Android Security Framework

Ernesto Rodriguez Gallardo

Resumen– Con el fin de optimizar el trabajo de un evaluador de seguridad, este proyecto consta de la elaboración de una metodología de evaluación de seguridad de aplicaciones para el sistema operativo Android y una herramienta modular basada en la metodología anterior con el fin de automatizar dichas evaluaciones. Debido a que no existe un estándar sobre la forma de trabajar en este ámbito, se ha estudiado el estado del arte respecto al tema y se ha desarrollado una metodología propia. La herramienta se ha implementado a modo de prueba de concepto y se han extraído unos resultados al respecto. Estos resultados demuestran que la herramienta es capaz de extraer una gran cantidad de información en un tiempo que para un humano sería impracticable, por ello esta prueba de concepto resulta muy prometedora para continuar con su desarrollo.

Palabras clave– Android, Seguridad, Framework, APK, Evaluación, Metodología

Abstract– In order to optimize the security evaluator's work, this project consists in the development of a methodology to assess the security of applications for an Android operating system and a modular tool based on the previous methodology to automate such evaluations. Due to the fact that there is no standard procedure on how to work in this field, the state of the art has been studied with regard to the subject and a particular methodology has been developed. The tool has been implemented as a proof of concept and some results have been extracted. These results show that the tool is capable of extracting a large amount of data in a period of time that would be impossible for a human. Thus, the proof of concept is extremely promising and encourages continuing with its development.

Keywords– Android, Security, Framework, APK, Evaluation, Metodology

1 Introducción

Hoy en día, es un hecho fehaciente que existe una tendencia creciente al uso que le damos al teléfono móvil en tareas cotidianas [1], como por ejemplo consultar nuestro saldo bancario. Ciertas de estas tareas, sin que muchos nos demos cuenta, ponen en peligro nuestra información personal al utilizar aplicaciones inseguras [2]. Debido a esto, muchas empresas que trabajan con información sensible de sus clientes vía aplicaciones móviles, están invirtiendo mucho dinero en que sus aplicaciones sean seguras. Las empresas que invierten en que terceros evalúen sus aplicaciones son tradicionalmente los bancos, dado que son los que tratan con lo más sensible para la mayoría, el dinero. Aún así también crece la apuesta de todo tipo de empresas por la seguridad de sus aplicaciones antes de lanzarlas al mercado. Dada la creciente demanda, este proyecto nace con el objetivo de generar una metodología y una herramienta basada en esta, que ayuden a los evaluadores profesionales a automatizar todo lo posible aquellas tareas repetitivas, en las que se desaprovechan las cualidades del evaluador, haciendo más eficientes las evaluaciones de seguridad de aplicaciones móviles.

Con el fin de hacer este proyecto viable, se ha llevado a cabo centrándonos en la evaluación de aplicaciones móviles para dispositivos Android, dado que es el sistema operativo más utilizado en este ámbito [3]. Además, en la prueba de concepto de la herramienta solo se implementan técnicas de análisis estático, debido a normalmente son más susceptibles de ser automatizadas.

Este documento consta de una primera parte de investigación en la que se encuentran la introducción al sistema operativo Android y sus aplicaciones, además de una sección con el estado del arte referente a este proyecto. El segundo bloque consta del desarrollo de la metodología de evaluación elaborada para el proyecto, junto a la documentación referente al diseño y funcionamiento de la herramienta. Por ultimo, se encuentran la exposición de los resultados del proyecto y las consecuentes conclusiones.

- E-mail de contacto: ernesto.rodriguezg@e-campus.uab.cat
- Mención realizada: Tecnologías de la Información
- Trabajo tutorizado por: Ángel Elbaz (dEIC)
- Curso 2017/18

2 Objetivos

El objetivo principal de este proyecto consta de elaborar una metodología de evaluación de seguridad para aplicaciones Android, y basada en esta desarrollar una herramienta modular que lleve a cabo evaluaciones de forma automática. Este objetivo, obviamente no es asumible por el grado de complejidad, como aproximación a este objetivo se ha llevado a cabo la implementación de una herramienta que haga más eficiente el trabajo de los evaluadores de seguridad automatizando procesos, de forma que el evaluador no tenga que llevarlos a cabo manualmente.
El objetivo principal de este proyecto es demasiado extenso para abordarlo directamente, por ello el proyecto se divide en varios sub-objetivos:

- Estudiar el estado del arte sobre las diferentes herramientas y técnicas utilizadas en una evaluación, con el fin de ser capaz de conocer su funcionamiento y posibilidades automatización.
- Definir una metodología de evaluación para aplicaciones móviles en dispositivos Android que permita identificar y priorizar las diferentes fases en una evaluación de seguridad, así como las herramientas y técnicas a utilizar, con el fin de identificar las susceptibles a una automatización.
- Diseñar el funcionamiento y la estructura de la herramienta modular y diseñar la plantilla para los módulos.
- Priorizar y definir los módulos en función de la posible mejora de tiempo respecto a la ejecución manual.
- Llevar a cabo la implementación de la herramienta modular y la implementación los módulos en orden de prioridad.
- Testear las implementaciones realizadas. Para asegurar su correcto funcionamiento.
- Ejecutar distintas pruebas, teniendo en cuenta el tiempo como métrica principal, entre la ejecución manual y automática.

3 Android

Android es un sistema operativo empleado principalmente en dispositivos móviles como teléfonos inteligentes y tabletas. Este sistema, es un sistema de código libre, actualmente propiedad de Google. Gracias a su arquitectura, confiere una gran flexibilidad ante la adaptación a dispositivos de características muy heterogéneas. Además, el hecho de que los desarrolladores de dispositivos no tengan que pagar licencias por su uso, ha hecho que sea el sistema operativo para dispositivos móviles más utilizado actualmente [3].

3.1. Sistema Operativo

Aunque Android esta basado en el núcleo de Linux, como se puede ver en la figura 1, en realidad tiene una arquitectura en forma de pila de software creada para ofrecer una gran flexibilidad ante la variedad de dispositivos y factores sobre los que se puede utilizar [4].

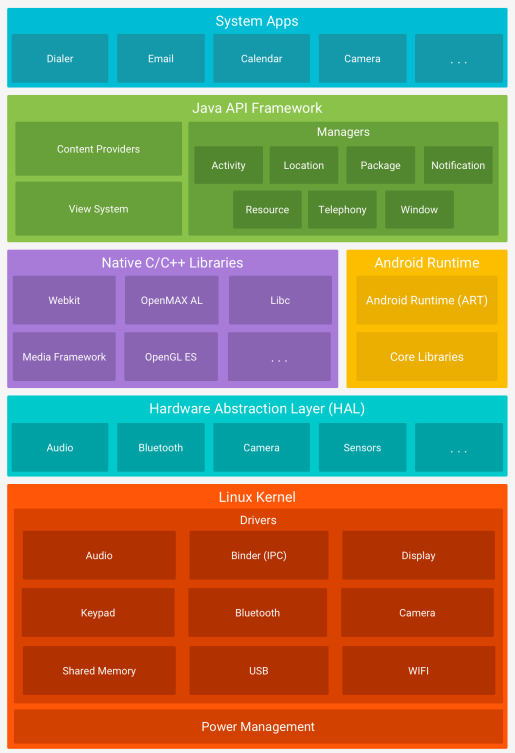


Fig. 1: Pila de software Android

3.1.1. Linux kernel

El kernel de Linux se sitúa en la parte más baja de esta arquitectura. Este kernel tiene ciertos cambios efectuados por Google con el fin de adaptarlo al uso en dispositivos móviles. Este nivel de la arquitectura provee a Android funcionalidades varias como son la gestión de los procesos o la memoria. Además, contiene los diferentes controladores para gestionar los dispositivos y funcionalidades como pueden ser las redes y la energía [5].

