilidades de seguridad encontradas en la aplicación móvil Banco 8 para Android, y es parte de las tareas de investigación tendientes al trabajo de Tesis de Magister de quien escribe. El informe aquí presentado contiene lo siguiente:

1. Vulnerabilidades encontradas en su aplicación y riesgos asociados.
2. Formas de mitigación de tales vulnerabilidades.

El objetivo es que el equipo de desarrollo de la aplicación bancaria del Banco 8 pueda accionar las recomendaciones aquí indicadas en ésta.

Detalles de la aplicación analizada

- Tipo de aplicación: Híbrida (Apache Cordova 3.7.2)
- Tipo de conexión: REST-JSON

Vulnerabilidades encontradas y su mitigación

Potencial Tampering de datos

La aplicación, al momento de realizar una transferencia de dinero, pudiera ser vulnerable a que un usuario malicioso, o algún malware instalado en el teléfono modifique los datos de transacciones de transferencias de dinero entre cuentas, antes de que esa información sea enviada al servidor vía HTTPS. Éste ataque funcionaría de la forma que se muestra en la figura:

1. La aplicación solicita al servidor la lista de destinatarios registrados por el usuario en su cuenta, ante lo cual éste se la envía.
2. El usuario selecciona un destinatario y el monto a transferir en la app, la cual envía esa información al servidor junto con la cuenta remitente. El servidor responde enviándole a la app los datos de la cuenta destinario, el monto a transferir y un desafío, el cual en el caso del Banco 8 es la solicitud de un número generado por un token.

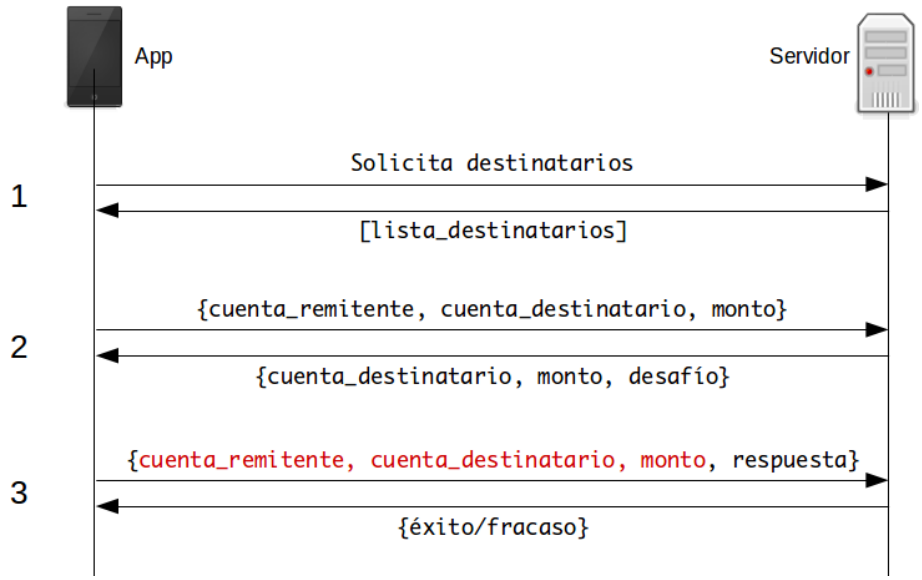


Figura 15: Modificación maliciosa de datos de transferencias

3. El usuario ingresa la respuesta al desafío solicitado, y la aplicación envía ese dato, junto a la cuenta remitente, la cuenta de destinatario y el monto a transferir. Estos datos pueden ser falseados de tal forma que la cuenta remitente no sea la correcta, la cuenta destino no sea la correcta o el monto no sea el que el usuario deseó transferir. Si el servidor no valida esta información, hay riesgo serio de robo de dinero mediante transferencias a cuentas no deseadas.

Para mitigar este problema, se propone el uso de un ID de transacción, como se muestra en la siguiente figura:

En este caso, en vez de dejar información validable en manos del cliente, la cual podría ser modificada maliciosamente, se deja esa información en el servidor y ésta se asocia con un ID de transacción, el cual es enviado a la app junto con el desafío. El usuario ingresa la respuesta, a la cual la app adosa este ID de transacción y la envía al servidor, el cual realiza los chequeos usuales (existencia de la cuenta de destinatario, el cliente tiene fondos suficientes, etc.), realiza la transacción, y responde al usuario si ésta fue exitosa o fallida.

Adicionalmente, antes de la validación, el servidor puede enviar en vez de una lista de destinatarios con mucha información, sólo enviar una lista con pares `{id_destinatario, nombre_destinatario}` en el paso 1, y cuando el usuario seleccione un destinatario, sólo enviar su id_destinatario en el paso 2.

