



**Escuela de Ingenierías Industrial, Informática y
Aeroespacial**

**MÁSTER UNIVERSITARIO EN
INVESTIGACIÓN EN CIBERSEGURIDAD**

Trabajo de Fin de Máster

**Análisis de la presencia de malware en aplicaciones
Android obtenidas de mercados alternativos**

**Analysis of the malware incidence in Android apps
obtained from alternative markets**

Autor: Sara El Kortbi Martínez

Tutor: Ángel Manuel Guerrero Higueras

Cotutor: Adrián Campazas Vega

(Septiembre, 2022)

UNIVERSIDAD DE LEÓN
Departamento de Matemáticas
MÁSTER UNIVERSITARIO EN INVESTIGACIÓN EN CIBERSEGURIDAD
Trabajo de Fin de Máster

ALUMNA: Sara El Kortbi Martínez

TUTOR: Ángel Manuel Guerrero Higueras

COTUTOR: Adrián Campazas Vega

TÍTULO: Análisis de la presencia de malware en aplicaciones Android obtenidas de mercados alternativos

TITLE: Analysis of the malware incidence in Android apps obtained from alternative markets

CONVOCATORIA: Septiembre, 2022

RESUMEN:

Con el aumento del uso de los smartphones en los últimos años, el malware que tiene como objetivo sistemas Android se ha visto incrementado. Este tipo de malware se distribuye principalmente a través de los denominados mercados de aplicaciones, mediante los cuales cualquier usuario puede descargar e instalar aplicaciones en su dispositivo Android.

El mercado oficial de Android es Google Play, sin embargo, existen muchos otros mercados alternativos. Es común que en este tipo de mercados exista una oferta de aplicaciones modificadas, denominadas *mods*. Estas son aplicaciones oficiales manipuladas por usuarios con cierto conocimiento con el objetivo de eliminar anuncios u ofrecer aplicaciones que son de pago en el mercado oficial de forma gratuita. En el presente trabajo se ha realizado un estudio acerca de la presencia de malware en los mercados que ofrecen este tipo de aplicaciones.

Para ello se ha realizado, en primer lugar, un estudio del estado del arte para determinar las características más importantes a la hora de detectar si una aplicación contiene malware. A partir de dicho estudio, se ha elaborado una herramienta que permite realizar un primer análisis automático de una muestra de aplicaciones recopiladas de distintos mercados en base a unas determinadas características. Esto ha permitido determinar cuáles de las aplicaciones podrían contener malware. Posteriormente, se ha realizado un análisis estático de aquellas aplicaciones más sospechosas. Los análisis realizados sobre las muestras seleccionadas indican que cerca del 70 % de las aplicaciones analizadas muestran algún indicio de presencia de malware. Esto supone un riesgo de seguridad elevado para los usuarios que consumen este tipo de aplicaciones.

ABSTRACT:

With the increased use of smartphones in recent years, malware targeting Android systems has been on the rise. This type of malware is mainly distributed through the so-called app markets, through which any user can download and install apps on their Android device. The official Android marketplace is Google Play, however, there are many other alternative marketplaces. It is common that in this type of markets there is an offer of modified applications, called "mods". These are official applications manipulated by users with certain knowledge with the aim of removing ads, payment requirements or adding new features. In the present work, a study has been carried out on the presence of malware in the marketplaces that offer this type of applications.

To achieve this, a review of the state of the art was performed to determine the most important characteristics to detect malware in an application. Based on this study, a tool was developed to perform an initial automatic analysis of a sample of applications collected from different markets. This made it possible to determine which of the applications could contain malware. Subsequently, a static analysis of the most suspicious applications was performed.

The analyses performed on the selected samples indicate that nearly 70 % of the analyzed applications show some indication of the presence of malware. This represents a high security risk for users who consume this type of applications.

Palabras clave: Malware, Android, apps

Firma de la alumna:



VºBº Tutor:

Índice general

Índice de figuras	v
Índice de tablas	vii
Glosario de términos	viii
Introducción	1
1. Estudio del problema	4
1.1. El contexto del problema	4
1.1.1. El sistema operativo Android	4
1.1.2. Malware en Android	10
1.1.3. Google Play y mercados alternativos	12
1.2. El estado de la cuestión	13
1.3. La definición del problema	14
2. Gestión de proyecto software	16
2.1. Alcance del proyecto	16
2.1.1. Definición del proyecto	16
2.1.2. Estimación de tareas y de recursos	17
2.1.3. Presupuesto	18
2.2. Plan de trabajo	20
2.2.1. Identificación de tareas	20
2.2.2. Estimación de tareas	20
2.2.3. Planificación de tareas	21
2.3. Gestión de recursos	22
2.3.1. Especificación de recursos	22
2.3.2. Asignación de recursos	22
2.4. Gestión de riesgos	22

2.4.1. Identificación de riesgos	23
2.4.2. Análisis de riesgos	23
2.5. Legislación y normativa	24
3. Solución	26
3.1. Descripción de la solución	26
3.1.1. Selección de los mercados	27
3.1.2. Selección de las aplicaciones	29
3.1.3. Descripción de la herramienta	34
3.2. El proceso de desarrollo	36
3.2.1. Análisis de requisitos	36
3.2.2. Diseño	36
3.2.3. Implementación	38
3.3. El producto del desarrollo	40
4. Evaluación	44
4.1. Proceso de evaluación	44
4.1.1. Forma de evaluación	47
4.1.2. Análisis Badoo Techbigs	48
4.1.3. Conclusiones	65
4.1.4. Análisis Netflix TechBigs	66
4.1.5. Conclusiones	77
4.1.6. Análisis DisneyPlus APKDone	77
4.1.7. Conclusiones	87
4.1.8. Análisis Amazon Techbigs	87
4.1.9. Conclusiones	90
Conclusión	91
Lista de referencias	95
A. Control de versiones	100
B. Seguimiento de proyecto fin de máster	102
B.1. Forma de seguimiento	102
B.2. Planificación inicial	102
B.3. Planificación final	103

Índice de figuras

1.1. Arquitectura de Android	6
1.2. Arquitectura de Android	7
2.1. Planificación de tareas	21
3.1. Ejemplo de resultado búsqueda	28
3.2. Diagrama de casos de uso	37
3.3. Diagrama de componentes	37
3.4. Diagrama de secuencia	38
3.5. Ejemplo de ejecución del programa	40
3.6. Ejemplo de json generado para la aplicación de Tantan original	42
3.7. Extracto del reporte generado en pdf	43
4.1. Aplicación de Badoo en mercado de TechBigs	49
4.2. Librerías Badoo TechBigs	49
4.3. Icono Badoo TechBigs	50
4.4. Referencia a dirección IP	50
4.5. Origen de la dirección IP	51
4.6. Paquete Appsflyer	51
4.7. Paquete Banalytics	52
4.8. Paquete dts.freefireth	52
4.9. Resultado de búsqueda de dts.freefireth	53
4.10. Clase RootCheck.class	53
4.11. Función checkIsRealPhone	53
4.12. Función readCpuInfo	54
4.13. Función hasBluetooth	54
4.14. Función GetSensorInfo	54
4.15. Función writeFile	55
4.16. Función checkAccessRootData	56

4.17. Función checkBusybox	56
4.18. Función executeCommand	57
4.19. Función checkSuperuserApk	57
4.20. Función checkRootPathSU	58
4.21. Función checkGetRootAuth	58
4.22. Función sendEvents	59
4.23. Función sendEventsIfNeeded	60
4.24. Función getPendingEvents	60
4.25. Paquete Beetalk	61
4.26. Búsqueda de Beetalk	61
4.27. Dirección IP	62
4.28. Paquete gamesafe.ano	62
4.29. Función sendCmd	63
4.30. Paquete pay.android	64
4.31. Paquetes de otras redes sociales	64
4.32. Referencia a nombre del mercado en base64	65
4.33. Paquetes Netflix Techbigs	66
4.34. Paquete teamseries.lotus	67
4.35. Código de la aplicación ofuscado	67
4.36. Presencia de anuncios en la aplicación	68
4.37. Referencia a mraid.js	68
4.38. Referencia a unityads de China	69
4.39. Método isChinaLocale	69
4.40. Método de comprobación de acceso root	70
4.41. API Keys	72
4.42. Descarga de subtítulos	73
4.43. Conexiones a localhost	74
A.1. Repositorio	100
A.2. Repositorio	101
B.1. Planificación incial	103
B.2. Planificación final	103

Índice de tablas

1.1.	Permisos en Android	7
2.1.	Costes de personal	18
2.2.	Costes totales de personal	18
2.3.	Coste de materiales	19
2.4.	Coste total del proyecto	19
2.5.	Estimación de las tareas	21
2.6.	Ánalisis de riesgos	23
3.1.	Mercados seleccionados y características	28
3.2.	Clasificación de aplicaciones en Google Play	29
4.1.	Resultados del análisis de Amazon Prime Video (MOD)	45
4.2.	Resultados del análisis de Disney Plus (MOD)	45
4.3.	Resultados del análisis de Youtube (MOD)	46
4.4.	Resultados del análisis de Netflix (MOD)	46
4.5.	Resultados del análisis de Adobe (MOD)	46
4.6.	Resultados del análisis de Tinder	47
4.7.	Resultados del análisis de Badoo	47

Glosario de términos

Malware : Programa malicioso que realiza intencionalmente acciones dañinas en un sistema y sin conocimiento del usuario.

ciberseguridad : Protección de los sistemas informáticos y de sus redes de comunicaciones, con el objetivo de mantener segura la información que procesan.

DEX : Dalvik Executable. Es un archivo de código compilado de una aplicación de Android.

APK : Android Application Package. Paquete para el sistema operativo Android.

Mod : Aplicación modificada creada a partir de una aplicación original, con el objetivo de añadir ciertas características nuevas, eliminar pagos, anuncios, etc.

ART : Android Run Time. Entorno de ejecución de aplicaciones de Android.

HAL : Hardware Abstraction Layer. Define una interfaz entre los servicios del sistema y los controladores.

JAR : Java Archives Package. Paquete que agrupa varias clases de Java, metadatos otros recursos.

Python : Lenguaje de programación de alto nivel, de propósito general e interpretado.

Emulador : Software que permite ejecutar programas o aplicaciones creados para una determinada plataforma en otra distinta.

Introducción

La rápida evolución de las nuevas tecnologías ha desencadenado un gran avance en el desarrollo de dispositivos móviles, lo que ha conllevado un gran aumento en su uso por parte de la población. Actualmente se estima que el 83 % de las personas tiene un smartphone. [1].

Este aumento en el uso de smartphones ha atraído la atención de los atacantes. De acuerdo con un estudio de Kaspersky [2], en 2021 se detectaron 3.464.756 paquetes maliciosos instalados en dispositivos móviles. Este número es significativamente más bajo que en 2020, año en el que se detectaron 5.683.694. Sin embargo, sigue siendo una amenaza muy grande para los usuarios de estos dispositivos.

Por otra parte, la detección de malware también ha visto mejoras en los últimos años. Con los avances en las ramas de aprendizaje automático, se han podido desarrollar nuevos métodos de detección más eficaces. Sin embargo, como se verá a lo largo de este documento, todavía queda mucho trabajo por hacer.

El malware generalmente se propaga a través de los mercados de aplicaciones. El mercado oficial de Android es el Google Play Store [3], sin embargo, existen numerosos mercados alternativos a través de los cuales se pueden obtener aplicaciones. El sistema operativo de Android es inherentemente menos seguro que otros debido a que no impone restricciones a la hora de instalar aplicaciones de terceros provenientes de mercados no oficiales, las cuales se pueden instalar con relativa facilidad.

Muchas de las aplicaciones existentes en los mercados alternativos son *mods* o aplicaciones modificadas. Este tipo de aplicaciones son aplicaciones oficiales que han sido modificadas para que el usuario obtenga algún tipo de beneficio. Por ejemplo, eliminar anuncios u obtener una versión premium. Generalmente estas aplicaciones son más propensas a tener virus [22] [23].

Planteamiento del problema

En este trabajo se propone realizar un estudio acerca de la presencia de malware en mercados alternativos. Para ello se realizará una selección de mercados y de aplicaciones que se descargarán de cada mercado. Para determinar la presencia de malware, se compararán dichas aplicaciones con sus versiones oficiales obtenidas de Google Play Store. Posteriormente, aquellas aplicaciones más sospechosas serán analizadas más en profundidad.

El trabajo se ha centrado en el sistema operativo de Android por ser el más utilizado [4] y por permitir la instalación de forma sencilla de aplicaciones provenientes de mercados no oficiales, tal y como se ha explicado anteriormente. Además, se ha centrado principalmente en el análisis de aplicaciones modificadas, ya que son el tipo de aplicaciones más propensas a contener malware.

Para realizar la comparación entre los *mods* y las aplicaciones originales, se ha desarrollado una herramienta que permite comparar de forma sencilla diferentes características de dos aplicaciones dadas. Además, esta herramienta también realiza un primer análisis de malware básico y genera un reporte con los resultados. A continuación, se ha realizado un análisis estático manual de aquellas aplicaciones más sospechosas para confirmar la presencia de malware.

Objetivos

El objetivo principal de este trabajo es realizar un estudio comparativo entre una muestra de aplicaciones legítimas obtenidas del mercado oficial de Android, Google Play, y las mismas aplicaciones de una serie de mercados no oficiales, así como realizar un análisis estático de las últimas para comprobar si contienen malware. El objetivo final es determinar si la existencia de malware en aplicaciones descargadas de páginas no oficiales es sustancial como para suponer un problema de seguridad, y poder concienciar a los usuarios de los dispositivos móviles de los peligros de descargar este tipo de aplicaciones. Para el cumplimiento de estos objetivos, se han detectado los siguientes subobjetivos:

1. Realización de un estado del arte en materia de detección de malware en Android
2. Desarrollo de una herramienta de comparación de aplicaciones

3. Identificación de las posibles aplicaciones sospechosas y realización de un análisis estático del código de las mismas.

Metodología

Como metodología de desarrollo se ha utilizado la metodología ágil SCRUM. Se ha escogido esta por proporcionar gran rapidez en la identificación de las incidencias que tengan lugar a lo largo del proyecto, así como una mayor adaptabilidad de las funciones de la herramienta en caso de variación de los requisitos.

Tal y como establece esta metodología, la cual se explicará con más detalle en el capítulo 2, se han mantenido reuniones semanales con los tutores del proyecto para comentar los avances del proyecto y los siguientes pasos que se seguirán en cada sprint.

Estructura del trabajo

El trabajo se estructura de la siguiente forma:

1. En el capítulo 1 se introduce el problema identificado y su contexto, así como el estado del arte en la materia de dicho problema.
2. En el capítulo 2 se explica la gestión de proyecto software. Se detalla el alcance del proyecto, el plan de trabajo a seguir, la gestión de los recursos y la gestión de los riesgos.
3. En el capítulo 3 se describe la solución y el proceso de desarrollo.
4. En el capítulo 4 se comenta el proceso de evaluación, en el cual se detallarán los resultados de los análisis realizados de las aplicaciones que contenían indicios de la presencia de malware.
5. Por último, se comentan las conclusiones del trabajo.

Capítulo 1

Estudio del problema

En este capítulo se expondrá el contexto que llevó a la identificación del problema y se detallará el estudio del estado del arte en materia de malware en Android realizado. Posteriormente, se definirá en detalle el problema que se ha tratado en este trabajo.

1.1. El contexto del problema

En primer lugar, se comentará en detalle el funcionamiento del sistema operativo de Android, su arquitectura, el malware existente para este sistema y los procedimientos para analizar aplicaciones. Por otra parte, se hablará del mercado oficial de aplicaciones y de otros mercados alternativos.

1.1.1. El sistema operativo Android

A lo largo de los últimos años se ha extendido notablemente el uso de dispositivos móviles o smartphones [1], almacenándose cada vez más información confidencial en ellos y haciendo que sean un objetivo muy atractivo para los atacantes.

El sistema operativo Android es uno de los más utilizados hoy en día [4]. Está basado en el kernel de Linux y actualmente es desarrollado por Google. Es un proyecto abierto, y eso hace que muchos desarrolladores y clientes se decanten por esta opción.

La arquitectura de Android se podría dividir en cuatro capas, tal y como se puede observar en la Figura 1.1. En la capa más superficial se encuentran las propias aplicaciones de Android, las cuales proporcionan la mayor parte de la funcionalidad

disponible para el usuario. En la siguiente capa se encuentra la capa de aplicación, la cual provee a los desarrolladores de distintos tipos de servicios y sistemas para desarrollar aplicaciones, como por ejemplo un manejador de notificaciones, un manejador de ventanas, etc.

La siguiente capa está constituida por el ART (*Android Run Time*) y librerías de Java. Esta capa permite al sistema ejecutar aplicaciones escritas en Java. Las clases de Java son compiladas en archivos *.dex* que son ejecutados por varias máquinas virtuales. A continuación, se encuentra la capa HAL (*Hardware Abstraction Layer*). Esta capa define diferentes interfaces que los distintos proveedores de hardware deben implementar de forma que no sea necesario modificar las capas superiores.

Por último, se encuentra la capa del kernel de Linux, el cual es responsable de diferentes funciones como el manejo de memoria, de procesos, sistema de archivos, etc.

Las aplicaciones de Android utilizan el formato *apk*, que es en esencia un archivo comprimido que contiene los archivos *.dex*, los recursos, las librerías y el archivo de manifiesto. Los archivos *apk* deben ser firmados por el desarrollador, de forma que se pueda identificar al autor y realizar actualizaciones de forma sencilla. Las aplicaciones pueden estar firmadas por un tercero o autofirmadas.

A diferencia de iOS, en Android no se requiere ser root o tener permisos de administrador para poder instalar aplicaciones de terceros provenientes de otros mercados. La opción está desactivada por defecto, pero es sencillo activarla sin poseer ningún conocimiento especial.

La principal característica de seguridad de Android radica en el modelo de permisos. Cuando un usuario instala una aplicación, este debe aceptar manualmente una serie de permisos que la aplicación solicita, de forma que el usuario autoriza o no a la aplicación a utilizar determinados recursos del sistema. Por ejemplo, una aplicación que permita editar fotos podría necesitar acceso a la cámara. Los permisos que una aplicación necesita se especifican en el archivo *AndroidManifest.xml*.

Por otra parte, en Android existen los permisos de ejecución, también denominados permisos peligrosos, ya que acceden a información sensible del usuario, como pueden ser por ejemplo el acceso a la localización, a la cámara o al micrófono. Este tipo de permisos, además de ser declarado en el archivo de manifiesto, deben pedirse de forma explícita al usuario. En la Figura 1.2 se muestra el flujo de trabajo con los permisos de Android.

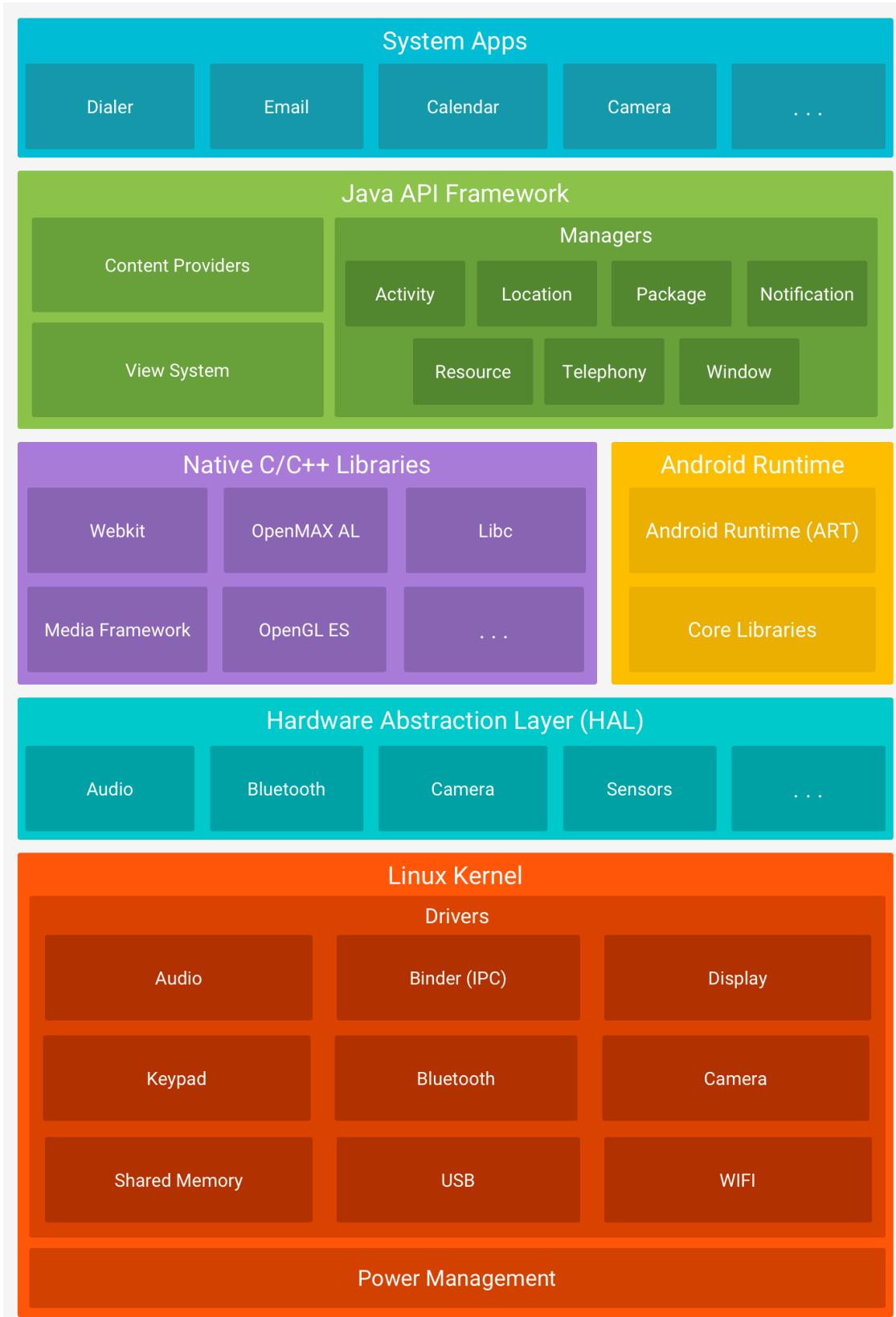


Figura 1.1: Arquitectura de Android

Fuente: <http://geraintw.blogspot.com.es/2012/09/cia-infosec.html>

**Figura 1.2:** Flujo de trabajo con los permisos de AndroidFuente: <https://developer.android.com/guide/topics/permissions/overviewruntime>

A continuación, se muestra una tabla en la que se resumen los permisos en tiempo de ejecución de Android y su descripción:

Tabla 1.1: Permisos en Android

Permiso	Descripción
ACCESS_COARSE_LOCATION	Permite a una aplicación acceder a la localización aproximada
ACCESS_FINE_LOCATION	Permite a una aplicación acceder a la localización exacta
ACCESS_MEDIA_LOCATION	Permite a una aplicación acceder a cualquiera de las localizaciones geográficas almacenadas por el usuario
ACTIVITY_RECOGNITION	Permite a una aplicación reconocer actividad física
ADD_VOICEMAIL	Permite a una aplicación añadir mensajes de voz al sistema
ANSWER_PHONE_CALLS	Permite a una aplicación responder a una llamada entrante
BLUETOOTH_ADVERTISE	Permite a una aplicación hacer que el dispositivo se muestre a otros dispositivos Bluetooth
BLUETOOTH_CONNECT	Permite a la aplicación conectar dos dispositivos por Bluetooth

BLUETOOTH_SCAN	Permite a la aplicación buscar otros dispositivos Bluetooth
BODY_SENSORS	Permite a la aplicación acceder a datos de los sensores que permiten al usuario medir ciertas constantes de su cuerpo. Por ejemplo, sus latidos.
BODY_SENSORS_BACKGROUND	Este permiso es similar al anterior, pero permite el acceso cuando la aplicación no está activa. Sin embargo, para poder tener acceso a los sensores sigue siendo necesario el permiso anterior.
CALL_PHONE	Permite a una aplicación realizar llamadas sin que el usuario tenga que confirmarlas
CAMERA	Permite a una aplicación tener acceso a la cámara/cámaras del dispositivo
GET_ACCOUNTS	Permite a una aplicación acceder a la lista de cuentas del Accounts Service.
NEARBY_WIFI_DEVICES	Este permiso se necesita para poder conectarse a dispositivos cercanos vía WiFi.
POST_NOTIFICATIONS	Permite a una aplicación enviar notificaciones.
PROCESS_OUTGOING_CALLS	Permite a la aplicación ver el número marcado durante una llamada con la opción de redireccionar la llamada a otro número o abortar la llamada completamente.
READ_CALENDAR	Concede a la aplicación permisos de lectura del calendario del usuario.
READ_CALL_LOG	Permite a una aplicación leer los registros de llamadas del usuario.
READ_CONTACTS	Permite a una aplicación leer la información de los contactos del usuario.

READ_EXTERNAL_STORAGE	Concede a una aplicación permisos de lectura sobre el almacenamiento externo.
READ_MEDIA_AUDIO	Concede a una aplicación permisos de lectura sobre el audio del almacenamiento externo.
READ_MEDIA_IMAGES	Concede a una aplicación permisos de lectura sobre las imágenes del almacenamiento externo.
READ_MEDIA_VIDEO	Concede a una aplicación permisos de lectura sobre los vídeos del almacenamiento externo.
READ_PHONE_NUMBERS	Concede a una aplicación permisos de lectura del número/s de teléfono del dispositivo.
READ_PHONE_STATE	Permite a una aplicación acceder al estado del dispositivo, incluyendo la información de red, las llamadas, la lista de cuentas del dispositivo, etc.
READ_SMS	Permite a una aplicación leer mensajes SMS.
RECEIVE_MMS	Permite a una aplicación monitorizar los mensajes MMS.
RECEIVE_SMS	Permite a una aplicación recibir mensajes SMS.
RECEIVE_WAP_PUSH	Permite a una aplicación recibir mensajes WAP.
RECORD_AUDIO	Permite a una aplicación grabar audio.
SEND_SMS	Permite a una aplicación enviar mensajes SMS.
USE_SIP	Permite a una aplicación utilizar el servicio SIP.
UWB_RANGING	Este permiso es requerido para poder utilizar ultra-ancho de banda- Required to be able to range to devices using ultra-wideband.