3.1.2. Capa de abstracción de hardware(HAL)

La HAL consiste en varios módulos de biblioteca y cada uno de estos implementa una interfaz para un tipo específico de componente hardware [4]. Gracias a esto es posible que los niveles superiores de la arquitectura ejecuten llamadas estándar al hardware, facilitando así la adaptación distintos hardware por parte del sistema.

3.1.3. Android Runtime

Android Runtime o ART reemplaza a Dalvik desde Android 5.0. Dalvik es la máquina virtual utilizada originalmente por Android, esta lleva a cabo la transformación de la aplicación en instrucciones de máquina, que posteriormente son ejecutadas por el entorno de ejecución nativo del dispositivo [6]. Una de las principales diferencias es el hecho de que Dalvik compilaba en cada ejecución el código de bytes y ART es capaz de compilarlo en tiempo de instalación y ejecutarlo cuando se abre la aplicación. Con el fin de mantener la compatibilidad, ART mantiene el mismo *bytecode* utilizado por Dalvik VM, el llamado DEX.

3.1.4. Bibliotecas C/C++ nativas

Esta capa proporciona a los diferentes niveles de la arquitectura las librerías necesarias para la ejecución nativa de servicios y componentes, obteniendo así mayor rendimiento y seguridad.

Android también permite la ejecución de código nativo por parte de las aplicaciones mediante el *Android Native Development Kit* que entre otros aspectos ofrece acceso a estas librerías desde el código.

3.1.5. API Java

Todo el conjunto de funciones de Android está disponible mediante APIs escritas en el lenguaje Java. Estas APIs son los cimientos que se necesitan para crear aplicaciones de Android simplificando la reutilización de componentes del sistema, servicios centrales y modulares [4]

3.1.6. Aplicaciones del sistema

En Android, se incluye un conjunto de aplicaciones centrales para correo electrónico, mensajería SMS, calendarios, navegación en Internet y contactos, entre otros elementos. Las aplicaciones incluidas en la plataforma no tienen un estado especial entre las aplicaciones que el usuario elige instalar. Por ello, una aplicación externa se puede convertir en el navegador web, el sistema de mensajería SMS o, incluso, el teclado predeterminado del usuario [4].

3.2. Aplicaciones Android

Las aplicaciones Android se desarrollan en lenguaje Java con el añadido del *Android Software Development Kit*. Además, como ya se ha mencionado anteriormente, gracias a su pila de software Android también puede incluir en sus aplicaciones librerías de código nativo con fines de rendimiento o seguridad. Además, utiliza su propio canal de distribución de aplicaciones llamado Google Play [7]. Gracias a esta plataforma se puede acceder a los archivos contenedores de miles de aplicaciones, estos archivos contenedores son los APK o Android Application Package. Un APK en realidad es un archivo extendido de JAR que a su vez extiende del ZIP, por lo que el APK no es la aplicación en sí, sino que este contiene todos los componentes de esta.

3.2.1. Estructura de un APK

Dado que el APK es un archivo contenedor, cabe destacar cuales son los archivos y directorios que este contiene, ya que estos son los que realmente forman la aplicación.

Archivos

- **AndroidManifest.xml:** Todas las aplicaciones deben tener un archivo AndroidManifest.xml (con ese nombre exacto) en el directorio raíz [8]. Este archivo XML provee la información necesaria al sistema para poder instalar y ejecutar la aplicación. Entre otros muchos datos cabe destacar que contiene el nombre del paquete que identifica la aplicación, declara permisos necesarios para su ejecución, declara el mínimo nivel de API necesario para su ejecución, servicios, etc.
- **classes.dex:** Es un archivo que contiene el código de la aplicación compilado de manera que sea comprensible por el ART o la Dalvik VM.
- **resources.arsc:** Este archivo contiene todos aquellos recursos que han sido precompilados para su uso por la aplicación.

Directorios

- **res:** El directorio res es donde se alojan todos los recursos requeridos por la aplicación que al contrario que en resources.arsc no han sido precompilados.
- **assets:** El directorio assets contiene todos aquellos recursos que pueden ser requeridos por la clase AssetManager.
- **lib:** Este directorio contiene las librerías nativas de la aplicación. Dentro de este pueden encontrarse distintos directorios en función de si la aplicación ha sido compilada para distintas arquitecturas o no.
- **META-INF:** Este último directorio contiene los archivos referentes a la firma de la aplicación que será comentada más adelante.

3.3. Seguridad en aplicaciones Android

En lo que respecta a la seguridad de las aplicaciones, Android implementa un sistema de verificación mediante firma digital en el momento de la instalación de las mismas, así como un sistema de permisos para controlar el acceso de las aplicaciones a los diferentes recursos del sistema. Android también hereda directamente de Linux un sistema para proteger sus aplicaciones mediante *Sandbox*.
Cabe destacar que a pesar de las mediadas de seguridad impuestas por el sistema operativo, la facilidad para obtener permisos de superusuario en Android, facilitan evitar y/o anular la capacidad de protección de estas medidas.
Debido a la posible hostilidad del entorno, dada por la facilidad para obtener permisos de superusuario, la seguridad de las aplicaciones recae en los desarrolladores y en cómo estos protegen la información. Esta seguridad no es fácil de definir ya que en realidad no existe una forma correcta de aplicarla, cada desarrollador intenta proteger sus aplicaciones lo mejor posible, aplicando múltiples contra medidas como comprobaciones de todo tipo e intentando ocultar al atacante la funcionalidad real de ciertas partes sensibles, con el objetivo de distraer la atención y dificultar su análisis.

3.3.1. Firma de aplicaciones

La firma de las aplicaciones asegura la integridad de la aplicación. A pesar de ello, cualquiera puede desempaquetar la aplicación y volver a firmarla sin que el sistema se oponga a volver a instalarla. Por ello, su seguridad no es tan firme como en otros sistemas. Aún así, la firma ya no será la misma debido a que tanto clave de firma como el contenido

no serán los mismos.
Actualmente la firma de aplicaciones Android consta de dos versiones compatibles entre sí.

V1 La primera versión de firma que se utilizó, se basa en el sistema de firma utilizado por el estándar JAR [9]. Esta firma está contenida en el directorio *META-INF* dentro del APK. Este estándar recopila los resultados de ejecutar funciones resumen a todas las entradas del APK en un fichero llamado *MANIFEST.MF*. Llegados a este punto cada firmante genera un fichero con extensión *.SF* en el que se incluyen todas las entradas contenidas en el *MANIFEST.MF* y además se añade la función resumen de este. Por último, cada firmante genera otro fichero donde se firma el anterior. Este tipo de firma asegura la integridad de las entradas del fichero APK, pero no la integridad de este al completo.

V2 Desde el lanzamiento de Android 7.0 existe un nuevo tipo de firma creada por los desarrolladores de Android llamada *APK Signature Scheme v2* [10]. Este esquema de firma, protege la integridad del APK al completo y lo incrusta entre los bytes del propio APK en un bloque del fichero denominado *APK Signing Block*.

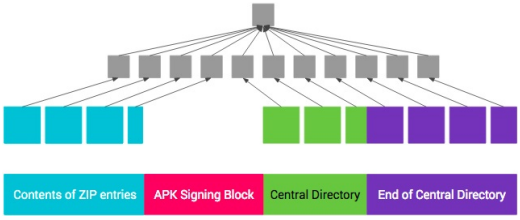


Fig. 2: Diagrama de firma V2

3.3.2. Permisos

En Android, la gestión y/o uso de ciertos recursos por parte de las aplicaciones esta controlado por un sistema de permisos. De esta manera una aplicación sin ningún tipo de permisos, no podrá afectar de ninguna manera ni a la experiencia de usuario ni a los datos contenidos en el dispositivo. Para que esta aplicación pueda obtener dichos permisos, estos deben estar definidos en el fichero XML contenido en el APK llamado *AndroidManifest.xml*.
El uso de los diferentes recursos del sistema, a los que se accede mediante permisos, tienen diferentes riesgos asociados. Por ello los permisos también tienen un nivel de seguridad a asociado que permite gestionar los diferentes permisos en función del riesgo que pueden suponer. En el caso de tratarse de un permiso de nivel normal, con el simple hecho de estar definido en el *AndroidManifest.xml*, este es concedido por el sistema. En cambio si el recurso al que se quiere acceder conlleva un riesgo más alto, el sistema preguntará directamente al usuario pidiendo permiso explicito [11].