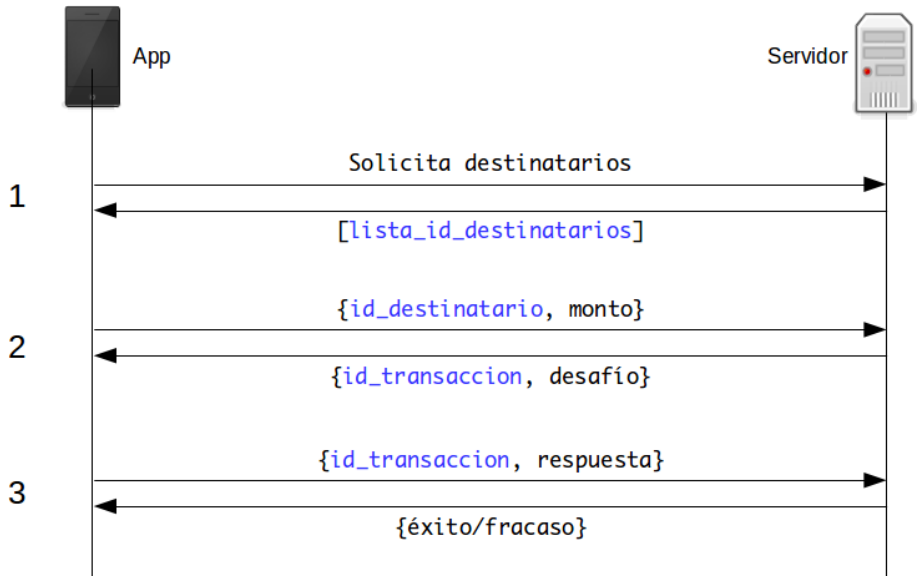


Figura 16: Mitigación ante potencial tampering de datos de transferencias

Mal uso de cifrado en reposo

La aplicación Banco 8 almacena en sus Shared Preferences (un archivo XML ubicado en su área de almacenamiento) el RUT del usuario cifrado con una biblioteca desarrollada específicamente para este propósito[47]. Sin embargo, existen problemas en cómo realiza el proceso de cifrado. A continuación se detallan estos problemas junto con sus respectivas medidas de mitigación:

1. La biblioteca usa como llave de cifrado el ANDROID_ID, valor asignado al teléfono al momento de su activación con Google, el cual permanece en el teléfono hasta que es formateado y es simple de obtener: http://developer.android.com/reference/android/provider/Settings.Secure.html#ANDROID_ID. Se recomienda la generación de una llave de cifrado aleatoria y que ésta sea almacenada, no en el área de almacenamiento de la aplicación, sino que en un sistema llamado Android Keystore, el cual almacena claves en forma segura, cifrándolos con una llave basada en el método de bloqueo del sistema elegido por el usuario, sea éste un PIN, password o patrón con el cual se autentifica ante el teléfono. Para mayor información al respecto, ir a la sección "Recursos Interesantes".
2. Para el cifrado, se genera lo que se conoce como un vector de inicialización (IV). Ese valor debe ser aleatorio, sin embargo en la función `getIv()`, que

[47] `cl.banco8.mibanco.util.SecurePreferences`

es la que genera tal vector, se observa que usa un valor fijo. Se recomienda generar el IV en forma aleatoria, de preferencia utilizando la función `java.security.SecureRandom`, disponible en la API de Java para Android.

Almacenamiento de información sensible en base de datos SQLite

Como parte de su funcionamiento, muchas aplicaciones almacenan información en bases de datos SQLite, sin cifrado alguno. En el caso de la aplicación Banco 8, almacena el nombre del usuario, su RUT y sus cookies de sesión. Se recomienda almacenar tal información en forma cifrada, mediante las recomendaciones indicadas en el punto anterior.

Uso innecesario del PAN de Tarjeta de Crédito en tránsito

La aplicación, al momento del usuario consultar su estado de tarjeta de crédito, muestra al usuario una versión anonimizada del PAN de ésta, de la forma `XXXX-XXXX-XXXX-4448`. Sin embargo, el servidor envía a la aplicación el PAN completo (en la forma `4444-4444-4444-4448`)[48] y es la aplicación la que la anonimiza. Además de ser innecesario enviar el PAN completo desde el servidor a la aplicación, es potencialmente inseguro, ya que mediante un proxy con soporte SSL se podría extraer esa información encriptada en tránsito. Se recomienda anonimizar tal valor antes de ser enviado desde el servidor, en conformidad con el Requisito 3 de la versión 3.1 del estándar PCI-DSS: https://www.pcisecuritystandards.org/documents/PCI_DSS_v3-1.pdf

Problemas con la conexión HTTPS

Uso del protocolo SSL3 El protocolo SSL en su tercera versión (SSL3) ya está a punto de cumplir 20 años en funcionamiento y ya ha sido deprecado por ser inseguro[49]. Se recomienda su remoción de la lista de protocolos, y sólo dejar TLSv1.0, TLSv1.1 y TLSv1.2. Adicionalmente su uso en servidores hace que estos sean vulnerables ante el ataque POODLE, indicado en la sección "POODLE y falta de protección ante downgrades de protocolo".

[48]El PAN es el número de tarjeta de crédito (Primary Account Number). El PAN mostrado en los ejemplos es un número de prueba para tarjetas VISA obtenido desde https://www.auricsystems.com/support-center/sample-credit-card-numbers/

[49]https://tools.ietf.org/html/rfc7568

Uso del cifrador RC4 EL cifrador RC4 es vulnerable ante múltiples ataques criptoanalíticos y su uso está prohibido en configuraciones modernas[50]. La recomendación es eliminarlo de la lista de cifradores, lo cual se tratará en la sección "Mitigación".

Uso del algoritmo SHA1 para validación de mensajes Si bien no existen aún ataques prácticos sobre el algoritmo de validación de mensajes SHA1, se recomienda que este se elimine de los cifradores ofrecidos a los clientes.

Carencia de Forward Secrecy Forward Secrecy (secreto hacia el futuro) es una propiedad que causa que, si un atacante obtiene la llave secreta del servidor HTTPS, no podrá obtener las llaves de todas las sesiones que alguna vez se hayan generado con esa llave, y por lo tanto no podrá desencriptar el contenido de conversaciones que haya tenido el servidor con sus clientes.

El algoritmo DH (Diffie-Hellman) permite implementar Forward Secrecy. Una de las razones por las cuales los servidores no acostumbran implementarlo es por el hecho de que consume más recursos computacionales que el algoritmo RSA, usado profusamente hoy en día. Sin embargo, el uso de Diffie-Hellman con Curva Elíptica reduce significativamente el consumo de recursos. Generalmente este algoritmo se denomina ECDHE[51].

