GUÍA PRÁCTICA DE PROGRAMACIÓN SEGURA PARA APLICACIONES NATIVAS EN ANDROID

EDUAR MORALES OSORIO

UNIVERSIDAD NACIONAL ABIERTA Y A DISTANCIA

ESPECIALIZACIÓN EN SEGURIDAD INFORMÁTICA

ESCUELA DE CIENCIAS BÁSICAS, TECNOLOGÍA E INGENIERÍA – ECBTI

MANIZALES

2019

GUÍA PRÁCTICA DE PROGRAMACIÓN SEGURA PARA APLICACIONES NATIVAS EN ANDROID

EDUAR MORALES OSORIO

Monografía como opción de grado para optar por el título de especialista en seguridad informática

Director de proyecto:

EDUARD ANTONIO MANTILLA TORRES

UNIVERSIDAD NACIONAL ABIERTA Y A DISTANCIA

ESPECIALIZACIÓN EN SEGURIDAD INFORMÁTICA

ESCUELA DE CIENCIAS BÁSICAS, TECNOLOGÍA E INGENIERÍA – ECBTI

MANIZALES

2019

Nota de aceptación

_____

_____

_____

_____

_____

_____

Firmal del presidente del jurado

_____

Firma del jurado

_____

Firma del jurado

Manizales, 23-05-2019

AGRADECIMIENTOS

A mi familia y personas cercanas que siempre me han apoyado en la consecución de mis objetivos.

CONTENIDO

Pág.

LISTA DE FIGURAS

RESUMEN

En el presente documento se realiza una revisión del panorama de la seguridad en el desarrollo de las aplicaciones móviles en Android, donde se establece que es una problemática importante en la que los desarrolladores juegan un papel importante, por tal razón se recomienda que éstos tengan un conocimiento más profundo sobre los temas de seguridad y la arquitectura del sistema operativo, para poder pensar en pro de la seguridad de los datos. Este documento aborda la premisa anterior haciendo un recorrido desde los principales conceptos de seguridad, arquitectura del sistema, seguridad en las aplicaciones móviles y las principales recomendaciones para almacenamiento seguro de datos, transmisión de datos y manejo de permisos, que son los principales elementos esenciales en la seguridad de las aplicaciones móviles de Android, además, se tratan otras recomendaciones y buenas prácticas de seguridad relacionadas con el almacenamiento interno, almacenamiento externo y manejo del archivo manifest. En la parte final del documento se pueden encontrar códigos de ejemplo para ayudan a implementar y entender de mejor manera los conceptos tratados en el documento.

INTRODUCCIÓN

Android es uno de los sistemas operativos más populares a nivel mundial, junto con iOS, estos sistemas operativos móviles han ganado su popularidad debido a que los smartphones se han convertido en dispositivos casi que indispensables para las personas, ya que son de fácil acceso y permiten realizar tareas cotidianas, por ejemplo, facilitan la comunicación, transferencia y consulta datos, pasar tiempo de ocio y apoyar labores profesionales. Gracias a la cantidad de funcionalidades que tienen los dispositivos móviles, tanto Android como iOS cuentan con millones de usuarios a nivel mundial, lo que significa que hay millones de datos que manejan estos dispositivos, convirtiéndolos así en un objetivo valioso de ataque para los ciberdelincuentes. Esta es una situación de la cual están conscientes en Google y Apple ya que cada vez se han implementado nuevos y mejores mecanismos de seguridad relacionados con la concepción de la arquitectura de la plataforma, sin embargo, uno de los elementos más importantes de los dispositivos móviles son las aplicaciones o apps que son las que permiten a los usuarios realizar diferentes labores, pero a la vez representan muchas veces una vulnerabilidad para el sistema, debido a que por malas prácticas de programación permiten inyectar códigos maliciosos para escala privilegios y ganar acceso root, por ejemplo, las vulnerabilidades de validación de entrada es una de las amenazas más series en la seguridad de las aplicaciones que puede llevar a ataques tipo confused deputy y denegación de servicios[1].

Considerando que el código fuente de una aplicación móvil puede representar una vulnerabilidad para ganar acceso root, modificar el comportamiento de una aplicación e incluso para infectar un dispositivo y realizar otro tipo de ataque, este documento busca aportar una serie de lineamientos que permitan a los desarrolladores de aplicaciones móviles nativas de Android implementar buenas

[1] FANG, Zhejun, *et al.* A static technique for detecting input validation vulnerabilities in Android apps. En: SCIENCE China Information Sciences. Mayo, 2017. Vol. 60, no. 5, p. 2.

prácticas para desarrollar aplicaciones móviles más seguras que consideren el manejo de actividades, permisos, persistencia de datos y aprovechamiento de los recursos de los smartphone.

El desarrollo del documento empieza con una contextualización sobre la arquitectura de Android y los principales conceptos de seguridad sobre el mismo, continuando con la identificación de malas prácticas durante el desarrollo y se finaliza con una serie de recomendaciones y ejemplos prácticos que permitan comprender la razón y la manera de implementar buenas prácticas de programación en el desarrollo móvil.

1. TÍTULO DE LA MONOGRAFÍA

Guía práctica de programación segura para aplicaciones nativas de Android.

2. DEFINICIÓN DEL PROBLEMA

Con la constante evolución de los sistemas operativos iOS y Android, también se han visto en auge las aplicaciones móviles que funcionan sobre dichos sistemas operativos. Estas aplicaciones, conocidas como Apps, abarcan una gran parte del mercado y son preferidas por los usuarios, aunque existan páginas web que cumplen la misma funcionalidad, por ejemplo, redes sociales, juegos, aplicaciones para buscar empleo, entre otras. El encanto de las Apps radica en que son más personales y se ajustan a las necesidades de los usuarios al articularse con los smartphones, permitiendo que los usuarios encuentren funcionalidad en las Apps que les ayudan en tareas cotidianas o simplemente en el tiempo de ocio. Sin embargo, como es normal en el mundo de la tecnología, todo beneficio tiene su desventaja, en este caso las aplicaciones móviles se han convertido en un objetivo de ataque por parte de los delincuentes informáticos que buscan robar datos de los usuarios, ya que estos últimos manejan a través de las Apps información sensible y que es valiosa para los delincuentes, por ejemplo, información financiera.

Debido a que las aplicaciones móviles procesan datos sensibles y cada vez son más utilizadas, es evidente que no sólo se debe abastecer la demanda de aplicaciones, también es necesario que se aumenten las medidas de seguridad que se tienen con las Apps, por ejemplo, los filtros utilizados por las tiendas oficiales de aplicaciones deben ser más estrictos, ya que actualmente se han descubierto aplicaciones populares en la tienda de aplicaciones de Google que incluían códigos maliciosos y contaban con alrededor de 560.000 descargas[2]. Estas aplicaciones además de instalar malware, solicitaban permisos del usuario llegando a tener acceso al tráfico de red permitiendo el robo de datos sensibles[3], lo que demuestra

[2] ABC. Estas son las trece “apps” de juegos con “malware” oculto que Google ha retirado de la Google Play, [En línea]. Disponible en: https://www.abc.es/tecnologia/moviles/aplicaciones/abci-estas-trece-apps-juegos-malware-oculto-android-retirado-google-play-201811271717_noticia.html

[3] TECHRADAR. Half a billion Android users downloaded malware from Play Store, [En línea]. Disponible en: https://www.techradar.com/news/half-a-billion-android-users-downloaded-malware-from-play-store

que los usuarios deben ser más conscientes sobre el uso y descarga de aplicaciones móviles. Finalmente, una de las medidas de seguridad más importante es la responsabilidad de los desarrolladores al momento de desarrollar aplicaciones móviles, quienes deberían hacer uso de buenas prácticas que ayuden a minimizar las vulnerabilidades y amenazas asociadas, ya que como señalan Fang *et al.*[4], los desarrolladores son la primera línea de defensa contra los ataques, pero lamentablemente tienen poco conocimiento en temas de seguridad.

Con base en la premisa anterior, se evidencia que debido a la falta de conocimiento y dominio en temas de seguridad por parte de los desarrolladores de aplicaciones móviles hay una falta de mecanismos de seguridad en el código fuente que generan vulnerabilidades en las aplicaciones, por tal razón se decide aportar en el ámbito del desarrollo de aplicaciones móviles con el desarrollo de una guía práctica de programación segura para aplicaciones nativas de Android, que pueda servir de insumo para los nuevos desarrolladores o desarrolladores experimentados en la implementación de buenas prácticas de desarrollo que permitan tener como producto final una aplicación móvil con buenos mecanismos de seguridad para la protección de datos de los usuarios, por ende, se plantea la siguiente pregunta relacionada a la problemática ¿Es posible recopilar una serie de lineamientos relacionados con la programación segura de aplicaciones móviles nativas en Android que le sirva a los desarrolladores como un insumo para tener un enfoque no sólo de desarrollo funcional, si no, también de desarrollo seguro?

[4] FANG, Zhejun, *et al.* Op. cit., p. 2.

3. JUSTIFICACIÓN

En la actualidad Android es uno de los principales sistemas operativos móviles preferidos por las personas, que a inicios del 2017 llegó a alcanzar el 85% de la población de este mercado[5] y que ha sabido seguir manteniendo su liderazgo. Por esta misma razón se puede pensar que se ha convertido en un objetivo de ataque atractivo, ya que hay más posibilidades de robar datos sensibles de los usuarios. A pesar de que desde sus inicios Google ha desarrollado y fortalecido los mecanismos de seguridad en el sistema operativo, siguen existiendo nuevas vulnerabilidades y métodos de explotación que evolucionan para aprovechar las nuevas vulnerabilidades, que muchas veces son propias del diseño de la plataforma y la arquitectura del sistema, pero también, muchas vulnerabilidades están relacionadas con problemas en el código fuente de las aplicaciones y malas prácticas de programación. Como señalan Meng *et al.*[6], "Los exploits de Android son inicializados por programas ejecutables nativos y estos exploits se pueden categorizar en cuatro métodos de ataque, falta de validación de las entradas, reasignación de memoria compartida, restricción de acceso anónimo a la memoria compartida y desbordamiento de número de proceso". Entre los vectores de ataque se encuentran las fallas de validación en los datos de entrada, una acción que es responsabilidad de los desarrolladores de aplicaciones y que puede culminar en una serie de vulnerabilidades que pueden ser explotadas para acceder al sistema, escalar privilegios o modificar la ejecución normal de una aplicación, "si un desarrollador de aplicaciones no valida apropiadamente las entradas provenientes de fuentes externas y no confiables, se puede inyectar código malicioso"[7].

---

[5] IDC. IDC: Smartphone OS Market Share, 2017. Disponible en: https://www.idc.com/promo/smartphone-market-share/os Citado por MENG, Huasong, e*t al.* A survey of Android exploits in the wild. En: Computers & Security. Julio, 2018. Vol. 76, p. 71.

[6] HÖBARTH, S and MAYRHOFER, R. A framework for on-device privilege escalation exploit execution on Android, Citado por MENG, Huasong, e*t al.* A survey of Android exploits in the wild. En: Computers & Security. Julio, 2018. Vol. 76, p. 74.

[7] FANG, Zhejun, *et al.* Op cit., p. 2.