3.3.3. Sandbox

El Sandbox en Android, a pesar de ser una característica heredada desde Linux, se utiliza de una forma peculiar. En la mayoría de sistemas cada usuario ejecuta diferentes aplicaciones, en cambio en Android el sistema operativo asigna un identificador de usuario único a cada aplicación y la ejecuta con ese usuario, de este modo el Sandbox de usuario de Linux se convierte en el Sandbox de las aplicaciones en Android.

4 Estado del arte

Para la elaboración de este proyecto se ha efectuado una búsqueda de herramientas similares a la que se pretende desarrollar, con el fin de conocer el estado del arte en este sector. A pesar de la exhaustiva búsqueda, únicamente se ha encontrado una herramienta que tenga una funcionalidad relacionada. Por ello también se elaborado estado del arte más general respecto a técnicas y herramientas utilizadas en evaluaciones de seguridad sobre aplicaciones Android.

4.1. Mobile Security Framework

Mobile Security Framework es una herramienta automática de evaluación de seguridad de aplicaciones para dispositivos móviles [12]. Esta herramienta es capaz de analizar aplicaciones para distintos sistemas operativos móviles como Android, iOS y Windows. Por ello, esta herramienta es capaz de evaluar APKs de Android, IPAs de IOS, APPXs de Windows y el propio código fuente de estas encapsulado en un archivo ZIP. Además, esta herramienta es capaz de llevar a cabo diferentes tipos de análisis como pueden ser el estático, dinámico y de código malicioso.
Esta herramienta sigue en desarrollo después de varios años de dedicación por parte sus creadores, pero aun así ha conseguido diversos reconocimientos en certámenes de seguridad informática a nivel global, entre ellos formar parte del *BalckHat Arsenal*.

4.2. Técnicas

Durante la investigación las técnicas en este sector han resultado ser muchas, en esta sección se recogen las mas utilizadas.

4.2.1. Debugging

El debugging se basa en la capacidad de ver y controlar el comportamiento de un programa en tiempo de ejecución. Las herramientas de debugging permiten lanzar un programa, detener su ejecución mediante puntos de parada, ver que está sucediendo y modificar el entorno mediante aspectos como las variables o la memoria [13].

4.2.2. Hooking

El hooking, es una técnica muy importante en el mundo de la evaluación de seguridad de aplicaciones móviles [14]. Esta técnica se basa en la intercepción de llamadas a funciones. En una ejecución normal, cuando se efectúa una llamada a función, esta se ejecutaría y finalizada su ejecución, volvería al punto en el que fue llamada.En cambio, cuando se efectúa una llamada concreta a la que le estamos haciendo hook, esta se redirige a el código que se quiere inyectar. Con esta redirección, se pueden conseguir distintos objetivos como el descubrir los parámetros que recibe esta función, evitar la ejecución de la función llamada, modificar su retorno, etc.

4.2.3. Application directories analysis

Una parte muy importante de toda evaluación es el análisis de los datos que la aplicación almacena. En su mayoría, las aplicaciones almacenan sus datos en unos directorios independientes protegidos por el sistema mediante el *Sandbox*. Si estos no están debidamente protegidos puede ser una brecha de seguridad. Aún así, hay que proteger los datos almacenados debido a la posible hostilidad del entorno [15].

4.2.4. Reverse Engineering

La ingeniería inversa de software es una combinación de diferentes artes como son la programación, el análisis lógico, el descifrado de códigos y la resolución de problemas [16]. Desde el punto de vista de la seguridad del software el planteamiento es muy sencillo, analizar cómo funcionan las medidas de seguridad, con el fin de conocer si se pueden vulnerar y por tanto si son seguras.

4.2.5. Miss configurations

Una mala configuración en el código o selección de los permisos por parte del desarrollador puede generar vulnerabilidades [17]. Por ello en una evaluación estos permisos y configuraciones han de ser revisados y puestos a prueba si fuese necesario para comprobar si son seguros.

4.2.6. Patching

El patching es una técnica basada en la modificación de forma estática de la aplicación con el fin de modificar su comportamiento [18]

4.2.7. Logging

Esta técnica consiste en monitorizar el funcionamiento de la aplicación mediante trazas que los desarrolladores pueden haber dejado atrás. No siempre es fructífera, pero en ocasiones los desarrolladores pueden haber dejado en el código trazas útiles para entender cómo funciona la aplicación y en algunos casos pueden haber dejado trazas de desarrollo activas que pueden exponer información sensible.

4.2.8. Networking

En el mundo de la seguridad en las redes, el abanico de posibilidades es muy extenso, pero cuando lo enfocamos a la evaluación del uso de la red por parte de una aplicación, el alcance es más concreto. Para evaluar una aplicación, la técnica por excelencia sería el Sniffing [19], esta permite analizar cómo se establece la comunicación segura entre la aplicación y el servidor gracias a la captura de los paquetes. De esta manera se puede establecer si se utilizan protocolos seguros y si se utilizan de forma correcta.

4.2.9. Crypto

La criptografía es muy importante desde el punto de vista de la seguridad ya que está puede proveer de Confidencialidad e Integridad a los elementos que se quieren proteger. Pero no toda la criptografía es igualmente segura, existen diferentes tipos de algoritmo e incluso algoritmos en desuso porque han sido vulnerados.

4.3. Herramientas

Al igual que con las técnicas, las herramientas halladas durante la investigación han sido muchas, en esta sección se introducen las que han sido utilizadas para este proyecto.

4.3.1. Apktool

ApkTool es una herramienta utilizada para desempaquetar el APK completo y devolverle una forma muy parecida a la que tenía previamente de ser empaquetada. Además, también permite el reempaquetado de la misma. Esto es muy útil de cara a aplicar técnicas de patching, en la que se modifica el APK [20].

4.3.2. JADX

Esta herramienta es un proyecto de código libre que se utiliza para extraer y decompilar el código contenido en el archivo *classes.dex* obteniendo así el código Java de nuevo [21].

4.3.3. Apksigner

Apksigner es una herramienta muy útil para la firma y verificación de esta sobre archivos APK desarrollada por el equipo de Android para cubrir las necesidades de los esquemas propios de firma actuales y futuros.
En el ámbito de una evaluación de seguridad apksigner resulta útil cuando se ha realizado un patching a una aplicación, ya que en hacer cambios sobre esta su firma ya no es válida y se ha de generar otra nueva.

4.3.4. YARA

YARA es una herramienta de código abierto diseñada para ayudar a los investigadores de malware a identificar y clasificar muestras de malware. Hace posible crear descripciones (o reglas) para familias de malware basadas en patrones textuales y/o binarios [22]. En este trabajo se ha utilizado para identificar patrones de código en base a reglas.

5 METODOLOGÍA

Como primer paso antes de desarrollar la metodología en sí, se hizo un trabajo de análisis del conocimiento adquirido, para definir un diagrama simplificado del flujo de tareas en una evaluación de seguridad sobre aplicaciones Android. El resultado de este análisis puede verse en la figura 3.