WRITE_CALENDAR	Permite a una aplicación escribir en el calendario del usuario.
WRITE_CALL_LOG	Permite a una aplicación escribir (pero no leer) en los registros de llamadas del usuario.
WRITE_CONTACTS	Concede a la aplicación permisos de escritura en los contactos del usuario.
WRITE_EXTERNAL_STORAGE	Permite a una aplicación escribir en el almacenamiento externo.

1.1.2. Malware en Android

Existen diferentes categorías en las cuales se pueden agrupar la mayoría de malware para dispositivos Android existente hoy en día. A continuación, se describen un total de 11 categorías recopiladas del artículo de IT World Canada [5].

1. **File Infector.** Este es un tipo de malware que se instala junto al APK y se ejecuta con ella. Colecciona diversos datos del dispositivo, y generalmente puede interaccionar con otras aplicaciones y configuraciones.
2. **Billing Fraud.** Este malware tiene como objetivo obtener dinero del usuario. Para ello el malware podría enviar SMS o realizar llamadas premium sin consentimiento, o engañar al usuario para que se suscriba o adquiera determinado contenido.
3. **Riskware.** El Riskware es un programa legítimo pero que supone un riesgo de seguridad en el dispositivo. La aplicación podría robar información del dispositivo, redirigir al usuario a webs maliciosas, instalar aplicaciones maliciosas, o modificar configuraciones del sistema.
4. **Ransomware.** Es un tipo de malware que cifra los archivos del dispositivo haciendo que sean inaccesibles para el usuario.
5. **Adware.** Este tipo de malware está relacionado con la publicidad. Su objetivo es mostrar al usuario anuncios y tratar de que el usuario interaccione con ellos, ya que de esta forma el desarrollador del adware obtiene generalmente beneficio económico. Además, es común que este tipo de aplicaciones coleccionen información personal.

6. **Backdoor.** El objetivo detrás de este tipo de malware es lograr obtener una “puerta trasera” que le dé al atacante acceso al dispositivo en cualquier momento, de forma que se puedan realizar ataques remotos sin necesitar acceso físico al dispositivo. Este tipo de malware suele colecciónar información personal, obtener acceso a mensajes, llamadas, etc.
7. **Scareware.** El scareware pretende asustar o preocupar al usuario para tratar de que este instale determinadas aplicaciones maliciosas. Por ejemplo, un scareware podría mostrar un anuncio por pantalla de que el dispositivo tiene un virus y es preciso instalar un antivirus.
8. **Spyware.** Es un malware cuyo objetivo es robar información confidencial del usuario.
9. **Troyano.** Este malware se comporta como un programa legítimo, sin embargo, roba información del dispositivo y realizan otro tipo de acciones peligrosas sin que el usuario tenga conocimiento.
10. **DDOS.** Malware que ejecuta una denegación de servicio (DoS) como parte de un ataque de DoS distribuido sin que el usuario tenga conocimiento de ello.
11. **Phishing.** Este tipo de malware engaña al usuario haciéndose pasar por una entidad real para robar los datos de autenticación del usuario.
12. **PUA (Potentially Unwanted Applications).** Este tipo de aplicaciones generalmente vienen incluidas con otras aplicaciones genuinas gratuitas. No siempre son destructivas, pero generalmente pueden presentarse como adware, spyware u otros. Suelen colecciónar información del usuario.
13. **Rooting.** Las aplicaciones de rooting buscan obtener permisos de administrador en el dispositivo sin informar al usuario, generalmente para ejecutar código malicioso. La principal forma de expansión del malware para Android es a través de los mercados de aplicaciones. Estos contienen aplicaciones propias y de terceros que los usuarios pueden descargar e instalar en el dispositivo. La detección de malware es una tarea compleja, y los mercados no pueden permitirse realizar un análisis exhaustivo por cada aplicación que los usuarios suben a la plataforma.

Existen varios tipos de análisis de malware posibles. En primer lugar, el **análisis estático** se basa en analizar la aplicación sin ejecutarla. De esta forma, se puede obtener información acerca del código, la estructura de los archivos, los permisos, etc. La principal desventaja de este tipo de análisis es que muchas veces las aplicaciones

que contienen malware están ofuscadas de forma que no se pueda interpretar el código a simple vista.

Por otra parte, el **análisis dinámico** tiene como objetivo analizar la aplicación mientras esta está en ejecución, de forma que se pueda analizar su comportamiento al ejecutarse. La principal desventaja de este método es que algunas aplicaciones contienen métodos para detectar si una aplicación se está ejecutando en un dispositivo real o en un emulador, lo que podría hacer que cambie su funcionamiento si se trata del segundo caso. Por último, el **análisis híbrido** se basa tanto en el análisis estático como en el dinámico.

1.1.3. Google Play y mercados alternativos

El mercado de Google Play es el mercado de aplicaciones oficial y es operado por Google. Sin embargo, con la creciente popularidad de Android y la imposibilidad de acceder a este mercado desde determinados países [6] han ido surgiendo nuevos mercados alternativos.

Es común que en algunos mercados alternativos se encuentren aplicaciones conocidas que fueron modificadas para añadir algún tipo de beneficio extra al usuario. Estas comúnmente se conocen como *mods*. Una aplicación podría ser modificada para eliminar anuncios, para obtener una versión premium sin pagar, o para añadir nuevas características interesantes al usuario. Un ejemplo es el conocido Whatsapp Plus [7], que permite, entre otras funcionalidades, establecer contraseñas para leer ciertos chats, mayor personalización de la apariencia, etc.

Sin embargo, este tipo de aplicaciones también son propensas a contener malware o adware. En primer lugar, son llamativas para el usuario, ya que ofrecen algún tipo de beneficio atractivo. Por otra parte, un usuario sin experiencia aceptará fácilmente todos los permisos de la aplicación, pensando que dichos permisos son necesarios para el correcto funcionamiento de la aplicación, sin tener en cuenta las consecuencias.

Los mercados tienen una gran responsabilidad a la hora de detectar y eliminar el malware, ya que, como se ha mencionado, estos mercados son la principal vía por la cual se extiende el malware para Android. Sin embargo, muchas veces el malware no se detecta hasta que ya fue descargado por un número considerable de personas.

Google utiliza un sistema basado en aprendizaje automático para la detección de malware, denominado Google Play Protect. Antes de aparecer en el mercado, las aplicaciones son sometidas a varios tests y comprobaciones de seguridad. Google Play Protect realiza un análisis de 100 mil millones de aplicaciones diarias [8].

Se ha demostrado que incluso en el mercado oficial es posible encontrar malware, por lo que la descarga de aplicaciones nunca es completamente segura. A pesar de los esfuerzos de Google por mantener su mercado libre de malware, se han encontrado numerosas aplicaciones en el mercado de Google Play que contenían malware para robar credenciales [9].

1.2. El estado de la cuestión

La detección de malware en aplicaciones de Android es un tema ampliamente estudiado. En concreto, acerca de la seguridad de las aplicaciones descargadas en entornos alternativos, destaca el trabajo de Zhou *et al.* [10] realizado en 2012. En él los autores analizaron cerca de 200.000 aplicaciones de cinco mercados, tanto oficiales como no oficiales, y comprobaron que la presencia de malware era bastante baja. La principal característica analizada en las aplicaciones son los permisos, ya que esto permite a los autores filtrar aquellas aplicaciones que no tengan comportamientos sospechosos.

Posteriormente, realizan un análisis del comportamiento para determinar si este concuerda con el comportamiento de algún tipo de malware existente. Por último, realizan un último análisis basado en métodos heurísticos para poder detectar malware desconocido.

Otro trabajo relacionado fue el desarrollado en 2014 por Lindorfer *et al.* [11] En este trabajo los autores realizaron un análisis de distintas aplicaciones obtenidas de dieciséis mercados diferentes. En dicho estudio determinaron que aquellas aplicaciones provenientes de mercados alternativos contenían un número mucho mayor de aplicaciones con adware y malware. Posteriormente, Kikuchi *et al.* [12] realizaron un estudio similar en el que buscaban comprobar la efectividad de los operadores de mercados alternativos de identificar y eliminar del mercado las aplicaciones que contienen malware. Determinaron que en los mercados alternativos el malware tenía diez veces más posibilidades de mantenerse en el mercado que en Google Play.

En cuanto a la detección de malware en Android, recientemente han surgido nuevas aproximaciones basadas en aprendizaje automático. El trabajo de Qiu *et al.* [13] resume algunos de los métodos actuales. La mayor parte de los trabajos utilizan como características a analizar los permisos y las llamadas a las APIs, sin embargo, existen otros trabajos basados en otras características como el grafo del flujo de datos (Xu *et al.* [14]), el bytecode generado (Hota *et al.* [15]), etc.

Por otra parte, también se continúan desarrollando métodos basados en métodos tradicionales, como por ejemplo el trabajo desarrollado por Yao Du *et al* [16]. En este artículo los autores desarrollaron un método de detección de malware basado en los grafos de llamadas a funciones, los permisos sensibles y las APIs para la clasificación del malware mediante un proceso de matching con familias de malware conocidas.

A pesar de todos los métodos existentes, aun queda mucho camino por recorrer con relación a la detección de malware. Como se ha observado, todavía resulta una tarea compleja para los mercados de aplicaciones detectar de forma efectiva el malware de las aplicaciones que suben los usuarios.

1.3. La definición del problema

Como se ha podido observar a lo largo de este capítulo, el aumento progresivo del uso de smartphones ha hecho que el número de malware existente para estos dispositivos crezca significativamente. Puesto que la vía principal de distribución de este tipo de malware son los mercados de aplicaciones, es fundamental que dichos mercados comprueben exhaustivamente las aplicaciones subidas por los usuarios. Sin embargo, esta es una tarea compleja, ya que se ha visto que incluso en los mercados oficiales no todas las aplicaciones son 100 % seguras, y siempre existe un determinado riesgo.

Por otra parte, el uso de aplicaciones modificadas o mods es relativamente común y muchas personas se sienten atraídas por los beneficios que este tipo de aplicaciones ofrecen. Como se ha comentado anteriormente, este tipo de aplicaciones son especialmente propensas a contener malware.

En este trabajo se ha realizado un estudio acerca de la presencia de aplicaciones maliciosas en mercados alternativos realizando una comparación entre una serie de aplicaciones recopiladas de dichos mercados y las correspondientes originales obtenidas de Google Play, y posteriormente un análisis estático de las aplicaciones consideradas sospechosas.

Muchos de los estudios existentes realizaron el análisis basado en todo tipo de aplicaciones. Este trabajo se ha centrado principalmente en el análisis de aplicaciones modificadas que añadan algún tipo de beneficio extra para el usuario. En los casos en los que no ha sido posible encontrar una versión modificada en los mercados seleccionados se ha analizado la aplicación no modificada obtenida de dicho mercado.

En el presente trabajo se ha realizado, por tanto, un análisis de una serie de aplicaciones modificadas obtenidas de diferentes mercados para estudiar si resultaría seguro descargar este tipo de aplicaciones en mercados alternativos.

Capítulo 2

Gestión de proyecto software

En este capítulo se detallará la planificación del proyecto software. En primer lugar, en el apartado 2.1 se definirá el alcance del proyecto; en el cual se incluyen la definición del proyecto, la estimación de las tareas y los recursos, y los presupuestos. Seguidamente, en el apartado 2.2 se determinará el plan de trabajo, seguido de la gestión de los recursos y la gestión de los riesgos en los apartados 2.3 e 2.4 respectivamente.

2.1. Alcance del proyecto

2.1.1. Definición del proyecto

La herramienta desarrollada para llevar a cabo el estudio descrito en apartados anteriores es un producto software que permite realizar una comparación entre dos aplicaciones. Además, la herramienta realizará un análisis de malware básico para determinar si la aplicación sospechosa fue detectada por algún antivirus, y finalmente genera un reporte en el que se muestran los resultados obtenidos. En el capítulo 3 se describe en mayor detalle el funcionamiento de la herramienta y sus características.

La idea detrás de esta herramienta es tener ofrecer una primera visión acerca de la posibilidad de la aplicación obtenida del mercado alternativo de contener malware. Esto permite realizar un primer filtrado de las aplicaciones. Aquellas que sean sospechosas fueron analizadas posteriormente de forma manual. Dado que el estudio se basa en sistemas Android, la herramienta fue desarrollada para las aplicaciones desarrolladas para dicho sistema.

2.1.2. Estimación de tareas y de recursos

Tareas

En primer lugar, es preciso definir la metodología de trabajo que se ha seguido a lo largo del desarrollo del proyecto. Como se ha comentado anteriormente, se ha optado por seguir una metodología ágil, en concreto, la metodología SCRUM, ya que estas permiten una mayor flexibilidad y adaptabilidad durante el desarrollo del proyecto.

La metodología SCRUM sigue un modelo iterativo e incremental, al contrario que otras metodologías tradicionales basadas en cascada en las que se sigue un modelo ordenado en el que cada etapa debe ejecutarse solo después de la finalización de la anterior. Se divide el proyecto en tareas denominadas historias de usuario que generalmente implementan una funcionalidad concreta. El desarrollo se divide en *sprints*. Un sprint es un período de tiempo definido en el cual se implementan un número de historias de usuario seleccionadas.

En la metodología SCRUM es común tener reuniones denominadas *daily* (una vez al día) o *weekly* (una vez a la semana) en las cuales se comentan los avances, los problemas y, en caso de ser una reunión de sprint, se deciden las nuevas historias de usuario a implementar.

Siguiendo con la metodología descrita, durante el desarrollo del proyecto se ha dividido el proyecto en historias de usuario y se ha dividido el proceso de desarrollo en sprint. Se han mantenido reuniones semanales en las cuales se han comentando los avances realizados y las siguientes tareas a realizar. Las tareas concretas se describirán en detalle en el apartado 2.2.

Como se ha comentado, esta metodología se escogió debido a la naturaleza del proyecto y la posible variación de requisitos durante el desarrollo del mismo. SCRUM está especialmente recomendado para este tipo de proyectos.

Recursos

Los recursos necesarios para el desarrollo de la herramienta descrita son los siguientes:

- Ordenador portátil
 - Procesador: Intel Core i7
 - Memoria RAM: 32 GB

- Tarjeta gráfica: NVIDIA GeForce GTX 1060
- Almacenamiento: 250 SDD
- Sistema operativo: Windows 10
- Dispositivo Móvil
 - Sistema operativo: Android 11
 - Almacenamiento: 2 GB

2.1.3. Presupuesto

A continuación se detalla un presupuesto estimado para el coste total de este proyecto.

Coste de personal

Para calcular los costes de personal es necesario tener en cuenta a todas las personas involucradas en el desarrollo del proyecto. En este caso, el proyecto fue llevado a cabo por un equipo compuesto por los siguientes perfiles profesionales:

- **Jefe de proyecto.** Su principal tarea es supervisar el proyecto y la ejecución de las tareas, así como gestionar la carga de trabajo de cada sprint y fijar reuniones de seguimiento.
- **Desarrollador de software Junior.** Su tarea principal es el diseño y desarrollo de la herramienta software a implementar. Los salarios de ambos perfiles se reflejan en la tabla siguiente. Los datos se han obtenido de la página de Glassdoor [17]. Los costes están expresados en euros.

Tabla 2.1: Costes de personal

Rol	Bruto anual	Seguridad Social	Total anual	Coste/Hora
Jefe de proyecto	47.002	15.040	62.042	32,31
Desarrollador Junior	12.240	5.760	18.000	9,34

Los costes totales se muestran a continuación:

Tabla 2.2: Costes totales de personal

Rol	Coste/Hora	Horas	Total

Jefe de proyecto	32,31	10	323
Desarrollador Junior	9,34	150	1.401
Total	41,65	160	6.664

Coste del material

Para calcular los costes de material, se ha tenido en cuenta tanto los costes del software como del hardware.

- **Software:** Para el desarrollo del código se ha utilizado el entorno de desarrollo de Visual Studio 2017, el cual es gratuito. Además, se ha utilizado git para el control de versiones, herramienta que también es gratuita. También, se ha utilizado VirtualBox. Ninguna de las librerías o herramientas utilizadas supuso un coste, por lo que no hay costes derivados del software utilizado para el desarrollo del proyecto.
- **Equipo de trabajo:** Se utilizó un ordenador portátil, del que se estima un coste de 1500€. Además, se utilizó un dispositivo móvil del que se estima un coste de 300€. El total del material resulta en 1.500€. Se estima una vida media útil del hardware de 4 años. Puesto que se ha utilizado 4 meses para la realización del proyecto, los costes totales resultan en 125€ para el hardware, tal y como se resume en la Tabla 3.2.

Tabla 2.3: Coste de materiales

Meses de vida	Meses de uso	Total	Total proyecto
48	4	1.500	125
48	4	300	25

Coste total del proyecto

Tabla 2.4: Coste total del proyecto

Concepto	Coste (€)
Costes de personal	6.664
Costes de hardware	150

Costes Totales	6.814
----------------	-------

2.2. Plan de trabajo

En este apartado se explicará en detalle la metodología de desarrollo seguida, así como la identificación de tareas y su estimación y planificación para lograr la realización del proyecto de forma satisfactoria.

2.2.1. Identificación de tareas

Para la realización completa del trabajo se han identificado las tareas que se muestran a continuación:

1. Estudio del problema
2. Revisión del estado del arte
3. Especificación de requisitos
4. Diseño de la herramienta:
 - a) Identificación de los casos de uso
 - b) Identificación de las clases y los componentes
5. Desarrollo de la herramienta
 - a) Desarrollo de la funcionalidad de comparación
 - b) Desarrollo de la funcionalidad de análisis mediante VirusTotal
 - c) Desarrollo de la funcionalidad de generación de un reporte
6. Análisis de resultados
7. Análisis estático

2.2.2. Estimación de tareas

Se ha realizado una estimación de la duración de las tareas, la cual se puede observar en la tabla siguiente. No se han tenido en cuenta fines de semana, y se ha estimado un total de 3 horas diarias de trabajo.

Tabla 2.5: Estimación de las tareas

Tarea	Número de días	Fecha de inicio	Fecha de fin
Tarea 1	5	29 mayo de 2022	5 junio de 2022
Tarea 2	5	5 junio de 2022	12 junio de 2022
Tarea 3	2	12 junio de 2022	14 junio de 2022
Tarea 4	8	14 junio de 2022	26 junio de 2022
Tarea 4.a	4	14 junio de 2022	19 junio de 2022
Tarea 4.b	4	19 junio de 2022	26 junio de 2022
Tarea 5	15	26 junio de 2022	17 julio de 2022
Tarea 5.a	10	26 junio de 2022	10 julio de 2022
Tarea 5.b	2	10 julio de 2022	13 julio de 2022
Tarea 5.c	3	13 julio de 2022	17 julio de 2022
Tarea 6	4	17 julio de 2022	24 julio de 2022
Tarea 7	12	24 julio de 2022	7 agosto de 2022

2.2.3. Planificación de tareas

A continuación, se muestra el diagrama de Gantt que ilustra la estimación anterior:

**Figura 2.1:** Planificación de tareas

2.3. Gestión de recursos

En el siguiente apartado se definirán y asignarán los recursos necesarios para el desarrollo del proyecto.

2.3.1. Especificación de recursos

En cuanto a los recursos necesarios para el proyecto podemos distinguir los siguientes:

- Recursos humanos
 - Desarrollador Junior
 - Jefe de proyecto
- Materiales
 - Ordenador portátil
 - Dispositivo móvil con sistema operativo Android
- Económicos
 - Los recursos económicos ascienden a un total de 6.814€ tal y como se ha descrito en el apartado 2.1.3.

2.3.2. Asignación de recursos

El desarrollador contará con un ordenador portátil para la realización del estudio y la elaboración de la herramienta, y con un dispositivo móvil que permita realizar la instalación y extracción de las aplicaciones a analizar.

2.4. Gestión de riesgos

Una parte importante del desarrollo de un proyecto de software es la gestión de riesgos. Esta etapa consiste en la identificación y gestión de los posibles riesgos que puedan tener lugar durante el desarrollo del proyecto.

Para llevar a cabo dicha gestión, se ha seguido el PMBOK (Guía de los fundamentos para la dirección de proyectos) [18]. Según la definición del PMBOK, un riesgo es un “evento o condición incierta que, si se produce, tiene un efecto positivo o negativo en uno o más de los objetivos de un proyecto”.

A continuación, se detallarán los riesgos identificados y el análisis realizado de los mismos.

2.4.1. Identificación de riesgos

A continuación, se muestra la lista de los riesgos del proyecto que se han identificado:

- RSG-01: Pérdida de información
- RSG-02: Planificación temporal incorrecta
- RSG-03: Curva de aprendizaje de las tecnologías demasiado alta
- RSG-04: Aplicaciones ofuscadas cuyo análisis resulta complejo
- RSG-05: Equipo de desarrollo averiado.

2.4.2. Análisis de riesgos

Una vez identificados los riesgos, es preciso planificar una estrategia a seguir para tratar de minimizar o anular su impacto en el proyecto. En el siguiente apartado se determinará una estrategia para cada riesgo identificado anteriormente. Se seguirá alguna de las siguientes:

- **Evitación:** Se ejecutará una respuesta de manera que el riesgo no pueda progresar.
- **Contingencia:** Se planificarán una serie de medidas a ejecutar cuando el riesgo se haya dado.
- **Transferencia:** Traspaso de la gestión del riesgo a una entidad externa.
- **Mitigación:** Se planificarán una serie de acciones destinadas a reducir el impacto o la probabilidad del riesgo.
- **Aceptación:** Se asumirá el impacto asociado al riesgo.

Tabla 2.6: Análisis de riesgos

ID	Descripción	Estrategia
RSG-01	Pérdida de información	Prevención: Mantener uno o varios repositorios con la información del proyecto

RSG-02	Planificación temporal incorrecta	Aceptación: En caso de ser preciso se replanificará el proyecto
RSG-03	Curva de aprendizaje de las tecnologías demasiado alta	Aceptación: En caso de ser preciso se realizará una replanificación destinando ciertas horas al estudio de las tecnologías necesarias
RSG-04	Aplicaciones ofuscadas cuyo análisis resulta complejo	Aceptación: Se realizará una nueva selección de aplicaciones.
RSG-05	Equipo de desarrollo averiado	Prevención: Se dispondrá de un segundo equipo de trabajo en buenas condiciones

2.5. Legislación y normativa

En el siguiente apartado se describirán las leyes y las normativas aplicables en este proyecto.

En primer lugar, cabe destacar diversos artículos del Código Penal Español [19]. Los artículos 270 y 273 están relacionados con las infracciones de la propiedad intelectual y derechos afines. Puesto que, como se verá posteriormente, gran parte del malware se hace pasar por una aplicación oficial generalmente clonada y distribuida de forma ilegal, se podría afirmar que las personas que distribuyan este tipo de contenido están incurriendo en un delito contra la propiedad intelectual.

Por otra parte, muchas de estas aplicaciones contienen malware cuyo objetivo es estafar al usuario, por lo que los distribuidores de este tipo de aplicaciones también estarían incurriendo en un delito de estafa de acuerdo con los artículos 248.2, 255 y 256 del Código Penal Español.

En el caso del malware para Android que entre dentro de la clasificación de Troiano, se podría considerar que aquellos que lo distribuyen incurren en un delito contra la confidencialidad, integridad y disponibilidad de datos y sistemas informáticos, tal y como se establece en el artículo 197. En el caso de que se utilice el malware para descubrir secretos de empresa, se estaría incurriendo además en el delito descrito en el artículo 278.1.

Por último, puesto que gran parte del malware y adware existente recopila datos personales del usuario sin consentimiento, también cabe hablar del Reglamento General de Protección de Datos [20]. Este establece que la recogida de datos debe ser concisa, transparente e inteligible, y que se utilizará un lenguaje sencillo y claro para el usuario. Como se verá a lo largo de este trabajo, todas las aplicaciones analizadas que recopilaban datos del usuario no pedían el consentimiento.

Capítulo 3

Solución

En el siguiente capítulo se explicará la solución desarrollada para llevar a cabo el estudio. En primer lugar, en el apartado 3.1, se explicará el proceso de selección de los mercados y de las aplicaciones a analizar, y posteriormente se describirá la herramienta desarrollada para llevar a cabo la primera fase del análisis. Seguidamente, en el apartado 3.2 se detallará el proceso de desarrollo de dicha herramienta, tanto su diseño como su implementación.

3.1. Descripción de la solución

Como se ha visto en capítulos anteriores, determinar la presencia de malware en una aplicación no es una tarea trivial. Es común que el malware evolucione rápidamente para adaptarse y evitar ser detectado por antivirus. Además, muchas aplicaciones infectadas con malware ofuscan su código para evitar ser analizadas de forma estática, o contienen métodos para detectar si se están ejecutando en un emulador, es decir, si se está realizando algún tipo de análisis dinámico.

Se ha visto que el mercado oficial de Google Play es el más popular y a pesar de que no todas las aplicaciones son seguras al 100 % [9], existen más garantías de seguridad que las aplicaciones que se descargan de mercados no oficiales.

En este trabajo el objetivo era analizar cuán seguro es descargar aplicaciones de dichos mercados alternativos. Para ello se ha realizado una selección de algunas de las aplicaciones más populares que se pueden encontrar en el mercado de Google Play. Estas aplicaciones fueron posteriormente descargadas de una serie de mercados no oficiales que ofrezcan su versión gratuita o modificada, para poder realizar una

comparativa con sus correspondientes versiones oficiales y determinar si existe un peligro real al descargar aplicaciones de mercados no oficiales.