Aunque existen diferentes modelos y marcos de trabajos para detectar amenazas y vulnerabilidades en Android, estas medidas de detección y corrección son validas para el después de haber desarrollado una aplicación, pero también es importante encontrar ayudas que permitan empezar a considerar el código fuente como la primera línea de defensa y que sean los desarrolladores de aplicaciones los responsables de crear aplicaciones móviles seguras, ya que finalmente de nada sirve una plataforma segura sobre la que funcionan aplicaciones vulnerables y más cuando una de las características más llamativas de un sistema operativo es la cantidad y diversidad de aplicaciones móviles que existan para el mismo. Es por esta razón que los mecanismos de seguridad preventivos que funcionen antes y durante el desarrollo de una aplicación móvil son necesarios.

Teniendo en cuenta lo anterior, las aplicaciones móviles requieren que se desarrollen bajo mejores prácticas que permitan garantizar buenos niveles de seguridad y funcionamiento a los usuarios, además de que se encuentra poca documentación relacionada a métodos de programación segura para Android, que consideren el manejo de actividades, permisos, persistencia de datos y aprovechamiento de los recursos de los smartphones, se decide aportar un documento que funcione como guía práctica para los desarrolladores de aplicaciones móviles, ofreciendo una serie de lineamientos basados en documentación, recomendaciones oficiales y otros estudios que ayude a entender la necesidad de prácticas de programación que aporten al desarrollo seguro, a fomentar una nueva cultura en los desarrolladores y finalmente que ayude a construir aplicaciones móviles más seguras.

4. OBJETIVOS

4.1 Objetivo general

Desarrollar una guía práctica de programación segura para aplicaciones nativas de Android

4.2 Objetivos específicos

- Realizar una revisión y contextualización de la arquitectura de Android y sus principales conceptos.
- Realizar una definición de los principales conceptos relacionados con la seguridad en Android.
- Identificar malas prácticas y códigos inseguros en el desarrollo de aplicaciones nativas de Android.
- Diseñar una guía en donde se describa con ejemplos prácticos el desarrollo seguro de aplicaciones nativas de Android.

5. MARCO REFERENCIAL

5.1 Marco teórico

Debido a que el enfoque de este documento es plantear una serie de lineamientos para ayudar a crear aplicaciones móviles nativas seguras de Android, donde se requiere que los desarrolladores de aplicaciones sean más conscientes sobre las necesidades de implementar buenas prácticas de desarrollo y la responsabilidad que ellos tienen para crear aplicaciones seguras, es necesario hacer una revisión de estudios enfocados en la seguridad del sistema operativo Android, donde se hable sobre diferentes mecanismos de seguridad existentes, propuestas que buscan implementar mecanismos de seguridad adicionales, vulnerabilidades existentes, propuestas enfocadas a la identificación de vulnerabilidades, entre otros temas relacionados a la seguridad en general de Android.

La plataforma Android cuenta con diferentes mecanismos que buscan asegurar los dispositivos y proteger la privacidad de los datos, estos mecanismos de seguridad pueden ir desde el nivel de aplicación hasta el nivel de la arquitectura del sistema, por ejemplo, Google Play Protect[8] que es un mecanismo de defensa incorporado en el sistema operativo Android con el objetivo de proteger los dispositivos contra diferentes tipos de malware, adaptándose y mejorando constantemente gracias al respaldo del algoritmo Machine Learning de Google. Entre las principales funciones de Google Play Protect se encuentran el escaneo continuo del dispositivo y actualizaciones de seguridad, escaneo y verificación de aplicaciones, asistencia de navegación segura, entre otras.

Otros mecanismos de seguridad importantes en Android son los permisos que permiten al usuario tener el control sobre los recursos que determinada aplicación requiere acceder, conocido como mecanismo basado en permisos y el Sandbox de aplicaciones que funciona a nivel de Kernel. El Sandbox de aplicaciones permite

[8] Android. Google Play Protect. The most widely deployed mobile threat protection service in the world, [En línea]. Disponible en: https://www.android.com/play-protect/

aislar cada aplicación en su propio proceso, evitando así que las aplicaciones interactúen directamente entre ellas y que requieran previamente de permisos para comunicarse.

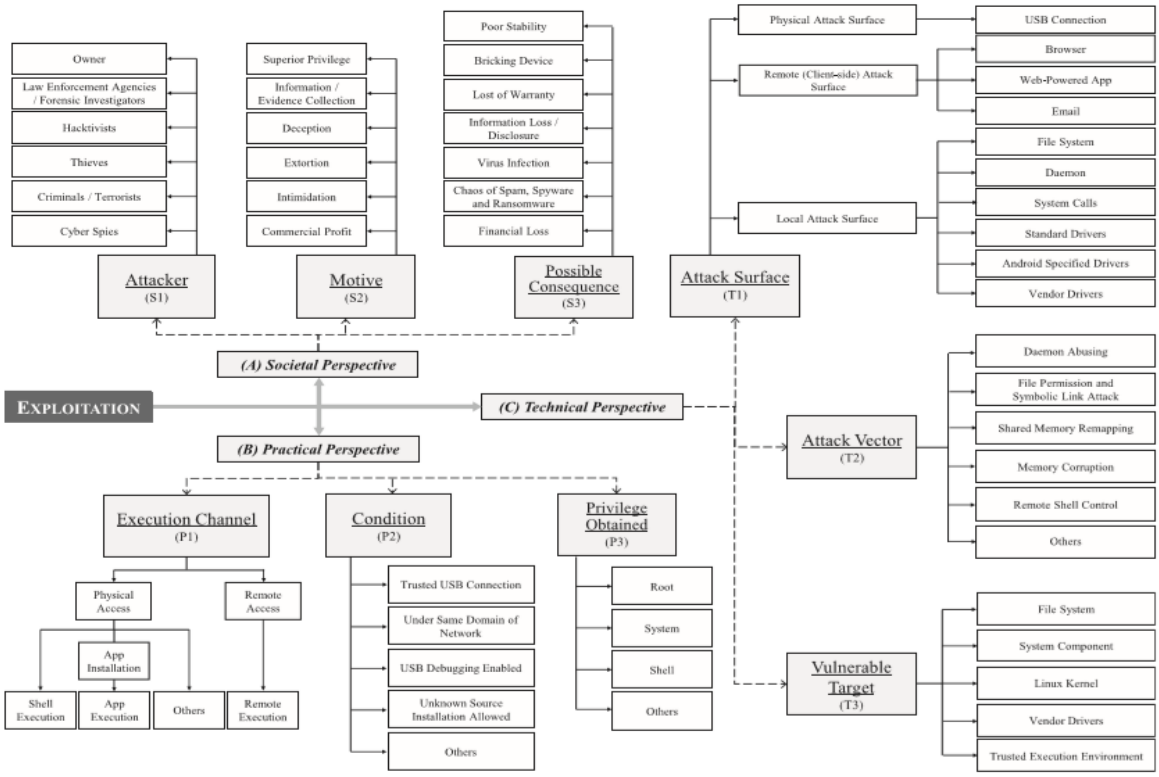
Desde su lanzamiento en 2008, Android ha ido evolucionando y mejorando sus mecanismos de seguridad debido a la existencia de vulnerabilidades y formas de explotar esas vulnerabilidades para escalar privilegios y tener control de los dispositivos Android. Un enfoque interesante de este tema es expuesto por MENG, Huasong, e*t al*. en el artículo A survey of Android exploits in the wild, donde se realiza un análisis de diferentes exploits existentes para Android con el fin de comprenderlos y comparar diferentes cualidades, para finalmente construir la taxonomía de un ataque en Android, dicha taxonomía permite comprender de manera global las diferentes variables que se pueden ver involucradas durante la explotación de una vulnerabilidad en Android considerando 3 perspectivas: social, práctica y técnica, como se puede observar en la Figura 1.

- En la perspectiva social[9] se consideran el atacante, quien es la entidad que realiza la explotación de la vulnerabilidad independiente de si malintencionado o no, los motivos que lo llevan a realizar la explotación de la vulnerabilidad y las posibles consecuencias de sus actos.
- En la perspectiva práctica[10] MENG, Huasong, e*t al*. consideran los canales de ejecución mediante los cuales se lleva a cabo el ataque, las condiciones que deben existir para realizar el ataque y finalmente lo que consideran como privilegio esperado que es el objetivo que tiene la realización de dicho ataque.
- Finalmente, la perspectiva técnica[11] considera tres aspectos que son: la superficie de ataque o componentes sobre los cuales se realiza el ataque, los vectores de ataque o métodos utilizados para realizar el ataque y el

[9] MENG, Huasong, e*t al.* A survey of Android exploits in the wild. En: Computers & Security. Julio, 2018. Vol. 76, p. 74.
[10] Ibid., p. 75.
[11] Ibid., p. 76.

objetivo vulnerable o la fuente donde se encuentra la vulnerabilidad que permitió el ataque.

Figura 1. *Taxonomía de un ataque en Android*



Fuente: *MENG, Huasong, et al. A survey of Android exploits in the wild. En: Computers & Security. Julio, 2018. Vol. 76, p. 75.*

Considerando la figura anterior se puede deducir que la explotación de una vulnerabilidad depende de la combinación de diferentes elementos los cuales son nombrados en las tres perspectivas, y por tal razón existe una gran cantidad de variaciones de vulnerabilidades y exploits, entre una de esas vulnerabilidades están las de tipo validación de entrada y como se nombró anteriormente, es una de las principales vulnerabilidades en Android que ocurren en las aplicaciones cuando los desarrolladores no verifican ni sanitizan los datos de entradas, lo que puede

ocasionar que se inyecten códigos maliciosos, por esta razón, las aplicaciones móviles se pueden considerar como otra superficie de ataque más donde los códigos fuentes son vulnerables. Es esta última premisa la que se desea abordar con la presente propuesta ya que existen otros trabajos que tratan el tema de seguridad amenazas y vulnerabilidad en Android, pero desde otra perspectiva donde no se considera el código fuente como principal línea de defensa

5.2 Marco conceptual

Es necesario entender y conocer los diferentes conceptos que actuaran como base para la construcción de este documento con el fin de tener una mejor comprensión del tema tratado, por esta razón se presenta a continuación una descripción más detallada de dichos temas.

Para hablar sobre la arquitectura de Android es necesario entender que Android está construido sobre el Kernel de Linux, por tal razón, existen muchos componentes similares que han sido adaptados para el contexto del sistema operativo móvil, por ejemplo, entre algunos mecanismos clave de seguridad se encuentran:

- El aislamiento de procesos
- Un modelo de permisos basados en el usuario
- Mecanismos extensibles para asegurar el componente IPC (Inter-process communications) encargado de gestionar la comunicación entre las aplicaciones
- La capacidad de remover partes innecesarias del Kernel y que, además, pueden representar un riesgo.

Como señalan Fang, Zhejun, e*t al*.[12], los mecanismos de seguridad en Android están basados en el Sandbox y los permisos y al igual que MENG, Huasong, e*t al*.[13] estos mecanismos de seguridad son los dos principales mecanismos de seguridad en Android que funcionan a nivel de Kernel y a nivel de aplicación respectivamente, por tal razón es importante saber qué son.