Tal y como se puede observar en la figura 3, en una evaluación de la seguridad de aplicaciones para sistemas Android, se utiliza como entrada el fichero contenedor de estas, el fichero APK. En lo que respecta al resultado de aplicar la metodología, se elabora un informe en el que se encuentra la información relevante a todos los procesos llevados a cabo durante la evaluación.
Partiendo del flujo de evaluación presentado en la figura 3, se extraen las siguientes fases.

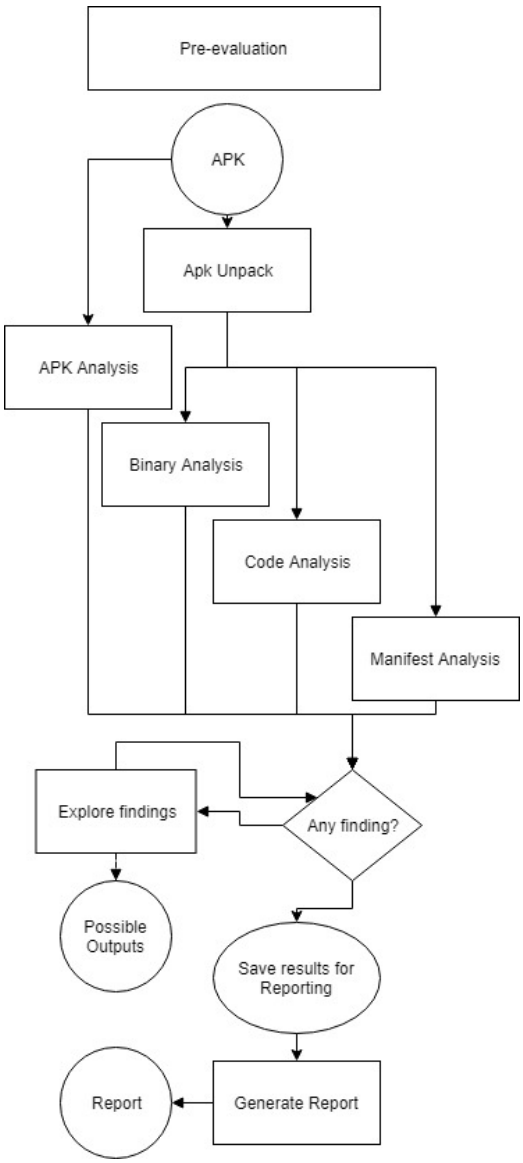


Fig. 3: Flujo de evaluación

5.1. Fase Previa

En esta fase se lleva a cabo la configuración del entono de trabajo. En todo trabajo de ingeniería, la organización es muy importante tanto física como digitalmente.
Con el fin de obtener esta organización, se deben llevar a cabo las tareas que sean necesarias para que de forma inequívoca que se puedan identificar las entradas y los resultados de una evaluación respecto a otra.

5.2. Fase de preparación

Tal y como se especifica anteriormente, la entrada de esta metodología es la aplicación es su estado de distribución, es decir, un fichero APK. Aunque como se puede ver en el flujo de evaluación, algunas de las fases del flujo no analizan este fichero directamente, sino que trabajan sobre ciertos contenidos de este fichero. Por ello, en esta fase se llevan a cabo todas las transformaciones necesarias para que estas fases tengan las entradas necesarias.

5.3. Fase de análisis

5.3.1. Análisis del APK

El APK como fichero en sí, sin modificaciones previas, es útil dado que permite comprobar la integridad de este. Para comprobar la integridad de este, el APK contiene su firma digital, haciendo posible su verificación.
Los APK no son un tipo de contenedor precisamente nuevo, por ello existen hasta el momento dos formas distintas de contener la firma digital, llamadas *Jar-Signature* o *V1* y *APK Signature Scheme v2* [10]. Ambas contienen información del firmante y la propia firma, pero de formas completamente distintas. Además, para asegurar la compatibilidad de aplicaciones nuevas con los dispositivos antiguos, ambas pueden convivir.
El proceso a seguir en esta fase comienza verificando las firmas de la aplicación. Con las firmas verificadas se procederá a extraer la información relevante tonto de la v1 como de la v2.

5.3.2. Análisis del manifest

Todo fichero APK ha de contener un fichero *AndroidManifest.xml* en el que se declaran diferentes aspectos de la aplicación [8]. Estos aspectos son muy variados, pero algunos de ellos pueden ser muy relevantes en lo que a la seguridad de la aplicación.
En esta fase se aplican dos procesos muy diferenciados, el primero será principalmente la extracción de la información, para su posterior adhesión al informe. Tras esto, se llevará a cabo un análisis de este mismo fichero, con el fin de encontrar los posibles hallazgos respecto a la seguridad de la aplicación.

5.3.3. Análisis del código

El código en la aplicación esta compilado para ser interpretado por el sistema, pero tras la fase de preparación, el código en Java estará disponible en el Working Directory.
En esta fase, se analiza el código en busca de hallazgos sobre la seguridad de la aplicación.

5.3.4. Análisis de binarios

Debido a que en Android existe la opción de ejecutar código nativo con el fin de obtener mejor rendimiento y seguridad en la ejecución de ciertas funciones [23], algunas APK contienen binarios compilados para algunas arquitecturas.
En esta fase, se extrae información de estos binarios para ser añadida al informe.

5.4. Fase de hallazgos

Tras analizar los componentes de la aplicación, se han obtenido una serie de hallazgos, que serán reflejados en el informe y en los cuales se podría profundizar.
Llegados a este punto, en el caso que se tengan hallazgos con los que se pueda o deba ir más en profundidad, para esclarecer la verdadera naturaleza o peligrosidad del hallazgo, se perpetuaran los análisis y/o ataques que sean necesarios.
En caso de que para la ejecución de los módulos se tengan

que generar archivos intermedios o cualquier tipo de output estos se contendrán el Working Directory.

5.5. Fase de reporting

Esta fase se recaba toda la información obtenida durante la evaluación, para así poder generar un documento en el que se plasma toda esta información relevante a la seguridad de la aplicación Android.

6 Diseño

Tal y como se recogió en los objetivos principales de este proyecto, se quiere implementar una herramienta modular para evaluar la seguridad de aplicaciones Android. Con este fin se ha diseñado una herramienta para ser implementada con el lenguaje de programación Python en su versión 2.7. Gracias a este lenguaje se podrá obtener capacidades multiplataforma además de ser un lenguaje muy flexible, potente y con un amplio abanico de APIs disponibles [24].
Este diseño se va a realizar siguiendo la metodología desarrollada anteriormente como requisitos de diseño, de forma que el funcionamiento de la herramienta final sea ejecutar esta metodología de principio a fin.
Esta herramienta tendrá la capacidad de ser completamente modular, pero además va tener dos tipos de módulos. Estos módulos serán exactamente iguales en forma pero se diferenciaran en el modo que se utilizarán. Los primeros serán los módulos que conforman la herramienta en sí y los segundos serán módulos que podrá el usuario añadir en su correspondiente directorio y no hará falta que modifique el código para ser ejecutados. Para ello, el usuario solo tendrá que añadir las reglas de hallazgo que desea y definir la relación de ese hallazgo con un módulo mediante dos ficheros de configuración que se explicarán más adelante.
Para organizar este diseño se va a tratar en dos partes, primer la parte de la herramienta en sí y después cada uno de sus módulos por separado.

6.1. Herramienta

La herramienta desarrollada para este proyecto consta de múltiples ficheros y directorios, los más significativos de estos van a ser comentados en esta sección. En lo que respecta a la estructura de estos ficheros y directorios, se puede ver en las capturas de pantalla añadidas al apéndice.