CRIME CRIME[52] es un ataque que permite robar información de las cookies de sesión, aún cuando éstas estén encriptadas en tránsito. El ataque se realiza a través de la función de compresión TLS. La mitigación en este caso consiste en deshabilitar la función de compresión.

POODLE y falta de protección ante downgrades de protocolo POODLE (Padding Oracle On Downgraded Legacy Encryption), es un ataque el cual es montado mediante la técnica de *hombre en el medio*[53] por algún inescrupuloso que pudiera estar monitoreando redes Wi-Fi, por ejemplo, y en el cual causa que una conexión que utiliza una versión del protocolo de comunicación segura (en este caso TLSv1.0), realice un *down-*

[50]https://tools.ietf.org/html/rfc7465

[51]Qualys SSL Labs, "Deploying Forward Secrecy": https://community.qualys.com/blogs/securitylabs/2013/06/25/ssl-labs-deploying-forward-secrecy

[52]Compression Ratio Info-leak Made Easy: https://community.qualys.com/blogs/securitylabs/2012/09/14/crime-information-leakage-attack-against-ssltls

[53]Man-In-The-Middle: https://en.wikipedia.org/wiki/Man_in_the_middle_attack

grade hacia una versión insegura (como es SSL3). La solución para esta vulnerabilidad particular es eliminar SSL3 de la lista de protocolos soportados por el servidor.

Adicionalmente se recomienda que el servidor utilice la biblioteca OpenSSL versión 1.0.1j o superior, la cual soporta la propiedad `TLS_FALLBACK_SCSV` y permite una mayor protección ante downgrades de protocolo.

Mitigación Las vulnerabilidades mencionadas en esta sección pueden ser corregidas cambiando la configuración SSL del servidor web al cual se conecta la aplicación. De acuerdo con el análisis, el servidor HTTP utilizado por la aplicación es un servidor Apache. La Fundación Mozilla dispuso en línea una herramienta para generar automáticamente configuraciones SSL para servidores Apache. Se puede encontrar aquí: https://mozilla.github.io/server-side-tls/ssl-config-generator/. Posterior a su configuración, ésta se puede probar con las herramientas Cipherscan y Qualys SSL Test, indicadas en la sección "Recursos Interesantes".

Abuso de logging (código de pruebas olvidado)

Android, como todo sistema Linux, contiene un log interno el cual puede ser utilizado para reportar eventos ocurridos dentro del sistema. Generalmente, las aplicaciones reportan automáticamente al log eventos relacionados con uso de recursos.

Asimismo, los mismos desarrolladores pueden utilizar el log para realizar tareas de corrección de errores (debugging). Por ejemplo, verificar valores de variables y su progresión durante la ejecución de la aplicación. Lamentablemente, muchas veces ese código de prueba es olvidado en la aplicación y pasa a producción, provocando filtración de información aprovechable por cualquiera que pudiera leer el log, por ejemplo:

- Otras aplicaciones que tengan permisos para leerlo (mediante el permiso `android.permission.READ_LOGS`).
- Malware instalado en el equipo.
- Alguien que active el modo de desarrollador en el teléfono y pueda leer el log mediante la conexión USB desde un computador.

La aplicación Banco 8, envía al log interno de Android la password del usuario que se conecta al servicio. Se recomienda encarecidamente eliminar tal código de la versión de producción de la aplicación.

Abuso de permisos

En el archivo `AndroidManifest.xml` se definen los permisos que la aplicación solicita al usuario para hacer uso de sus recursos al momento de instalar la aplicación. Algunos de estos permisos son riesgosos, y para el caso del Banco 8, son los siguientes:

Acceso a contactos del usuario La aplicación hace uso de los siguientes permisos:

- `android.permission.READ_CONTACTS`
- `android.permission.WRITE_CONTACTS`

Esto puede dar pie a abusos de privacidad en contra del usuario. Se recomienda remover estos permisos de la aplicación.

Escritura en el almacenamiento externo del teléfono El teléfono incluye un área de almacenamiento externo, el cual no tiene las protecciones incluidas en el área de almacenamiento correspondiente a la aplicación, y por lo tanto, cualquier archivo almacenado en esta área externa podría ser accesible por otras aplicaciones y a través de conexiones USB al teléfono. Se recomienda remover el permiso `android.permission.WRITE_EXTERNAL_STORAGE`.

Recursos interesantes

- **NowSecure: 42+ Best Practices for Secure iOS and Android Development**. Este manual incluye una serie de consejos y buenas prácticas para desarrollar aplicaciones seguras, tanto en iOS como en Android: https://goo.gl/ivJD4M
- **Cipherscan**: Herramienta via linea de comandos que analiza la configuración de servidores HTTPS y entrega reportes. Puede ser usada para analizar servidores en estado de pre-producción: https://github.com/jvehent/cipherscan.
- **Qualys SSL Test**: Esta herramienta web gratuita fue utilizada para analizar la configuración actual del servidor en producción, y es utilizable ya para cuando el servidor con su nueva configuración ya se encuentre expuesto a Internet. Entrega información sobre configuración de certificados, protocolos, cifradores, etc., y le *pone nota* al servidor: https://www.ssllabs.com/ssltest/
- **Android Keystore**: Es un sistema de almacenamiento seguro de llaves o valores que se desea que sean secretos. Funciona como una tabla de datos encriptada

mediante el PIN, password o patrón del usuario del equipo, por lo tanto no es fácil de obtener. Puede ver la documentación oficial de Android al respecto en http://developer.android.com/intl/es/training/articles/keystore.html y un ejemplo de su uso en http://www.androidauthority.com/use-android-keystore-store-passwords-sensitive-information-623779/

Informe Banco 9

Introducción

El presente informe consiste en un conjunto de recomendaciones de mitigación de vulnerabilidades de seguridad encontradas en la aplicación móvil para Android de Banco 9, y es parte de las tareas de investigación tendientes al trabajo de Tesis de Magister de quien escribe. El informe aquí presentado contiene lo siguiente:

1. Vulnerabilidades encontradas en su aplicación y riesgos asociados.
2. Formas de mitigación de tales vulnerabilidades.

El objetivo es que el equipo de desarrollo de la aplicación bancaria de Banco 9 pueda accionar las recomendaciones aquí indicadas en ésta.