El Sandbox está basado en un mecanismo de privilegios de Linux basado en el usuario en el que el sistema asigna en este caso a las aplicaciones un identificador único de usuario (UID) permitiendo de esta manera que las aplicaciones funcionen bajo su propio proceso.[14] El sandbox de aplicaciones tiene como objetivo aislar las aplicaciones como si fueran procesos independientes haciendo que cuando requieran comunicarse o interactuar entre ellas se verifique si tienen los permisos y privilegios necesarios; de esta manera se busca evitar que aplicaciones maliciosas tengan acceso a recursos del sistema o de otras aplicaciones y debido a que el Sandbox de aplicaciones funciona a nivel del Kernel, todos los niveles superiores de la arquitectura Android donde están las librerías del sistema y diferentes aplicaciones, estén protegidos por este mecanismo.

El mecanismo basado en permisos de Android, también conocido como modelo de permisos de Android, permite que se definan los permisos específicos que requiere una aplicación para acceder a algunos recursos del sistema, debido a que, por defecto, las aplicaciones están aisladas ya que su ejecución está contenida sobre el Sandbox. Existen cuatro niveles de permisos en Android, yendo desde el de menor privilegio hasta el más alto

- *Normal* – Este nivel es para los permisos que han sido declarados en el archivo Manifest de una aplicación móvil, por ejemplo, permiso para acceder a Internet, permiso para vibración[15].

[12] FANG, Zhejun, *et al.* Op. cit., p. 1.
[13] MENG, Huasong, *et al.* Op. cit., p. 73.
[14] Android Open Source Project. Application Sandbox. [En línea]. Disponible en: https://source.android.com/security/app-sandbox
[15] MENG, Huasong, *et al.* Op. cit., p. 73.

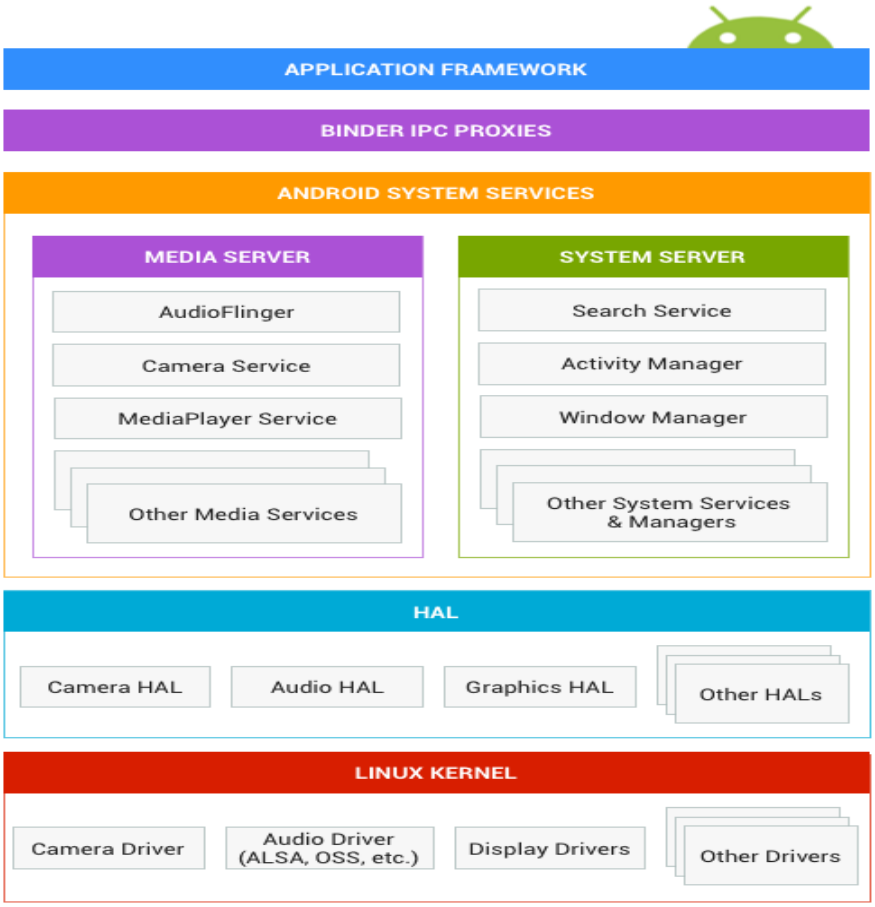
- *Dangerous (Peligroso)* – Este nivel está destinado para los permisos que han sido otorgados por el usuario mientras la aplicación se encontraba en ejecución, por ejemplo, permiso para acceder a las fotos del usuario, permiso para acceder a los contactos del usuario[16].
- *Signature (Firma)* – Este nivel de permiso empieza a representar una categoría más riesgosa en donde se manejan los permisos otorgados a aplicaciones desarrolladas y firmadas por terceros de confianza[17].
- *Signature and System (Firma y Sistema)* – Este nivel está destinado para los permisos con el mayor nivel de privilegio que son otorgados a aplicaciones desarrolladas y firmadas por Google y compañías comercializadoras de celulares[18].

[16] MENG, Huasong, *et al.* Op. cit., p. 73.
[17] MENG, Huasong, *et al.* Op. cit., p. 73.
[18] MENG, Huasong, *et al.* Op. cit., p. 73.

6. ARQUITECTURA DE ANDROID

Antes de tratar el tema de seguridad en Android, es necesario comprender de manera global la arquitectura de Android, para ésto, se hará una revisión de sus componentes.

Figura 2. *Arquitectura del sistema - Android*



Fuente: *Android Open Source Project. Architecture. [En línea]. Disponible en: https://source.android.com/devices/architecture*

Como se puede observar en la Figura 2, la arquitectura del sistema de Android está compuesta por cinco componentes los cuales son:

- **Kernel de Linux –** Este es el componente esencial sobre el cual funciona Android, ya que es la base para muchas de las funcionalidades del sistema, por ejemplo, administración de memoria, administración de energía, manejo de los drivers de Hardware.

- **HAL (Hardware Abstraction layer) –** La capa de abstracción de hardware es un componente que brinda una interfaz estandarizada de módulos para manejar los Dirvers del nivel inferior.

- **Servicios del sistema de Android –** En este componente hay una serie de módulos que permiten que el framework de aplicaciones se comunique con el hardware subyacente a través de los servicios que ofrece el sistema, tales como servicios de búsqueda, administrador de notificaciones.

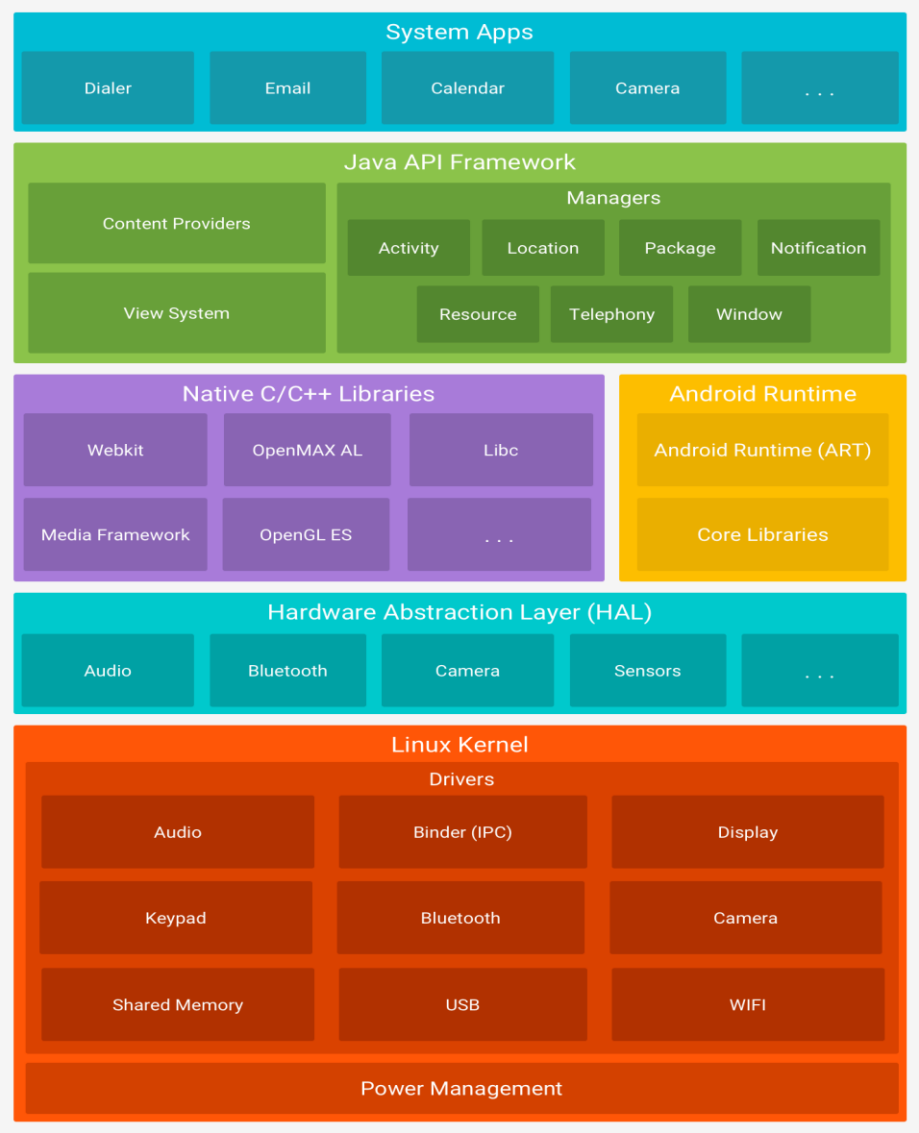
- **Binder IPC –** Es un mecanismo que se encarga de gestionar la comunicación entre las aplicaciones, permitiendo que interactúen las aplicaciones con los servicios del sistema.

- **Framework de aplicación –** En este componente se encuentran definidas diferentes APIs por parte de los desarrolladores que permiten gestionar la comunicación de los componentes subyacentes y los controladores.

Sin embargo, ésta no es la única arquitectura de Android que se puede encontrar ya que existe otro tipo de diagrama relacionado a la arquitectura de Android y que es de interés para el tema tratado en este documento, este diagrama es conocido como la pila de software de Android. Al igual que la arquitectura del sistema de Android, es un diagrama que se maneja por capas y se puede entender como otra

concepción de la arquitectura de Android ya que se tratan de los mismos componentes, pero desde un enfoque diferente.