6.1.1. androidsf.py

Dado que la herramienta será completamente automática y el output final es un informe, la interfaz de usuario de la herramienta será por línea de comandos, siendo así más simple de implementar e igual de funcional. Para implementar esta línea de comandos se utiliza el fichero principal de la herramienta *androidsf.py*.
En este fichero se implementa una simple interfaz por consola que continuamente está a la espera de la entrada de usuario para instanciar o llamar al *EvaluationDriver*. Esta interfaz de usuario tiene cuatro comandos:

- Exit: escribiendo *exit*,*e*, *q* o *quit* se cierra la aplicación.
- Run: escribiendo *run* o *r* se lanza la evaluación.
- Driver: escribiendo *driver* o *d* y la ruta a un APK se instancia el driver.
- Help: escribiendo en la consola *help* o *h* se nos muestran los tres comandos anteriores.

Este fichero puede ser lanzado con parámetro o sin él, este parámetro es básicamente la ruta al APK que se quiere analizar. Si no se utiliza este parámetro, habrá que instanciar el driver manualmente. Si se proporciona la ruta por parámetro se comenzará con el driver ya instanciado.

6.1.2. module.py

Este fichero contiene la clase Module, esta clase sirve para que todos los módulos que hereden de esta tengan los mismos métodos a los que llamar desde el driver. Los métodos genéricos de un módulo son:

- __init__(): Inicializa el módulo.
- run(): Contiene el código a ejecutar por el modulo.
- report(): Genera el fichero HTML con la información a incluir en el report.
- get_findings(): En caso de haber hallazgos los retorna.

6.1.3. evaluation_driver.py

Este archivo Python ya no solo es un script a modo de lanzador, sino que es una clase que como su nombre indica es la que contiene la lógica de la evaluación. Esta clase será la encargada de ir ejecutando los distintos módulos.
En el momento que la clase *EvaluationDriver* sea instanciada, se ejecutará la función llamada *__init__()*, en esta función se implementa la funcionalidad correspondiente a la fase previa de la metodología. Para poder diferenciar inequívocamente los resultados de una evaluación de otra se creará un directorio específico para el día y la hora de la evaluación. Dentro de este directorio llamado *Working Directory*, se colocará una copia del APK original a la que se le aplicarán y guardarán distintas funciones resumen almacenadas en el fichero *fingerprints*. Cuando en la línea de comandos se ejecute el comando *run*, teniendo un driver instanciado, se ejecutará la función *run()* del driver. En esta función se lleva a cabo la ejecución de los distintos módulos. Para listar los módulos a ejecutar se utiliza una lista de Python llamada *flow*. Los módulos están contendidos en la carpeta *modules*. Existen dos excepciones que no están en este *flow*, son los módulos de *unpack* y de report. Estos se ejecutan independientemente al inicio y final del método *run* correspondiendo respectivamente con las fases de preparación y reporting respectivamente. Entre las llamadas al *unpack* y *report*, se halla la sección de la función *run* que llevará a cabo las fases de análisis y de hallazgos paralelamente. Para hacer esto, por cada módulo que se ejecute se comprobará si existen hallazgos, en caso positivo y si estos hallazgos tienen un módulo relacionado, este también se ejecutará. Esta relación estará definida en un fichero llamado *findings* que relacionará el código del hallazgo con el nombre del módulo. Además, también indicará si se ha de ejecutar por cada finding o una única vez.

6.2. Modulos

En esta segunda sección se definen los diferentes módulos y su funcionalidad. Todos los módulos deberán estar contenidos en la carpeta modules. Además, en la carpeta modules existen otros dos directorios auxiliares llamados *utils* y *rules*. El primero contiene distintas herramientas que pueden ser ejecutadas desde los módulos y el segundo contiene los archivos que definen las reglas para el análisis para los diferentes módulos de análisis.

6.2.1. rules

Pese a no ser un módulo, este directorio contiene unos ficheros muy importantes para la ejecución de algunos de los módulos, por lo que es importante tenerlo en cuenta. Los ficheros que contiene este directorio son ficheros *.yara*. Yara es una herramienta de reconocimiento de patrones muy sencilla de utilizar pero muy potente. Gracias a Yara podemos definir una regla simplemente escribiendo una serie de *strings* objetivo y con ellas formar una condición mediante operadores lógicos. Además, permite añadir meta-datos a la regla de forma que estos sean accesibles cuando se encuentra una coincidencia. Estos han sido muy utiles para añadir un identificador al hallazgo con el que comparar contra las entradas de el fichero *findings* en caso de coincidencia en el análisis [22].

6.2.2. apk_unpack.py

Este módulo corresponde directamente con la fase de preparación en la metodología de evaluación. El cometido de este módulo es coger el APK y aplicarle las transformaciones necesarias para obtener las entradas del resto de módulos y almacenarlos en el Working Directory.
La implementación de este módulo se basa en la utilización de herramientas para desempaquetar el APK y obtener diferentes entradas. Con el fin de obtener el *AndroidManifest.xml*, los binarios y el directorio *META-INF* que contiene la firma v1, se utilizara la herramienta *apktool* [20]. Después, para obtener el código en lenguaje Java, se utilizará una herramienta llamada *jadx* [21] que extrae y decompila el código del archivo *classes.dex*.

6.2.3. apk_analysis.py

Este módulo se basa en analizar las firmas del fichero APK con el fin de obtener la información del firmante y el estado de la firma. Este módulo se basa en tres funciones principales además de las predefinidas en todos los módulos. Estas tres funciones se encargarán de verificar y extraer las firmas del fichero APK:

- Verify(): Esta función ejecuta una herramienta llamada *apksigner* con la opción *-verify* y el APK como paramtro. Esta llamada retorna el resultado de la verificación del APK que es parseada y guardada en un diccionario para su posterior adhesión al informe.
- v1(meta_dir): Esta función y todas a las que esta llama, obtienen todos los datos de la firma v1. Para ello genera una lista de firmantes en la que cada firmante es en realidad un diccionario Python con todos los campos de la firma para una vez ejecutado el modulo poder obtener todos los campos por nombre y añadirlos al informe.
- v2(apk): Esta función y todas a las que esta llama, obtiene todos los datos de la firma v2. Para ello genera una lista de firmantes en la que cada firmante es en realidad un diccionario Python con todos los campos de la firma para una vez ejecutado el modulo poder obtener todos los campos por nombre y añadirlos al informe.

6.2.4. code_analysis.py

Para analizar el código en busca de posibles hallazgos de seguridad utilizaremos Yara [22].
La función *run()* de este módulo únicamente llama a la función de *scan_code()*. Tras esto se recorren todos los directorios y ficheros obtenidos gracias a la herramienta *jadx*. Entonces por cada fichero java se recorre línea a línea comparándolo con las reglas de Yara. De esta forma cuando en una línea obtenemos un hallazgo podemos guardar estos datos en una lista Python.
El formato de los hallazgos sería una lista Python en la que por cada hallazgo que se hace se guarda una estructura de lista que tiene el fichero donde se ha encontrado en la primera posición, la línea en la que se ha hecho el hallazgo en la segunda posición y la lista de hallazgos en esa línea en la tercera posición. Finalmente, como en todos los módulos que se obtienen hallazgos, y estos se guardan en un atributo de la clase llamado *findings*.
Aprovechando el recorrer todos los ficheros java para su análisis estos son listados para ser mostrados en el informe.

6.2.5. manifest_analysis.py

Este módulo de análisis del manifest tiene dos funcionalidades, la primera recupera la información que contiene para plasmarla en posteriormente en el informe y la segunda lo analiza de la misma forma que se hace con el código, con la excepción de que también hace un análisis de componentes que no están presentes gracias al fichero de reglas *manifest_miss_rules.yara*.

6.2.6. binary_analysis.py

El módulo de análisis de binarios se basa en un script llamado *readelf.py*, este utiliza la librería *pyelftools* para emular el comando de Linux *readelf*.
Utilizando este script obtendremos tres de las partes más importantes del ELF en lo que respecta a la seguridad, el header, los símbolos y las librerías que utiliza este ELF. Para ello utilizaremos los parámetros *-h*, *-d* y *-s*.