Detalles de la aplicación analizada

- Tipo de aplicación: Nativa
- Tipo de conexión: HTTP-JSON

Vulnerabilidades encontradas y su mitigación

Potencial Tampering de datos

La aplicación, al momento de realizar una transferencia de dinero, pudiera ser vulnerable a que un usuario malicioso, o algún malware instalado en el teléfono modifique los datos de transacciones de transferencias de dinero entre cuentas, antes de que esa información sea enviada al servidor vía HTTPS. Éste ataque funcionaría de la forma que se muestra en la figura:

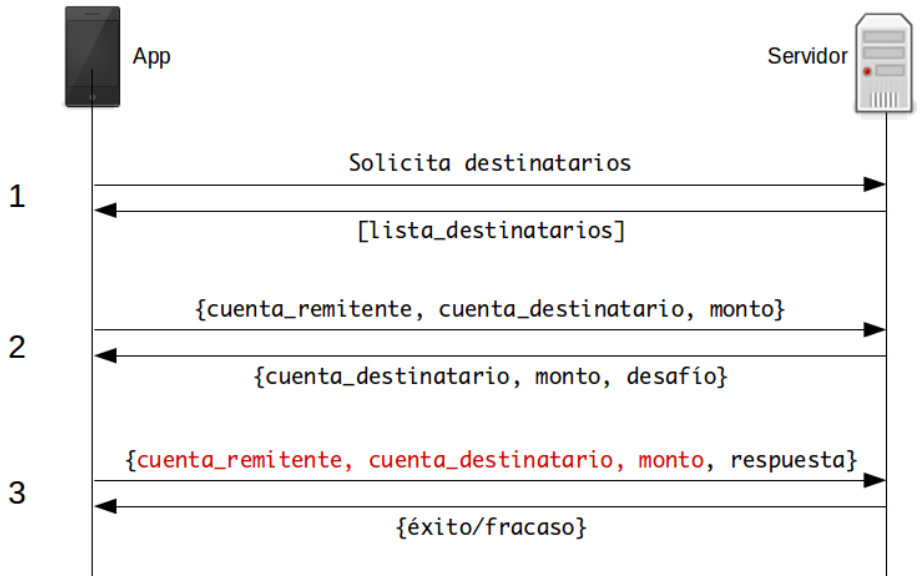


Figura 17: Modificación maliciosa de datos de transferencias

1. La aplicación solicita al servidor la lista de destinatarios registrados por el usuario en su cuenta, ante lo cual éste se la envía.
2. El usuario selecciona un destinatario y el monto a transferir en la app, la cual envía esa información al servidor junto con la cuenta remitente. El servidor responde enviándole a la app los datos de la cuenta destinario, el monto a transferir y un desafío, el cual en el caso del Banco 9 son tres coordenadas de la tarjeta de coordenadas del usuario.
3. El usuario ingresa la respuesta al desafío solicitado, y la aplicación envía ese dato, junto a la cuenta remitente, la cuenta de destinatario y el monto a transferir. Estos datos pueden ser falseados de tal forma que la cuenta remitente no sea la correcta, la cuenta destino no sea la correcta o el monto no sea el que el usuario deseó transferir. Si el servidor no valida esta información, hay riesgo serio de robo de dinero mediante transferencias a cuentas no deseadas.

Para mitigar este problema, se propone el uso de un ID de transacción, como se muestra en la siguiente figura:

En este caso, en vez de dejar información validable en manos del cliente, la cual podría ser modificada maliciosamente, se deja esa información en el servidor y ésta se asocia con un ID de transacción, el cual es enviado a la app junto con el desafío. El usuario ingresa la respuesta, a la cual la app adosa este ID de transacción y la envía al servidor, el cual realiza los chequeos usuales (existencia de la cuenta de destinatario, el cliente tiene fondos suficientes, etc.), realiza la transacción, y responde al usuario si ésta fue

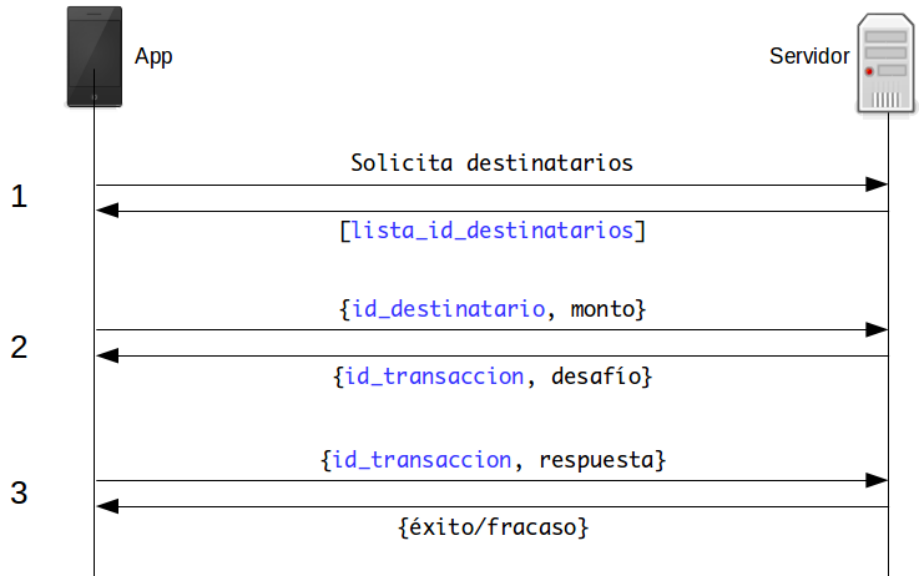


Figura 18: Mitigación ante potencial tampering de datos de transferencias

exitosa o fallida.