Figura 3. *Pila de software de Android*



Fuente: *Android Developers. Arquitectura de la plataforma. [En línea]. Disponible en: https://developer.android.com/guide/platform*

En la pila de software se puede observar la similitud entre las primeras dos capas, en Kernel de Linux que es la base para las funcionalidades del sistema operativo y la capa de abstracción de hardware donde se definen interfaces estandarizadas para algunos componentes de Hardware y es a partir de esta capa que se pueden observar unos nuevos componentes en comparación con la arquitectura del sistema. En la misma capa se puede encontrar las librerías nativas de C y C++ y el componente del tiempo de ejecución de Android. Las librerías nativas C/C++ funcionan como una capa intermediaria entre el Kernel y el Framework de aplicación[19], ya que muchos componentes de la plataforma Android están escritos en C o C++ y en ocasiones se requiere acceder a sus funcionalidades desde la capa superior Framework de aplicación. El otro componente que comparte la misma capa que las librerías nativas C/C++ es Android Runtime (ART) o en español, tiempo de ejecución de Android. Este componente permite ejecutar varias máquinas virtuales en formato DEX, que es un formato especial de Android que busca optimizar el uso mínimo de memoria[20], además de otras funcionalidades como compilación, depuración, informes de fallos, entre otros. Una capa más arriba se puede encontrar el Framework de Java API que contiene un conjunto de funciones del sistema operativo escritas en el lenguaje Java para que los desarrolladores puedan hacer uso de ellas y reutilizar diferentes componentes de una manera más fácil ya que se encuentran estandarizadas en el mismo lenguaje con el que se desarrollan las aplicaciones nativas de Android. Finalmente, en la capa superior se encuentran las Apps del sistema, que sencillamente son un conjunto de aplicaciones predefinidas que vienen incluidas en el sistema operativo, pero pueden ser reemplazadas por aplicaciones de terceros, entre estas aplicaciones se encuentran la aplicación de contactos, de mensajes SMS, Calendario, navegador web, cámara, entre otras.

[19] GUNASEKERA, Sheran. Android Security Apps. Berkeley, CA, Apress, 2012. Cap 1, p. 4.
[20] Android Open Source Project. Arquitectura de la plataforma. [En línea]. Disponible en: https://developer.android.com/guide/platform

7. CONCEPTOS DE SEGURIDAD EN ANDROID

Android busca ser un sistema operativo seguro que protege los datos de los usuarios a través de controles de seguridad que protegen los recursos del sistema, por ejemplo, el sandbox de aplicaciones y el sistema de permisos sobre los cuales se habló anteriormente. Sin embargo, los mecanismos de seguridad no son los únicos conceptos relacionados a la seguridad del sistema operativo, a continuación, se discute este tema con más detalle al hablar de los principales conceptos de seguridad en Android.

7.1 Separación de privilegios

La separación de privilegios es un concepto que se relaciona con el Sandbox de aplicaciones, donde se hace uso de identificadores de usuario asignados a cada una de las aplicaciones, haciendo que cada aplicación se ejecute como un proceso separado, de esta manera se garantiza que una aplicación no pueda acceder a los datos de otra y que la única manera de comunicarse entre ellas es que explícitamente la aplicación solicite permiso para acceder a los datos[21].

La separación de privilegios es una buena medida de seguridad que dificulta la realización de un ataque informático, ya que, si un atacante obtiene acceso a uno de los componentes de un sistema buscando llegar a un componente más importante para comprometer el sistema en su totalidad, debe intentar ganar acceso de manera individual en cada uno de los componentes debido a que no todos tienen el mismo nivel de privilegios, lo que genera que el ataque sea más difícil de realizar. Debido a que el Sandbox es una característica del Kernel de Linux y gracias a la arquitectura de Android, el modelo de seguridad de separación de privilegios permite aislar los diferentes componentes que se encuentran sobre la capa del

[21] GUNASEKERA, Sheran. Op. cit., Cap. 1, p.10.

Kernel, por ejemplo, librerías nativas y aplicaciones del sistema operativo. Sin embargo, como recomienda Google[22], el Sandbox de aplicaciones no es totalmente seguro, por lo que se recomienda implementar un modelo de defensa en profundidad para evitar que una correlación de ataques en un vector diferente pueda comprometer la seguridad del sandbox.

7.2 Permisos

El sistema de permisos tiene como objetivo que el usuario final del dispositivo sea el responsable de decidir si autoriza o no a una aplicación para que tenga acceso a algún recurso del sistema. Este mecanismo de seguridad tiene dos actores importantes, el primero son los usuarios a quienes se les otorga la facultad de otorgar acceso a los recursos del sistema según sus necesidades y el segundo actor es el desarrollador quien debe definir de manera explícita los permisos que requiere su aplicación para que funcione correctamente.

7.3 Firma del código de las aplicaciones

Las firmas de las aplicaciones móviles son un mecanismo que busca identificar a los desarrolladores y sus aplicaciones, ya que la única manera de que una aplicación sea ejecutada es que cuente con un certificado firmado por una autoridad o por el mismo desarrollador. La verificación de la firma de la aplicación es realizada en el momento de instalación de la aplicación, si un certificado ha vencido antes de la instalación de una aplicación entonces no se podrá instalar y ejecutar en el dispositivo. Es importante aclarar que estos certificados son obligatorios para las aplicaciones que están en fase de lanzamiento, a pesar de que también existen

[22] Android Open Source Project. Application Sandbox. [En línea]. Disponible en: https://source.android.com/security/app-sandbox

certificados para aplicaciones en fase de depuración, generalmente son certificados creados por el entorno de desarrollo.

7.4 Root

Debido a que el sistema operativo Android está basado en un Kernel de Linux, existen algunos componentes y conceptos de Linux que también están en el sistema operativo móvil, uno de esos conceptos es el súper usuario conocido como root. Algunas funcionalidades del Kernel de Android requieren permisos Root para su ejecución, sin embargo, de manera predeterminada para los usuarios finales no existen una opción para tener permisos root, ya que representaría un riesgo en la estabilidad del sistema al tener control sobre lo que un usuario puede hacer con los diferentes componentes del sistema, lo que aumenta la exposición a fallas de seguridad[23]. A pesar de que tener acceso root representa un peligro, no es algo prohibido, muchos usuarios, por ejemplo, los desarrolladores, logran habilitar el modo root para realizar funciones de depuración o utilizar otras características de los dispositivos como personalización o manipulación de los datos.

[23] Android Open Source Project. Kernel Security. [En línea]. Disponible en: https://source.android.com/security/overview/kernel-security

8. SEGURIDAD EN LAS APLICACIONES

Después de revisar los principales conceptos de la seguridad en Android, es momento de empezar a tratar la seguridad desde la perspectiva de los desarrolladores.

Dentro de las mejores prácticas para desarrollar aplicaciones seguras, existen cuatro aspectos esenciales dentro de las revisiones de seguridad que debe tener en cuenta un desarrollador de aplicaciones:

- Almacenamiento seguro de datos
- Implementar comunicaciones seguras
- Actualización del proveedor de seguridad (Security Provider)
- Prestar atención a los permisos

8.1 Almacenamiento seguro de datos

Con el fin de proteger los datos de los usuarios, los desarrolladores deben tener en consideración la verificación de datos externos y el uso de APIs que accedan a datos sensibles de los usuarios. Se recomienda que se minimice el contacto directo con los datos privados y que estos datos sean manejados a través de hashes y no en texto plano cuando las APIs tengan acceso a los datos sensibles.

El almacenamiento de datos en Android funciona principalmente en dos ambientes, el almacenamiento interno y el almacenamiento externo, de manera predeterminada las apps sólo pueden acceder a los datos del almacenamiento interno y para muchas aplicaciones ésto es suficiente, sólo se debe tener en consideración los permisos que se deben utilizar para el almacenamiento de datos. En caso de que los datos que se deban almacenar sean privados, se pueden guardar en el almacenamiento interno, pero evitando utilizar MODE_WORLD_WRITABLE y

MODE_WORLD_READABLE para limitar el acceso a los datos por parte de otras aplicaciones.

Cuando los datos que se deben almacenar son datos con los cuales interactúa el usuario, por ejemplo, fotos, son los adecuados para guardar en el almacenamiento externo, ya que sólo serán accedidos por el usuario bajo el mismo proceso de la aplicación, por lo que no se requieren muchas medidas de seguridad. El almacenamiento externo tiene una variante cuando los datos se deben compartir con otras aplicaciones, los Content Provider son un mecanismo de almacenamiento que permite otorgar permisos de escritura y lectura a otras aplicaciones o simplemente limitarlas al contexto propio de la aplicación. Los Content Provider se pueden gestionar desde el archivo Manifest con la propiedad *android:exported* y su valor true o false para indicar si se permite que otras aplicaciones accedan a los datos almacenados de la aplicación. Al utilizar Content Provider, Google recomienda hacer uso de métodos predeterminados de acceso al Content Provider que ayudan a evitar las inyecciones SQL, estos métodos de consulta con parámetros con *query()* para retornar datos, *update()* para actualizar datos del Content Provider y *delete()* para eliminar datos del Content Provider. En resumen, los tres aspectos que se deben considerar respecto al almacenamiento datos son el almacenamiento interno, almacenamiento externo y los proveedores de contenido (content providers).

8.2 Implementar comunicaciones seguras

Con el fin de asegurar comunicaciones seguras en el contexto de las aplicaciones, se recomienda que los desarrolladores de aplicaciones hagan uso de la seguridad de la capa de transporte (TLS) para encriptar comunicaciones entre clientes y servidores. Esta capa de transporte permite gestionar las comunicaciones que utilizan un acuerdo cliente servidor de llave privada y llave pública.

Las comunicaciones seguras en Android han sido implementadas para “evitar que los datos de una app sean interceptados por entidades maliciosas a través de conexiones de red”[24], ya que esto representa un algo riesgo para la seguridad de los datos de un usuario que pueden ser privados. Android permite utilizar protocolos seguros para la transmisión de datos privados, por ejemplo, para la implementación de Secure Socket Layer se puede hacer uso de la clase HttpsURLConnection proporcionada por Android. La anterior no es la única medida para implementar comunicaciones cifradas, a nivel de socket Android proporciona clases adicionales como SSLSocket. Debido a que una de las principales vulnerabilidades se relaciona con la entrada de datos, respecto a las comunicaciones seguras Android recomienda no confiar en ningún dato proveniente del protocolo HTTP, por lo que se deben validar todos los datos que provengan de un Intent o un WebView que utilice este protocolo[25].

8.3 Actualización del proveedor de seguridad

Android cuenta con un proveedor de seguridad (Security Provider) que permite garantizar la comunicación segura a través de redes seguras, ya que es importante que los dispositivos estén actualizados constantemente en las librerías que garantizan las comunicaciones seguras, debido a que constantemente resultan nuevas vulnerabilidades y de esta manera se busca evitar que alguna aplicación pueda ser interceptada y robar sus datos.

[24] Android Developers. Seguridad con HTTPS y SSL. [En línea]. Disponible en: https://developer.android.com/training/articles/security-ssl?hl=es-419
[25] Android Developers. Sugerencias de seguridad. [En línea]. Disponible en: https://developer.android.com/training/articles/security-tips?hl=es-419#UserData

8.4 Prestar atención a los permisos

Debido a que el sistema de permisos es uno de los mecanismos de seguridad esenciales en Android, es de vital importancia que los desarrolladores de aplicaciones móviles sepan cuando utilizarlos. Por esta razón existen algunas consideraciones que se deben tener en cuenta para hacer uso de los permisos, ya que los permisos involucran al desarrollador de aplicaciones y al usuario quien es la persona que decide sobre la autorización de los permisos.