El estudio comparativo consta de dos fases:

- Una primera fase automática, en la cual se ha desarrollado una herramienta que permita obtener ciertas características de las aplicaciones y permita comparar varias aplicaciones en base a dichas características. Esta se ha utilizado para comparar las aplicaciones modificadas y su correspondiente original.
- Una segunda fase manual, en la cual se ha realizado un análisis estático de aquellas aplicaciones modificadas que hayan mostrado algún tipo de indicación de presencia de malware en el análisis previo.

3.1.1. Selección de los mercados

La selección de los mercados resultó una tarea compleja, ya que en los mercados alternativos a Google Play más populares no fue posible encontrar muchas de las aplicaciones presentes en el mercado oficial, ya que estas o bien no estaban o bien no eran mods.

Por tanto, se ha optado por seleccionar aquellos que cuenten con mejor posicionamiento SEO, es decir, aquellos que tengan mayor visibilidad en el buscador de Google. Se ha decidido utilizar esta aproximación para la selección de los mercados ya que ofrece un resultado realista de los mercados más accesibles a los usuarios. Se ha demostrado que cerca del 25 % por ciento de los clicks de los usuarios son el primer resultado de la búsqueda [21]. Para ello se ha realizado la búsqueda en Google a fecha 8 de junio con las siguientes palabras:

“nombre de la aplicación” apk mod

En la Figura 3.1 se puede observar un ejemplo de los resultados de esta búsqueda. Se han seleccionado aquellos en los que se han encontrado el mayor número de aplicaciones. Cada aplicación se ha descargado de cinco mercados no oficiales diferentes. A continuación, se muestra en la Tabla 3.1 las características de los mercados seleccionados.

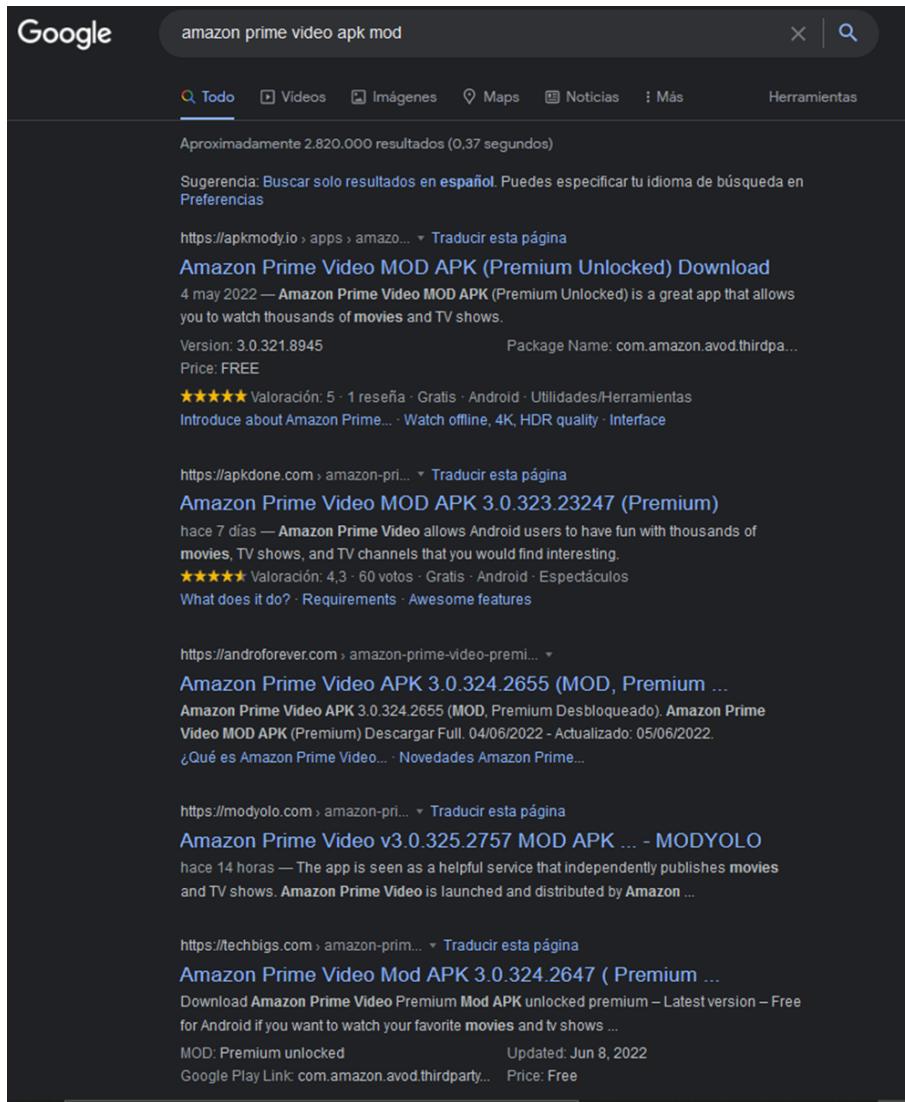


Figura 3.1: Ejemplo de resultado búsqueda

Tabla 3.1: Mercados seleccionados y características

Mercado	Publicidad	Posibilidad de crear cuenta	Pago	Permite dejar comentarios o puntuación	Permite a cualquier usuario subir aplicaciones desde la página
APKMody	Sí	No	No	Sí	No
APKDone	Sí	No	No	Sí	No
Androforever	Sí	No	No	Sí	No
Modyolo	Sí	No	No	Sí	No
Techbigs	Sí	Sí	No	Sí	No

3.1.2. Selección de las aplicaciones

Para realizar la selección de las aplicaciones a analizar, se ha partido de la lista de aplicaciones más populares [24] en función de la categoría. Google Play categoriza las aplicaciones en las siguientes categorías [25]:

Tabla 3.2: Clasificación de aplicaciones en Google Play

Categoría	Ejemplos
Arte y diseño	Cuaderno de bocetos, herramientas para pintores, herramientas de arte y diseño, libros para colorear
Autos y vehículos	Tiendas para autos, seguros para autos, comparación de precios de autos, seguridad vial, opiniones y noticias sobre autos
Belleza	Tutoriales de maquillaje, herramientas de maquillaje, peinados, tiendas de productos de belleza, simuladores de maquillaje
Libros y referencias	Lectores de libros, libros de referencia, libros de texto, diccionarios, tesauros, wikis
Empresa	Lectores y editores de documentos, seguimiento de paquetes, escritorio remoto, administración de correo electrónico, búsqueda de empleo
Cómics	Personajes de cómics, títulos de cómics
Comunicaciones	Mensajes, chats/mensajería instantánea, favoritos, libretas de direcciones, navegadores, administración de llamadas
Citas	Formación de parejas, noviazgo, creación de relaciones, encuentros con gente nueva, búsqueda del amor

Educación	Preparación de exámenes, ayudas para el estudio, vocabulario, juegos educativos, aprendizaje de idiomas
Entretenimiento	Transmisión de video, películas, programas de TV y entretenimiento interactivo
Eventos	Entradas para conciertos, eventos deportivos y cines, y reventa de entradas
Finanzas	Banca, pagos, buscadores de cajeros, noticias financieras, seguros, impuestos, cartera y comercio, calculadoras de propinas
Comida y bebida	Recetas, restaurantes, guías gastronómicas, descubrimiento y degustación de vinos, recetas de tragos
Salud y bienestar	Bienestar personal, seguimiento de ejercicios, sugerencias sobre dietas y nutrición, salud y seguridad, etc.
Casa y hogar	Búsqueda de casas y apartamentos, mejoras para el hogar, decoración de interiores, hipotecas, bienes raíces
Bibliotecas y demostración	Bibliotecas de software y demostraciones técnicas
Estilo de vida	Instructivos, planificación de bodas y eventos, y guías prácticas
Mapas y navegación	Herramientas de navegación, GPS, cartografía, herramientas de tránsito y transporte público

Medicina	Referencias clínicas y de medicamentos, calculadoras, manuales para proveedores de atención médica, noticias y revistas médicas
Música y audio	Servicios musicales, radios y reproductores de música
Noticias y revistas	Periódicos, agregadores de noticias, revistas y blogs
Paternidad y maternidad	Embarazo, cuidado y vigilancia de bebés, cuidado de niños
Personalización	Fondos de pantalla, fondos de pantalla animados, pantalla principal, pantalla bloqueada, tonos
Fotografía	Cámaras, herramientas de edición de fotografías, apps para administrar y compartir fotos
Productividad	Blocs de notas, listas de tareas, teclados, impresión, calendarios, copias de seguridad, calculadoras, herramientas de conversión
Compras	Compras en línea, subastas, cupones, comparaciones de precios, listas de compras, reseñas de productos
Sociales	Redes sociales y registro
Deportes	Comentarios y noticias deportivas, seguimiento de resultados, administración de equipos virtuales y cobertura de partidos
Herramientas	Herramientas para dispositivos Android

Viajes y local	Herramientas para hacer reservas, viajes compartidos, taxis, guías de ciudades, información sobre empresas locales, herramientas de administración de viajes y reserva de recorridos
Reproductores y editores de video	Reproductores de video, editores de video, almacenamiento de medios
Clima	Informes meteorológicos.

Para llevar a cabo el estudio se han seleccionado 5 aplicaciones de 3 categorías distintas. Las categorías seleccionadas fueron las siguientes:

- **Categoría de entretenimiento.** Este tipo de aplicaciones generalmente ofrecen servicios de pago, y son comúnmente utilizadas por todo tipo de público, por lo que podría ser un objetivo para los atacantes.
- **Categoría de citas.** Se ha seleccionado esta categoría debido a que generalmente este tipo de aplicaciones son de pago o poseen algún tipo de servicio premium de pago o suscripción. Por tanto, muchas personas podrían buscar versiones modificadas que permitan podrían ser un objetivo común para los atacantes.
- **Categoría de negocio.** Por último, se ha escogido la categoría de aplicaciones de negocio, ya que actualmente han visto un incremento en su uso [41]. Además, puesto que el perfil de usuario principal de este tipo de aplicaciones son los trabajadores de las empresas, estas aplicaciones podrían suponer un objetivo tentador a los atacantes, ya que podrían conseguir información sensible del dispositivo.

Las aplicaciones seleccionadas fueron las siguientes:

- **Categoría de entretenimiento:** En este caso solo cuatro aplicaciones fueron seleccionadas, ya que una de las cinco era la propia aplicación de Google Play.
 - **Netflix** [26]: es un servicio de streaming que ofrece una amplia variedad de contenido. Para consumir el contenido es preciso suscribirse mediante un pago mensual.

- **Disney Plus** [27]: es un servicio de streaming propiedad de The Walt Disney que ofrece contenido producido por The Walt Disney Studios y Walt Disney Television. De nuevo, también es preciso suscribirse mediante un pago para poder consumir el contenido.
- **Youtube** [28]: Sitio web dedicado a la compartición de vídeos. Es gratuito pero tiene anuncios.
- **Amazon Prime Video** [29]: servicio de streaming de películas y series creado y administrado por Amazon. Como las dos primeras, es preciso suscribirse mediante un pago mensual o anual para consumir el contenido.

- **Categoría de citas:**

- **Tinder** [30]: Aplicación de citas y encuentros. Es gratis pero ofrece diferentes niveles de suscripción que mejoran la experiencia de usuario. A mayor nivel, mayor precio.
- **Badoo** [31]: También es una aplicación de citas que ofrece un servicio de suscripción basado en niveles.
- **Bumble** [32]: Como las anteriores, es gratuita, pero para disfrutar de la máxima experiencia de usuario es preciso suscribirse por un pago.
- **Tantan** [33]: Aplicación de citas gratuita pero que ofrece un paquete premium.
- **CuteU** [34]: Aplicación de citas que se basa en hacer "match" por vídeo. Es gratuita y ofrece también mejorar la experiencia mediante pago.

- **Categoría de negocios:**

- **Adobe Reader** [35]: Visor de PDF gratuito desarrollado por Adobe. Ofrece una versión PRO de pago.
- **Zoom** [36]: Programa de software de videochat. Es gratuito pero ofrece una versión PRO y una versión Business de pago.
- **Google Meet** [37]: Es un servicio de videoconferencia desarrollado por Google. Es gratuito pero ofrece una versión premium de pago.
- **Microsoft Teams** [38]: Plataforma de comunicación desarrollada por Microsoft. Es gratuito y ofrece otras opciones económicas para mejorar ciertos aspectos como el número máximo de participantes en una llamada.

Esta aplicación no se ha podido analizar ya que no fue encontrada en ninguno de los mercados seleccionados.

- **Whatsapp Business [39]:** Aplicación gratuita desarrollada especialmente para pequeñas y medianas empresas con el objetivo de mejorar las interacciones con los clientes. Es totalmente gratuita. Esta aplicación tampoco se ha podido analizar ya que no fue encontrada en ninguno de los mercados seleccionados.

3.1.3. Descripción de la herramienta

Para el desarrollo del estudio, se ha desarrollado un programa que permite realizar una comparativa entre dos aplicaciones dadas. La idea de este programa es que ofrezca una primera idea acerca de la posibilidad de que la aplicación sea maliciosa o no mediante la comparativa de la aplicación original y la modificada descargada del correspondiente mercado.

En primer lugar, el programa recopila información de cada aplicación acerca de los siguientes aspectos:

- **Nombre** de la aplicación
- Si la aplicación está **firmada** o no. Como se ha visto, esto podría servir para identificar al autor de la aplicación.
- **Permisos.** Los permisos de una aplicación determinan los recursos o las acciones que una aplicación puede realizar en el sistema. Es probablemente el punto más crítico, ya que, si una aplicación modificada contiene algún permiso más que la aplicación original, podemos determinar que se trata de una aplicación sospechosa, y así lo ha demostrado la literatura.
- **Actividades.** Una actividad en Android se corresponde con una pantalla de la aplicación.
- Nombres de **ficheros**

Posteriormente, el programa realiza una comparativa de los permisos de cada una de las aplicaciones para determinar si se ha añadido o eliminado algún permiso. En este caso interesa especialmente conocer si se ha añadido algún permiso peligroso, ya que como se ha visto en la revisión de la literatura, esto podría indicar la presencia de malware. Además, el script realiza un análisis mediante la API de VirusTotal

para determinar si la aplicación contiene ya algún tipo de malware que ya haya sido detectado anteriormente.

Por último, la herramienta genera un reporte con los resultados obtenidos, en los cuales se muestran los permisos añadidos y si la aplicación ha sido detectada por algún antivirus.

En base a los resultados obtenidos, se ha realizado un análisis manual de una selección de las aplicaciones más sospechosas, para determinar en mayor detalle si se trata de aplicaciones peligrosas. Este apartado se centrará en proceso de desarrollo del análisis automatizado, mientras que el siguiente capítulo describirá en mayor profundidad el análisis manual.

3.2. El proceso de desarrollo

En este apartado se detallará el proceso de desarrollo de la herramienta implementada para facilitar el estudio llevado a cabo. En primer lugar, se comentará el proceso de análisis, que embarcará la definición y especificación de requisitos. En segundo lugar, se detallará el diseño de la solución. Seguidamente, se hablará del proceso de implementación y finalmente se explicarán las pruebas realizadas.

3.2.1. Análisis de requisitos

A partir de la etapa de diseño, se determinaron una serie de requisitos que la solución desarrollada debe cumplir. Estos se listan a continuación:

1. La herramienta debe recibir como entrada la ruta de la aplicación original y de la aplicación no oficial
2. La herramienta realizará una extracción de datos
3. La herramienta realizará una comparativa de los permisos entre ambas aplicaciones
4. La herramienta guardará estos datos en formato json en el directorio de trabajo
5. La herramienta utilizará la API de VirusTotal para realizar un análisis de la aplicación no oficial para determinar si es detectada por algún antivirus.
6. La herramienta generará un reporte en pdf que contendrá la información estructurada en tablas

3.2.2. Diseño

En la etapa de diseño se ha utilizado la información obtenida de la etapa de análisis de requisitos para tratar de diseñar el producto a desarrollar, es decir, determinar las especificaciones de la herramienta o componentes del sistema.

Diseño de sistema

La herramienta se ha desarrollado en Python 3.10, y se han utilizado diferentes librerías, de las cuales se hablará con mayor detalle en el siguiente apartado. En las figuras que se muestran a continuación, se pueden observar los diagramas de caso de uso, de componentes y de secuencia de la herramienta.

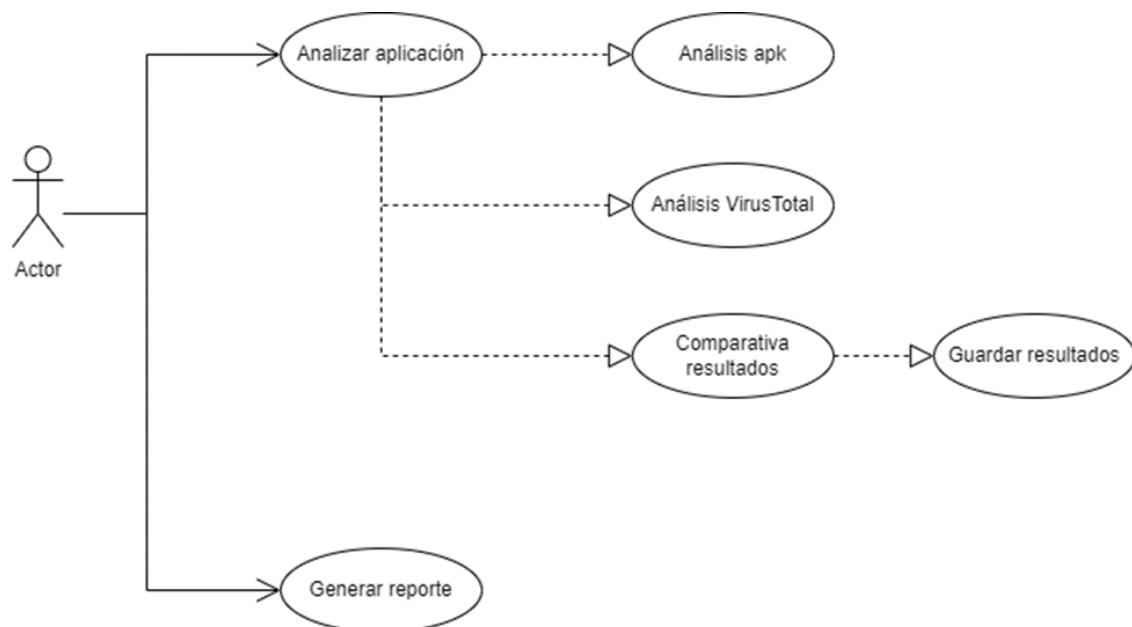


Figura 3.2: Diagrama de casos de uso

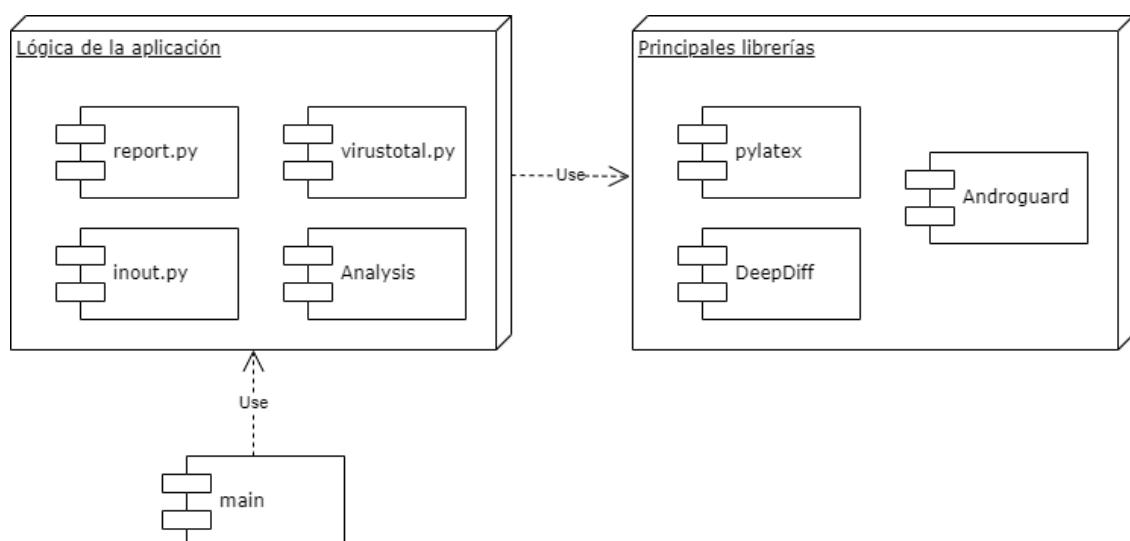
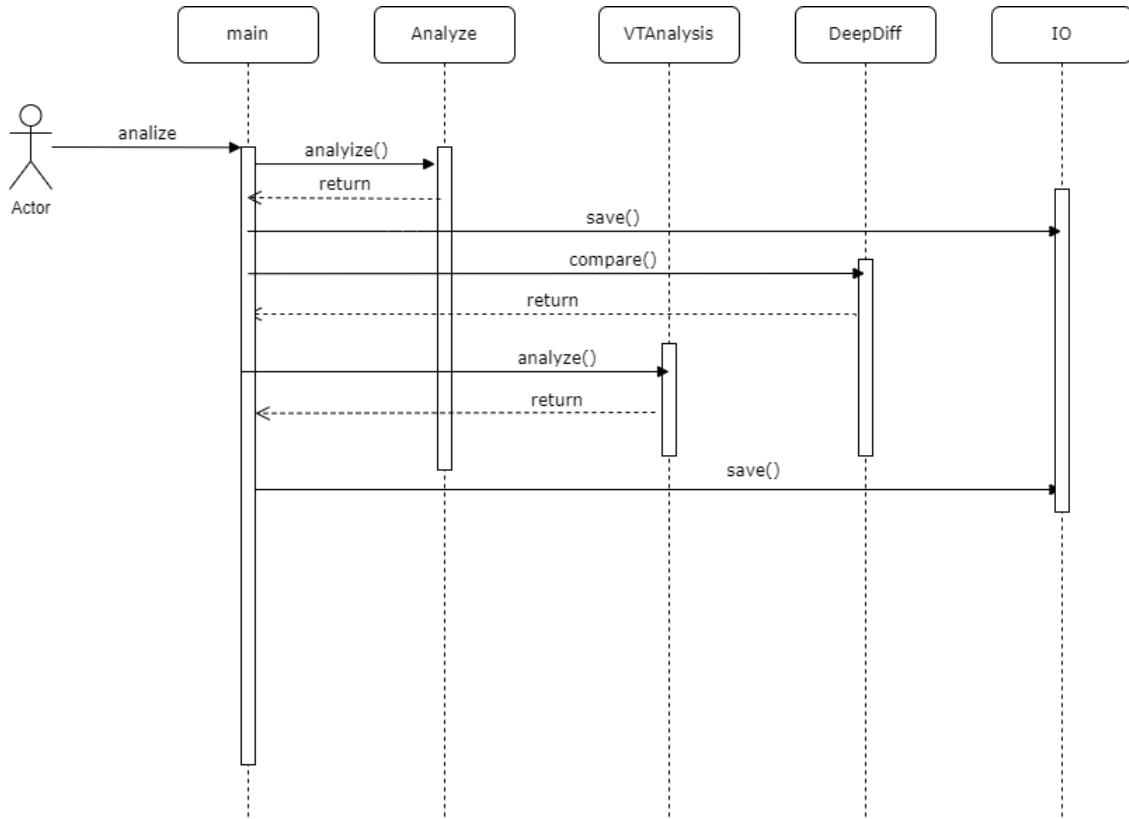


Figura 3.3: Diagrama de componentes

**Figura 3.4:** Diagrama de secuencia

3.2.3. Implementación

A continuación, se explicarán diversos aspectos relacionados con la implementación de la herramienta utilizada para el desarrollo del estudio. Como se ha comentado en el apartado anterior, la herramienta se ha desarrollado en Python 3.10 y se han utilizado las siguientes herramientas:

- **Git.** Tal y como se ha planificado en la gestión de riesgos, se ha utilizado una herramienta de control de versiones para llevar un seguimiento de los cambios realizados y almacenar el código de forma segura en un repositorio. La herramienta utilizada para ello es git, un sistema de control de versiones gratuito y ampliamente utilizado. Para gestionar el repositorio se ha utilizado GitHub, una plataforma online para alojar proyectos utilizando git.
- **Androguard.** Androguard es una herramienta desarrollada en Python para manipular archivos de Android (APK, DEX, xml, etc). Se ha seleccionado por su facilidad de integración en el proyecto al estar desarrollada en Python, así como por las amplias funcionalidades que proporciona para extraer distintas características que resultaban de interés de los archivos APK.

- **DeepDiff.** DeepDiff es una librería de Python que permite encontrar diferencias entre diccionarios, strings y otros objetos. De nuevo, esta librería se ha seleccionado por estar desarrollada en Python y por proporcionar funcionalidad necesaria para trabajar con diccionarios. Se ha utilizado para determinar las diferencias entre las características de las dos aplicaciones dadas, las cuales se almacenan en formato JSON.
- **JSON.** Se ha utilizado la librería de JSON en Python para tratar con este tipo de datos, ya que, como se ha comentado anteriormente, es el tipo utilizado para almacenar las características extraídas.
- **Pylatex.** Librería de Python que permite crear y compilar archivos LaTeX. Se ha utilizado para generar el reporte final.
- **Argparse.** Librería de Python para leer los comandos que el usuario le pasa al programa.
- **ADB (Android Debug Bridge).** Es una herramienta de línea de comandos que permite controlar un dispositivo móvil conectado a un puerto USB del ordenador. Esta herramienta se utilizó para poder extraer las aplicaciones originales que fueron instaladas en el teléfono mediante el mercado oficial de Google Play.