Adicionalmente, antes de la validación, el servidor puede enviar en vez de una lista de destinatarios con mucha información, sólo enviar una lista con pares `{id_destinatario, nombre_destinatario}` en el paso 1, y cuando el usuario seleccione un destinatario, sólo enviar su id_destinatario en el paso 2.

Timeout mal implementado

De acuerdo con un estudio desarrollado por la Universidad de Berkeley y Google, el 62 % de los usuarios de teléfonos móviles no utiliza ningún método de bloqueo de sus equipos, sea éste el uso de un PIN, password o patrón[54]. Es por esto que la implementación de un timeout de cierre en una aplicación móvil es de extrema importancia, ya que un teléfono desbloqueado puede ser fuente, al menos, de filtración de información bancaria.

Si bien la aplicación del Banco 9 implementa un timeout, éste debe mejorar su implementación. Su funcionamiento consiste en que al finalizar el tiempo de la sesión, aparece una ventana alertando al usuario del cierre automático de ésta. Tal ventana no cubre toda la pantalla del aparato, y deja ver parte del contenido. En vez de eso, la aplicación debería volver a la pantalla de login automáticamente.

[54]https://www.eecs.berkeley.edu/~daw/papers/lock-ccs14.pdf

Problemas con la conexión HTTPS

Cadena de certificación mal hecha De acuerdo con el análisis realizado, se determinó que la cadena de certificación del servidor está incompleta. Falta incluir el siguiente certificado:

- Symantec Class 3 Secure Server CA - G4
  (fingerprint `ff67367c5cd4de4ae18bcce1d70fdabd7c866135`)

Si bien esto puede no ser una vulnerabilidad en el corto plazo, eventualmente versiones de Android nuevas pudieran entregar errores de validación de certificados y no permitir la conexión al servidor del todo.

Uso del protocolo SSL3 El protocolo SSL en su tercera versión (SSL3) ya está a punto de cumplir 20 años en funcionamiento y ya ha sido deprecado por ser inseguro[55]. Se recomienda su remoción de la lista de protocolos, y sólo dejar TLSv1.0, TLSv1.1 y TLSv1.2. Adicionalmente su uso en servidores hace que estos sean vulnerables ante el ataque POODLE, indicado en la sección "POODLE y falta de protección ante downgrades de protocolo".

Uso del cifrador RC4 EL cifrador RC4 es vulnerable ante múltiples ataques criptoanalíticos y su uso está prohibido en configuraciones modernas[56]. La recomendación es eliminarlo de la lista de cifradores, lo cual se tratará en la sección "Mitigación".

Uso del algoritmo SHA1 para validación de mensajes Si bien no existen aún ataques prácticos sobre el algoritmo de validación de mensajes SHA1, se recomienda que este se elimine de los cifradores ofrecidos a los clientes.

Carencia de Forward Secrecy Forward Secrecy (secreto hacia el futuro) es una propiedad que causa que, si un atacante obtiene la llave secreta del servidor HTTPS, no podrá obtener las llaves de todas las sesiones que alguna vez se hayan generado con esa llave, y por lo tanto no podrá desencriptar el contenido de conversaciones que haya tenido el servidor con sus clientes.

[55] https://tools.ietf.org/html/rfc7568
[56] https://tools.ietf.org/html/rfc7465

El algoritmo DH (Diffie-Hellman) permite implementar Forward Secrecy. Una de las razones por las cuales los servidores no acostumbran implementarlo es por el hecho de que consume más recursos computacionales que el algoritmo RSA, usado profusamente hoy en día. Sin embargo, el uso de Diffie-Hellman con Curva Elíptica reduce significativamente el consumo de recursos. Generalmente este algoritmo se denomina ECDHE[57].

POODLE y falta de protección ante downgrades de protocolo POODLE (Padding Oracle On Downgraded Legacy Encryption), es un ataque el cual es montado mediante la técnica de *hombre en el medio*[58] por algún inescrupuloso que pudiera estar monitoreando redes Wi-Fi, por ejemplo, y en el cual causa que una conexión que utiliza una versión del protocolo de comunicación segura (en este caso TLSv1.0), baje a una versión insegura (como es SSL3). La solución para esta vulnerabilidad particular es eliminar SSL3 de la lista de protocolos soportados por el servidor.

Mitigación Las vulnerabilidades mencionadas en esta sección pueden ser corregidas cambiando la configuración SSL del servidor web al cual se conecta la aplicación. De acuerdo con el análisis, el servidor HTTP utilizado por la aplicación es un IBM HTTP Server, el cual al ser basado en Apache, permite que los mismos parámetros de configuración puedan ser utilizados. La documentación sobre SSL para el IBM HTTP Server se encuentra en http://www-01.ibm.com/support/docview.wss?uid=swg21179559.