- **Hacer uso únicamente de los permisos necesarios –** Los permisos buscan que una aplicación tenga acceso a algunos recursos del sistema, por tal razón, el desarrollador debe considerar los escenarios en los que se debe hacer uso de un permiso, para esta situación es importante considerar que los permisos interrumpen la actividad del usuario para que éste pueda tomar una decisión, esto sucede en las versiones de Android 6.0 en adelante, en versiones anteriores, los permisos se gestionan durante la instalación de la aplicación. Así que es importante saber si de verdad se requiere el permiso o si por otro lado la acción se puede realizar a través de otro método.
- **Tener en cuenta los permisos cuando se utilizan bibliotecas –** Cuando un desarrollador de aplicaciones móviles utiliza una librería en su aplicación, debe tener conocimiento sobre el uso de permisos de la librería, ya que estos permisos serán heredados y el desarrollador debe hacerse cargo de gestionar estos permisos en su propia aplicación.
- **Explicar por qué se utilizan los permisos –** Debido a que son los usuarios finales de los dispositivos los que deben otorgar los permisos, se recomienda que se incluya información sobre la razón que se requieren utilizar los permisos para facilitar la toma decisiones en los usuarios.
- **Ser explícito con el acceso al sistema –** Con el fin de generar confianza en el usuario, es recomendable indicar en qué momentos las aplicaciones tienen acceso a funciones y recursos del sistema, ya que el no avisar, se puede interpretar como un acceso escondido.

Un elemento importante que debe tener toda aplicación y que está estrechamente relacionada con el mecanismo de permisos es el Archivo Manifest. Este archivo tiene como objetivo recolectar información importante sobre la aplicación, que es necesaria para que el sistema operativo pueda ejecutar la App, además de que juega un papel importante en la seguridad de la aplicación, por lo que es necesario que este documento sea comprendido por los desarrolladores. El archivo Manifest puede contener los siguientes campos:

- Nombre de paquete de la aplicación – Este campo permite a la aplicación tener un nombre de paquete que es un identificador único.
- Descripción de los componentes de la aplicación – En este campo se establece los diferentes componentes que requiere la aplicación para su funcionamiento y que son requeridos por parte del sistema operativo para poder ejecutar la aplicación.
- Declaración de permisos – Este campo es de vital importancia, ya que permite establecer los permisos necesarios para solicitar permisos al sistema operativo para acceder a funciones restringidas de las APIs, para comunicarse con otras aplicaciones y los permisos requeridos para que otras aplicaciones pueden acceder sus propios componentes.
- Declaración de API – Este campo establece el valor mínimo de la API que es requerido para que la aplicación pueda funcionar sobre determinada versión del sistema operativo Android.
- Declaración de bibliotecas – Este campo es utilizado para establecer las bibliotecas que requiere la aplicación para funcionar.

El archivo Manifest define sus diferentes valores y atributos a través del lenguaje XML, por lo que es importante que los desarrolladores conozcan la sintaxis utilizada para definir de manera correcta los diferentes elementos y sus atributos que hacen parte del Manifest y hacer que este funcione correctamente.

Los permisos en el archivo Manifest imponen limitaciones “para proteger datos y códigos clave que podrían usarse incorrectamente para distorsionar o afectar la experiencia del usuario”[26]. Android tiene un enfoque particular en cuanto al manejo de los permisos y la interacción con los usuarios, prefiriendo mostrar los diferentes permisos que requiere la aplicación antes de instalarla, ya que se considera que es en este preciso momento que el usuario está atento a revisar con cuidado la información sobre la App y sus funcionalidades, para finalmente decidir si quiere instalar la App en el dispositivo o no. Otra de las razones por las que se decide mostrar esta información importante antes de instalar la aplicación, es que se busca garantizar con certeza que el usuario revise y entienda esta información en lugar de empezarla a ignorar y aceptarla sin comprender, como sucede cuando se muestran este tipo de diálogos durante los tiempos de ejecución.

[26] Android Developers. Manifesto de la App. [En línea]. Disponible en: *https://developer.android.com/guide/topics/manifest/manifest-intro.html*

Figura 4 *Estructura Android Manifest.xml*

```
<?xml version="1.0" encoding="utf-8"?>

<manifest>

    <uses-permission />
    <permission />
    <permission-tree />
    <permission-group />
    <instrumentation />
    <uses-sdk />
    <uses-configuration />
    <uses-feature />
    <supports-screens />
    <compatible-screens />
    <supports-gl-texture />

    <application>

        <activity>
            <intent-filter>
                <action />
                <category />
                <data />
            </intent-filter>
            <meta-data />
        </activity>

        <activity-alias>
            <intent-filter> . . . </intent-filter>
            <meta-data />
        </activity-alias>

        <service>
            <intent-filter> . . . </intent-filter>
            <meta-data/>
        </service>

        <receiver>
            <intent-filter> . . . </intent-filter>
            <meta-data />
        </receiver>

        <provider>
            <grant-uri-permission />
            <meta-data />
            <path-permission />
        </provider>

        <uses-library />

    </application>

</manifest>
```

Fuente. *Android Developers. Manifesto de la App. [En línea]. Disponible en: https://developer.android.com/guide/topics/manifest/manifest-intro.html*

A nivel de criptografía Android proporciona métodos que buscan evitar que los desarrolladores deban implementar sus propios algoritmos de cifrado, ya que a través de la clase *Cipher* permite implementar algoritmos de cifrado existentes como AES y RSA. Además, también proporcionan mecanismos adecuados para la generación de las claves aleatorias para dichos algoritmos, garantizando que no se pierda la fortaleza del algoritmo al generar claves con números aleatorios, haciendo uso de las clases *SecureRandom* y *KeyGenerator*.

A pesar de que son cuatros los aspectos de seguridad en Android, existen medidas adicionales que aportan a la seguridad en las aplicaciones y que apoyan esos cuatro aspectos esenciales, por ejemplo, una de las principales medidas es la validación de datos de entrada, ya que un dato de entrada malicioso puede generar la explotación de un desbordamiento de buffer hasta una inyección SQL que afectaría la disponibilidad, integridad y confidencialidad de la aplicación y sus datos. Se recomienda que se verifique el uso de formatos de datos que cumplan con los datos de entrada esperado, utilizar listas negras para eliminar o reemplazar caracteres especiales y en el caso de utilizar consultas SQL utilizar consultas parametrizadas[27].

Otra de las recomendaciones que deben ser consideradas por parte de los desarrolladores es el manejo de los datos de los usuarios, por ejemplo, en caso de que una aplicación deba acceder o transmitir datos sensibles de usuarios se debe pensar en la manera de evitar que los datos sean expuestos de forma accidental o de que un atacante pueda interceptarlos, por tal razón, los datos sensibles jamás deben ser almacenados en texto plano, por el contrario se recomienda utilizar un hash no reversible de los datos.

[27] Android Developers. Sugerencias de seguridad. [En línea]. Disponible en: https://developer.android.com/training/articles/security-tips?hl=es-419#UserData

De forma general, Android recomienda minimizar el contacto con los datos de los usuarios, sin embargo, hay ocasiones en los que la lógica de la aplicación obliga a procesar dichos datos, en tales casos se deben tener las siguientes consideraciones.

- Implementar políticas de seguridad donde se explique la razón de uso de los datos privados de los usuarios.
- Evitar que componentes externos puedan acceder a datos privados y evitar darles permiso si no se conoce la razón de su solicitud, de esta manera se evita una posible fuga de información.
- Evaluar la necesidad de transmitir datos sensibles hasta un servidor para su procesamiento y verificar si el procesamiento requerido por la aplicación se puede realizar directamente en el cliente.

De igual manera, también se deben tener consideraciones especiales con el manejo de credenciales de usuario, por ejemplo, se recomienda minimizar la frecuencia de solicitud de credenciales de usuario, con el fin de que los ataques de suplantación de identidad tengan menos probabilidades de ser realizados[28]. Como alternativa a solicitar las credenciales, se recomienda utilizar un token de autorización que se pueda actualizar con una periodicidad baja, es decir, un token de corta duración. Otra de las recomendaciones es no almacenar nombres de usuario ni contraseña y utilizar como alternativa un servicio en la nube que se puede invocar con una de las clases que provee Android, AccountManager.

Otra de las medidas adicionales que apoyan el aislamiento y almacenamiento de datos sensibles, es la posibilidad de utilizar algoritmos de cifrado para encriptar los

[28] Android Developers. Sugerencias de seguridad. [En línea]. Disponible en: https://developer.android.com/training/articles/security-tips.html

datos al momento de almacenarlos y para transmitirlos mediante canales seguros. Anteriormente se habló sobre la implementación de comunicaciones seguras en la que se recomienda el uso de HttpsURLConnection o SSLSocket, adicionalmente, Android proporciona la clase Cipher para el uso de algoritmos de cifrado como AES y RSA con 224 o 256 bits para el tamaño de la clave pública.

Finalmente, otro de los mecanismos importantes que un desarrollador debe considerar al momento de realizar comunicación entre procesos es utilizar las funciones del Sistema de Android para realizar dichas comunicaciones, Intents, Binder, Messenger, Service y BroadcastReceiver. Estas interfaces de comunicación sin importantes ya que a nivel de seguridad permiten verificar la identidad de las aplicaciones que utilizan el Inter-Process Communication.

9. MALAS PRÁCTICAS EN EL DESARROLLO DE APLICACIONES

Las recomendaciones de seguridad planteadas por Google buscan cubrir las principales problemáticas de seguridad durante el desarrollo de aplicaciones, estas problemáticas están relacionadas directamente con los datos y su procesamiento, como es nombrado por el mismo Google, el problema de seguridad más común para una aplicación en Android es si otras aplicaciones pueden acceder a los datos que se guardan en el dispositivo[29], sin embargo este problema está relacionado con la configuración de la App, los permisos que maneje, la implementación de criptografía y comunicaciones seguras.

A nivel de aplicación existe una gran problemática con los datos de entrada, una situación que es bastante común en los diferentes lenguajes y diferentes plataformas que dejan vulnerabilidades que pueden ser explotadas a través de ataques tipo Cross-Site Scripting. Android no es una plataforma indiferente a los problemas que se generan por las validaciones de entradas, provee una serie de mecanismos de prevención que no siempre son utilizados por los desarrolladores cuando una aplicación requiere leer datos ingresados por el usuario o provenientes de la red.

Uno de los problemas más comunes derivados de los problemas de validación de entrada es el Buffer Overflow, que consiste en un comportamiento anormal cuando una aplicación sobrepasa los límites de memoria para escribir datos en otras ubicaciones de la memoria, llegando incluso a eliminar datos. Aunque Android tiene métodos que ayudan a mitigar el riesgo de explotación de dichas vulnerabilidades, la primera línea de defensa sigue siendo un problema, es decir, el código de los desarrolladores, ya que no realizan las validaciones necesarias, ni utilizan las mejores prácticas de programación recomendadas. La validación de datos de entrada no sólo sucede en el código nativo utilizado para las aplicaciones, también

[29] Android Developers. Sugerencias de seguridad. [En línea]. Disponible en: https://developer.android.com/training/articles/security-tips?hl=es-419#UserData

existe en ámbitos en los que se utilizan otros lenguajes para el procesamiento de datos, por ejemplo, JavaScript y SQL, por lo que se recomienda para todos los casos en los que se lean o reciban datos, verificar que los datos de entrada correspondan a los parámetros esperados.

A continuación, se listarán una serie de ejemplos de malas práctica de programación que se presentan comúnmente y que pueden afectar la integridad, confidencialidad o disponibilidad de los datos de una aplicación móvil, haciendo que ésta se considere insegura.