El programa está compuesto por los siguientes archivos:

- **analysis.py:** Es el fichero que inicia la herramienta. Lee los datos introducidos por el usuario, realiza la extracción de las características de las aplicaciones y realiza la comparación entre dichas características.
- **vt.py:** Contiene las funciones necesarias para calcular el hash del APK y realizar la petición necesaria a la API de VirusTotal para realizar un primer análisis de la aplicación.
- **report.py:** Es el fichero que contiene las funciones encargadas de generar el reporte en formato PDF o LaTeX.
- **ionout.py:** Fichero con funciones auxiliares que se encargan del tratamiento de los datos.

3.3. El producto del desarrollo

En esta sección se ilustra el producto desarrollado y su funcionamiento. La herramienta no posee interfaz gráfica, ya que no se ha considerado necesaria al ser una herramienta sencilla de utilizar mediante línea de comandos.

Para obtener las aplicaciones originales de Google Play, estas se instalaron en el dispositivo móvil y posteriormente fueron extraídas para poder ser utilizadas en la herramienta. El resto de aplicaciones se descargaron de los mercados correspondientes. En la figura siguiente se ilustra un ejemplo de ejecución del programa:

```
PS C:\Program Files (x86)\Microsoft Visual Studio\Shared\Python36-64> python D:\android_malware_detection\analysis.py --original "D:\\master\\TFM\\citas\\tantan\\tantan_original.apk" --mod "D:\\master\\TFM\\citas\\tantan\\techbigs\\tantantechbigs.apk" --o "D:\\android_malware_detection\\result.json"
#### Starting analysis #####
Original APK: D:\\master\\TFM\\citas\\tantan\\tantan_original.apk
MOD APK: D:\\master\\TFM\\citas\\tantan\\techbigs\\tantantechbigs.apk
```

Figura 3.5: Ejemplo de ejecución del programa

Como se puede observar, se ejecuta el script *analysis.py* pasándole la ruta de la aplicación original mediante el argumento *-original* y la ruta de la aplicación modificada mediante el argumento *-mod*. Por último, se especifica el nombre del archivo de salida con el argumento *-o*.

El programa almacena los resultados intermedios del análisis de ambas aplicaciones, incluyendo las características mencionadas en el apartado 3.1.3, y el archivo que contiene la comparación entre ambas aplicaciones, así como el resultado del análisis de virus total. Estos archivos se almacenan en formato JSON. Un ejemplo del contenido de uno de estos ficheros para la aplicación de Tantan original se puede observar en las figuras siguientes.

```
 {
    "Name": "Tantan",
    "is_signed": true,
    "permissions": [103],
    "details_permission": [...],
    "activities": [374],
    "files": [15212],
    "main": [1]
}
```

```
    ],
    "Name":"Tantan",
    "is_signed":true,
    "permissions": [
        "android.permission.READ_PHONE_STATE",
        "com.htc.launcher.permission.READ_SETTINGS",
        "android.permission.REQUEST_INSTALL_PACKAGES",
        "android.permission.WAKE_LOCK",
        "com.sec.android.provider.badge.permission.WRITE",
        "com.tencent.qqlauncher.permission.READ_SETTINGS",
        "org.adwfreak.launcher.permission.READ_SETTINGS",
        "com.miui.mihome2.permission.WRITE_SETTINGS",
        "com.google.android.finsky.permission.BIND_GET_INSTALL_REFERRER_SERVICE",
        "net.qihoo.launcher.permission.WRITE_SETTINGS",
        "com.pl.mobile.putong.permission.PROCESS_PUSH_MSG",
        "android.permission.SYSTEM_ALERT_WINDOW",
        "me.everything.badger.permission.BADGE_COUNT_READ",
        "android.permission.ACCESS_FINE_LOCATION",
        "com.htc.launcher.permission.UPDATE_SHORTCUT",
        "telecom.mdesk.permission.READ_SETTINGS",
        "android.permission.VIBRATE",
        "com.miui.mihome2.permission.READ_SETTINGS",
        "net.oneplus.launcher.permission.WRITE_SETTINGS",
        "android.permission.WRITE_SETTINGS",
        "com.android.launcher2.permission.WRITE_SETTINGS",
        "org.adw.launcher.permission.WRITE_SETTINGS",
        "android.permission.READ_EXTERNAL_STORAGE",
        "com.fede.launcher.permission.READ_SETTINGS",
        "net.qihoo.launcher.permission.READ_SETTINGS",
        "android.permission.WRITE_APP_BADGE",
        "com.ktc.launcher.permission.WRITE_SETTINGS"
    ],
    "Name":"Tantan",
    "is_signed":true,
    "permissions": [
        [103],
        "details_permission": [
            {
                "android.permission.READ_PHONE_STATE": [
                    "dangerous",
                    "read phone status and identity",
                    "Allows the app to access the phone features of the device. This permission allows the app to determine the phone number and device IDs, whether a call is active, and the remote number connected by a call."
                ],
                "com.htc.launcher.permission.READ_SETTINGS": [
                    "normal",
                    "Unknown permission from android reference",
                    "Unknown permission from android reference"
                ],
                "android.permission.REQUEST_INSTALL_PACKAGES": [
                    "signature|appop",
                    "request install packages",
                    "Allows an application to request installation of packages."
                ],
                "android.permission.WAKE_LOCK": [
                    "normal|instant",
                    "prevent phone from sleeping",
                    "Allows the app to prevent the phone from going to sleep."
                ],
                "com.miui.mihome2.permission.WRITE_SETTINGS": [
                    "dangerous",
                    "write settings",
                    "Allows the app to change system settings such as screen brightness, volume, and connectivity options." ]
            }
        ]
    ]
}
```

```
    "activities": [",
        "com.p1.mobile.putong.account.ui.welcome.WelcomeAct",
        "com.p1.mobile.putong.account.ui.accountnew.PhoneNumberIn
        putAct",
        "com.p1.mobile.putong.account.ui.accountnew.loginopt.act.
        PhoneNumberLoginOptAct",
        "com.p1.mobile.putong.account.ui.accountnew.loginopt.act.
        CCodeChooseAct",
        "com.p1.mobile.putong.account.ui.accountnew.loginopt.act.
        CropperAct",
        "com.p1.mobile.putong.account.ui.accountnew.VerifyCodeInp
        utAct",
        "com.p1.mobile.putong.account.ui.accountnew.SignUpDetails
        NewAct",
        "com.p1.mobile.putong.account.ui.accountnew.SignUpProfile
        ImageAct",
        "com.p1.mobile.putong.account.ui.accountnew.facebook.Face
        bookSignUpProfileImageAct",
        "com.p1.mobile.putong.account.ui.accountnew.loginopt.act.
        ShortCutLoginOptActivity",
        "com.p1.mobile.putong.account.ui.accountnew.loginopt.act.
        VerifyCodeAct",
        "com.p1.mobile.putong.account.ui.accountnew.loginopt.act.
        WebViewActOpt",
        "files": [
            "DebugProbesKt.bin",
            "LICENSE-junit.txt",
            "LICENSE_OFL",
            "LICENSE_UNICODE",
            "META-INF/android_release.kotlin_module",
            "META-INF/androidx.activity_activity.version",
            "META-INF/androidx.annotation_annotation-
            experimental.version",
            "META-INF/androidx.appcompat_appcompat-
            resources.version",
            "META-INF/androidx.appcompat_appcompat.version",
            "META-INF/androidx.arch.core_core-runtime.version",
            "META-
            INF/androidx.asynclayoutinflater_asynclayoutinflater.ver
            sion",
            "META-INF/androidx.browser_browser.version",
            "META-INF/androidx.cardview_cardview.version",
            "META-
            INF/androidx.coordinatorlayout_coordinatorlayout.version"
            ,
            "META-INF/androidx.core_core-ktx.version",
            "META-INF/androidx.core_core.version",
            "META-INF/androidx.cursoradapter_cursoradapter.version",
            "META-INF/androidx.customview_customview.version",
            "META-INF/androidx.documentfile_documentfile.version",
```

Figura 3.6: Ejemplo de json generado para la aplicación de Tantan original

Además, es posible generar un reporte en pdf en el cual se muestren los resultados de la comparación en una tabla que resulte más sencilla de visualizar, tal y como se observa en la siguiente figura:

5 Netflix

5.1 diff_androforever_netflix.json

```
Added Permissions
'android.permission.VIBRATE'
'com.mozillaonline.downloads.com.teamseries.ACCESS_DOWNLOAD_MANAGER'
'android.permission.READ_EXTERNAL_STORAGE'
'android.permission.DOWNLOAD_WITHOUT_NOTIFICATION'
'com.mozillaonline.downloads.com.teamseries.SEND_DOWNLOAD_COMPLETED_INTENTS'
'android.permission.SYSTEM_ALERT_WINDOW'
'com.mozillaonline.downloads.com.teamseries.ACCESS_DOWNLOAD_MANAGER_ADVANCED'
'com.google.android.providers.gsf.permission.READ_GSERVICES'
'com.google.android.providers.gsf.permission.WRITE_GSERVICES'
```

5.2 diff_apkdone_netflix.json

```
Added Permissions
'android.permission.VIBRATE'
'com.mozillaonline.downloads.com.teamseries.ACCESS_DOWNLOAD_MANAGER'
'android.permission.READ_EXTERNAL_STORAGE'
'android.permission.DOWNLOAD_WITHOUT_NOTIFICATION'
'android.permission.ACCESS_COARSE_LOCATION'
'com.mozillaonline.downloads.com.teamseries.SEND_DOWNLOAD_COMPLETED_INTENTS'
'android.permission.SYSTEM_ALERT_WINDOW'
'com.mozillaonline.downloads.com.teamseries.ACCESS_DOWNLOAD_MANAGER_ADVANCED'
'com.google.android.providers.gsf.permission.READ_GSERVICES'
'com.google.android.providers.gsf.permission.WRITE_GSERVICES'
```

Figura 3.7: Extracto del reporte generado en pdf

Capítulo 4

Evaluación

4.1. Proceso de evaluación

Los resultados obtenidos con el análisis automatizado han permitido hacer una selección de aquellas aplicaciones que a priori parecen más peligrosas, para realizar posteriormente un análisis manual en mayor profundidad. Para determinar las aplicaciones que se han analizado manualmente, se ha seguido el siguiente criterio:

- Puesto que las aplicaciones de entretenimiento son las que más permisos peligrosos modifican, de cada aplicación se ha seleccionado aquella descargada del mercado que más permisos modifique o que más haya sido detectada por antivirus. Se ha exceptuado Youtube, ya que los permisos peligrosos modificados son mucho menores.
- De las aplicaciones de negocio no se ha seleccionado ninguna, ya que no presentaban ninguna amenaza aparente.
- De las aplicaciones de citas se ha seleccionado Badoo, ya que es la única en la que se encontró algo significativo en el análisis automático.

En las siguientes tablas se resumen los resultados obtenidos en análisis automatizado. Se han marcado en azul aquellas aplicaciones seleccionadas para analizar manualmente:

	Amazon Prime Video (MOD)				
	Androforever	Apkdone	Apkmody	Modyolo	Techbigs
SYSTEM ALERT WINDOW	X	X	X	X	
DOWNLOAD WITHOUT NOTIFICATION			X		
READ EXTERNAL STORAGE					X
READ INTERNAL STORAGE					X
WRITE EXTERNAL STORAGE					X
WRITE INTERNAL STORAGE					X
ACCESS COARSE LOCATION					X
ACCESS FINE LOCATION					X
Detectado por VT		1			12

Tabla 4.1: Resultados del análisis de Amazon Prime Video (MOD)

	Disney Plus (MOD)				
	Androforever	Apkdone	Apkmody	Modyolo	Techbigs
CAMERA		X			
READ EXTERNAL STORAGE	X	X		X	X
WRITE EXTERNAL STORAGE	X	X		X	X
SYSTEM ALERT WINDOW	X	X		X	X
READ LOGS		X			
Detectado por VT					

Tabla 4.2: Resultados del análisis de Disney Plus (MOD)

	Youtube (MOD)				
	Androforever	Apkdone	Apkmody	Modyolo	Techbigs
SEND SMS	X				X
RECEIVE SMS	X				X
REQUEST INSTALL PACKAGES			X		
REQUEST DELETE PACKAGES			X		
Detectado por VT					

Tabla 4.3: Resultados del análisis de Youtube (MOD)

	Netflix (MOD)				
	Androforever	Apkdone	Apkmody	Modyolo	Techbigs
ACCESS COARSE LOCATION	X				X
READ PHONE STATE	X	X	X	X	X
READ EXTERNAL STORAGE WRITE EXTERNAL STORAGE	X	X	X		X
SYSTEM ALERT WINDOW	X	X	X	X	X
DOWNLOAD WITHOUT NOTIFICATION		X			
Detectado por VT		1		2	

Tabla 4.4: Resultados del análisis de Netflix (MOD)

	Adobe (MOD)				
	Androforever	Apkdone	Apkmody	Modyolo	Techbigs
Detectado por VT	1	1			1

Tabla 4.5: Resultados del análisis de Adobe (MOD)

	Tinder				
	Androforever	Apkdone	Apkmody	Modyolo	Techbigs
Detectado por VT	1	1	1	1	1

Tabla 4.6: Resultados del análisis de Tinder

	Badoo				
	Androforever	Apkdone	Apkmody	Modyolo	Techbigs
Detectado por VT					2

Tabla 4.7: Resultados del análisis de Badoo

4.1.1. Forma de evaluación

En el apartado anterior se realizó una selección de las aplicaciones en base a los resultados obtenidos del primer análisis automatizado. En este capítulo, se ha llevado a cabo un análisis en mayor profundidad de dichas aplicaciones para determinar el comportamiento de las mismas y comprobar si existen indicios de la presencia de malware.

Para ello, en primer lugar, se realizó un proceso de ingeniería inversa con cada una de las aplicaciones seleccionadas para poder tener acceso al código en claro, así como a las librerías utilizadas, los recursos y el AndroidManifest. El proceso de ingeniería inversa consistió en los siguientes pasos:

1. En primer lugar, se utilizó la herramienta **apktool** para obtener el archivo AndroidManifest.xml y los archivos *.dex*, así como acceso a los recursos y a las librerías.
2. Seguidamente, se utilizó la herramienta **dex2jar** para transformar los archivos *.dex* a *jar* y que pudiesen ser abiertos con un descompilador de Java.
3. Se ha utilizado las aplicaciones **jd-gui** y **jadx-gui** para poder leer los archivos *.class*. Se han utilizado ambas ya que cada una proporcionaba funcionalidades ligeramente diferentes.

Es posible que el código de algunas de las aplicaciones esté parcial o totalmente ofuscado, y el análisis no pueda ser posible. Esto es especialmente común en aplicaciones para proteger los derechos de autor, pero también es muy común en aplicaciones con malware para evitar su detección.

Una vez descompilada la aplicación, se han analizado diversos aspectos del código en busca de elementos indicativos de presencia de malware/adware, entre ellos, se han analizado los siguientes elementos:

- Librerías utilizadas
- Recursos
- AndroidManifest.xml
- Código descompilado. Interesa especialmente analizar:
 - Accesos a directorios del sistema
 - IPs o dominios existentes en el código
 - Presencia de cadenas sospechosas
 - Uso de permisos
 - Conexiones http/https

No se ha realizado un análisis dinámico, ya que este quedaba fuera del alcance del presente trabajo. En los siguientes subapartados se detallarán los análisis realizados y se mostrarán las distintas evidencias recopiladas que podrían indicar la presencia de algún tipo de malware, adware o comportamiento sospechoso.

4.1.2. Análisis Badoo Techbigs

La primera aplicación analizada fue Badoo obtenida del mercado de TechBigs. En la página de descarga se indicaba que dicha aplicación era la original, sin ningún tipo de modificación, como se puede ver en la Figura 4.1.

Sin embargo, puesto que la ejecución del programa automatizado determinó que existían diferencias entre la aplicación original y la descargada de TechBigs, y que se añadían nuevos permisos peligrosos, se puede asumir que no se trata de la aplicación original.

Lo primero que se ha analizado fueron las librerías de la aplicación. Como se puede ver en la Figura 4.2, la aplicación hace uso de librerías de Firebase, lo cual indica que se podría conectar a un servidor. Además, también contiene la librería de Unity, una herramienta para el desarrollo de videojuegos. Por último, contiene una librería denominada *Ghinhaba_Ryuuka*.

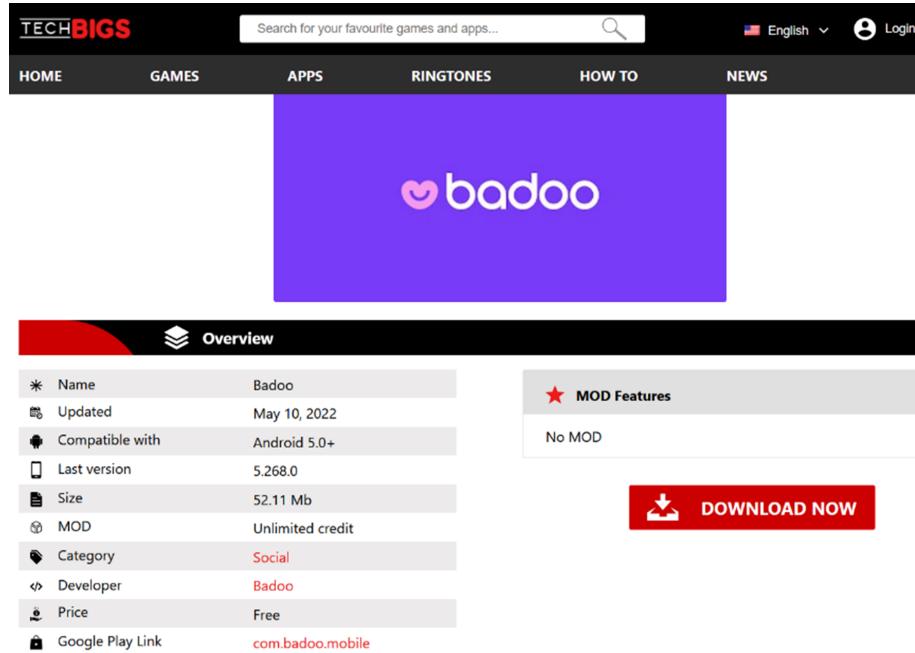


Figura 4.1: Aplicación de Badoo en mercado de TechBigs

Nombre	Fecha de modificación	Tipo	Tamaño
libanogs.so	03/10/2021 10:41	Archivo SO	3.591 KB
libanort.so	03/10/2021 10:41	Archivo SO	1.230 KB
libcrashmonitor.so	03/10/2021 10:41	Archivo SO	102 KB
libFirebaseCppAnalytics.so	03/10/2021 10:41	Archivo SO	26 KB
libFirebaseCppApp-6_16_1.so	03/10/2021 10:41	Archivo SO	2.112 KB
libFirebaseCppCrashlytics.so	03/10/2021 10:41	Archivo SO	31 KB
libFirebaseCppMessaging.so	03/10/2021 10:41	Archivo SO	47 KB
libfreetype.so	03/10/2021 10:41	Archivo SO	926 KB
libGGP.so	03/10/2021 10:41	Archivo SO	1.040 KB
libGhinhaba_Ryuuka.so	03/10/2021 10:41	Archivo SO	666 KB
libharfbuzz.so	03/10/2021 10:41	Archivo SO	2.063 KB
libl2cpp.so	03/10/2021 10:41	Archivo SO	71.499 KB
libmain.so	03/10/2021 10:41	Archivo SO	22 KB
libunity.so	03/10/2021 10:41	Archivo SO	14.684 KB
libyoume_voice_engine.so	03/10/2021 10:41	Archivo SO	2.108 KB

Figura 4.2: Librerías Badoo TechBigs

No se ha podido determinar de qué se trata la librería de *Ginhara_Ryuuka*, ya que no existe ninguna información acerca de la misma. Esto podría indicar que se trata de una librería no oficial.

Revisando los recursos de la aplicación, se ha encontrado un ícono que no está relacionado de ninguna forma con la aplicación original, tal y como se puede ver en la Figura 4.3.

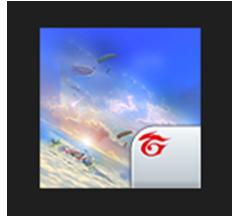


Figura 4.3: Ícono Badoo TechBigs

En este punto se podría asumir que la aplicación descargada no es Badoo a pesar de lo que se anuncia en el mercado de TechBigs, sino que posiblemente sea otra aplicación distinta. También se ha analizado el archivo de AndroidManifest.xml, en el cual se ha encontrado una referencia a una dirección IP, como se puede apreciar en la Figura 4.4.

```
<meta-data android:name="notch.config" android:value="portrait|landscape"/>
<meta-data android:name="com.garena.sdk.applicationId" android:value="100067"/>
<meta-data android:name="com.garena.sdk.applicationVariant" android:value="th"/>
<meta-data android:name="com.garena.sdk.push.applicationId" android:value="100067"/>
<meta-data android:name="com.beetalklib.ganalytics.report_url" android:value="http://122.11.128.69:2205"/>
<meta-data android:name="com.facebook.sdk.ApplicationId" android:value="@string/facebook_appid"/>
```

Figura 4.4: Referencia a dirección IP

Se ha realizado una búsqueda de dicha dirección IP, y se ha averiguado que su origen es Singapur, tal y como se puede observar en la Figura 4.5.

Checking IP Address

IP Address: 122.11.128.69

Geolocation: SG (Singapore), 00, N/A, N/A Singapore - [Google Maps](#)

Figura 4.5: Origen de la dirección IP

Se ha continuado el proceso de análisis revisando los paquetes de la aplicación. Uno de los paquetes encontrados es el de appsflyer, una plataforma de análisis y atribución de marketing móvil, lo que a primera vista parece señalar la presencia de anuncios en la aplicación.

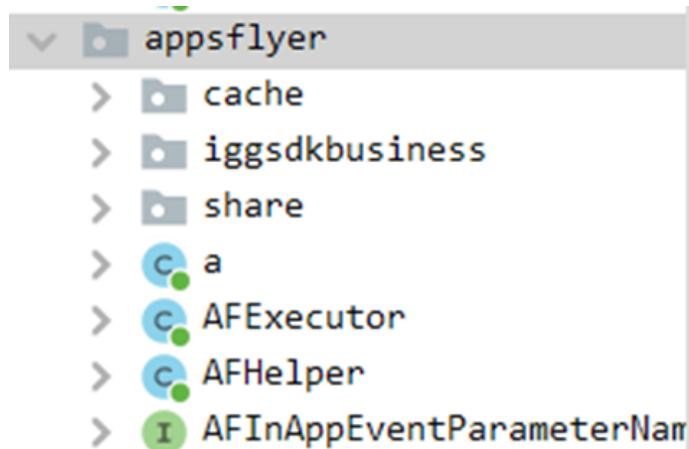


Figura 4.6: Paquete Appsflyer

Además, se ha encontrado otro paquete denominado *banalytics* que parece estar relacionado con algún tipo de analítica y recolección de datos, como se puede observar en la Figura 4.7.

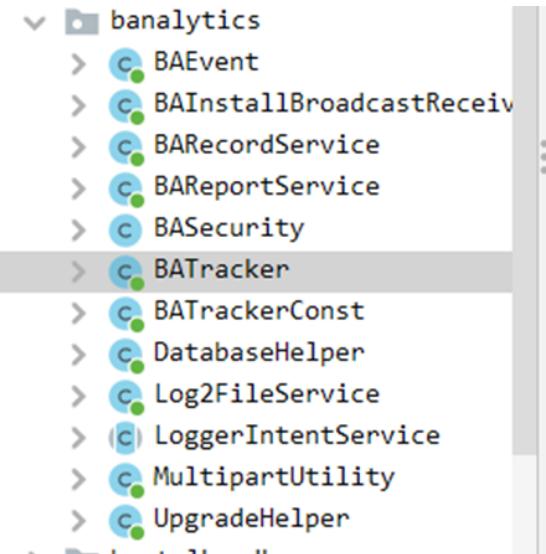


Figura 4.7: Paquete Banalytics

También se ha encontrado un paquete denominado dts.freefireth, el cual se muestra en la Figura 4.8.

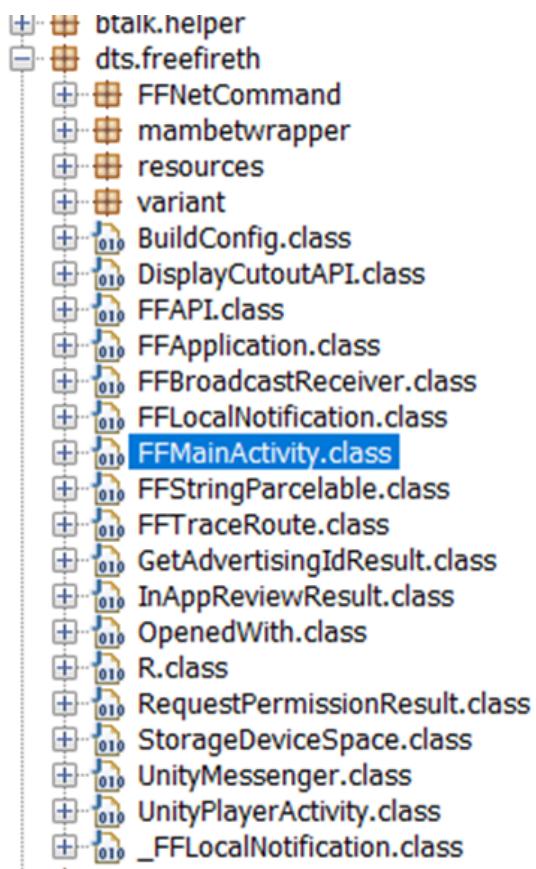


Figura 4.8: Paquete dts.freefireth

Realizando una búsqueda en Google, se ha determinado que esta aplicación en realidad se trata de un juego, tal y como se puede apreciar en la Figura 4.9, cuyo ícono se corresponde con el encontrado en la carpeta de recursos.