6.2.7. report.py

Este fichero implementa el módulo referente al último punto de la metodología, la generación del informe. El módulo de report utilizará una interfaz Python llamado *pdfkit* que en realidad utiliza la herramienta *wkhtmltopdf*. Gracias a esta herramienta se genera un PDF a base de una lista de ficheros HTML, que en este caso serán los resultados para añadir al informe de cada uno de los módulos ejecutados.

7 RESULTADOS

Como principal resultado cabe destacar que el objetivo principal del proyecto ha sido llevado a cabo satisfactoriamente. Tal como se ha comentado anteriormente, se han desglosado en diferentes sub-objetivos y cada uno de ellos ha tenido unos resultados independientes.

- En lo que respecta a el estado del arte, al comenzar el proyecto se hizo un estudio de este y se documentó. A pesar de esto, durante toda la duración del proyecto el conocimiento respecto a este estado del arte y su enfoque para el proyecto ha ido cambiando y por ello este se ha actualizado antes de finalizar el proyecto.
- Uno de los mayores retos de este proyecto ha sido el de cumplir el objetivo de definir una metodología de evaluación de seguridad para aplicaciones Android. Al principio se intentó buscar una metodología estándar de la que partir para poder asegurar una cobertura mayor. Tras esta búsqueda, que no fue fructífera, se desarrolló una metodología propia en base a las necesidades del proyecto y que posteriormente fuese aplicable a una herramienta desarrollada dentro del tiempo de trabajo del que se disponía.
- Una vez se consiguió cumplir el segundo objetivo, el objetivo de diseñar la herramienta fue muy natural y fluido gracias a que la metodología propia estaba pensada de forma que también fuese fácil extraer de ella un diseño modular.
- Tras el diseño de la herramienta, el siguiente objetivo es el de priorizar los módulos a desarrollar con el fin de poder llevar a cabo la implementación de esta herramienta en el tiempo disponible para ello. Esta priorización de módulos, debido al retraso que ya se llevaba debido a los problemas con la metodología, se basó en la propia funcionalidad de la herramienta. Como se nombra tanto en la metodología y en el diseño de la herramienta, hay lo que se podrían llamar dos tipos de módulos, los que se ejecutan siempre y los que únicamente se lanzan en función de los hallazgos de estos primeros. La decisión que se tomó para poder acabar la herramienta a tiempo, fue completar a toda costa los módulos estáticos ya que son los que realmente implican la funcionalidad principal de la herramienta y dejar para trabajo futuro los módulos dinámicos. Aun así se implementaron dos módulos de test para verificar la funcionalidad.
- En lo que a la implementación de la herramienta respecta, se ha desarrollado la herramienta hasta el punto que se ha podido dentro de las restricciones de tiempo, obteniendo una prueba de concepto de la herramienta, funcional y con la que poder sacar conclusiones respecto al proyecto.
- Ligado a la implementación ha ido el objetivo de los test de funcionamiento. Durante toda la implementación se han ido haciendo test continuos, para confirmar que lo que se estaba implementando funcionaba correctamente.
- El ultimo objetivo es la extracción de resultados de la herramienta. El principal resultado de la herramienta es obviamente el informe que esta genera. Este informe agrupa toda la infamación recabada durante la evaluación automática, por ello gracias a este informe un evaluador humano puede ahorrar gran cantidad de tiempo simplemente buscando lo que necesita en el informe.

Con el fin de poder comprobar hasta que punto esta herramienta puede ser útil, se han llevado a acabo cinco ejecuciones de la herramienta con aplicaciones de bancos que ya están en el mercado y se han tomado distintos datos significativos. A pesar que es muy difícil comparar lo que tardaría un humano en hacer justo lo mismo que la herramienta, en la tabla 1 se pueden apreciar los tiempos de ejecución del método *run()* de la clase *EvaluationDriver*. Los otros dos

	Tiempo de evaluación
BBVA	42.82 segundos
Banco Sabadell	36.24 segundos
Banco Santander	48.84 segundos
ING	45.97 segundos
CaixaBank	67.84 segundos

TABLA 1: RESULTADOS DE TIEMPO EN LA EJECUCIÓN DE LAS APLICACIONES DE TEST.

datos medidos respecto a la ejecución esta herramienta, tal como se puede ver en la tabla 2, han sido el numero de archivos de código y las lineas que estos contienen. Se han escogido estos dos indicadores debido a que otros posibles indicadores tenían muy poca variabilidad entre cualquier ejecución, debido a que la herramienta lleva a cabo los mismos procesos sobre todas ellas y ciertas partes de una aplicación Android a pesar de sus diferencias no son tan significativas como en lo referente al código.

	Numero ficheros	Numero lineas
BBVA	3325	265610
Banco Sabadell	2571	235224
Banco Santander	3157	345060
ING	3806	343962
CaixaBank	5096	458393

TABLA 2: NUMERO DE FICHEROS Y LINEAS DE CÓDIGO EN LA EJECUCIÓN DE LAS APLICACIONES DE TEST.

8 CONCLUSIONES

De este trabajo se pueden extraer diferentes conclusiones.

- A nivel personal y como futuro profesional, este proyecto ha sido muy enriquecedor dado que me ha permitido afianzar conocimientos y trabajar con una planificación. Además, he podido llevar a buen puerto un proyecto de esta magnitud de principio a fin, con sus problemas y dificultades.
- Respecto a la herramienta, tiene mucho potencial y los resultados obtenidos son muy prometedores. Claramente un evaluador puede llegar a ganar mucho tiempo utilizando la herramienta con las reglas adecuadas.

- Aún y con los resultados obtenidos en este proyecto, esta herramienta es solo una prueba de concepto con la que demostrar que es viable seguir desarrollando esta herramienta y metodología. De cara a continuar este proyecto hay varias lineas de mejora con las que continuar, teniendo el tiempo necesario para investigar y desarrollar estas lineas la herramienta podría llegar a ser muy potente gracias a su modularidad.

    1. Una de las lineas de trabajo futuro más importantes de cara a que un evaluador pueda utilizar esta herramienta de forma eficiente, es trabajar en optimizar el informe final mediante algún sistema de base de datos, facilitando que el evaluador pueda encontrar lo que necesita de forma sencilla ya que actualmente los informes llegan a ser muy extensos.
    2. La segunda posible linea de trabajo es optimizar el sistema de reglas para que sea posible hacer búsquedas mas complejas, capacitando así a la herramienta a hallar ciertos patrones más complejos.
    3. Como ultima linea de trabajo se podrían efectuar pequeños cambios en la herramienta con el fin de que aprovechando su modularidad esta pudiese perpetrar análisis dinámicos.

REFERENCIAS

[1] Eldia, "Usar el celular para todo, un hábito que causa furor en nuestro país." [Online]. Available: https://www.eldia.com/nota/2017-12-26-2-50-20-usar-el-celular-para-todo-un-habito-que-causa-furor-en-nuestro-pais-informacion-general

[2] D. G. Bilić, "Aplicaciones inseguras se propagan entre usuarios de android." [Online]. Available: https://www.welivesecurity.com/la-es/2018/03/02/aplicaciones-inseguras-propagan-usuarios-android/

[3] A. MOSCARITOLO, "El 99.6 % del mercado móvil le pertenece a android y ios," 2 2017. [Online]. Available: http://latam.pcmag.com/sistemas-operativos-moviles/18490/news/el-996-del-mercado-movil-le-pertenece-a-android-y-ios

[4] "Arquitectura de la plataforma." [Online]. Available: https://developer.android.com/guide/platform/?hl=es-419

[5] R. Singh, "An overview of android operating system and its securityfeatures," *Rajinder Singh Int. Journal of Engineering Research and Applications*, vol. 4, no. 2, pp. 519–521, 2 2014.