En el siguiente ejemplo, se puede observar una porción de código donde no se verifica el origen del Intent, por lo que una aplicación desconocida podría intentar comunicarse con esta aplicación para ejecutar la eliminación de un usuario. Se recomienda crear listas blancas de aplicaciones de confianza que puede intentar comunicarse a través de un Intent.

Figura 5. *Falta de validación del origen de la comunicación*

```
IntentFilter filter = new IntentFilter("com.example.RemoveUser");
MyReceiver receiver = new MyReceiver();
registerReceiver(receiver, filter);

public class DeleteReceiver extends BroadcastReceiver {
  @Override
  public void onReceive(Context context, Intent intent) {
    int userID = intent.getIntExtra("userID");
    destroyUserData(userID);
  }
}
```

Fuente. *Common Weakness Enumeration. CWE-346: Origin Validation Error. [En línea]. Disponible en: https://cwe.mitre.org/data/definitions/346.html*

Otra vulnerabilidad presente en Android y que se relaciona con los Intents, es enviar información sensible a través de éstos, además de utilizar Intents en los que no se especifica qué aplicaciones tienen permiso para recibir el Intent con datos sensibles. En el siguiente ejemplo, se observa una porción de código en la que se utiliza un Intent que puede ser escuchado por diferentes aplicaciones de confianza para crear una cuenta de usuario, sin embargo, siempre se debe validar las aplicaciones de confianza que pueden escuchar el Intent, para ésto, se recomienda utilizar una lista blanca de aplicaciones de confianza, de lo contrario una atacante podría interceptar la comunicación haciendo uso de un IntentFilter.

Figura 6. *Intent con información sensible y sin validar receptores*

```
Intent intent = new Intent();
intent.setAction("com.example.CreateUser");
intent.putExtra("Username", uname_string);
intent.putExtra("Password", pw_string);
sendBroadcast(intent);
```

Fuente. *Common Weakness Enumeration. CWE-927: Use of implicit Intent for Sensitive Communication. [En línea]. Disponible en: https://cwe.mitre.org/data/definitions/927.html*

Otra de los principales errores que cometen los desarrolladores es guardar información sensible en el código como texto plano, por lo que un atacante que llegue a ganar acceso podría leerlas desde el código fuente. Este problema se presenta principalmente con credenciales de usuario, tokens de sesión o variables en archivos de configuración. La recomendación es siempre guardar la información sensible de manera cifrada, definir variables de entorno administrativas o recuperarlas desde una bóveda de datos.

Figura 7. *Cookie de usuario en texto plano*

```
response.addCookie( new Cookie("userAccountID", acctID);
```

Fuente. *Common Weakness Eumeration. CWE-312: Cleartext Storage of Sensitive Information. [En línea]. Disponible en: https://cwe.mitre.org/data/definitions/312.html*

Figura 8. *Archivo de configuración con información en texto plano*

```
# Java Web App ResourceBundle properties file
...
webapp.ldap.username=secretUsername
webapp.ldap.password=secretPassword
...
```

Fuente. *Common Weakness Eumeration. CWE-312: Cleartext Storage of Sensitive Information. [En línea]. Disponible en: https://cwe.mitre.org/data/definitions/312.html*

Otra de las principales vulnerabilidades relacionadas con la falta de validación de datos, son las inyecciones SQL. Este tipo de vulnerabilidades se relacionan con datos maliciosos que no son validados y pueden alterar la sentencia SQL para modificar o mostrar datos sin autorización. En el siguiente ejemplo se muestra la autenticación en un sistema utilizando el nombre de usuario y la contraseña como parámetros anexados a la consulta, además de que no se realiza validación de datos sobre estos parámetros, lo que genera que sean vulnerables a inyecciones SQL.

***Figura 9**. Código vulnerable a inyección SQL*

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

class Login {
  public Connection getConnection() throws SQLException {
    DriverManager.registerDriver(new
            com.microsoft.sqlserver.jdbc.SQLServerDriver());
    String dbConnection =
      PropertyManager.getProperty("db.connection");
    // Can hold some value like
    // "jdbc:microsoft:sqlserver://<HOST>:1433,<UID>,<PWD>"
    return DriverManager.getConnection(dbConnection);
  }

  String hashPassword(char[] password) {
    // Create hash of password
  }

  public void doPrivilegedAction(String username, char[] password)
                                 throws SQLException {
    Connection connection = getConnection();
    if (connection == null) {
      // Handle error
    }
    try {
      String pwd = hashPassword(password);

      String sqlString = "SELECT * FROM db_user WHERE username = '"
                         + username +
                         "' AND password = '" + pwd + "'";
      Statement stmt = connection.createStatement();
      ResultSet rs = stmt.executeQuery(sqlString);

      if (!rs.next()) {
        throw new SecurityException(
          "User name or password incorrect"
        );
      }

      // Authenticated; proceed
    } finally {
      try {
        connection.close();
      } catch (SQLException x) {
        // Forward to handler
      }
    }
  }
}
```

***Fuente.** Carnegie Mellon University. SEI CERT Oracle Coding Standard for Java. Prevent SQL Injection [En línea]. Disponble en: https://wiki.sei.cmu.edu/confluence/display/java/IDS00-J.+Prevent+SQL+injection*

10. RECOMENDACIONES DE PROGRAMACIÓN SEGURA

Con base en las principales problemáticas en el desarrollo de aplicaciones móviles en Android, se propone una serie de guías que abarquen las mejores prácticas a nivel de protección de datos, considerando el almacenamiento, la comunicación y los permisos para que los datos puedan ser accedidos por otras aplicaciones. A continuación, se presentan las recomendaciones acompañadas del código que permite implementarlas

10.1 Asegurar las comunicaciones

10.1.1 Uso de Intents

Cuando una aplicación requiere ejecutar una acción y esa acción puede ser ejecutada por dos aplicaciones diferentes a través de Intent, se debe permitir al usuario realizar una elección sobre la aplicación que confía para ejecutar la acción.

Figura 10. *Uso de Intents*

```
Intent intent = new Intent(Intent.ACTION_SEND);
List<ResolveInfo> possibleActivitiesList =
        queryIntentActivities(intent, PackageManager.MATCH_ALL);

// Verify that an activity in at least two apps on the user's device
// can handle the intent. Otherwise, start the intent only if an app
// on the user's device can handle the intent.
if (possibleActivitiesList.size() > 1) {

    // Create intent to show chooser.
    // Title is something similar to "Share this photo with".

    String title = getResources().getString(R.string.chooser_title);
    Intent chooser = Intent.createChooser(intent, title);
    startActivity(chooser);
} else if (intent.resolveActivity(getPackageManager()) != null) {
    startActivity(intent);
}
```

Fuente. *Android Developers. App Security Best Practices. [En línea]. Disponible en: https://developer.android.com/topic/security/best-practices?hl=es-419*

10.1.2 Aplicar permisos basados en firmas

Este tipo de permisos se utilizan cuando se comparten datos entre dos aplicaciones propias de confianza, por lo que no se requiere que el usuario autorice conceder el permiso para que las apps interactúen, debido a que ambas aplicaciones están firmadas con la misma llave.

Figura 11. *Uso de permisos basados en firmas*

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.myapp">
    <permission android:name="my_custom_permission_name"
                android:protectionLevel="signature" />
```

Fuente. *Android Developers. App Security Best Practices. [En línea]. Disponible en: https://developer.android.com/topic/security/best-practices?hl=es-419*

10.1.3 Desactivar el acceso al Content Provider

Cuando se requiere compartir datos entre con una aplicación de otro desarrollar, se debe deshabilitar el acceso al Content Provider para evitar que aplicaciones de terceros puedan acceder a los datos con permisos de escritura y lectura. Por defecto el atributo android:exported está en true, permitiendo que el Content Provider esté activo.

Figura 12. *Desactivación del Content Provider*

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.myapp">
    <application ... >
        <provider
            android:name="android.support.v4.content.FileProvider"
            android:authorities="com.example.myapp.fileprovider"
            ...
            android:exported="false">
            <!-- Place child elements of <provider> here. -->
        </provider>
        ...
    </application>
</manifest>
```

Fuente. *Android Developers. App Security Best Practices. [En línea]. Disponible en: https://developer.android.com/topic/security/best-practices?hl=es-419*

10.1.4 Implementación de SSL

A continuación, se explica cómo implementar una conexión SSL con un sitio de confianza que tiene un certificado firmado por un CA. Android apoya bastante la implementación de protocolos seguros para transmitir datos en la red.

Figura 13. *Implementación conexión SSL*

```
URL url = new URL("https://www.google.com");
HttpsURLConnection urlConnection = (HttpsURLConnection) url.openConnection();
urlConnection.connect();
InputStream in = urlConnection.getInputStream();
```

Fuente. *Android Developers. App Security Best Practices. [En línea]. Disponible en: https://developer.android.com/topic/security/best-practices?hl=es-419*

10.1.5 Configuración de seguridad de red

En el caso de que una aplicación CA (Certificate Authority) personalizadas, se puede crear un archivo para declarar las configuraciones de seguridad de la red sin modificar el código de la aplicación. Para realizar este proceso, se debe declarar la propiedad *android:networkSecurityConfig* en el archivo Manifest:

Figura 14*. Declaración del archivo de seguridad de red*

```
<manifest ... >
    <application
        android:networkSecurityConfig="@xml/network_security_config"
        ... >
        <!-- Place child elements of <application> element here. -->
    </application>
</manifest>
```

Fuente. *Android Developers. App Security Best Practices. [En línea]. Disponible en: https://developer.android.com/topic/security/best-practices?hl=es-419*

Posteriormente se debe crear el recurso que se especificó en un archivo XML, en el caso de la Figura 9, *network_security_config* y se establece el valor de la propiedad

cleartextTrafficPermitted en *false*, con el objetivo de establecer que el tráfico de red utilice el protocolo HTTPS.

Figura 15. *Especificación de uso de HTTPS*

```
<network-security-config>
    <domain-config cleartextTrafficPermitted="false">
        <domain includeSubdomains="true">secure.example.com</domain>
        ...
    </domain-config>
</network-security-config>
```

Fuente. *Android Developers. App Security Best Practices. [En línea]. Disponible en: https://developer.android.com/topic/security/best-practices?hl=es-419*

10.2 Establecer los permisos adecuados

10.2.1 Uso de Intents para evitar el manejo de permisos

Cuando se requiere que una aplicación interactúe con otra y se conoce que existe otra aplicación con los permisos necesarios para interactuar con la aplicación que se necesita, en lugar de agregar un nuevo permiso a la primera aplicación, se recomienda implementar un Intent hacia la aplicación con los permisos necesarios y que sea ésta la que se encargue de manejar la petición.

Figura 16. *Delegación de permisos*

```
// Delegates the responsibility of creating the contact to a contacts app,
// which has already been granted the appropriate WRITE_CONTACTS permission.
Intent insertContactIntent = new Intent(Intent.ACTION_INSERT);
insertContactIntent.setType(ContactsContract.Contacts.CONTENT_TYPE);