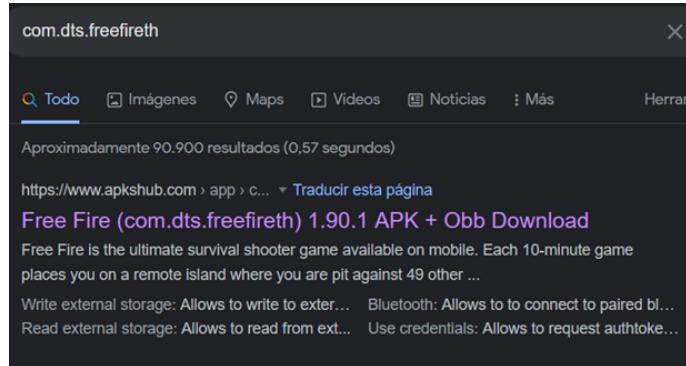


Figura 4.9: Resultado de búsqueda de dts.freefireth

En uno de los paquetes se encontró un archivo denominado “RootCheck.class”. En esta clase se encontraron diversos métodos sospechosos que serán descritos a continuación.

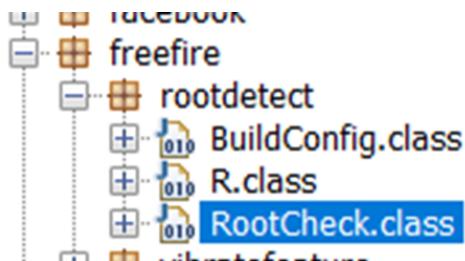


Figura 4.10: Clase RootCheck.class

En primer lugar, se ha observado la existencia de una función que comprueba si la aplicación se está ejecutando en un móvil real y no en un emulador. Para ello, lee la información de la CPU y comprueba que sea real:

```

public static boolean checkIsRealPhone() {
    String readCpuInfo = readCpuInfo();
    return readCpuInfo.contains("intel") || readCpuInfo.contains("amd");
}

```

Figura 4.11: Función checkIsRealPhone

El método anterior llama a la función `readCpuInfo`, la cual se puede observar en la Figura 4.12. Obtiene la información de la CPU mediante realizando una llamada al comando `cat` para leer el archivo `/proc/cpuinfo` y obtener información de la CPU del dispositivo.

```

public static String readCpuInfo() {
    try {
        Process start = new ProcessBuilder("/system/bin/cat", "/proc/cpuinfo").start();
        StringBuffer stringBuffer = new StringBuffer();
        BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(start.getInputStream(), "utf-8"));
        while (true) {
            String readLine = bufferedReader.readLine();
            if (readLine != null) {
                stringBuffer.append(readLine);
            } else {
                bufferedReader.close();
                return stringBuffer.toString().toLowerCase();
            }
        }
    } catch (IOException unused) {
        return "";
    }
}

```

Figura 4.12: Función readCpuInfo

Continuando con el análisis de esta misma clase, se han encontrado otros métodos sospechosos, como un método para determinar si el Bluetooth está activado o para obtener detalles sobre el servicio de telefonía, el cual se muestra a continuación:

```

public boolean hasBlueTooth() {
    BluetoothAdapter defaultAdapter = BluetoothAdapter.getDefaultAdapter();
    return defaultAdapter != null && !TextUtils.isEmpty(defaultAdapter.getName());
}

public String getProvidersName(TelephonyManager telephonyManager, Activity activity) {
    TelephonyManager telephonyManager2 = (TelephonyManager) activity.getSystemService("phone");
    return null;
}

```

Figura 4.13: Función hasBluetoot

También se ha encontrado otro método que obtiene información del sensor del dispositivo:

```

public static String GetSensorInfo(Activity activity) {
    StringBuilder sb = new StringBuilder();
    SensorManager sensorManager = (SensorManager) activity.getSystemService("sensor");
    if (sensorManager.getDefaultSensor(5) == null) {
        sb.append("0");
    } else {
        sb.append("1");
    }
    if (sensorManager.getDefaultSensor(6) == null) {
        sb.append("0");
    } else {
        sb.append("1");
    }
    if (sensorManager.getDefaultSensor(8) == null) {
        sb.append("0");
    } else {
        sb.append("1");
    }
    return sb.toString();
}

```

Figura 4.14: Función GetSensorInfo

Asimismo, se han encontrado dos funciones para leer y escribir archivos:

```
public static Boolean writeFile(String str, String str2) {
    try {
        FileOutputStream fileOutputStream = new FileOutputStream(str);
        fileOutputStream.write(str2.getBytes());
        fileOutputStream.close();
        return true;
    } catch (Exception e) {
        e.printStackTrace();
        return false;
    }
}

public static String readFile(String str) {
    try {
        FileInputStream fileInputStream = new FileInputStream(new File(str));
        byte[] bArr = new byte[1024];
        ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();
        while (true) {
            int read = fileInputStream.read(bArr);
            if (read > 0) {
                byteArrayOutputStream.write(bArr, 0, read);
            } else {
                return new String(byteArrayOutputStream.toByteArray());
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
        return null;
    }
}
```

Figura 4.15: Función writeFile y readFile

La función de escritura se utiliza en esa misma clase en un método denominado **checkAccessRootData** para escribir en el directorio data, de forma que determina si se tienen permisos de administrador (ya que son necesarios para acceder a dicho directorio).

```
public static synchronized boolean checkAccessRootData() {  
    synchronized (RootCheck.class) {  
        try {  
            writeFile("/data/su_test", "test_ok").booleanValue();  
            return "test_ok".equals(readFile("/data/su_test"));  
        } catch (Exception unused) {  
            return false;  
        }  
    }  
}
```

Figura 4.16: Función checkAccessRootData

También se ha encontrado un archivo que comprueba la ejecución de *busybox*:

```
public static synchronized boolean checkBusybox() {  
    synchronized (RootCheck.class) {  
        try {  
            return executeCommand(new String[]{"busybox", "df"}) != null;  
        } catch (Exception unused) {  
            return false;  
        }  
    }  
}
```

Figura 4.17: Función checkBusybox

Y una función denominada **executeCommand** cuyo argumento es una lista de comandos que serán ejecutados.

```
public static ArrayList<String> executeCommand(String[] strArr) {
    ArrayList<String> arrayList = new ArrayList<>();
    try {
        Process exec = Runtime.getRuntime().exec(strArr);
        new BufferedWriter(new OutputStreamWriter(exec.getOutputStream()));
        BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(exec.getInputStream()));
        while (true) {
            try {
                String readLine = bufferedReader.readLine();
                if (readLine == null) {
                    break;
                }
                arrayList.add(readLine);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        return arrayList;
    } catch (Exception unused) {
        return null;
    }
}
```

Figura 4.18: Función executeCommand

Otro método relacionado es el **checkSuperuserApk**, el cual comprueba si existe una aplicación llamada Superuser.apk, para obtener acceso root al sistema:

```
public static boolean checkSuperuserApk() {
    try {
        return new File("/system/app/Superuser.apk").exists();
    } catch (Exception unused) {
        return false;
    }
}
```

Figura 4.19: Función checkSuperuserApk

También se encontró otro método denominado **checkRootPathSU**, que comprueba a que directorios tiene acceso la aplicación:

```

public static boolean checkRootPathSU() {
    String[] strArr = {"system/bin/", "system/xbin/", "system/sbin/", "sbin/", "vendor/bin/"};
    for (int i = 0; i < 5; i++) {
        try {
            if (new File(strArr[i] + "su").exists()) {
                return true;
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    return false;
}

public static boolean checkRootWhichSU() {
    return executeCommand(new String[]{"system/xbin/which", "su"}) != null;
}

```

Figura 4.20: Función checkRootPathSU

Por último, existe otro método denominado **checkGetRootAuth**, que trata de obtener permisos de superusuario:

```

public static synchronized boolean checkGetRootAuth() {
    Process process;
    Throwable th;
    synchronized (RootCheck.class) {
        DataOutputStream dataOutputStream = null;
        try {
            process = Runtime.getRuntime().exec("su");
            try {
                DataOutputStream dataOutputStream2 = new DataOutputStream(process.getOutputStream());
                try {
                    dataOutputStream2.writeBytes("exit\n");
                    dataOutputStream2.flush();
                    if (process.waitFor() == 0) {
                        try {
                            dataOutputStream2.close();
                            process.destroy();
                        } catch (Exception e) {
                            e.printStackTrace();
                        }
                        return true;
                    }
                    try {
                        dataOutputStream2.close();
                        process.destroy();
                    } catch (Exception e2) {
                        e2.printStackTrace();
                    }
                    return false;
                } catch (Exception unused) {
                    dataOutputStream = dataOutputStream2;
                    if (dataOutputStream != null) {
                        try {
                            dataOutputStream.close();
                        } catch (Exception e3) {
                            e3.printStackTrace();
                            return false;
                        }
                    }
                    process.destroy();
                    return false;
                } catch (Throwable th2) {
                    th = th2;
                    dataOutputStream = dataOutputStream2;
                    if (dataOutputStream != null) {
                        try {
                            dataOutputStream.close();
                        } catch (Exception e4) {
                            e4.printStackTrace();
                            throw th;
                        }
                    }
                    process.destroy();
                    throw th;
                }
            }
        }
    }
}

```

Figura 4.21: Función checkGetRootAuth

Además de estos métodos, se han encontrado diferentes métodos en este paquete que indican que se realizan peticiones a un servidor, como por ejemplo el que se muestra a continuación:

```
private void sendEvents(String str, List<BAEvent> list) {
    Throwable th;
    HttpURLConnection httpURLConnection;
    HttpURLConnection httpURLConnection2 = null;
    try {
        try {
            httpURLConnection = (HttpURLConnection) new URL(str).openConnection();
        } catch (IOException unused) {
        }
    } catch (Throwable th2) {
        th = th2;
    }
    try {
        httpURLConnection.setRequestMethod("POST");
        httpURLConnection.setRequestProperty("content-type", AbstractSpiCall.ACCEPT_JSON_VALUE);
        httpURLConnection.setDoInput(true);
        httpURLConnection.setDoOutput(true);
        OutputStream outputStream = httpURLConnection.getOutputStream();
        BufferedWriter bufferedWriter = new BufferedWriter(new OutputStreamWriter(outputStream, VKHttpClient.sDefaultStringEncoding));
        bufferedWriter.write(generateJSONString(list));
        bufferedWriter.flush();
        bufferedWriter.close();
        outputStream.close();
        httpURLConnection.connect();
        int responseCode = httpURLConnection.getResponseCode();
        if (responseCode != 404) {
            this._databaseHelper.removeSentEvents(list);
            BBAppLogger.i("events log posted to server %d", Integer.valueOf(responseCode));
        } else {
            _numOfFails++;
            BBAppLogger.i("server is likely down :X", new Object[0]);
        }
        if (httpURLConnection == null) {
            return;
        }
        httpURLConnection.disconnect();
    } catch (IOException unused2) {
        httpURLConnection2 = httpURLConnection;
        _numOfFails++;
        BBAppLogger.i("http request unsuccessful", new Object[0]);
        if (httpURLConnection2 == null) {
            return;
        }
        httpURLConnection2.disconnect();
    } catch (Throwable th3) {
        th = th3;
        httpURLConnection2 = httpURLConnection;
        if (httpURLConnection2 != null) {
            httpURLConnection2.disconnect();
        }
        throw th;
    }
}
```

Figura 4.22: Función sendEvents

Este método se invoca desde otro método denominado **sendEventsIfNeeded** que se invoca pasándole una lista que se obtiene invocando a un método denominado **getPendingEvents**:

```

public boolean sendEventsIfNeeded(Context context) {
    if (_numOfFails >= 2 || this._databaseHelper.getPendingEventCount() <= 0) {
        return false;
    }
    List<BAEvent> pendingEvents = this._databaseHelper.getPendingEvents();
    try {
        Bundle bundle = context.getPackageManager().getApplicationInfo(context.getPackageName(), 128).metaData;
        if (bundle != null && bundle.containsKey(BATrackerConst.MANIFEST_REPORT_URL)) {
            BATrackerConst.URL = bundle.getString(BATrackerConst.MANIFEST_REPORT_URL);
        }
    } catch (PackageManager.NameNotFoundException unused) {
    }
    BBAppLogger.i("report URL %s", BATrackerConst.URL);
    if (!TextUtils.isEmpty(BATrackerConst.URL)) {
        sendEvents(BATrackerConst.URL, pendingEvents);
    }
}
return true;
}

```

Figura 4.23: Función sendEventsIfNeeded

Dicho método obtiene diversos datos del dispositivo que previamente fueron insertados en una base de datos. Entre estos datos, destacan la información del teléfono o el tipo de línea de comandos.

```

/* JADK INFO: Access modifiers changed from: protected */
public List<BAEvent> getPendingEvents() {
    SQLiteDatabase readableDatabase = getReadableDatabase();
    String str = BAEvent.EventEntry.COLUMN_NAME_EVENT_ID;
    String str2 = BAEvent.EventEntry.COLUMN_NAME_TIMESTAMP;
    String str3 = "status";
    String str4 = "description";
    String str5 = BAEvent.EventEntry.COLUMN_NAME_CMD;
    String str6 = "user_id";
    String str7 = "device_id";
    String str8 = "country";
    Cursor query = readableDatabase.query(BAEvent.EventEntry.TABLE_NAME, new String[]{str, str2, str3, str4, str5, str6, str7, "channel", "api_level", "de
    ArrayList arrayList = new ArrayList();
    if (query.moveToFirst()) {
        while (true) {
            int i = query.getInt(query.getColumnIndex(str));
            int i2 = query.getInt(query.getColumnIndex(str2));
            int i3 = query.getInt(query.getColumnIndex(str3));
            String string = query.getString(query.getColumnIndex(str4));
            String string2 = query.getString(query.getColumnIndex(str5));
            String string3 = query.getString(query.getColumnIndex(str6));
            String string4 = query.getString(query.getColumnIndex(str7));
            String str9 = str;
            String str10 = str2;
            String str11 = str3;
            String str12 = str4;
            String str13 = str5;
            String str14 = str6;
            String str15 = str7;
            String str16 = str8;
            String str17 = query.getString(query.getColumnIndex("app_version"));
            String str18 = query.getString(query.getColumnIndex("referrer"));
            String str19 = str7;
            String str20 = query.getString(query.getColumnIndex("manufacturer"));
            ArrayList arrayList2 = arrayList;
            String str21 = query.getString(query.getColumnIndex("app_key"));
            String str22 = str8;
            String str23 = query.getString(query.getColumnIndex(str16));
            BAEvent bAEEvent = new BAEvent();
            bAEEvent.setEventId(i);
            bAEEvent.setTimestamp(i2);
            bAEEvent.setStatus(i3);
            bAEEvent.setDescription(string);
            bAEEvent.setCmdType(string2);
            bAEEvent.setUserid(string3);
            bAEEvent.setDeviceId(string4);
            bAEEvent.setChannel(string5);
            bAEEvent.setApiLevel(i4);
            bAEEvent.setDeviceInfo(string6);
            bAEEvent.setAppVersion(i5);
        }
    }
}

```

Figura 4.24: Función getPendingEvents

A continuación, se ha encontrado otro paquete desconocido denominado *beetalk*, el cual se muestra en la Figura 4.25. Realizando una búsqueda en Google, se ha podido averiguar que se trata de una aplicación similar a Badoo, tal y como se puede apreciar en la Figura 4.26.

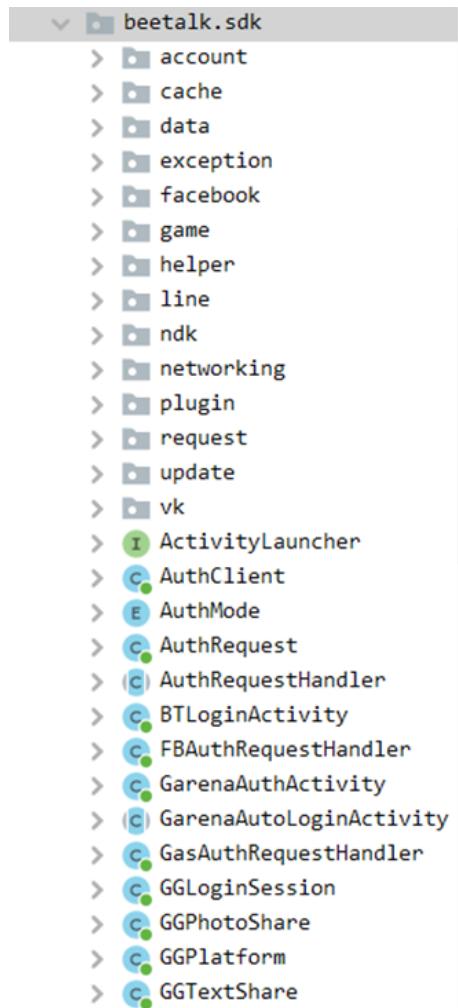


Figura 4.25: Paquete Beetalk

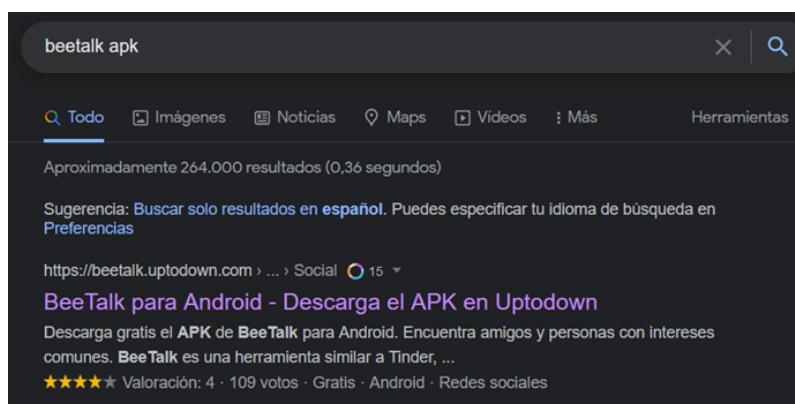


Figura 4.26: Búsqueda de Beetalk

En este paquete se ha encontrado un método que parece enviar logs a un servidor, cuya IP se corresponde con la vista en el archivo AndroidManifest.xml.

```
/* Loaded from: D:\master\TFM\decompile\citas\badoo\techbigs\classes.dex */
public class Log2fileService extends LoggerIntentService {
    private static final int BUFF_SIZE = 16;
    private static String DIR = null;
    private static String EXTRA_DIR_PATH = "dir_path";
    private static final String EXTRA_FILE = "file";
    private static final String EXTRA_FORCE_FLUSH = "forceFlush";
    private static final String EXTRA_IS_FATAL = "isFatal";
    private static final String EXTRA_LOG = "log";
    private static final String EXTRA_TIME = "time";
    private static final int MAX_LOGS_COUNT = 168;
    private static final String REPORT_FILE = "report.gz";
    private static final String TAIL = ".txt";
    private HashMap<String, ArrayList<LogInfo>> logBuff = new HashMap<>();
    private static final String TAG = Log2fileService.class.getSimpleName();
    private static SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH", Locale.ENGLISH);
    private static SimpleDateFormat logsdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss.SSS", Locale.ENGLISH);
    public static String CRASH_SERVER_ADDRESS = "http://203.116.180.99/crash/logs/";
    public static String CRASH_SERVER_HOST = "crashreporter.beetalkmobile.com";
}
```

Figura 4.27: Dirección IP

Otro de los paquetes analizados es el que se muestra en la figura siguiente. El código dentro de este paquete estaba parcialmente ofuscado.

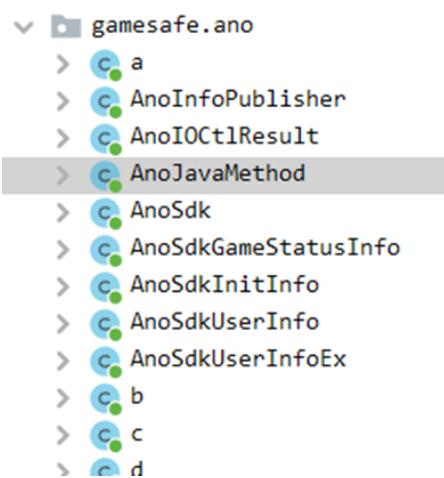


Figura 4.28: Paquete gamesafe.ano

Dentro del mismo, se encontraron métodos en los cuales se accede a la línea de comandos y se ejecutan determinados comandos ofuscados:

```
public static void sendCmd(String str) {
    sendCmdEx(str);
}

public static int sendCmdEx(String str) {
    String str2;
    if (str == null) {
        return -1;
    }
    if (str.compareTo(a.a("didodvgduz")) == 0) {
        f.a();
        return 0;
    } else if (str.startsWith(a.a("ho:"))) {
        MainThreadDispatcher2.SendCmd(str.substring(3));
        return 0;
    } else if (str.startsWith(a.a("dia_xg:"))) {
        a(str.substring(7));
        return 0;
    } else {
        if (str.startsWith(a.a("dn_hvdi_gjjkzm"))) {
            if (Looper.myLooper() != Looper.getMainLooper()) {
                return 0;
            }
            str2 = a.a("DnHvdiOcmzvy");
        } else if (str.startsWith(a.a("hnbwjs:")) || str.startsWith(a.a("cdyz_hnbwjs:"))) {
            n.a().a(str);
            return 0;
        } else if (str.startsWith(a.a("yg_adgz:"))) {
            return 0;
        } else {
            if (str.startsWith(a.a("pkyvoz_vyw_zivwgzy_jqzm_pnw:"))) {
                b(str);
                return 0;
            } else if (str.startsWith(a.a("zszx|"))) {
                return 0;
            } else {
                str2 = "*#07#: " + str;
            }
        }
        b.a(str2);
        return 0;
    }
}
```

Figura 4.29: Función sendCmd

No se ha podido determinar más información acerca de esta clase debido a la dificultad para interpretar el código ofuscado. Asimismo, se ha encontrado un paquete llamado *pay.android*, lo que indica que podría haber compras en la aplicación:

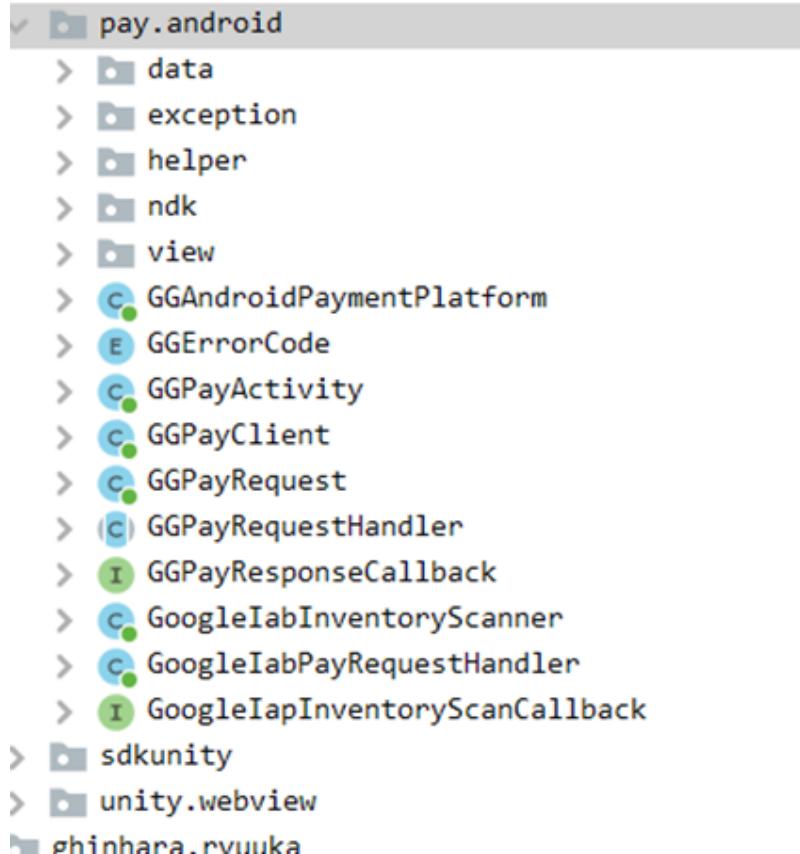


Figura 4.30: Paquete *pay.android*

Otros paquetes encontrados hacen referencia a distintas redes sociales como twitter o vk, lo que indica que quizás haya algún tipo de interacción con dichas redes:

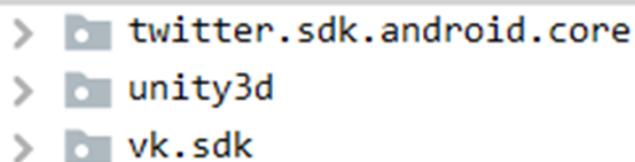


Figura 4.31: Paquetes de otras redes sociales

Por último, se ha encontrado un método que contiene texto codificado en base64. Este método crea un mensaje al abrir la aplicación que muestra el texto “Tech-Bigs.com”. Esto indica que dicha aplicación fue modificada o creada para dicho mercado.

```

protected void onCreate(Bundle paramBundle) {
    Toast.makeText((Context)this, new String(Base64.decode("VEVDSE3JR1MuQ09N", 0)), 1).show();
    Toast.makeText((Context)this, new String(Base64.decode("VEVDSE3JR1MuQ09N", 0)), 1).show();
    Toast.makeText((Context)this, new String(Base64.decode("VEVDSE3JR1MuQ09N", 0)), 1).show();
    GhinharaApp.Start((Context)this);
    if (usingFCM() && this.mUnityPlayer != null) {
        this.mUnityPlayer.quit();
        this.mUnityPlayer = null;
    }
    super.onCreate(paramBundle);
    String str = getMetaDataString("HwAcc");
    if (str != null && str.equalsIgnoreCase("on"))
        getWindow().setFlags(16777216, 16777216);
    str = getMetaDataString("Bugly");
    if (str != null && str.equalsIgnoreCase("on"))
        CrashReport.initCrashReport((Context)this);
    WebViewManager.onActivityResult((Activity)this);
    WebViewManager.setUnityMessenger(UnityMessenger.I());
    FFAPI.MainActivity = this;
    SdkUnity.setActivity((Activity)this);
    MambetWrapper.setActivity((Activity)this);
    MambetWrapper.setUnityMessenger(UnityMessenger.I());
    FFAPI.MetaDataGarenaAppId = Helper.getMetaDataAppId((Context)this);
    watchBattery();
    watchWifi();
    handleIntent(getIntent());
}

```

Figura 4.32: Referencia a nombre del mercado en base64

4.1.3. Conclusiones

En este apartado se ha mostrado el análisis estático de la aplicación de Badoo obtenida del mercado de TechBigs. El objetivo era determinar si la aplicación contiene algún indicio de presencia de malware, adware, spyware, etc.