Adicionalmente, la Fundación Mozilla dispuso en línea una herramienta para generar automáticamente configuraciones SSL para servidores Apache, la cual se puede aplicar para IBM HTTP Server. Se puede encontrar aquí: https://mozilla.github.io/server-side-tls/ssl-config-generator/. Posterior a su configuración, ésta se puede probar con las herramientas Cipherscan y Qualys SSL Test, indicadas en la sección "Recursos Interesantes".

Abuso de permisos

En el archivo `AndroidManifest.xml` se definen los permisos que la aplicación solicita al usuario para hacer uso de sus recursos al momento de instalar la aplicación. Algunos de estos permisos son riesgosos, y para el caso del Banco 9, son los siguientes:

[57] Qualys SSL Labs, "Deploying Forward Secrecy": https://community.qualys.com/blogs/securitylabs/2013/06/25/ssl-labs-deploying-forward-secrecy

[58] Man-In-The-Middle: https://en.wikipedia.org/wiki/Man_in_the_middle_attack

Escritura en el almacenamiento externo del teléfono El teléfono incluye un área de almacenamiento externo, el cual no tiene las protecciones incluidas en el área de almacenamiento correspondiente a la aplicación, y por lo tanto, cualquier archivo almacenado en esta área externa podría ser accesible por otras aplicaciones y a través de conexiones USB al teléfono. Se recomienda remover el permiso `android.permission.WRITE_EXTERNAL_STORAGE`.

Uso de georreferenciación Una función ofrecida por las aplicaciones bancarias nacionales es la que permite al usuario obtener una lista o mapa de los cajeros Redbanc o sucursales del banco más cercanas al lugar en el que se encuentra. Para ello se requiere acceso a los sistemas de georreferenciación del teléfono, que son principalmente de dos tipos:

- Georreferenciación fina: Éste tipo de georreferenciación se basa en el uso de satélites GPS y se puede acceder a ello mediante el permiso `android.permission.ACCESS_FINE_LOCATION`. Su precisión es de entre 2 y 20 metros.
- Georreferenciación gruesa: Se basa en el uso de torres celulares y hotspots Wifi para entregar una ubicación aproximada del aparato. También se conoce como A-GPS (Assisted GPS) o Network Location Provider, y su precisión es de entre 100 y 200 metros. Se puede acceder a este tipo de georreferenciación con el permiso `android.permission.ACCESS_COARSE_LOCATION`[59].

Dado que la función de localización de cajeros y sucursales no requiere mayor precisión, la recomendación en este caso es quitar el permiso `ACCESS_FINE_LOCATION` y dejar el `ACCESS_COARSE_LOCATION`. Además del beneficio para la privacidad del usuario, la geolocalización gruesa consume menos energía de la batería del aparato y es mucho más rápida en la obtención de información geográfica de éste.

Recursos interesantes

- **NowSecure: 42+ Best Practices for Secure iOS and Android Development**. Este manual incluye una serie de consejos y buenas prácticas para desarrollar aplicaciones seguras, tanto en iOS como en Android: https://goo.gl/ivJD4M

[59]The Shine Blog, "A Good Look at Android Location Data": http://blog.shinetech.com/2011/10/14/a-good-look-at-android-location-data/

- **Cipherscan**: Herramienta via linea de comandos que analiza la configuración de servidores HTTPS y entrega reportes. Puede ser usada para analizar servidores en estado de pre-producción: https://github.com/jvehent/cipherscan.
- **Qualys SSL Test**: Esta herramienta web gratuita fue utilizada para analizar la configuración actual del servidor en producción, y es utilizable ya para cuando el servidor con su nueva configuración ya se encuentre expuesto a Internet. Entrega información sobre configuración de certificados, protocolos, cifradores, etc., y le *pone nota* al servidor: https://www.ssllabs.com/ssltest/

Informe Banco 10

Introducción

El presente informe consiste en un conjunto de recomendaciones de mitigación de vulnerabilidades de seguridad encontradas en la aplicación móvil para Android de Banco 10, y es parte de las tareas de investigación tendientes al trabajo de Tesis de Magister de quien escribe. El informe aquí presentado contiene lo siguiente:

1. Vulnerabilidades encontradas en su aplicación y riesgos asociados.
2. Formas de mitigación de tales vulnerabilidades.

El objetivo es que el equipo de desarrollo de la aplicación bancaria de Banco 10 pueda accionar las recomendaciones aquí indicadas en ésta.

Detalles de la aplicación analizada

- Tipo de aplicación: Nativa
- Tipo de conexión: HTTP-XML

Vulnerabilidades encontradas y su mitigación

Potencial Tampering de datos

La aplicación, al momento de realizar una transferencia de dinero, pudiera ser vulnerable a que un usuario malicioso, o algún malware instalado en el teléfono modifique los datos de transacciones de transferencias de dinero entre cuentas, antes de que esa

información sea enviada al servidor vía HTTPS. Éste ataque funcionaría de la forma que se muestra en la figura:

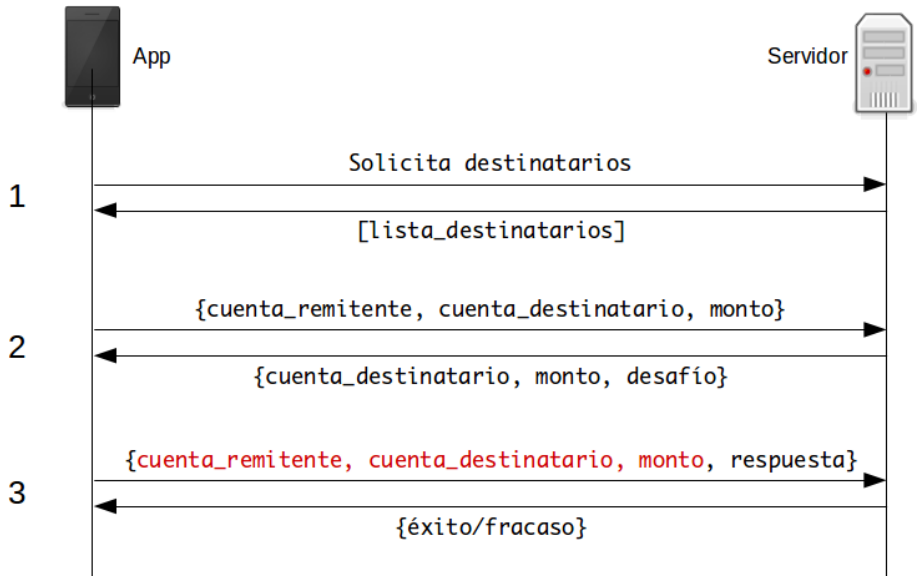


Figura 19: Modificación maliciosa de datos de transferencias

1. La aplicación solicita al servidor la lista de destinatarios registrados por el usuario en su cuenta, ante lo cual éste se la envía.
2. El usuario selecciona un destinatario y el monto a transferir en la app, la cual envía esa información al servidor junto con la cuenta remitente. El servidor responde enviándole a la app los datos de la cuenta destinario, el monto a transferir y un desafío, el cual en el caso del Banco 10 son tres coordenadas de la tarjeta de coordenadas del usuario.
3. El usuario ingresa la respuesta al desafío solicitado, y la aplicación envía ese dato, junto a la cuenta remitente, la cuenta de destinatario y el monto a transferir. Estos datos pueden ser falseados de tal forma que la cuenta remitente no sea la correcta, la cuenta destino no sea la correcta o el monto no sea el que el usuario deseó transferir. Si el servidor no valida esta información, hay riesgo serio de robo de dinero mediante transferencias a cuentas no deseadas.

Para mitigar este problema, se propone el uso de un ID de transacción, como se muestra en la siguiente figura:

En este caso, en vez de dejar información validable en manos del cliente, la cual podría ser modificada maliciosamente, se deja esa información en el servidor y ésta se asocia con un ID de transacción, el cual es enviado a la app junto con el desafío. El usuario

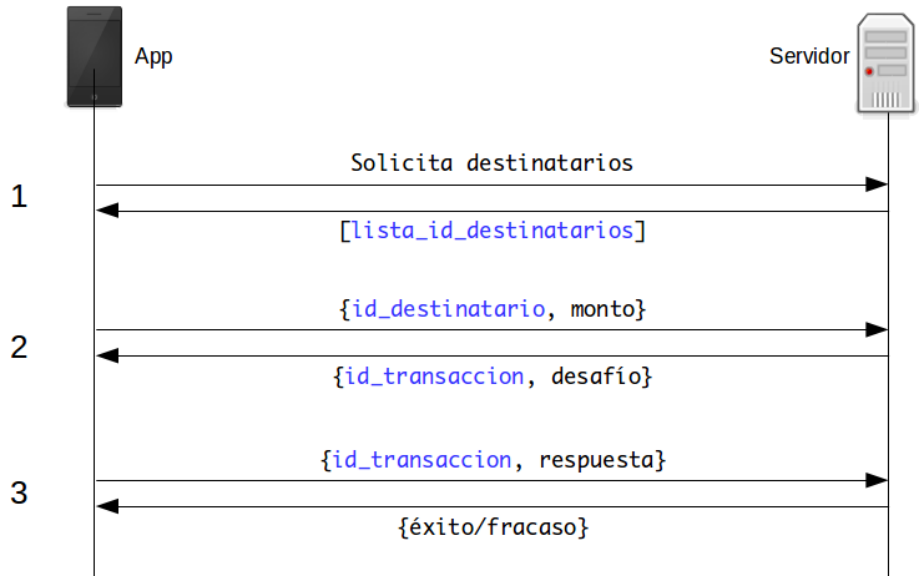


Figura 20: Mitigación ante potencial tampering de datos de transferencias

ingresa la respuesta, a la cual la app adosa este ID de transacción y la envía al servidor, el cual realiza los chequeos usuales (existencia de la cuenta de destinatario, el cliente tiene fondos suficientes, etc.), realiza la transacción, y responde al usuario si ésta fue exitosa o fallida.

Adicionalmente, antes de la validación, el servidor puede enviar en vez de una lista de destinatarios con mucha información, sólo enviar una lista con pares `{id_destinatario, nombre_destinatario}` en el paso 1, y cuando el usuario seleccione un destinatario, sólo enviar su id_destinatario en el paso 2.

Abuso de logging (código de pruebas olvidado)

Android, como todo sistema Linux, contiene un log interno el cual puede ser utilizado para reportar eventos ocurridos dentro del sistema. Generalmente, las aplicaciones reportan automáticamente al log eventos relacionados con uso de recursos.

Asimismo, los mismos desarrolladores pueden utilizar el log para realizar tareas de corrección de errores (debugging). Por ejemplo, verificar valores de variables y su progresión durante la ejecución de la aplicación. Lamentablemente, muchas veces ese código de prueba es olvidado en la aplicación y pasa a producción, provocando filtración de información aprovechable por cualquiera que pudiera leer el log, por ejemplo:

- Otras aplicaciones que tengan permisos para leerlo (mediante el permiso `android.permission.READ_LOGS`).

- Malware instalado en el equipo.
- Alguien que active el modo de desarrollador en el teléfono y pueda leer el log mediante la conexión USB desde un computador.

La aplicación Banco 10, envía al log interno de Android los siguientes datos sensibles:

- Información geográfica (extraída desde GPS).
- Cookies de sesión.
- RUT del usuario.
- Limite de monto a transferir a terceros.
- Desafío y respuesta a desafío de tarjeta de coordenadas.
- Datos del dastinatario de transferencia bancaria.