// Make sure that the user has a contacts app installed on their device.
if (insertContactIntent.resolveActivity(getPackageManager()) != null) {
    startActivity(insertContactIntent);
}
```

Fuente. *Android Developers. App Security Best Practices. [En línea]. Disponible en: https://developer.android.com/topic/security/best-practices?hl=es-419*

10.2.2 Compartir datos a través de aplicaciones

Al momento de compartir datos con otras aplicaciones, se recomienda seguir los siguientes lineamientos por parte de Google: Implementar permisos de sólo lectura (read-only) o sólo escritura (write-only) únicamente cuando sea necesario. Se puede dar acceso a los datos únicamente una vez utilizando los parámetros:

- FLAG_GRANT_READ_URI_PERMISSION
- FLAG_GRANT_WRITE_URI_PERMISSION

Figura 17. *Compartiendo el contenido de las Apps*

```
// Create an Intent to launch a PDF viewer for a file owned by this app.
Intent viewPdfIntent = new Intent(Intent.ACTION_VIEW);
viewPdfIntent.setData(Uri.parse("content://com.example/personal-info.pdf"));

// This flag gives the started app read access to the file.
viewPdfIntent.addFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION);

// Make sure that the user has a PDF viewer app installed on their device.
if (viewPdfIntent.resolveActivity(getPackageManager()) != null) {
    startActivity(viewPdfIntent);
}
```

Fuente. *Android Developers. App Security Best Practices. [En línea]. Disponible en: https://developer.android.com/topic/security/best-practices?hl=es-419*

10.3 Almacenamiento seguro de datos

10.3.1 Datos privados en el almacenamiento interno

Debido a que algunas aplicaciones requieren almacenar información sensible de los usuarios, se debe garantizar que se almacenan de forma segura. La información más sensible y confidencial debe ser guardada en el almacenamiento interno, ya que se aplica el concepto del Sandbox, cada aplicación por defecto puede acceder a sus datos creados en el almacenamiento interno, es decir, están aislados de otras aplicaciones.

Figura 18. *Almacenamiento de datos en la memoria interna*

```
// Creates a file with this name, or replaces an existing file
// that has the same name. Note that the file name cannot contain
// path separators.
final String FILE_NAME = "sensitive_info.txt";
String fileContents = "This is some top-secret information!";

FileOutputStream fos = openFileOutput(FILE_NAME, Context.MODE_PRIVATE);
fos.write(fileContents.getBytes());
fos.close();
```

Fuente. *Android Developers. App Security Best Practices. [En línea]. Disponible en: https://developer.android.com/topic/security/best-practices?hl=es-419*

10.3.2 Lectura de datos desde el almacenamiento interno

No sólo se debe considerar la manera adecuada de guardar los datos en el almacenamiento interno, también es necesario conocer la mejor manera de recuperar esos datos que están almacenados.

Figura 19. *Lectura de datos desde el almacenamiento interno*

```
// The file name cannot contain path separators.
final String FILE_NAME = "sensitive_info.txt";
FileInputStream fis = openFileInput(FILE_NAME);

// available() determines the approximate number of bytes that can be
// read without blocking.
int bytesAvailable = fis.available();
StringBuilder topSecretFileContents = new StringBuilder(bytesAvailable);

// Make sure that read() returns a number of bytes that is equal to the
// file's size.
byte[] fileBuffer = new byte[bytesAvailable];
while (fis.read(fileBuffer) != -1) {
    topSecretFileContents.append(fileBuffer);
}
```

Fuente. *Android Developers. App Security Best Practices. [En línea]. Disponible en: https://developer.android.com/topic/security/best-practices?hl=es-419*

10.3.3 Uso del almacenamiento externo

Debido a que el almacenamiento externo no tiene restricciones relacionadas al acceso de los datos, además de que no existe garantía de que el dispositivo de almacenamiento externo siempre esté conectado al dispositivo Android, se debe realizar validación de los datos para que no se almacenen como datos corruptos e inestables.

Figura 20. *Definición de permisos para el almacenamiento externo*

```
<manifest ... >
    <!-- Apps on devices running Android 4.4 (API level 19) or higher cannot
         access external storage outside their own "sandboxed" directory, so
         the READ_EXTERNAL_STORAGE (and WRITE_EXTERNAL_STORAGE) permissions
         aren't necessary. -->
    <uses-permission
          android:name="android.permission.READ_EXTERNAL_STORAGE"
          android:maxSdkVersion="18" />
    ...
</manifest>
```

Fuente. *Android Developers. App Security Best Practices. [En línea]. Disponible en: https://developer.android.com/topic/security/best-practices?hl=es-419*

Figura 21. *Validación de los datos que se guardan en el almacenamiento externo*

```
File ringtone = new File(getExternalFilesDir(DIRECTORY_RINGTONES,
        "my_awesome_new_ringtone.m4a"));
if (isExternalStorageEmulated(ringtone)) {
    Logger.e(TAG, "External storage is not present");
} else if (getExternalStorageState(ringtone) == MEDIA_REMOVED
        | MEDIA_UNMOUNTED | MEDIA_BAD_REMOVAL | MEDIA_UNMOUNTABLE) {
    Logger.e(TAG, "External storage is not available");
} else {
    FileInputStream fis = new FileInputStream(ringtone);

    // available() determines the approximate number of bytes that
    // can be read without blocking.
    int bytesAvailable = fis.available();
    StringBuilder fileContents = new StringBuilder(bytesAvailable);
    byte[] fileBuffer = new byte[bytesAvailable];
    while (fis.read(fileBuffer) != -1) {
        fileContents.append(fileBuffer);
    }

    // Implement appropriate logic for checking a file's validity.
    checkFileValidity(fileContents);
}
```

Fuente. *Android Developers. App Security Best Practices. [En línea]. Disponible en: https://developer.android.com/topic/security/best-practices?hl=es-419*

10.3.4 Almacenamiento de datos no sensibles en Caché

Android permite utilizar métodos para acceder rápidamente a información, pero dicha información no puede ser privada, por tal razón se recomienda que algunos datos que sean constantemente usados por las aplicaciones sean almacenados en Caché, considerando que no sean datos privados del usuario.

Figura 22. *Almacenamiento de datos no sensibles en caché*

```
File cacheDir = getCacheDir();
File fileToCache = new File(myDownloadedFileUri);
String fileToCacheName = fileToCache.getName();
File cacheFile = new File(cacheDir.getPath(), fileToCacheName);
```

Fuente. *Android Developers. App Security Best Practices. [En línea]. Disponible en: https://developer.android.com/topic/security/best-practices?hl=es-419*

10.4 Criptografía

Debido a que Android provee algoritmos de cifrado que permiten proteger los datos y asegurar las comunicaciones, a continuación, se muestran algunos ejemplos de cómo hacer uso de estas funcionalidades para trabajar con los datos de manera más segura.

10.4.1 Lectura y escritura de archivos

Android permite utilizar la clase *EncryptedFile* para realizar procesos de escritura y lectura de datos de manera más segura, ya que utiliza una librería creada específicamente para el cifrado de datos que requieren ser transmitidos.

Figura 23. *Lectura de datos con EncryptedFile*

```
val fileToRead = "my_sensitive_data.txt"
lateinit var byteStream: ByteArrayOutputStream
val encryptedFile = EncryptedFile.Builder(
    File(context.getFilesDir(), fileToRead),
    context,
    masterKeyAlias,
    EncryptedFile.FileEncryptionScheme.AES256_GCM_HKDF_4KB
).build()

try {
    encryptedFile.openFileInput().use { fileInputStream ->
        try {
            byteStream = ByteArrayOutputStream()
            var nextByte = fileInputStream.read()
            while (nextByte != -1) {
                byteStream.write(nextByte)
                nextByte = fileInputStream.read()
            }

            val fileContents = byteStream.toByteArray()

        } catch (ex: Exception) {
            // Error occurred opening raw file for reading.
        } finally {
            fileInputStream.close()
        }
    })
} catch (ex: IOException) {
    // Error occurred opening encrypted file for reading.
}
```

Fuente. *Android Developers. Work with data more securely. [En línea]. Disponible en: https://developer.android.com/topic/security/data*

Figura 24. *Escritura de datos con EncryptedFile*

```
val fileToWrite = "my_other_sensitive_data.txt"
val encryptedFile = EncryptedFile.Builder(
    File(context.getFilesDir(), fileToWrite),
    context,
    masterKeyAlias,
    EncryptedFile.FileEncryptionScheme.AES256_GCM_HKDF_4KB
).build()

// Write to a file.
try {
    val outputStream: FileOutputStream? = encryptedFile.openFileOutput()
    outputStream?.apply {
        write("MY SUPER SECRET INFORMATION"
            .toByteArray(Charset.forName("UTF-8")))
        flush()
        close()
    }
} catch (ex: IOException) {
    // Error occurred opening file for writing.
}
```

Fuente. *Android Developers. Work with data more securely. [En línea]. Disponible en: https://developer.android.com/topic/security/data*

10.5 Configuraciones de seguridad de la red

Este archivo de configuración permite personalizar los ajustes de seguridad de red de una aplicación mediante declaraciones específicas.

10.5.1 Creación del archivo de seguridad de red

Este archivo de configuración se crea como un archivo XML en el que se realizarán los ajustes de seguridad necesarios para la aplicación, que pueden ir desde elección de CA (autoridades de certificación) de confianza, protección del tráfico en texto plano, control de los certificados permitidos para realizar conexiones con la app, entre otros. Para crear el archivo, se hace uso del Manifest en el cuál se declarará el archivo de configuración de seguridad de red.

Figura 25. *Declarando el archivo de configuración de seguridad de red*

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
    <application android:networkSecurityConfig="@xml/network_security_config"
                    ... >
        ...
    </application>
</manifest>
```

Fuente. *Android Developers. Configuración de seguridad de la red. [En línea]. Disponible en: https://developer.android.com/training/articles/security-config*

10.5.2 Desactivación del tráfico Cleartext

Esta es una medida adiciona útil cuando se quiere garantizar que las conexiones de la aplicación sean sólo mediante HTTPS, de esta manera se evita redirecciones accidentales que permitan utilizar la transmisión de datos en texto plano. El siguiente ejemplo demuestra la manera de desactivar la transmisión de datos en texto plano para un determinado dominio haciendo uso del archivo de seguridad de red.

Figura 26. *Desactivando el tráfico Cleartext*

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <domain-config cleartextTrafficPermitted="false">
        <domain includeSubdomains="true">secure.example.com</domain>
    </domain-config>
</network-security-config>
```

Fuente. *Android Developers. Configuración de seguridad de la red. [En línea]. Disponible en: https://developer.android.com/training/articles/security-config*

10.6 Actualización del proveedor de seguridad

El objeto *Provider* de Android ayuda a mitigar las vulnerabilidades relacionadas con las comunicaciones de red ofreciendo un mecanismo de protección que es

necesario actualizar constantemente mediante los servicios de Google Play debido a las nuevas vulnerabilidades que constantemente aparecen.

10.6.1 Actualización sincrónica del proveedor de seguridad

La manera más sencilla de actualizar el proveedor de seguridad es haciendo uso de la clase *ProviderInstaller* y su método *InstallIfNeeded()*

Figura 27. *Actualización del proveedor de seguridad*

```
/**
 * Sample sync adapter using {@link ProviderInstaller}.
 */
public class SyncAdapter extends AbstractThreadedSyncAdapter {

  ...