Para empezar, a pesar de que se anuncia en el mercado como la aplicación de citas Badoo, no se ha encontrado ninguna referencia a dicha aplicación. En su lugar, se han encontrado diversos paquetes que referenciaban aplicaciones distintas, entre ellas, un juego y otra aplicación similar a la original.

Además, se ha podido ver que la aplicación contiene paquetes relacionados con analítica y recolección de determinados datos del dispositivo, los cuales se almacenan en una base de datos. También se han encontrado evidencias de anuncios en la aplicación.

Por otra parte, se han encontrado diversos métodos sospechosos indicativos de la presencia de malware en la aplicación. Entre ellos, se han encontrado métodos que obtienen información del dispositivo para determinar si se trata de un dispositivo real o de un emulador. Seguidamente, se observó que existían diversas funciones que accedían a la línea de comandos, comprobaban si se podían obtener permisos de administrador, etc.

Por tanto, se ha visto que la aplicación no solo no es la original, sino que además se han encontrado evidencias de la presencia de malware.

4.1.4. Análisis Netflix TechBigs

La siguiente aplicación analizada fue Netflix del mercado de TechBigs. En primer lugar, se han revisado los paquetes que componen la aplicación y se ha observado que esta contenía diversas librerías de terceros para la implementación de diversos aspectos de la aplicación, como por ejemplo *me.zhanghai.android.materialprogressbar*, *fr.castorflex.android.circularprogressbar*, *com.miguelcatalan.materialsearchview*, etc. Además, se puede observar que también se incluye la librería de Unity3d como en el caso anterior.

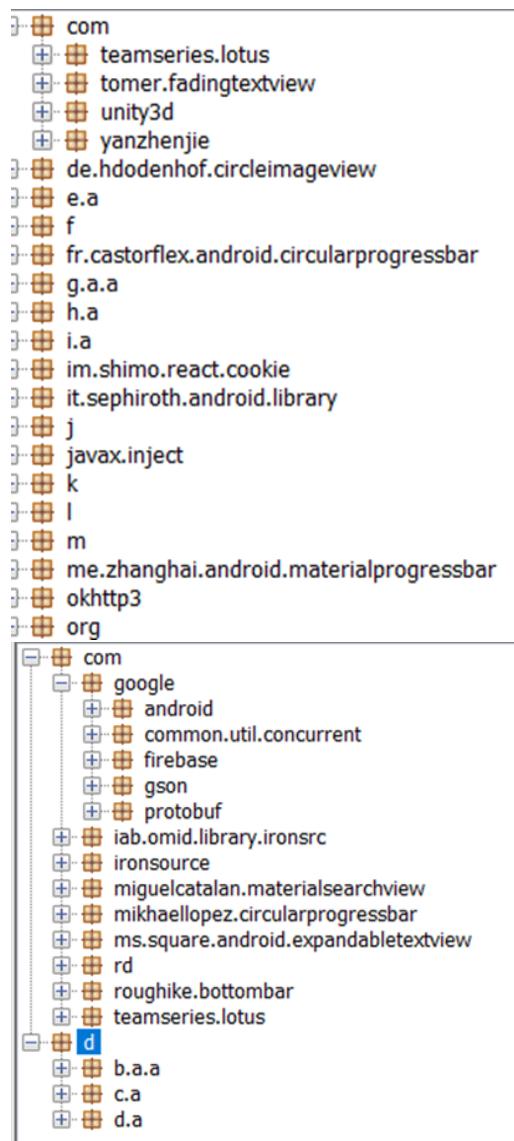


Figura 4.33: Paquetes Netflix Techbigs

Por último, la aplicación contiene un paquete denominado *teamseries.lotus*, el cual, realizando una búsqueda en Google, se ha podido observar que se trata de

la aplicación de TeaTV. En este punto, se puede determinar que no se trata de aplicación de Netflix, sino de una alternativa similar gratuita.

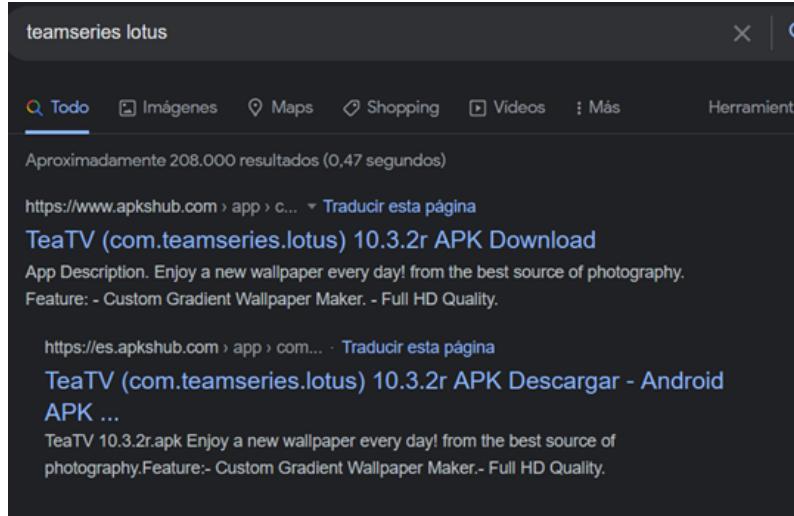


Figura 4.34: Paquete teamseries.lotus

Como sucedía en el caso anterior, el código de algunos de los paquetes de la aplicación estaba ofuscado y, por tanto, no pudo ser completamente analizado, tal y como se muestra en la figura siguiente:

```
private static zzfc zzb(Context context, boolean z) {
    boolean z2;
    boolean z3;
    boolean z4;
    if (zzdk.zzwC == null) {
        synchronized (zzxd) {
            if (zzdk.zzwC == null) {
                zzfc zza = zzfc.zza(context, "11+nowuQKLoZSE4zpeTvU13Gha6AS9UB10MB85g+5uQ=", "e8mp4jaCizT/LsNfz2/GdPr+FhTxb18eTB3yb59Ii1v1t5mZH53X/ZD16c616YtH63r7e8a2B3zPRATmVgCa
                if (zza.isInitialized()) {
                    try {
                        z2 = ((Boolean) zzw.zzqr().zzd(zzabp.zzcfc)).booleanValue();
                    } catch (IllegalStateException e2) {
                        z2 = false;
                    }
                }
                if (z2) {
                    zza("/*byekN5BCG0pEYkmkev8jYid2cU9nElfchq05aiwfVy65sZtpifn+5AI", "PJXTpPH5Q3jP8R73KmmofmTne3UwI4eaifskBpk4=", new Class[]{});
                }
            }
            zzd("STR7QE1n2Nae56fJRHdsAlh+RkDZLmzq8l1SixsR9TdmqE0af2eosWf09RrF9F7xV1", "1s62fuaYcsZqgiuoZcKchtaXoDXTG3ldfDayg76f0w=", Context.class);
            zzd("Q13Pj/nE6T13UShdmlhX9pHzk7Bv0t8s0gyxx65vur6ubTosqD4NeCuaF2H", "PSng0ubxv5jSIS10K01bba4x2Sw0pB55v9jcsu=", Context.class);
            zzd("n+rPrxVxuED4NjJ7p5khYn3L23M6eCDO0C17nP0DPFV27FrR8uzqggz3C6gJr", "UBLuxrBFVB6UPO/u88tFqcN0mwf2TBQ2oJDguA3w=", Context.class);
            zzd("1KMaeb61gb611gTb51ppbCnVAbMn0DTTyXcVBUk1m7R088F2rxeXuH", "Jntmd6ipTxb+if2v9j1lnq8ftlhbKmc/(XRMic8=", Context.class);
            zzd("rHkgI1358161gTb51ppbCnVAbMn0DTTyXcVBUk1m7R088F2rxeXuH", "d0pda04HpLm0jKGm3wDca0i1yveqHqpynaVhfew=", Context.class);
            zzd("j3EPB010H1R1v0PjU9k/1.67wEm124sVjy/1eqlqHYYf711ep/1DnLr+fcCPt58Ym9d97Q0zRzo", "Jntmd6ipTxb+if2v9j1lnq8ftlhbKmc/(XRMic8=", Context.class);
            zzd("Pb3bEQzXTR173k/Bmxiq4pkjDy5KFpSAz1yLw5D/xawc5akyhckyPvixjds", "udr+yQv2Tks7uKw1D0f3tzd5bi011PFB//M+o2Cku=", Context.class, Boolean.TYPE);
            zzd("Sn0p6h6fooyRjeo237gFBCikgofyQuoOhqf4g1+vBpAR/vcB0o3G6gpLqvgx", "GaekbaK5/u0/kv0escu/vCn6oAgj8hp7d8qy0kly=" , Context.class);
            zzd("4/jkG300u1nSP03h/Q0B19grdvMSag1Inx7GhBDYK5dd44nJvh75Xskr+o21Z", "k1N42J3NxXXU14t4m0jDdUp79r5Xkf10pyghYhUXQ=", MotionEvent.class, DisplayMetrics);
            zzd("41jqeaQhH0jle921AyfVnxn0nRDw0/uhExoVv0yIA16FK01TnpLcnJzPPNvZh", "Ubqr1nfw//bzB2e20u1Lf0m+d4L3d9r5Pymbdws5Y=", MotionEvent.class, DisplayMetrics);
            zzd("C91vawoCR/xAdhlin1fnUuLnxSpdxk0ut3y73yjwBhTE1C2w/m2FwsonndoD9", "og/heiru//pr/jaa0AX0sdZ9U14Ly10fV9P2pT3so=", new Class[]{});
            zzd("B4TR7m/Bq+TfByOrh//gxFeBebRXBuKj90cNcnfaoxkrJfspaAZ6GEnokp520", "Xg7RxuN0BpRUuOxgvE2glfJ7V33s4uGw16QoQvencs=", new Class[]{});
            zzd("nbPsfSCDR7wEzUUU/EV3UV5e010w/11xFpbY4z45GUQ3bGuJDDqj1Snpq4Zee", "hv9/8RfhfTS6D1p17knLUS9DwBxLY+fYtShuV7y=", new Class[]{});
            zzd("ME5k6h6focvy25ap8910Rx1z2Hvr07+7p7TpB7HB4pxY1cfUmqyRV02d81mw/Xw", "jw17Vys5wVvYfHW3d+q7st+k2swx61cm72p99y=", new Class[]{});
            zzd("38+C1L8111ckz2oZvFx6LDfbTOOX4v6V24s6fFduoRadympBzffrtgTY61Aqe", "22Bt4+15jov1yob2x2yEd8dpCpxQ5GoxThefIU2m07y=", new Class[]{});
            zzd("jKnA9a3w1jdsfP6NGymR8hX8L1fG/680v0EVKezorg+AGzgtIGAV0d7Zlq8htD", "9opNs06j5kfhoIuId+99PlunDxaApJ7uM1m1z18=", new Class[]{});
            zzd("v"uhy+EAlk1pH44kRKY3xjTgCn5auHH1175HL875AKd9+bh3Xrhikk1Ujk4X099", "y6Dr+0A0tOelH+QqId82r0v0shlwqEvlddy1009c=", Context.class, Boolean.TYPE);
            zzd("Zrgjlpv6CoFYR+X9h6fJfnnAxPtz7sE361IO80y3rZakvClpyZG1wJte324C2", "ROScE0l6mPn023olxLxovnL/S2h2HerkUgilrigc", StackTraceElement[], class);
            zzd("B1f9hZtD0n1k6Ax36FXxEpk5nqo6wTv7me:URv1t1Gk1A17kb6f4+H+hciq7Swf", "ynd1tk1C11y7kefK045Vj/yiandb0nEx1bgd0uVx0o=", View.class, DisplayMetrics.class);
            zzd("fZpWn43DvY3H+Alfkge4xxr1ptHw7BaFyjw7Xdp6kQ1HEPsBezyklip7K3J", "YqrFXTpE0zhqgXdnxu46L91fkB49h6q1edSVltTk=", Context.class, Boolean.TYPE);
            zzd("W187GCNEdGubbyq1s01Ccfexx1Ahm1z10yxtetzesVgqBQ8QxVtX2Ve757+", "Axpv7G/Ygsm2PLxyasiVhCsi+Z3isYbfhhkk0tW/Q3s=", View.class, Activity.class, Boolean);
            zzd("Gnr+tm558fqcKz1CaokD17w5tnxqj2AnkvilVlog1loq7+v+cqu0eljhnoVb1uVHD", "99fd4T4VL+CcQqohIB+ZSPesMTz3PoWVXNiqb2jY3Ak=", Long.TYPE);
        }
        z3 = ((Boolean) zzw.zzqr().zzd(zzabp.zzcsk)).booleanValue();
    } catch (IllegalStateException e3) {
        z3 = false;
    }
    if (z3) {
        zza("D9q6Pp7iuKLjvrfjEfTufrDzVuPRI0AEk5gpBh2xnatOnF8jgkD1/z48d3PucR", "3ok2i0qpxXA6wdGctKZBw1kUdo7fvz8PU1ipEtW1E=", Context.class);
    }
    zza("8tnV51Kfky7d5sou/L0zGp9CvTnR1x1NPZwzeYndAYt1he6k3/YSKoBCdPqe624V4", "8GsaZh20Tvj2/a2ObTwq/0y0K9P13h2Q3PbYuUhk57cc=", Context.class);
    try {
        z4 = ((Boolean) zzw.zzqr().zzd(zzabp.zzcsm)).booleanValue();
    } catch (IllegalStateException e4) {
        z4 = false;
    }
}
```

Figura 4.35: Código de la aplicación ofuscado

Se ha analizado el código legible de la aplicación, en el cual se han encontrado diversos fragmentos que indican la presencia de anuncios en la aplicación. Uno ejemplo se muestra a continuación:

```
public static String[] a(Context paramContext) {
    Class<?> clazz = Class.forName("com.google.android.gms.ads.identifier.AdvertisingIdClient");
    Object object = clazz.getMethod("getAdvertisingIdInfo", new Class[] { Context.class }).invoke(clazz, new Object[] { paramContext });
    Method method2 = object.getClass().getMethod("getId", new Class[0]);
    Method method1 = object.getClass().getMethod("isLimitAdTrackingEnabled", new Class[0]);
    String str = method2.invoke(object, new Object[0]).toString();
    boolean bool = ((Boolean)method1.invoke(object, new Object[0])).booleanValue();
    object = new StringBuilder();
    object.append(bool);
    return new String[] { str, object.toString() };
}
```

Figura 4.36: Presencia de anuncios en la aplicación

También se ha encontrado otro fragmento en el cual se descarga un archivo denominado *mraid.js*, el cual es utilizado para la inclusión de anuncios. MRAID (Mobile Rich Media Ad Interface Definitions) es una API que permite la comunicación entre los anuncios y las aplicaciones, así como acceder a ciertas funciones del dispositivo [40].

```
public final WebResourceResponse shouldInterceptRequest(WebView param1WebView, String paramString) {
    boolean bool;
    Logger.i("shouldInterceptRequest", paramString);
    try {
        bool = (new URL(paramString)).getFile().contains("mraid.js");
    } catch (MalformedURLException malformedURLException) {
        bool = false;
    }
    if (bool) {
        StringBuilder stringBuilder = new StringBuilder("file:///");
        stringBuilder.append(w.u(this.a));
        stringBuilder.append(File.separator);
        stringBuilder.append("mraid.js");
        String str = stringBuilder.toString();
        File file = new File(str);
        try {
            new FileInputStream(file);
            return new WebResourceResponse("text/javascript", "UTF-8", getClass().getResourceAsStream(str));
        } catch (FileNotFoundException fileNotFoundException) {}
    }
    return super.shouldInterceptRequest(param1WebView, paramString);
}
```

Figura 4.37: Referencia a mraid.js

Además, se han encontrado referencias a unityads de China. Esto indica que la aplicación podría ser de origen chino:

```
public static String getDefaultConfigUrl(String paramString) {
    String str;
    if (isChinaLocale(Device.getNetworkCountryISO())) {
        str = "https://config.unityads.unitychina.cn/webview/";
    } else {
        str = "https://config.unityads.unity3d.col/webview/";
    }
}
```

Figura 4.38: Referencia a unityads de China

Otro fragmento que evidencia esto es el método que se muestra en la figura siguiente, en el cual se comprueba si el dispositivo se está ejecutando en China:

```
public static String getDefaultConfigUrl(String str) {
    String str2 = isChinaLocale(Device.getNetworkCountryISO()) ? "https://config.unityads.unitychina.cn/webview/" : "https://config.unityads.unity3d.col/webview/";
    return str2 + getWebViewBranch() + "/" + str + "/config.json";
}

public static boolean isChinaLocale(String paramString) {
    return (paramString.equalsIgnoreCase("CN") || paramString.equalsIgnoreCase("CHN"));
}
```

Figura 4.39: Método isChinaLocale

Continuando con el análisis del código, se han encontrado fragmentos que señalan al acceso a diversos directorios del sistema. Esta función se utiliza para comprobar si el teléfono tiene acceso root, de forma similar a como sucedía con la aplicación anteriormente analizada:

```
private static boolean b(String paramString) {
    int i = 0;
    while (true) {
        if (i < 8) {
            try {
                (new String[8])[0] = "/sbin/";
                (new String[8])[1] = "/system/bin/";
                (new String[8])[2] = "/system/xbin/";
                (new String[8])[3] = "/data/local/xbin/";
                (new String[8])[4] = "/data/local/bin/";
                (new String[8])[5] = "/system/sd/xbin/";
                (new String[8])[6] = "/system/bin/failsafe/";
                (new String[8])[7] = "/data/local/";
                String str = (new String[8])[i];
                StringBuilder stringBuilder = new StringBuilder();
                stringBuilder.append(str);
                stringBuilder.append(paramString);
                boolean bool = (new File(stringBuilder.toString())).exists();
                if (bool)
                    return true;
                i++;
            } catch (Exception exception) {
                return false;
            }
            continue;
        }
        return false;
    }
}
```

Figura 4.40: Método de comprobación de acceso root

Además, se han encontrado en el código de la aplicación diversos enlaces a distintas páginas web de streaming, los cuales se listan a continuación:

- <https://secretlink.xyz>
- <https://supernova.to>
- <https://lookmovie2.to>
- <https://myflixer.to>
- <https://yesmovies.vc>
- <https://abcvideo.cc>
- <https://streamtape.com>
- <https://mixdrop.co>
- <https://vidoza.net>
- <https://aparat.cam>
- <https://wolfstream.tv>
- <https://mixdrop.co>

- <https://api.cartoonhd.biz>
- <https://api.teatv.live>
- <https://openload.pw>
- <https://supernova.to>
- <https://series9.la>
- <https://mixdrop.co>
- <https://streamlare.com/api/video/stream/get>
- https://sflix.to/ajax/get_link
- <https://voe.sx>
- <https://dood.ws>
- <https://dood.to>
- <https://dood.watch>
- <https://mzzcloud.file>
- <https://thenos.org>
- <https://upstream.to>
- <https://vf-film.co>
- <https://vf-serie.org>
- <https://vudeo.io>
- <https://vidhotel.com>
- <https://mixdrop.co/e>
- <https://videobin>
- <https://streamzz>
- <https://v2.vidsrc.me/src>
- <https://watchepisodes4.com>
- <https://watchseries.net>
- <https://sbplay2.xyz>
- <https://watchserieshd.co>

- <https://phimoi.net/tiem-kiem/>
- <https://primewire.ag/index.php>
- <https://gowatchfreemovies.to>
- <https://watchepisodeseries.com>

La aplicación se sirve de los diferentes enlaces mostrados anteriormente para ofrecer servicios de streaming de series y películas de forma similar a como lo hace Netflix. Sin embargo, estas páginas son ilegales y su uso conlleva riesgos de seguridad. También se han encontrado dos cadenas que podrían corresponderse con las API Keys de alguno de los servicios de streaming a los que accede la aplicación:

```
public void t() {
    this.L3 = new i1();
    if (g.o(getApplicationContext()) {
        this.L3.a(new k1[] { new k1(728, 90, "92952e92-04b3-4c3a-87fc-46502007cc43") });
    } else {
        this.L3.a(new k1[] { new k1(320, 50, "3d7efb40-5650-4853-8254-374fe014fd4f") });
    }
    this.L3.b((x0)new c0(this));
}
```

Figura 4.41: API Keys encontradas en Netflix de Techbigs

Se ha encontrado también el código correspondiente a la descarga de subtítulos. Estos se obtienen del API de opensubtitles y se almacenan en la carpeta de descargas:

```
protected String a(String... paramVarArgs) {
    if (paramVarArgs[0].contains("opensubtitles"))
        try {
            XmlRpcClientConfigImpl xmlRpcClientConfigImpl = new XmlRpcClientConfigImpl();
            xmlRpcClientConfigImpl.setServerURL(new URL("http://api.opensubtitles.org/xml-rpc"));
            XmlRpcClient xmlRpcClient = new XmlRpcClient();
            xmlRpcClient.setConfig((XmlRpcClientConfig)xmlRpcClientConfigImpl);
            Map map = (Map)xmlRpcClient.execute("LogOut", new Object[] { this.@m("token_user_default") });
        } catch (ClassCastException classCastException) {
            classCastException.printStackTrace();
        } catch (XmlRpcException xmlRpcException) {

        } catch (IndexOutOfBoundsException indexOutOfBoundsException) {

        } catch (NullPointerException nullPointerException) {

        } catch (MalformedURLException malformedURLException) {}
    try {
        StringBuilder stringBuilder1 = new StringBuilder();
        stringBuilder1.append(Environment.getExternalStorageDirectory().getAbsolutePath());
        stringBuilder1.append("/Download/");
        String str = stringBuilder1.toString();
        if (!TextUtils.isEmpty(a.g@().l()))
            str = a.g@().l();
        File file2 = new File(str);
        if (!file2.exists())
            file2.mkdirs();
        HttpURLConnection httpURLConnection = (HttpURLConnection)(new URL(paramVarArgs[0])).openConnection();
        StringBuilder stringBuilder2 = new StringBuilder();
        stringBuilder2.append(str);
        stringBuilder2.append("downloadfile");
        File file1 = new File(stringBuilder2.toString());
        if (file1.exists())
            file1.delete();
        FileOutputStream fileOutputStream = new FileOutputStream(file1);
        InputStream inputStream = httpURLConnection.getInputStream();
        byte[] arrayOfByte = new byte[4096];
        while (true) {
            int i = inputStream.read(arrayOfByte);
            if (i != -1) {
                fileOutputStream.write(arrayOfByte, 0, i);
                continue;
            }
            fileOutputStream.close();
            inputStream.close();
            return file1.getAbsolutePath();
        }
    } catch (Exception exception) {
        exception.printStackTrace();
    }
}
```

Figura 4.42: Descarga de subtítulos

Por último, se han encontrado unas conexiones a *localhost* cuyo objetivo no pudo ser determinado:

```
public static <T> a(int paramInt, ResponseBody paramResponseBody) {
    if (paramInt >= 400)
        return a(paramResponseBody, (new Response.Builder()).code(paramInt).protocol(Protocol.HTTP_1_1).request((new Request.Builder()).url("http://localhost/").build()).build());
    StringBuilder stringBuilder = new StringBuilder();
    stringBuilder.append("code < 400: ");
    stringBuilder.append(paramInt);
    throw new IllegalArgumentException(stringBuilder.toString());
}

public static <T> a(T paramT) {
    return a(paramT, (new Response.Builder()).code(200).message("OK").protocol(Protocol.HTTP_1_1).request((new Request.Builder()).url("http://localhost/").build()).build());
}
```

Figura 4.43: Conexiones a localhost

Para comprender más en profundidad de qué se trata la aplicación, se ha instalado en un emulador. La aplicación tiene el icono de Netflix y la interfaz es muy similar, sin embargo, como se ha visto se trata de una aplicación completamente diferente.

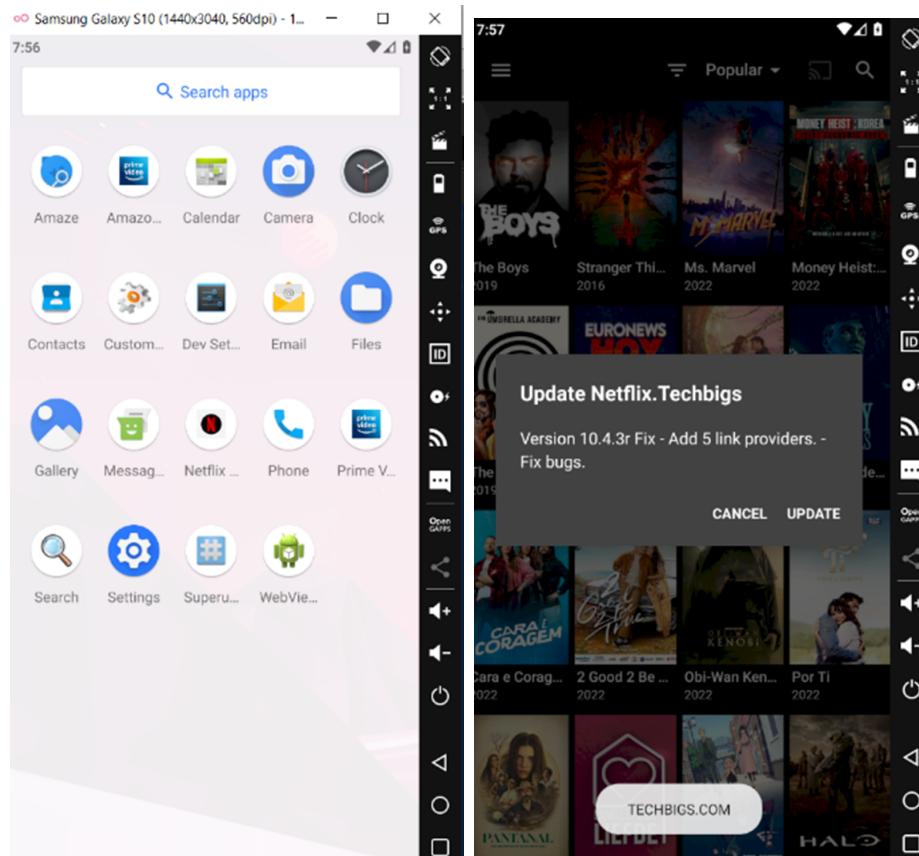


Figura 4.44: Mensaje de la aplicación

Como se puede ver en la figura anterior, al abrir la aplicación, esta notifica de la necesidad de actualizar. Al pulsar el botón de *update* la aplicación pide permisos para acceder al almacenamiento. Esto es muy peligroso, ya que un atacante podría instalar malware en el dispositivo de esta forma. Además, al tratar de reproducir

algún vídeo, salta de nuevo otra alerta con la recomendación de descargar otro reproductor de vídeo distinto, lo cual también podría resultar peligroso.