Se recomienda encarecidamente eliminar tal código de la versión de producción de la aplicación.

Envío de información sensible a través de canal inseguro (HTTP)

Si bien, la información manejada por la aplicación Banco 10 es cifrada, hay un aspecto que llama la atención. La aplicación ofrece al usuario la posibilidad de acceder a información de comercios cercanos a su ubicación geográfica. Para ello, toma la información desde el GPS del aparato y la envía a un servicio web sin cifrado alguno en el transporte. Se recomienda que tal servidor, y la conexión a éste, se manejen vía HTTPS.

Problemas con la conexión HTTPS

Cadena de certificación mal hecha De acuerdo con el análisis realizado, se determinó que la cadena de certificación del servidor contiene dos certificados que corren riesgo de deprecación. Estos certificados son los siguientes:

- www.banco10.cl (fingerprint anonimizado)
- GeoTrust SSL CA - G2 (fingerprint `4f56644858829ffb85a770171accf9f8407a137b`)

Ambos certificados son firmados con el algoritmo SHA1, lo cual causa que serán deprecados a partir del 1 de enero de 2016[60] y los clientes Android podrían arrojar errores de validación de certificados posterior a esa fecha. Se recomienda contactar a la autoridad certificadora y solicitar la renovación de ambos certificados.

[60] Qualys SSL Labs, "SHA1 Deprecation: What You Need to Know" https://community.qualys.com/blogs/securitylabs/2014/09/09/sha1-deprecation-what-you-need-to-know

Uso del cifrador RC4 EL cifrador RC4 es vulnerable ante múltiples ataques criptoanalíticos y su uso está prohibido en configuraciones modernas[61]. La recomendación es eliminarlo de la lista de cifradores, lo cual se tratará en la sección "Mitigación".

Mitigación Las vulnerabilidades mencionadas en esta sección pueden ser corregidas cambiando la configuración SSL del servidor web al cual se conecta la aplicación. El analisis no arrojó resultados concluyentes respecto de qué servidor usa ws.banco10.cl, por lo tanto no se pueden dar recomendaciones específicas para la configuración SSL del servidor. Sin embargo, existe una guía que puede ser útil para tal tarea, desarrollada por la Fundación Mozilla: https://wiki.mozilla.org/Security/Server_Side_TLS. Posterior a la configuración SSL del servidor, ésta se puede probar con las herramientas Cipherscan y Qualys SSL Test, indicadas en la sección "Recursos Interesantes".

Abuso de permisos

En el archivo `AndroidManifest.xml` se definen los permisos que la aplicación solicita al usuario para hacer uso de sus recursos al momento de instalar la aplicación. Algunos de estos permisos son riesgosos, y para el caso del Banco 10, son los siguientes:

Uso de georreferenciación Una función ofrecida por las aplicaciones bancarias nacionales es la que permite al usuario obtener una lista o mapa de los cajeros Redbanc o sucursales del banco más cercanas al lugar en el que se encuentra. Para ello se requiere acceso a los sistemas de georreferenciación del teléfono, que son principalmente de dos tipos:

- Georreferenciación fina: Éste tipo de georreferenciación se basa en el uso de satélites GPS y se puede acceder a ello mediante el permiso `android.permission.ACCESS_FINE_LOCATION`. Su precisión es de entre 2 y 20 metros.
- Georreferenciación gruesa: Se basa en el uso de torres celulares y hotspots Wifi para entregar una ubicación aproximada del aparato. También se conoce como A-GPS (Assisted GPS) o Network Location Provider, y su precisión es de entre 100 y 200 metros. Se puede acceder a este tipo de georreferenciación con el permiso `android.permission.ACCESS_COARSE_LOCATION`[62].

[61] https://tools.ietf.org/html/rfc7465

[62] The Shine Blog, "A Good Look at Android Location Data": http://blog.shinetech.com/2011/10/14/a-good-look-at-android-location-data/

Dado que la función de localización de cajeros y sucursales no requiere mayor precisión, la recomendación en este caso es quitar el permiso `ACCESS_FINE_LOCATION` y dejar el `ACCESS_COARSE_LOCATION`. Además del beneficio para la privacidad del usuario, la geolocalización gruesa consume menos energía de la batería del aparato y es mucho más rápida en la obtención de información geográfica de éste.

Escritura en el almacenamiento externo del teléfono El teléfono incluye un área de almacenamiento externo, el cual no tiene las protecciones incluidas en el área de almacenamiento correspondiente a la aplicación, y por lo tanto, cualquier archivo almacenado en esta área externa podría ser accesible por otras aplicaciones y a través de conexiones USB al teléfono. Se recomienda remover el permiso `android.permission.WRITE_EXTERNAL_STORAGE`.

Recursos interesantes

- **NowSecure: 42+ Best Practices for Secure iOS and Android Development**. Este manual incluye una serie de consejos y buenas prácticas para desarrollar aplicaciones seguras, tanto en iOS como en Android: https://goo.gl/ivJD4M
- **Cipherscan**: Herramienta via linea de comandos que analiza la configuración de servidores HTTPS y entrega reportes. Puede ser usada para analizar servidores en estado de pre-producción: https://github.com/jvehent/cipherscan.
- **Qualys SSL Test**: Esta herramienta web gratuita fue utilizada para analizar la configuración actual del servidor en producción, y es utilizable ya para cuando el servidor con su nueva configuración ya se encuentre expuesto a Internet. Entrega información sobre configuración de certificados, protocolos, cifradores, etc., y le *pone nota* al servidor: https://www.ssllabs.com/ssltest/