[6] "Art." [Online]. Available: https://es.wikipedia.org/wiki/Android_Runtime

[7] "Google play." [Online]. Available: https://es.wikipedia.org/wiki/Google_Play

[8] "Manifiesto de la app — android developers." [Online]. Available: https://developer.android.com/guide/topics/manifest/manifest-intro?hl=es-419

[9] "Jar file specification." [Online]. Available: https://docs.oracle.com/javase/8/docs/technotes/guides/jar/jar.html#Signed_JAR_File

[10] "Apk signature scheme v2." [Online]. Available: https://source.android.com/security/apksigning/v2

[11] "Permisos del sistema — android developers." [Online]. Available: https://developer.android.com/guide/topics/security/permissions?hl=es-419

[12] A. Abraham, D. Schlecht, M. Dobrushin, and V. Nadal, "Mobile security framework (mobsf)." [Online]. Available: https://github.com/MobSF/Mobile-Security-Framework-MobSF

[13] E. Corral, "Amenaza silenciosa iii – hooking (teoría)." [Online]. Available: https://www.gnu.org/software/gdb/

[14] GNU, "Gdb: The gnu project debugger," 12 2015. [Online]. Available: https://securityinside.info/amenaza-silenciosa-iii-hooking-teoria/

[15] A. Hoog and J. McCash, *Android forensics: investigation, analysis, and mobile security for Google Android*. Syngress, 2011.

[16] E. Eilam, *Reversing: the hackers guide to reverse engineering*. Wiley, 2005.

[17] Z. Han, L. Cheng, Y. Zhang, S. Zeng, Y. Deng, and X. Sun, "Systematic analysis and detection of misconfiguration vulnerabilities in android smartphones," *2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications*, 2014.

[18] W. You, B. Liang, W. Shi, S. Zhu, P. Wang, S. Xie, and X. Zhang, "Reference hijacking," *Proceedings of the 38th International Conference on Software Engineering - ICSE 16*, 2016.

[19] internetmania.net, "Qué es el sniffing." [Online]. Available: http://www.internetmania.net/int0/int93.htm

[20] C. Tumbleson, "Apktool." [Online]. Available: https://ibotpeaches.github.io/Apktool/documentation/

[21] skylot, "Jadx." [Online]. Available: https://github.com/skylot/jadx

[22] V. M. Alvarez, "Yara." [Online]. Available: https://virustotal.github.io/yara/

[23] "Kit de desarrollo nativo." [Online]. Available: https://developer.android.com/ndk/guides/

[24] A. DuVander, "What programming language is most popular with apis?" [Online]. Available: https://www.programmableweb.com/news/what-programming-language-most-popular-apis/2013/06/03

APÉNDICES

A.1. Estructura de la herramienta

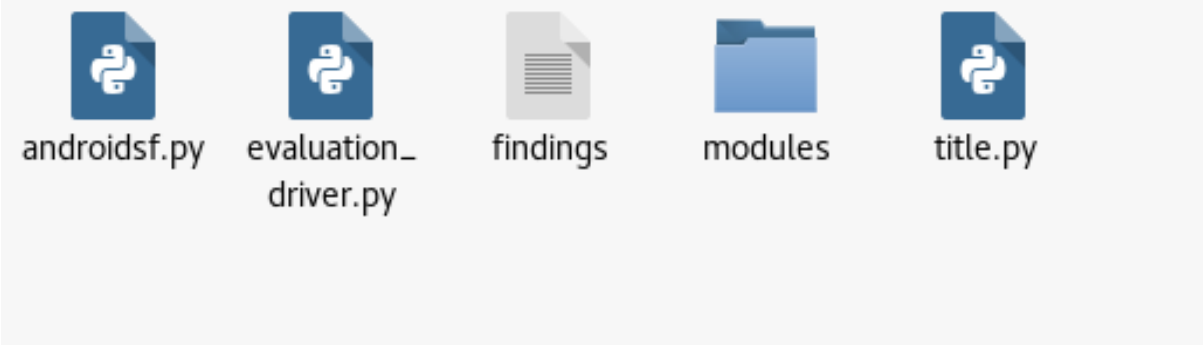


Fig. 4: Directorio base de la herramienta

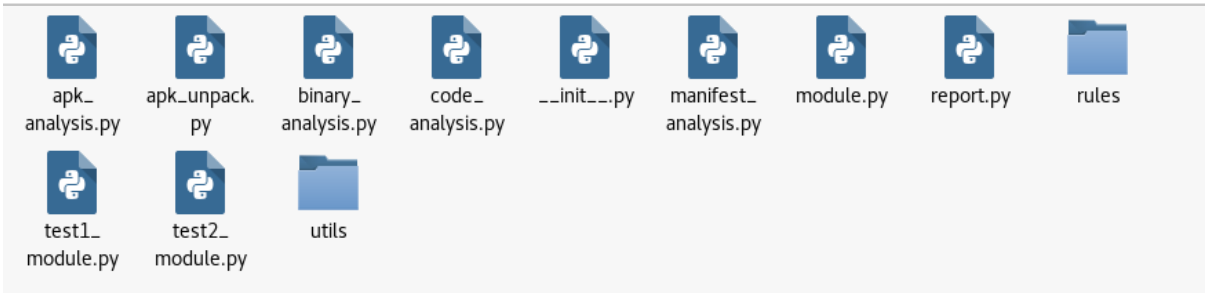


Fig. 5: Directorio *modules*

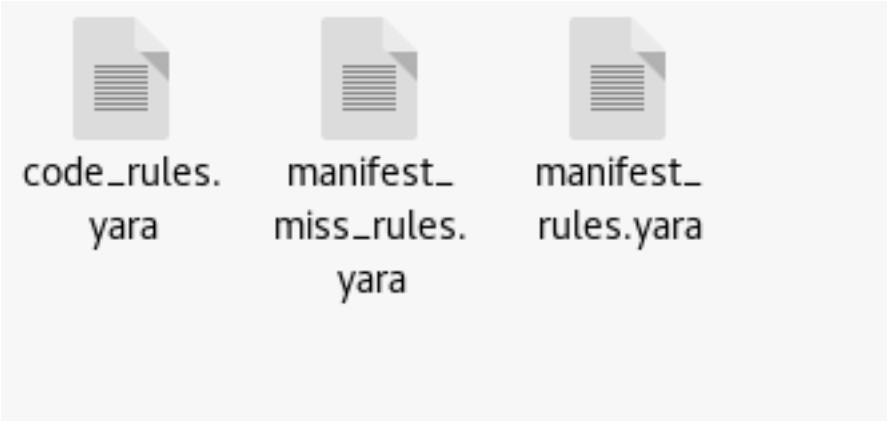


Fig. 6: Directorio *rules*

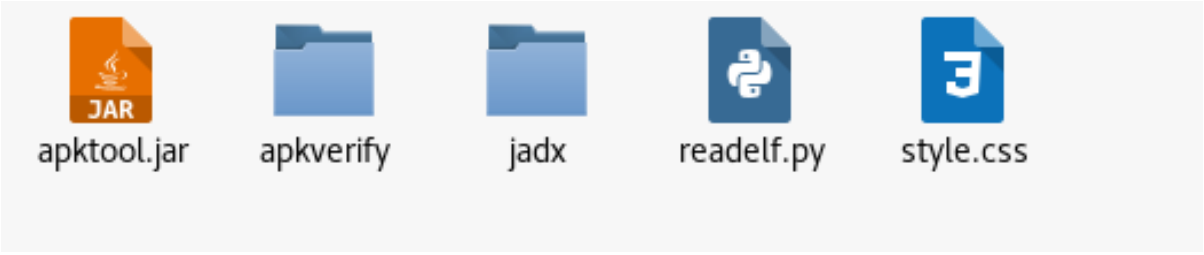


Fig. 7: Directorio *utils*

A.2. Extractos de informe

Report

Information

Working directory: /root/Desktop/tfg/wd 2018-06-16_16-12-57

Provided APK: /root/Desktop/tfg/Android-Security-Framework/testapps/net.inverline.bancosabadell.officelocator.android.apk