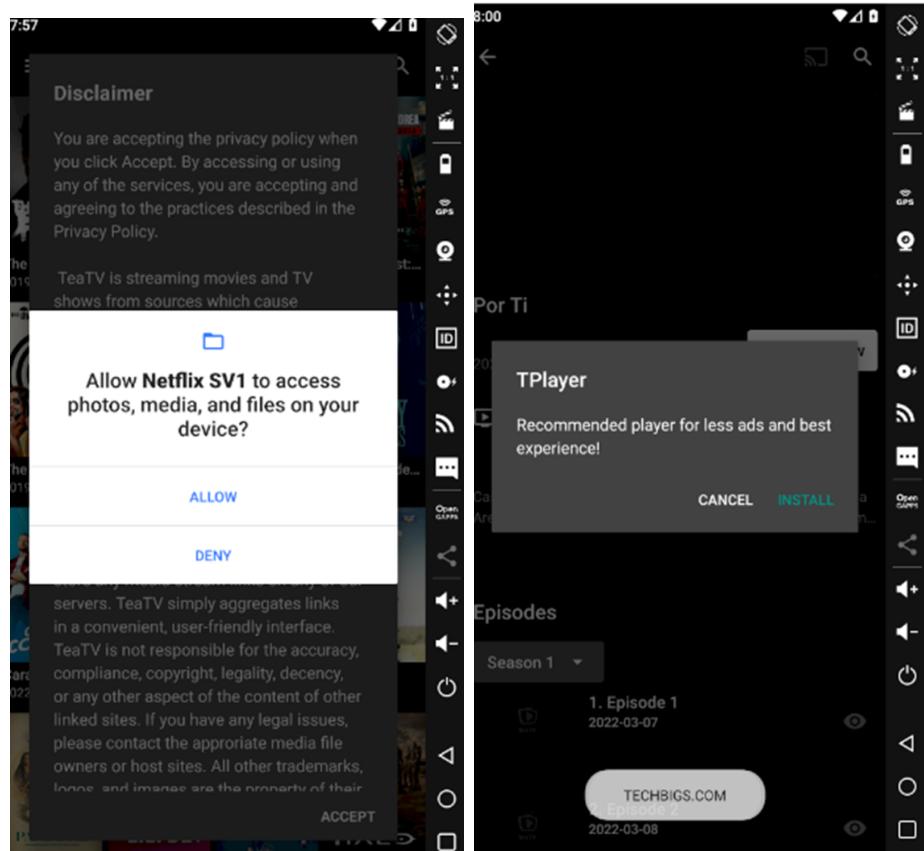


Figura 4.45: Intentos de acceso al almacenamiento

Observando la interfaz, podemos determinar que los enlaces que se observaron en el código son los que se muestran para visualizar las distintas series y películas en la aplicación:

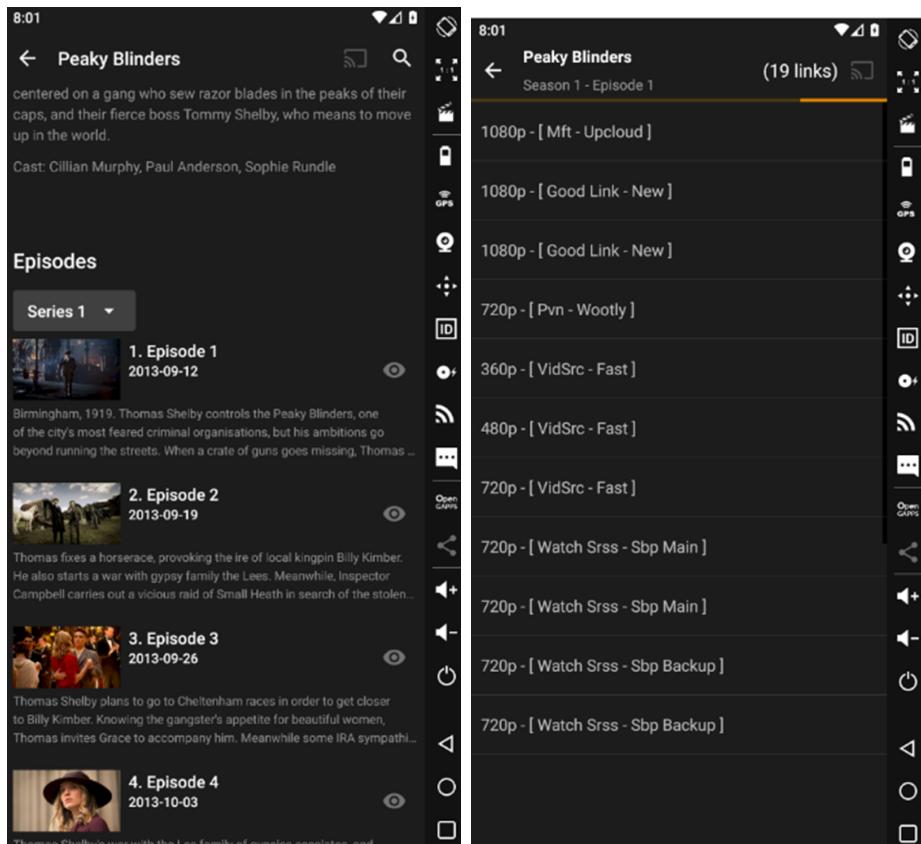


Figura 4.46: Interfaz de la aplicación de Netflix de Techbigs

4.1.5. Conclusiones

Se ha visto que la aplicación analizada, a pesar de ser anunciada como Netflix premium y tener el mismo icono, no está relacionada con ella. Se trata de una aplicación que imita la interfaz de la aplicación original, sin embargo, las series y películas que muestra son obtenidas de diversas páginas que ofrecen servicios de streaming ilegal.

No es posible determinar con seguridad si la aplicación contiene algún tipo de malware simplemente con el análisis estático, ya que como se ha visto parte del código estaba ofuscado y eso hace que sea difícil interpretarlo. Sin embargo, el hecho de que la aplicación acceda a tantos servicios de streaming ilegales y requiera instalar otras aplicaciones las cuales se instalarían en el dispositivo tras aceptar los permisos de escritura y lectura del almacenamiento es en sí un indicativo de que la aplicación no es segura.

Por otra parte, el hecho de que la aplicación descargue archivos de páginas externas en el dispositivo puede resultar muy peligroso, ya que no es una aplicación de confianza. Además, se ha visto que la aplicación contiene anuncios.

4.1.6. Análisis DisneyPlus APKDone

La siguiente aplicación analizada fue DisneyPlus descargada del mercado de APKDone. Lo primero que se ha observado es que en los paquetes existen referencias a “Disney”, al contrario que sucedía en las aplicaciones anteriores en las cuales no existía mención a la aplicación original. El código de la aplicación será analizado en búsqueda de fragmentos sospechosos.

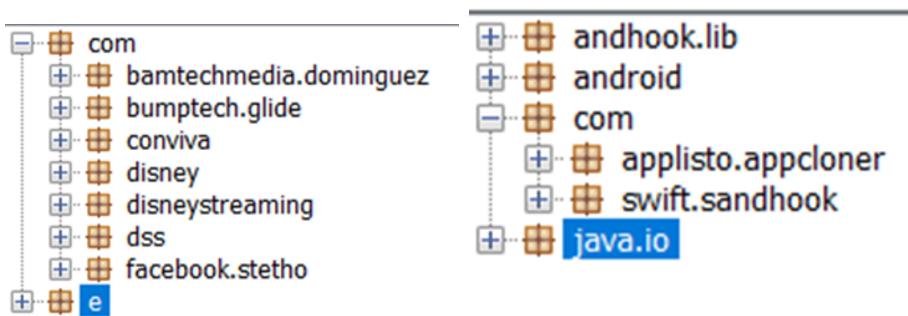


Figura 4.47: Paquetes de la aplicación Disney+ de APKDone

Lo primero que se encontró dentro del paquete *conviva* es un enlace a un web cuyo acceso muestra un error 404.

```
public Client(b paramb, d paramd, String paramString) {
    if (!paramb.a())
        return;
    try {
        if ((new URL("https://cws.conviva.com")).getHost().equals((new URL(paramb.c)).getHost()))
            this.d = true;
    } catch (MalformedURLException malformedURLException) {}
    if (paramString != null)
        this.e = paramString;
    this.f = "4.0.12.155";
    b b1 = new b(paramb);
    this.g = b1;
    b1.h = paramString;
    this.h = paramd;
    paramd.i("SDK", b1);
    com.conviva.utils.c c1 = this.e.c();
    this.i = c1;
    try {
        c1.b(new j(this, this, (Client)paramb), "Client.init");
        x();
        this.j = true;
        return;
    } catch (Exception exception) {
        this.j = false;
        this.e = null;
        this.i = null;
        SessionFactory sessionFactory = this.b;
        if (sessionFactory != null)
            sessionFactory.f();
        this.b = null;
    }
}
```

Figura 4.48: Enlace a dominio de *cws.conviva*

Además, en el mismo paquete se encontró otro enlace:

```
class m implements Callable<Void> {
    m(Client this$0) {}

    public Void a() throws Exception {
        String str = this.a.c();
        if (str != null && e.d.d.a.c != str) {
            String str1 = (Client.e(this.a)).c;
            str = "testonly.conviva.com";
            if (!str1.contains("testonly.conviva.com"))
                str = "cws.conviva.com";
            if (Client.n(this.a) < 0) {
                ContentMetadata contentMetadata = new ContentMetadata();
                HashMap<Object, Object> hashMap = new HashMap<Object, Object>();
                contentMetadata.b = (Map)hashMap;
                hashMap.put("c3.IPV4IPV6Collection", "T");
                Map<String, String> map = contentMetadata.b;
                StringBuilder stringBuilder = new StringBuilder();
                stringBuilder.append("ipv4.");
                stringBuilder.append(str);
                map.put("c3.domain", stringBuilder.toString());
                Client client = this.a;
                Client.o(client, Client.k(client).l(contentMetadata, SessionFactory.SessionType.HINTED_IPV4));
            }
            if (Client.p(this.a) < 0) {
                ContentMetadata contentMetadata = new ContentMetadata();
                HashMap<Object, Object> hashMap = new HashMap<Object, Object>();
                contentMetadata.b = (Map)hashMap;
                hashMap.put("c3.IPV4IPV6Collection", "T");
                Map<String, String> map = contentMetadata.b;
                StringBuilder stringBuilder = new StringBuilder();
                stringBuilder.append("ipv6.");
                stringBuilder.append(str);
                map.put("c3.domain", stringBuilder.toString());
                Client client = this.a;
                Client.q(client, Client.k(client).l(contentMetadata, SessionFactory.SessionType.HINTED_IPV6));
            }
        }
        return null;
    }
}
```

Figura 4.49: Enlace al dominio *testonly.conviva*

Se accedió a dicho dominio, el cual redirigió a otro:

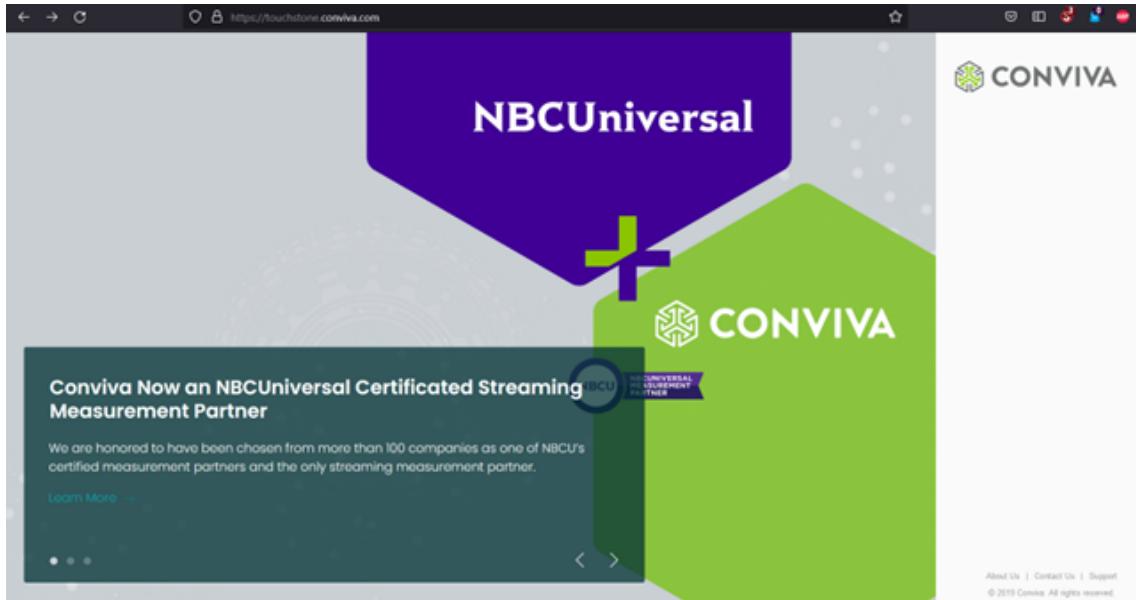


Figura 4.50

Sin embargo, se ha descubierto que se trata de una empresa relacionada con el streaming. Por lo que, a priori, se puede descartar como amenaza.

Por otra parte, la existencia de un paquete denominado *apk_cloner* indica que se podría haber usado alguna herramienta para clonar la aplicación original, que posteriormente sería modificada. En las siguientes figuras se muestran diferentes fragmentos de código en los cuales se evidencia esto:

```

public ParcelFileDescriptor openFile(Uri paramUri, String paramString) throws FileNotFoundException {
    File file;
    StringBuilder stringBuilder1;
    String str = TAG;
    StringBuilder stringBuilder2 = new StringBuilder();
    stringBuilder2.append("openFile; uri: ");
    stringBuilder2.append(paramUri);
    Log.i(str, stringBuilder2.toString());
    Context context = getContext();
    if ("/Image.png".equals(paramUri.getPath()) && "r".equals(paramString)) {
        file = new File(context.getCacheDir(), "share_image.png");
        paramString = TAG;
        stringBuilder1 = new StringBuilder();
        stringBuilder1.append("openFile; returning share image file descriptor; file: ");
        stringBuilder1.append(file);
        Log.i(paramString, stringBuilder1.toString());
        return ParcelFileDescriptor.open(file, 268435456);
    }
    if (Utils.checkNotNull((Context)stringBuilder1))
        try {
            if ("/cloneSettings".equals(file.getPath())) {
                File file1 = CloneSettings.getInstance((Context)stringBuilder1).getCloneSettingsFile();
                if ("r".equals(paramString)) {
                    Log.i(TAG, "openFile; returning clone settings file MODE_READ_ONLY file descriptor...");
                    return ParcelFileDescriptor.open(file1, 268435456);
                }
                if ("w".equals(paramString)) {
                    Log.i(TAG, "openFile; returning clone settings file MODE_WRITE_ONLY file descriptor...");
                    return ParcelFileDescriptor.open(file1, 738197504);
                }
            }
        } catch (FileNotFoundException fileNotFoundException) {
            throw fileNotFoundException;
        } catch (Exception exception) {
    }
}

public class DefaultProvider extends AbstractContentProvider {
    private static final int MAX_COUNT = 10;

    private static final String PREF_KEY_COUNT = "com.applisto.appcloner.classes.DefaultProvider.count";
    private static final String PREF_KEY_IDENTITY_SEED = "com.applisto.appcloner.classes.identitySeed";
    private static final String PREF_KEY_ORIGINAL_INSTALL_VERSION_CODE = "com.applisto.appcloner.classes.originalInstallVersionCode";
    public static final String PREF_KEY_PREFIX = "com.applisto.appcloner.classes.";
    private static final String TAG = DefaultProvider.class.getSimpleName();

    private static String getOriginalPackageName(Context paramContext) {
        try {
            Bundle bundle = (paramContext.getPackageManager().getApplicationInfo(paramContext.getPackageName(), 128)).metaData;
            if (bundle.getBoolean("com.applisto.appcloner.appClonerDevDevice")) {
                Utils.sDevDevice = Boolean.valueOf(true);
                Log.sEnabled = true;
            }
            return new String(Base64.decode(bundle.getString("com.applisto.appcloner.originalPackageName"), 0));
        } catch (Exception exception) {
            Log.w(TAG, exception);
            return null;
        }
    }

    public static Object invokeSecondaryInstance(Context paramContext, String paramString1, String paramString2, Object... paramVarArgs) throws Exception {
        ClassLoader classLoader = Utils.getSecondaryClassLoader(paramContext);
        StringBuilder stringBuilder = new StringBuilder();
        stringBuilder.append("com.applisto.appcloner.classes.secondary.");
        stringBuilder.append(paramString1);
        Class<?> clazz = classLoader.loadClass(stringBuilder.toString());
        for (Method method : clazz.getMethods()) {
    }
}

```

Figura 4.51: Referencias a AppCloner

Analizando en más profundidad las diferentes clases del paquete, se ha observado que algunas de ellas contienen nombres que podrían resultar sospechosos, entre ellas, FakeCalculator y FakeCamera, tal y como se puede apreciar en la figura que se muestra a continuación:

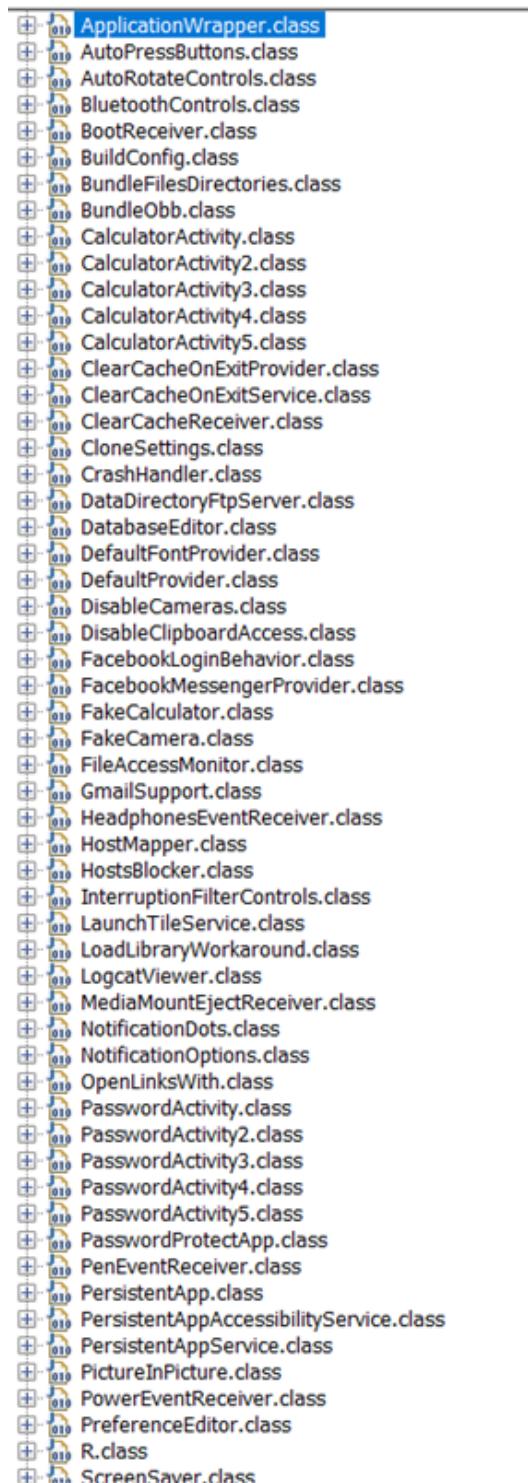


Figura 4.52: Clases sospechosas

Entre estas clases se ha encontrado también lo que parece ser evidencia de creación y acceso a un servidor FTP:

```
private static void startFtpServer(Context paramContext) {
    String str = TAG;
    StringBuilder stringBuilder = new StringBuilder();
    stringBuilder.append("startFtpServer; sStarted: ");
    stringBuilder.append(sStarted);
    Log.i(str, stringBuilder.toString());
    if (sStarted == null)
        try {
            String str1;
            Notification notification;
            Notification.Builder builder;
            int i = sPort;
            if (i == 0) {
                sPort = getFreePort();
                str = TAG;
                stringBuilder = new StringBuilder();
                stringBuilder.append("startFtpServer; generated new port; sPort: ");
                stringBuilder.append(sPort);
                Log.i(str, stringBuilder.toString());
            }
            str = paramContext.getFilesDir().getParent();
            Method method = Class.forName("com.applisto.appcloner.ftpserver.AppClonerFtpServer").getMethod("startFtpServer", new Class[] { int.class, String.class, String.class });
            try {
                method.invoke(null, new Object[] { Integer.valueOf(sPort), "appcloner", "appcloner", str });
            } catch (Exception exception) {
                Log.w(TAG, exception);
                sPort = getFreePort();
                String str4 = TAG;
                StringBuilder stringBuilder3 = new StringBuilder();
                stringBuilder3.append("startFtpServer; generated new port; sPort: ");
                stringBuilder3.append(sPort);
                Log.i(str4, stringBuilder3.toString());
                method.invoke(null, new Object[] { Integer.valueOf(sPort), "appcloner", "appcloner", str });
            }
            sStarted = Boolean.valueOf(true);
            String str2 = $StringsProperties.getProperty("data_directory_ftp_server_started_message");
            Inet4Address inet4Address = Utils.<Inet4Address>getWifiInetAddress(Inet4Address.class);
            StringBuilder stringBuilder2 = new StringBuilder();
            stringBuilder2.append("ftp://");
            stringBuilder2.append("appcloner");
            stringBuilder2.append(":");
            stringBuilder2.append("appcloner");
            stringBuilder2.append("@");
            if (inet4Address != null) {
                str1 = inet4Address.getHostAddress();
                str1 = str1 + str;
            }
        } catch (Exception exception) {
            Log.e(TAG, exception);
        }
}
```

Figura 4.53: Servidor FTP

Además, se ha encontrado un método que parece obtener un archivo json con opciones de clonado, el cual está cifrado:

```
private static String getDefaultCloneSettingsJson() {
    Log.i(TAG, "getDefaultCloneSettingsJson; ");
    try {
        InputStream inputStream = CloneSettings.class.getResourceAsStream("cloneSettings.json");
        try {
            String str = (new SimpleCrypt("UYGy723!Po-efjve")).decrypt(toString(inputStream));
        } finally {
            try {
                exception.close();
            } catch (Exception exception1) {}
        }
    } catch (Exception exception) {
        Log.w(TAG, exception);
        while (true) {
            try {
                while (true)
                    Thread.sleep(1000L);
                break;
            } catch (InterruptedException interruptedException) {}
        }
    }
}
```

Figura 4.54

A continuación, se puede observar un fragmento de la clase FakeCamera:

```

private static void showNotification() {
    Log.i(TAG, "shownotification");
    hideNotification();
    try {
        NotificationManager notificationManager = (NotificationManager)sContext.getSystemService("notification");
        if (notificationManager != null) {
            String str1;
            Intent intent = new Intent("com.applisto.appcloner.action.FAKE_CAMERA_SELECT_PICTURE");
            intent.setPackage(sContext.getPackageName());
            Context context = sContext;
            boolean bool = false;
            PendingIntent pendingIntent = PendingIntent.getBroadcast(context, 0, intent, 0);
            String str2 = $stringsProperties.getProperty("fake_camera_title");
            if ($bitmap != null) {
                str1 = $stringsProperties.getProperty("fake_camera_use_picture_message");
            } else {
                str1 = $stringsProperties.getProperty("fake_camera_select_picture_message");
            }
            NotificationBuilder builder = (new Notification.Builder(sContext)).setContentTitle(str2).setContentText(str1).setContentIntent(pendingIntent).setAutoCancel(true).setWhen(0L);
            if ($bitmap != null && Build.VERSION.SDK_INT >= 16) {
                builder.setStyle(NotificationStyle)(new Notification.BigPictureStyle()).setBigContentTitle(str2).setSummaryText(str1).bigPicture($bitmap));
            if (Build.VERSION.SDK_INT >= 21) {
                Intent intent1 = new Intent("com.applisto.appcloner.action.FAKE_CAMERA_ROTATE_CLOCKWISE");
                intent1.setPackage(sContext.getPackageName());
                PendingIntent pendingIntent1 = PendingIntent.getBroadcast(sContext, 0, intent1, 0);
                Intent intent2 = new Intent("com.applisto.appcloner.action.FAKE_CAMERA_ROTATE_ANTI_CLOCKWISE");
                intent2.setPackage(sContext.getPackageName());
                PendingIntent pendingIntent2 = PendingIntent.getBroadcast(sContext, 0, intent2, 0);
                builder.addAction(new Notification.Action(0, $stringsProperties.getProperty("fake_camera_rotate_right_label"), pendingIntent1));
                builder.addAction(new Notification.Action(0, $stringsProperties.getProperty("fake_camera_rotate_left_label"), pendingIntent2));
            }
        }
        if (System.currentTimeMillis() - sPictureTakenTimestamp > 1000L)
            bool = true;
        Utils.setSmallNotificationIcon(builder, bool);
        notificationManager.notify(556712456, builder.getNotification());
    } catch (Exception exception) {
        Log.w(TAG, exception);
    }
}

```

Figura 4.55: Clase *FakeCamera*

Parece ser en esta clase en donde se hace uso del permiso de cámara que se observó anteriormente. Sin embargo, el objetivo de la existencia de esta clase no pudo ser determinado. En esta misma clase se requiere el permiso “READ_EXTERNAL_STORAGE” para acceder a la galería. En general, se ha visto que en este paquete se requieren muchos de los permisos visualizados en el análisis automatizado.