  // This is called each time a sync is attempted; this is okay, since the
  // overhead is negligible if the security provider is up-to-date.
  @Override
  public void onPerformSync(Account account, Bundle extras, String authority,
      ContentProviderClient provider, SyncResult syncResult) {
    try {
      ProviderInstaller.installIfNeeded(getContext());
    } catch (GooglePlayServicesRepairableException e) {

      // Indicates that Google Play services is out of date, disabled, etc.

      // Prompt the user to install/update/enable Google Play services.
      GoogleApiAvailability.getInstance()
              .showErrorNotification(context, e.connectionStatusCode)

      // Notify the SyncManager that a soft error occurred.
      syncResult.stats.numIoExceptions++;
      return;

    } catch (GooglePlayServicesNotAvailableException e) {
      // Indicates a non-recoverable error; the ProviderInstaller is not able
      // to install an up-to-date Provider.

      // Notify the SyncManager that a hard error occurred.
      syncResult.stats.numAuthExceptions++;
      return;
    }

    // If this is reached, you know that the provider was already up-to-date,
    // or was successfully updated.
  }
}
```

Fuente. *Android Developers. Actualiza tu proveedor de seguridad para protegerte contra vulnerbilidades SSL. [En línea]. Disponible en: https://developer.android.com/training/articles/security-gms-provider*

10.7 Validación de datos

Una de las principales vulnerabilidades en las aplicaciones móviles y general en el software es la falta de validación de datos de entrada, lo que da pie a que las fuentes poco confiables puedan utilizar estos vectores de entrada para ingresar datos maliciosos que alteren el comportamiento de la implicación e incluso lleguen a comprometer su funcionamiento y por lo tanto sus datos. Siempre se recomienda como buena práctica de programación, desconfiar de los datos de entrada de los usuarios y realizar validación sobre éstos con el fin de garantizar que los datos de entrada corresponden a los datos esperados por la aplicación.

10.7.1 Prevención contra inyecciones SQL

El uso de sentencias preparadas ayuda a mitigar el impacto de las inyecciones SQL, ya que ayuda a validar el argumento de la sentencia SQL, evitando que se puedan inyectar códigos arbitrarios que podrían extraer datos. A continuación, un ejemplo de cómo utilizar adecuadamente la técnica *Prepared Statements*, en la que se hace uso del signo de interrogación (*?*) como carácter especial para indicar el argumento que será pasado en la sentencia SQL.

Figura 28. *Uso de Prepared Statements*

```
public void doPrivilegedAction(
  String username, char[] password
) throws SQLException {
  Connection connection = getConnection();
  if (connection == null) {
    // Handle error
  }
  try {
    String pwd = hashPassword(password);

    // Validate username length
    if (username.length() > 8) {
      // Handle error
    }

    String sqlString =
      "select * from db_user where username=? and password=?";
    PreparedStatement stmt = connection.prepareStatement(sqlString);
    stmt.setString(1, username);
    stmt.setString(2, pwd);
    ResultSet rs = stmt.executeQuery();
    if (!rs.next()) {
      throw new SecurityException("User name or password incorrect");
    }

    // Authenticated; proceed
  } finally {
    try {
      connection.close();
    } catch (SQLException x) {
      // Forward to handler
    }
  }
}
```

Fuente. *Carnegie Mellon University. SEI CERT Oracle Coding Standard for Java. Prevent SQL Injection [En línea]. Disponble en: https://wiki.sei.cmu.edu/confluence/display/java/IDS00-J.+Prevent+SQL+injection*

11. CONCLUSIONES

Los desarrolladores deben enfocarse no sólo en aprender a desarrollar de manera funcional, deben comprender conceptos de seguridad y articularlos a las metodologías de desarrollo. Se espera que esta guía aporte conocimiento en las etapas de análisis y diseño de una aplicación, para que lleguen menos vulnerabilidades a las etapas de prueba, por esta razón se ha realizado una revisión y contextualización de la arquitectura del sistema operativo Android que permitirá a los desarrolladores tener una mejor comprensión de los diferentes módulos que lo conforman.

Partiendo de la necesidad de conocer los componentes del sistema operativo y su funcionamiento, se desprende de igual manera la necesidad de contextualizarse sobre los principales conceptos asociados a la seguridad de Android, estos conceptos se relacionan con los principales cuatro componentes de la seguridad que deben estar presente en la seguridad de Android: Almacenamiento seguro de datos, comunicaciones seguras, actualización del proveedor de seguridad y los permisos, de los cuales se empiezan a desprender otros conceptos como el sandbox de aplicaciones, el manejo de permisos desde archivo de configuración, la implementación de cifrado y protocolos como HTTPS y SSL.

Ya que el objetivo principal del artículo es recopilar información relacionada a buenas prácticas de programación también se realiza una revisión de ejemplos de código vulnerable que generan huecos de seguridad que atentan contras los conceptos principales de seguridad recomendados por Android, por ejemplo, falta de validación de datos, que deriva en inyecciones de código malicioso, archivos de configuración con información sensible en texto plano.

Finalmente, para concluir con el objetivo de este trabajo, se realiza una recopilación de las mejores prácticas recomendadas por Google donde se ofrece ejemplos sobre la manera correcta de gestionar los permisos, de asegurar las comunicaciones, almacenar los datos, entre otros. De esta manera al realizar un recorrido teórico sobre los principales conceptos de Android para complementar con ejemplos prácticas de código vulnerable y código seguro, se dispone de este documento como una guía de programación segura que puede ser utilizada por desarrolladores de aplicaciones móviles y ayudará a concientizar sobre la necesidad de desarrollar con un enfoque seguro.

A manera de conclusiones más personales sobre el tema en general, se puede observar que Google se ha preocupado por establecer una serie de lineamientos y recomendaciones de seguridad donde especifican claramente cómo se deben manejar determinadas situaciones, por ejemplo, el almacenamiento de datos, el manejo de permisos, la transmisión de datos, sin embargo, las vulnerabilidades en etapa de desarrollo siguen existiendo, lo que demuestran que muchos desarrolladores no han profundizado en temas de seguridad y no conocen de manera adecuada la arquitectura del sistema para sacarle buen provecho.

Ya que se requiere que el desarrollador tenga conocimientos en general de seguridad y de los diferentes componentes y mecanismos que ofrece Android para articularlos y ofrecer aplicaciones seguras, por tal razón, sería ideal que en los cursos de desarrollo de aplicaciones móviles se incluya un componente enfocado a la seguridad de los datos.

La arquitectura de Android es bastante robusta y continúa en crecimiento y evolución, las recomendaciones anteriormente descritas no son la única alternativa

para abordar una situación, debido a que el contexto de las aplicaciones puede ser muy variable, por lo que depende de la experiencia del desarrollador conocer cuál es la mejor manera asegurar los datos.

Se espera que este tema pueda seguir siendo desarrollado a profundidad y tratando nuevos conceptos que se implementan con las nuevas versiones de sistemas operativos, además, es necesario comprender este tema no sólo desde el lenguaje de programación Java, también se debe considerar un lenguaje que ha sido fuertemente aceptado para el desarrollo nativo, Kotlin. Finalmente, el tema de seguridad no sólo se debe limitar a las aplicaciones nativas, también las aplicaciones híbridas deberían ofrecer unos lineamientos de programación segura para proteger los datos de los usuarios.

Finalmente, para dar respuesta a la pregunta planteada en la definición del problema, se considera que esta guía aunque limitada, considerando los diferentes lenguajes existentes y las diferentes tecnologías utilizadas para el desarrollo de aplicaciones móviles, trata de manera global las mejores prácticas que aportan a la protección de los datos de usuarios en cuanto a su almacenamiento, transporte y permisos de acceso, por lo que, si alguien que realiza una revisión de esta guía, comprenderá que aunque hay muchas alternativas para realizar una acción, hay principios que se deben cumplir, por ejemplo, el manejo de permisos, el cifrado de datos, la validación de datos de entrada, entre otros, que finalmente hacen que un desarrollador añada a su enfoque de desarrollo una mentalidad en pro de la protección de los datos y la seguridad de sus aplicaciones.

BIBLIOGRAFÍA

Android Developers. Actualiza tu proveedor de seguridad para protegerte contra vulnerbilidades SSL. [En línea]. Disponible en: https://developer.android.com/training/articles/security-gms-provider

Android Developers. App Security Best Practices. [En línea]. Disponible en: https://developer.android.com/topic/security/best-practices?hl=es-419

Android Developers. Arquitectura de la plataforma. [En línea]. Disponible en: https://developer.android.com/guide/platform

Android Developers. Configuración de seguridad de la red. [En línea]. Disponible en: https://developer.android.com/training/articles/security-config

Android Developers. Manifesto de la App. [En línea]. Disponible en: https://developer.android.com/guide/topics/manifest/manifest-intro.html

Android Developers. Security for Android Developers. [En línea]. Disponible en: https://developer.android.com/topic/security?hl=es-419

Android Developers. Seguridad con HTTPS y SSL. [En línea]. Disponible en: https://developer.android.com/training/articles/security-ssl?hl=es-419

Android Developers. Sugerencias de seguridad. [En línea]. Disponible en: https://developer.android.com/training/articles/security-tips?hl=es-419#UserData

Android Developers. Work with data more securely. [En línea]. Disponible en: https://developer.android.com/topic/security/data

Android Open Source Project. App Security. [En línea]. Disponible en: https://source.android.com/security/overview/app-security

Android Open Source Project. Application Sandbox. [En línea]. Disponible en: https://source.android.com/security/app-sandbox

Android Open Source Project. Architecture. [En línea]. Disponible en: https://source.android.com/devices/architecture

Android Open Source Project. Kernel Security. [En línea]. Disponible en: https://source.android.com/security/overview/kernel-security

Carnegie Mellon University. SEI CERT Oracle Coding Standard for Java. Prevent SQL Injection [En línea]. Disponble en: https://wiki.sei.cmu.edu/confluence/display/java/IDS00-J.+Prevent+SQL+injection

Common Weakness Eumeration. CWE-312: Cleartext Storage of Sensitive Information. [En línea]. Disponible en: https://cwe.mitre.org/data/definitions/312.html

Common Weakness Enumeration. CWE-346: Origin Validation Error. [En línea]. Disponible en: https://cwe.mitre.org/data/definitions/346.html

Common Weakness Enumeration. CWE-927: Use of implicit Intent for Sensitive Communication. [En línea]. Disponible en: https://cwe.mitre.org/data/definitions/927.html

FANG, Zhejun, *et al.* A static technique for detecting input validation vulnerabilities in Android apps. En: SCIENCE China Information Sciences. Mayo, 2017. Vol. 60, no. 5, p. 1-16.

GUNASEKERA, Sheran. Android Security Apps. Berkeley, CA, Apress, 2012. Print ISBN 978-1-4302-4062-4. Online ISBN 978-1-4302-4063-1

MENG, Huasong, e*t al.* A survey of Android exploits in the wild. En: Computers & Security. Julio, 2018. Vol. 76, p. 71-91.

MU, Zhang y HENG Yin. Android Application Security. A semantics and Context-Aware Approach. Cham, Springer, 2016. Print ISBN 978-3-319-47811-1. Online ISBN 978-3-319-47812-8

YANG, Xu, *et al.* An adaptative and configurable protection framework against Android privilege escalation threats. En: Future Generation Computer Systems. Marzo, 2019. Vol 92, p. 210-224.