APK fingerprints:

sha256: 8f5263fe3afd71077f8d7e11f1fac3518ad2c57d2e64bb32e324edeccd80836b

sha1: 3a07301be8f41294d96e05a0a874c003f26a85ec

md5: 2d92d05a4e1735dc64c1723151d5f46c

Fig. 8: Cabecera del informe

APK Analysis

V1

Certificates

Certificate:

Data:

Version: 3

Serial Number: 4b:30:aa:54

Signature Algoritm: sha1WithRSAEncryption

Issuer: 'C=ES, ST=Barcelona, L=Sabadell, O=BANCO DE SABADELL SA, OU=Direcci\xc3\xb3 Aplicacions d\xe2\x80\x99Acc\xc3\xa9s, CN=www.sabadellatlantico.mobi'

Valididty

Not Before: 2009-12-22 11:15:32 GMT

Not After: 2059-12-10 11:15:32 GMT

Subject: 'C=ES, ST=Barcelona, L=Sabadell, O=BANCO DE SABADELL SA, OU=Direcci\xc3\xb3 Aplicacions d\xe2\x80\x99Acc\xc3\xa9s, CN=www.sabadellatlantico.mobi'

Subject Public Key Info

Public Key Algorithim: rsaEncryption

Public Key

Bits: 1024

Modulus:

00:c5:f5:f7:fd:6d:94:03:a6:e9:fb:5f:95:97:af:43:a7:7f:a1:5c:
3d:3d:0b:11:ae:d9:2c:47:98:35:dc:66:66:e2:1a:7e:df:6d:f0:5b:
a4:7e:fa:08:3c:61:a6:70:55:df:a0:2a:de:9e:ea:4e:ab:6b:d5:c5:
62:5d:c0:de:00:97:ce:0e:b0:a8:f5:09:6b:62:6c:5b:2b:b0:fa:cb:
b6:36:c4:7f:99:1e:97:dd:22:c5:61:87:38:e8:3d:d7:75:3b:5d:e8:
07:56:61:73:72:5c:a0:53:e4:33:3d:3d:cb:5c:5a:7f:91:3c:11:18:
2b:63:4b:f2:61:b4:07:96:27

Exponent: 65537

Extensions

Thumbprints

MD5 04:2D:FF:87:90:1C:A1:4A:E5:93:FF:11:73:BF:BC:DC

Sha1 04:62:FB:36:D2:08:D9:E4:73:56:49:9D:76:D0:00:AE:5F:BF:C1:E0

Sha256 28:6C:6A:F6:12:3F:DE:08:27:67:38:CB:8F:D3:5E:1B:3F:AA:32:F5:B0:ED:46:B8:09:FB:C6:35:5C:35:A2:3F

Signatures

Signature :

9fad75fdbb116a40a5f47063c8016894e8c6d0f427897c1d07570e7a59d5
bf9043cc4c49ebc3088a1173ba06184e2521b75002c7a69b9f5df67c390a
939fb659899d015bcada112b1f99bd91d2a6be10ccbffb34df183553fabe
09423a7b7d318b05fad89446176 8ce67d40b0fa1f6dfe2c196809bb0e42d
9a328d8e4f64cd60

Fig. 9: Sección de firmas

Manifest Analysis

Manifest data

Package name: net.inverline.bancosabadell.officelocator.android

Main activity: net.inverline.bancosabadell.officelocator.android.sab.SplashActivity

Min SDK: 14

Max SDK:

Target SDK: 19

Android version: 1414093

Android version name: 15.3.2

Icons:

@drawable/ic_launcher

Services:

net.inverline.bancosabadell.officelocator.android.sab.push.GcmNotificationHandlerIntentService

Activities:

net.inverline.bancosabadell.officelocator.android.sab.SplashActivity

net.inverline.bancosabadell.officelocator.android.sab.ui.launcher.LauncherActivity

net.inverline.bancosabadell.officelocator.android.sab.ui.customer.CustomerActivity

net.inverline.bancosabadell.officelocator.android.sab.ui.customer.PromoLayerActivity

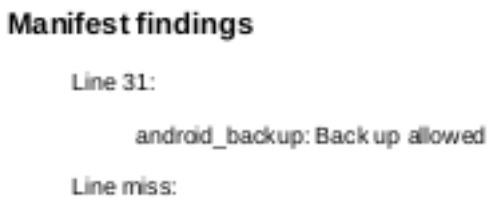
net.inverline.bancosabadell.officelocator.android.sab.ui.html.HtmlCustomerActivity

net.inverline.bancosabadell.officelocator.android.sab.ui.html.facade.HtmlFacadeActivity

net.inverline.bancosabadell.officelocator.android.sab.ui.vtpc.VtpcGenerationActivity

net.hockeyapp.android.UpdateActivity

Fig. 10: Información del AndroidManifest.xml



Manifest findings

Line 31:

android_backup: Backup allowed

Line miss:

Fig. 11: Hallazgos en el AndroidManifest.xml

Code Analysis

Total files inspected: 2571.

Total lines inspected: 235224.

Code files

java/sources/org/iban4j/'IbanFormatException.java'

java/sources/org/iban4j/'CountryCode.java'

java/sources/org/iban4j/'IbanUtil.java'

java/sources/org/iban4j/'InvalidCheckDigitException.java'

java/sources/org/iban4j/'UnsupportedCountryException.java'

java/sources/org/iban4j/support/'Assert.java'

java/sources/org/iban4j/bban/'BbanStructureEntry.java'

java/sources/org/iban4j/bban/'BbanStructure.java'

Fig. 12: Archivos y lineas de código analizados

Fig. 13: Hallazgos en el código

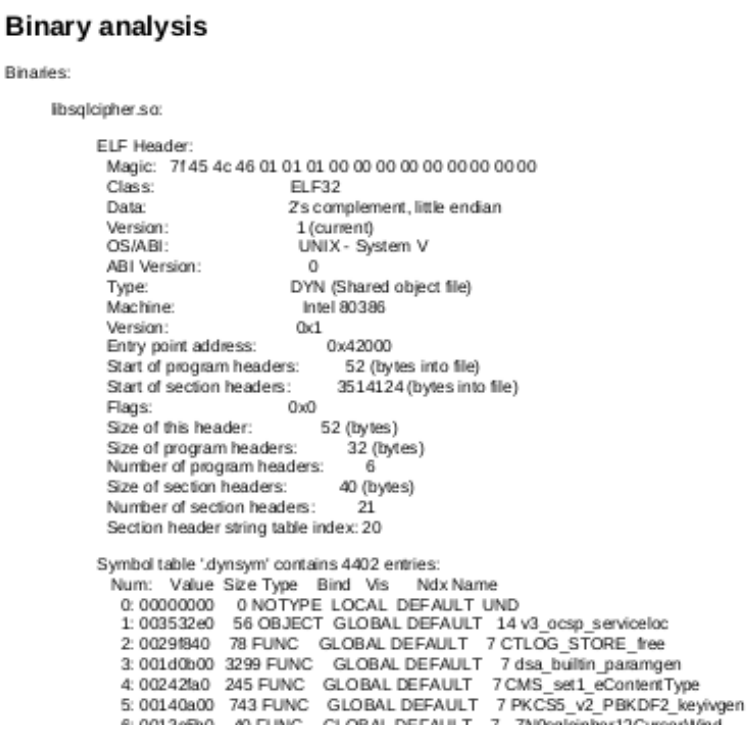


Fig. 14: Información relativa a los binarios