Por otra parte, se han encontrado diversos enlaces que parecen ser dominios legítimos de Disney:

- <https://registerdisney.go.com>
- <https://disneyplus.com>
- <https://disneyplus.com/downloads>
- <https://appconfigs.disney-plus.net>
- <https://prod-ripcut-delivery.disney-plus.net>

También se han encontrado otros enlaces a lo que parecen ser configuraciones almacenadas en github, tal y como se puede observar en la Figura 4.56.

```

public static final class a extends a<CreateGroup> {
    public a(String paramString1, String paramString2, String paramString3, String paramString4, long paramLong, String paramString5) {
        super("urn:dss:event:groupwatch:fed:group:createRequested", "https://github.bamtech.co/schema-registry/tree/master/dss/event/engagement/group-watch/client/1.0.0/group/create-requested.os2.yaml", "", new CCE());
    }
}

public static final class b extends a<CreatePlayhead> {
    public b(String paramString1, String paramString2, long paramLong, PlayState paramString3, String paramString4, String paramString5) {
        super("urn:dss:event:groupwatch:fed:playhead:createRequested", "https://github.bamtech.co/schema-registry/tree/master/dss/event/engagement/group-watch/client/1.0.0/playhead/create-requested.os2.yaml", "", new CCE());
    }
}

public static final class c extends a<LeaveGroup> {
    public c(String paramString) {
        super("urn:dss:event:groupwatch:fed:group:deviceLeaveRequested", "https://github.bamtech.co/schema-registry/tree/master/dss/event/engagement/group-watch/client/1.0.0/group/device-leave-requested.os2.yaml", "");
    }
}

public static final class d extends a<RequestGroupState> {
    public d(String paramString) {
        super("urn:dss:event:groupwatch:fed:group:groupStateRequested", "https://github.bamtech.co/schema-registry/tree/master/dss/event/engagement/group-watch/client/1.0.0/group/group-state-requested.os2.yaml", "");
    }
}

public static final class e extends a<JoinGroup> {
    public e(String paramString1, String paramString2, String paramString3, String paramString4) {
        super("urn:dss:event:groupwatch:fed:group:joinRequested", "https://github.bamtech.co/schema-registry/tree/master/dss/event/engagement/group-watch/client/1.0.0/group/join-requested.os2.yaml", "", new JoinGroup());
    }
}

```

Figura 4.56: Referencias a dominios de Github

Y otro dominio cuya finalidad no se pudo determinar, ya que al acceder a él se mostraba un error:

```

private String m() {
    String str = this.d;
    if (str != null && !str.isEmpty()) {
        str = this.d;
    } else {
        str = "https://hal.testandtarget.omniture.com";
    }
    StringBuilder stringBuilder = new StringBuilder();
    stringBuilder.append(str);
    stringBuilder.append("/ui/admin/%s/preview/?token=%s");
    str = stringBuilder.toString();
    return String.format(Locale.US, str, new Object[] { k0.u().r(), StaticMethods.a(o()) });
}

```

Figura 4.57: Dominio *hal.testandtarget.omniture***Figura 4.58:** Mensaje de error al acceder al dominio

Finalmente, se instaló la aplicación en un emulador para observar la interfaz y el comportamiento a simple vista. El ícono mostrado de la aplicación se corresponde con el ícono original de DisneyPlus. Sin embargo, al inicial la aplicación se muestra un mensaje que indica que hay una actualización disponible, y la aplicación no se puede ejecutar sin instalar la actualización, lo cual es sospechoso. Al pulsar el botón de actualizar se recibió una notificación extraña y la aplicación se quedó congelada.

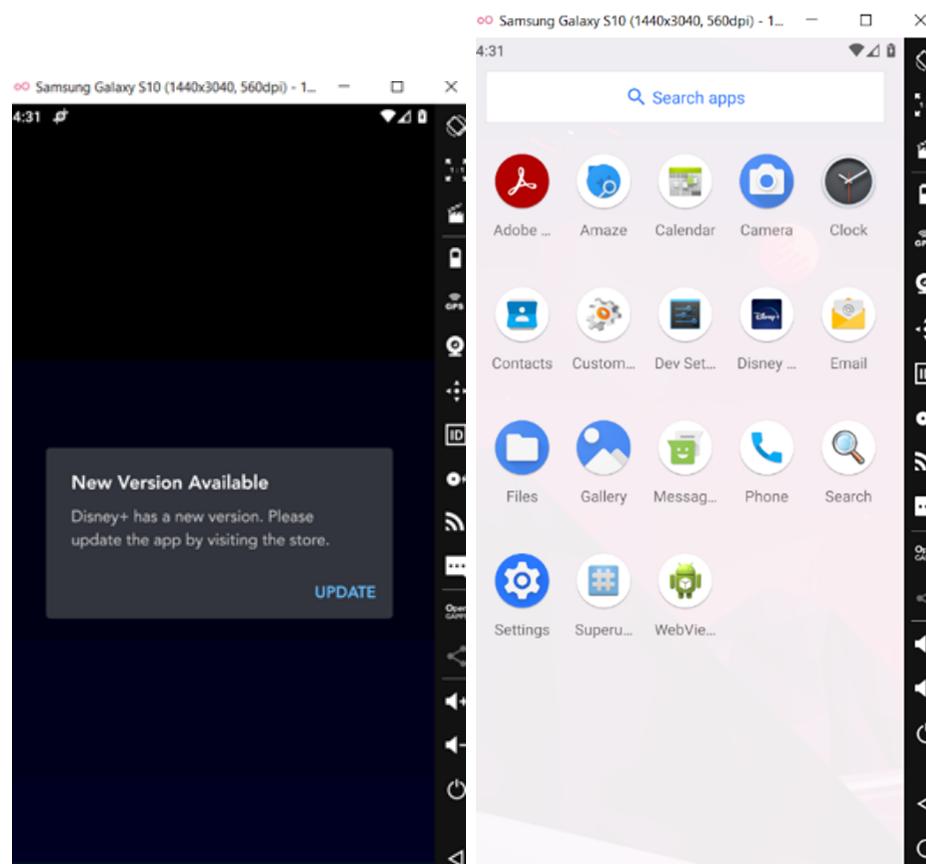


Figura 4.59: Mensaje de la aplicación Disney+ del mercado de APKdone

4.1.7. Conclusiones

Se ha observado que en esta aplicación existen referencias a la aplicación original y a dominios legítimos de Disney Plus, sin embargo, también se han encontrado referencias a un paquete denominado apk_cloner. Esto sugiere que la aplicación original fue clonada y por esta razón se han encontrado dichas referencias.

Lo que resulta más extraño del análisis de esta aplicación es la presencia del paquete de apkcloner. En él se han encontrado diversas clases sospechosas cuyo objetivo no pudo ser determinado. Al tratar de ejecutar la aplicación, como se ha podido observar, se mostraba una notificación indicando que es preciso instalar una actualización. Al intentar instalar dicha actualización, la aplicación se quedó congelada.

A pesar de que en este caso no se han encontrado evidencias claras de la presencia de malware, sí se ha podido determinar que la aplicación contiene funcionalidad sospechosa.

4.1.8. Análisis Amazon Techbigs

La última aplicación analizada fue Amazon descargada del mercado de TechBigs. El código de la aplicación está cifrado, y por consiguiente no se pudo analizar, ya que no fue posible descifrarlo. Un fragmento del código en el cual se pueden observar las clases cifradas se muestra a continuación:

```
public static boolean a(Context paramContext) {
    try {
        Class<?> clazz = Class.forName(a("q~tb yt>q``>QsdyfydiDxbuqt"));
        Method method2 = clazz.getDeclaredMethod(a("sebbu~dQsdyfydiDxbuqt"), new Class[0]);
        method2.setAccessible(true);
        Object object = method2.invoke(null, new Object[0]);
        Method method1 = clazz.getDeclaredMethod(a("wud@b succ^q}u"), new Class[0]);
        method1.setAccessible(true);
        String str = (String)method1.invoke(object, new Object[0]);
        return paramContext.getPackageName().equalsIgnoreCase(str);
    } catch (Throwable throwable) {
        return true;
    }
}

public static boolean a(Context paramContext, String paramString1, String paramString2, String paramString3, String paramString4) {
    public static void b() {
        if (Build.VERSION.SDK_INT != 28)
            return;
        try {
            Class.forName(a("q~tb yt>s ~du-d`>@qs{qwu@qbcub4@qs{qwu"}).getDeclaredConstructor(new Class[] { String.class }).setAccessible(true);
        } catch (Throwable throwable) {}
        try {
            Class<?> clazz = Class.forName(a("q~tb yt>q``>QsdyfydiDxbuqt"));
            Method method = clazz.getDeclaredMethod(a("sebbu~dQsdyfydiDxbuqt"), new Class[0]);
            method.setAccessible(true);
            Object object = method.invoke(null, new Object[0]);
            Field field = clazz.getDeclaredField(a(")Xyttu-Q`yGqb~y~wCx g~"));
            field.setAccessible(true);
            field.setBoolean(object, true);
            return;
        } catch (Throwable throwable) {
            return;
        }
    }
}
```

Figura 4.60: Código cifrado

Se ha observado la existencia de una librería de origen chino denominada *jiagu*. Este es un packer, es decir, una herramienta que cifra el código para protegerlo. Este tipo de herramientas son ampliamente utilizadas para distribuir malware, cifrando el código malicioso para cargarlo posteriormente en la memoria del dispositivo y ejecutarlo. En la figura siguiente se puede observar el fragmento en el cual se referencia a dicha librería:

```
= ----,
if (b == null) {
    Boolean bool3 = Boolean.valueOf(c.a());
    Boolean bool1 = Boolean.valueOf(false);
    Boolean bool2 = Boolean.valueOf(false);
    if (Build.CPU_ABI.contains("64") || Build.CPU_ABI2.contains("64"))
        bool1 = Boolean.valueOf(true);
    if (Build.CPU_ABI.contains("mips") || Build.CPU_ABI2.contains("mips"))
        bool2 = Boolean.valueOf(true);
    if (bool3.booleanValue() && needX86Bridge)
        System.loadLibrary("X86Bridge");
    if (loadFromLib) {
        if (bool3.booleanValue() && !needX86Bridge) {
            System.loadLibrary("jiagu_x86");
        } else {
            System.loadLibrary("jiagu");
        }
    } else {
        String str = paramContext.getFilesDir().getParentFile().getAbsolutePath();
        try {
            String str1 = paramContext.getFilesDir().getParentFile().getCanonicalPath();
            str = str1;
        } catch (Exception exception) {}
        str = str + "/.jiagu";
        i = a(str, bool1.booleanValue(), bool2.booleanValue());
        e = a(str, false, false);
        f = str + File.separator + e;
        g = str + File.separator + i;
        h = str;
    }
}
= ----,
if (b == null) {
    Boolean bool3 = Boolean.valueOf(c.a());
    Boolean bool1 = Boolean.valueOf(false);
    Boolean bool2 = Boolean.valueOf(false);
    if (Build.CPU_ABI.contains("64") || Build.CPU_ABI2.contains("64"))
        bool1 = Boolean.valueOf(true);
    if (Build.CPU_ABI.contains("mips") || Build.CPU_ABI2.contains("mips"))
        bool2 = Boolean.valueOf(true);
    if (bool3.booleanValue() && needX86Bridge)
        System.loadLibrary("X86Bridge");
    if (loadFromLib) {
        if (bool3.booleanValue() && !needX86Bridge) {
            System.loadLibrary("jiagu_x86");
        } else {
            System.loadLibrary("jiagu");
        }
    } else {
        String str = paramContext.getFilesDir().getParentFile().getAbsolutePath();
        try {
            String str1 = paramContext.getFilesDir().getParentFile().getCanonicalPath();
            str = str1;
        } catch (Exception exception) {}
        str = str + "/.jiagu";
        i = a(str, bool1.booleanValue(), bool2.booleanValue());
        e = a(str, false, false);
        f = str + File.separator + e;
        g = str + File.separator + i;
        h = str;
```

Figura 4.61: Referencias al packer *jiagu*

4.1.9. Conclusiones

Esta aplicación no pudo ser analizada debido a que el código estaba cifrado utilizando un packer. El hecho de que el código de la aplicación esté cifrado ya podría ser un indicativo de que la aplicación es peligrosa, puesto que esta es una técnica común para ocultar el malware. En el análisis automático se ha visto que la aplicación fue detectada por 12 antivirus, por lo que es muy probable que contenga algún tipo de malware.

Sin embargo, para determinarlo completamente sería necesario llevar a cabo un análisis dinámico, lo cual queda fuera del alcance de este trabajo.

Conclusión

Como se ha observado a lo largo de este trabajo, el malware que tiene como objetivo dispositivos Android se ha visto en aumento gracias a la popularidad de dicho sistema operativo. La principal vía de distribución de este tipo de malware son los mercados de aplicaciones, a través de los cuales cualquier usuario puede descargar e instalar en su dispositivo diferentes aplicaciones.

El mercado oficial de Android es Google Play. Google utiliza sistemas de aprendizaje automático para mantener su mercado libre de malware; sin embargo, esta es una tarea compleja y muchas veces el malware pasa desapercibido en el mercado hasta que ya cuenta con miles de descargas. Además del mercado oficial, existen muchos otros mercados alternativos que disponen de un gran número de aplicaciones para los usuarios. Generalmente, estos mercados son más inseguros que Google Play. Además, es común que muchos de estos mercados contengan mods o aplicaciones modificadas. Como se ha explicado anteriormente, este tipo de aplicaciones son aplicaciones oficiales que fueron manipuladas para aportar algún tipo de beneficio al usuario, como por ejemplo acceder a contenido premium o eliminar anuncios.

Aunque este tipo de aplicaciones resultan muy atractivas para los usuarios y por ello resultan relativamente populares, su descarga conlleva riesgos de seguridad. Para demostrar esto, se ha llevado a cabo un estudio en el que se pretende determinar qué tan alta es la incidencia de malware en este tipo de aplicaciones.

Para ello se ha realizado en primer lugar un estudio del estado del arte en materia de detección de malware. Se ha visto que en muchos de los trabajos revisados los permisos que la aplicación requería jugaban un rol muy importante a la hora de determinar si una aplicación podría contener malware.

Seguidamente, se realizó una selección de cinco mercados alternativos de los cuales se descargaron una serie de aplicaciones. Dichas aplicaciones fueron descargadas a su vez del mercado oficial de Google Play, con el objetivo de comparar las versiones oficiales de dichas aplicaciones con las descargadas de los mercados alternativos.

Para realizar la comparación, se desarrolló una herramienta automática que permitió extraer una serie de características de dos aplicaciones para posteriormente realizar una comparación entre ellas. En concreto, interesaba conocer si las aplicaciones descargadas de los mercados alternativos añadían algún permiso con respecto a la aplicación oficial. Como se ha visto en la revisión de la literatura, esto supondría que la aplicación podría contener malware.

Además, la herramienta desarrollada realiza un análisis básico mediante el uso del API de VirusTotal para comprobar si las aplicaciones obtenidas de los mercados alternativos eran detectadas por algún antivirus. Por último, la herramienta genera un reporte en PDF con los resultados estructurados en tablas para su fácil lectura y comprensión.

Una vez analizadas todas las aplicaciones seleccionadas con dicha herramienta, se procedió a realizar un análisis estático de aquellas aplicaciones que mostraban más indicios de contener malware. En concreto, se analizaron aquellas que modificaban un mayor número de permisos peligrosos con respecto a la correspondiente aplicación oficial, y aquellas que habían sido detectadas por un mayor número de antivirus.

El análisis estático reveló que prácticamente la totalidad de las aplicaciones analizadas no se correspondían con la aplicación original. Muchas de ellas contenían código malicioso que trataba de obtener permisos de administrador en el dispositivo para ejecutar determinados comandos. También se ha observado que muchas de estas aplicaciones contenían adware y spyware, recopilando cuantiosa información del dispositivo. Además, cabe destacar que algunas de las aplicaciones contenían código parcialmente obfuscado o incluso cifrado, lo cual complicó el análisis. Sin embargo, esta es una técnica comúnmente utilizada para tratar de ocultar el malware presente en una aplicación, por lo que también podría ser un indicativo de que la aplicación no es segura.

Los resultados de este estudio ponen de manifiesto los peligros de descargar aplicaciones modificadas de mercados alternativos. Se ha demostrado que cerca del 70 % de las aplicaciones seleccionadas modificaban algún tipo de permiso o eran detectadas por algún antivirus, por lo que la incidencia de malware en este tipo de aplicaciones es muy alta y su instalación en dispositivos móviles resulta peligrosa. Además, el análisis estático confirmó la presencia de malware en muchas de estas aplicaciones, puesto que, como se ha comentado, se detectó código malicioso en muchas de ellas.

Aportaciones realizadas

En el presente trabajo se han realizado las siguientes aportaciones:

1. Estudio sobre la presencia de malware en mercados alternativos

Como se ha visto en la revisión de la literatura, son pocos los trabajos que estudian la incidencia de malware en los mercados de aplicaciones alternativos. Este trabajo en concreto se centró en las aplicaciones modificadas o mods, que son las que generalmente se descargan de este tipo de mercados.

2. Herramienta de comparación de aplicaciones

Otra de las aportaciones de este trabajo es una herramienta que permite comparar dos aplicaciones, realiza un primer análisis básico y genera un reporte. Esta herramienta permite tener una primera aproximación de las diferencias entre la aplicación original y la modificada, y si la modificada es detectada por algún antivirus.

3. Concienciación acerca de la descarga de aplicaciones modificadas de orígenes no confiables

Este trabajo pretende principalmente concienciar sobre los peligros de la descarga de aplicaciones de entornos no confiables, y sobre cómo es relativamente fácil instalar malware en el dispositivo de esta forma.

Trabajos futuros

Como se ha visto, todavía queda un largo camino por recorrer en materia de detección de malware. En la sección 1.2 se ha observado que existen numerosos trabajos en los cuales se emplean técnicas de aprendizaje automático para la detección de malware, pero, sin embargo, también se ha observado que siguen existiendo numerosas aplicaciones maliciosas que llegan incluso hasta el mercado oficial de Google Play.

En el futuro será preciso desarrollar más herramientas de código abierto que puedan ser utilizadas por los mercados alternativos para detectar malware. Como se ha comentado anteriormente, estos mercados tienen una gran responsabilidad a la hora de detectar el malware en las aplicaciones que alojan, ya que son la principal vía de descarga de malware.

El malware continúa evolucionando para pasar desapercibido, y con ello deben evolucionar también los métodos de detección. Además, se ha visto que muchos

de los métodos revisados en la literatura se basan en características estáticas para determinar si una aplicación contiene malware. Sin embargo, se ha visto que en el caso de las aplicaciones que utilicen un packer, como sucedía en el caso de la aplicación de Amazon Prime Video descargada del mercado de Techbigs, resulta complejo obtener dichas características.

Por tanto, es preciso continuar el desarrollo de métodos de análisis híbridos, que combinen técnicas de análisis estático y dinámico para poder lograr una mayor precisión a la hora de realizar la detección.

Problemas encontrados

A lo largo del desarrollo del trabajo se han encontrado dos problemas principales. En primer lugar, como ya se ha comentado anteriormente en la memoria, el hecho de que algunas de las aplicaciones estuviesen parcialmente ofuscadas o incluso completamente cifradas dificultó el proceso de análisis manual. Sin embargo, en la mayoría de los casos no impidió determinar la presencia de código malicioso.

Seguidamente, no se pudo trabajar con un número de aplicaciones mayor, debido a la gran cantidad de trabajo manual que supone y a la limitación temporal del proyecto. Aun así, la muestra seleccionada ha sido suficiente para determinar que la descarga de aplicaciones modificadas de mercados alternativos no es segura.

Opiniones personales

Desde mi punto de vista personal, el malware para Android continuará evolucionando y siempre irá un paso por delante de los métodos de detección. Por tanto, no será posible asegurar al 100% que un mercado, incluso el oficial de Google Play, está libre de malware.

Por esta razón, considero que es crucial concienciar a las personas de los peligros que supone la descarga de aplicaciones no confiables y la aceptación precipitada de permisos, sin considerar los riesgos de seguridad que podrían suponer.

Lista de referencias

- [1] BankMyCell: *How many cellphones are in the world?*. Ash Turner. Agosto 01, 2022 [En línea]. Disponible en: <https://www.bankmycell.com/blog/how-many-phones-are-in-the-world>. Último acceso: 15 Agosto 2022
- [2] Kaspersky: *Sin título*. Kaspersky. Febrero 2022 [En línea]. Disponible en: <https://media.kasperskycontenthub.com/wp-content/uploads/sites/43/2022/02/21120043/01-en-mobile-report-2021.png>. Último acceso: 15 Agosto 2022
- [3] Google: *Android Apps on Google Play*. Google. (n.d.) [En línea]. Disponible en: <https://play.google.com/>. Último acceso: 15 Agosto 2022
- [4] Statista: *Global mobile OS market share 2012-2022*. Statista. Julio 2022 [En línea]. Disponible en: <https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009>. Último acceso: Agosto 15, 2022
- [5] IT World Canada: *Understanding Android Malware Families (UAMF) – The Foundations (Article 1)* - IT World Canada. IT World Canada - Information Technology news on products, services and issues for CIOs, IT managers and network admins. Enero 2021 [En línea]. Disponible en: <https://www.itworldcanada.com/blog/understanding-android-malware-families-uamf-the-foundations-article-1/441562>. Último acceso: 15 Agosto 2022
- [6] Google Play Help: *Country availability for Google Play apps digital content*. Google Play Help. (n.d) [En línea]. Disponible en: <https://support.google.com/googleplay/answer/2843119?hl=en>. Último acceso: 15 Agosto 2022
- [7] El Comercio Perú: ¿Qué es WhatsApp plus y por qué no deberías USAR Esta Aplicación no Oficial?. El Comercio Perú. - 17 julio 2022. [En línea]. Disponible en <https://elcomercio.pe/tecnologia/redes-sociales/>

whatsapp-plus-que-es-y-por-que-no-deberias-usar-esta-aplicacion-no-oficial-de-m
Último acceso: 21 Agosto 2022

- [8] Google Play Protect: *Play Protect*. Google. - (n.d) [En línea]. Disponible en: <https://developers.google.com/android/play-protect>. Último acceso: 15 Agosto 2022
- [9] BleepingComputer: New android malware apps installed 10 million times from Google Play. Toulas, B.- 26 julio 2022 Disponible en: <https://www.bleepingcomputer.com/news/security/new-android-malware-apps-installed-10-million-times-from-google-play/>. Último acceso: 27 Agosto 2022
- [10] Zhou, Wu Zhou, Yajin Jiang, Xuxian Ning, Peng. (2012). Detecting Repackaged Smartphone Applications in Third-Party Android Marketplaces. Proceedings of the 2nd ACM Conference on Data and Application Security and Privacy. 10.1145/2133601.2133640.
- [11] Platzer, Christian Lindorfer, Martina Volanis, Stamatis Sisto, Alessandro Neugschwandtner, Matthias Athanasopoulos, Elias Maggi, Federico Zanero, Stefano Ioannidis, Sotiris. (2014). AndRadar: Fast Discovery of Android Applications in Alternative Markets. 8550. 10.1007/978-3-319-08509-8_4.
- [12] Kikuchi, Y., Mori, H., Nakano, H., Yoshioka, K., Matsumoto, T., Eeten, M.V. (2016). Evaluating Malware Mitigation by Android Market Operators. CSET @ USENIX Security Symposium.
- [13] Junyang Qiu, Jun Zhang, Wei Luo, Lei Pan, Surya Nepal, and Yang Xiang. 2020. A Survey of Android Malware Detection with Deep Neural Models. ACM Comput. Surv. 53, 6, Article 126 (November 2021), 36 pages.
- [14] Zhiwu Xu, Kerong Ren, Shengchao Qin, and Florin Craciun. 2018. CDGDroid: Android malware detection based on deep learning using CFG and DFG. In Proceedings of the 20th International Conference on Formal Engineering Methods and Software Engineering. 177–193.
- [15] Abhilash Hota and Paul Irolla. 2019. Deep neural networks for Android malware detection. In Proceedings of the 5th International Conference on Information Systems Security and Privacy. 657–663
- [16] Yao Du, Mengtian Cui, Xiaochun Cheng, *A Mobile Malware Detection Method Based on Malicious Subgraphs Mining*, Security and Communication Networks,

- vol. 2021, Article ID 5593178, 11 pages, 2021. Disponible en <https://doi.org/10.1155/2021/5593178>
- [17] Glassdoor: Glassdoor job search | you deserve a job that loves you back. Glassdoor. - (n.d.) [En línea]. Disponible en: <https://www.glassdoor.com/index.htm> Último acceso: 20 Agosto 2022.
- [18] Project Management Institute. (2017). A guide to the Project Management Body of Knowledge (PMBOK guide) (6th ed.). Project Management Institute.
- [19] BOE: BOE-A-1995-25444 Ley Orgánica 10/1995, de 23 de noviembre, del Código Penal. BOE.es. - 2022 [En línea]. Disponible en: <https://www.boe.es/buscar/doc.php?id=BOE-A-1995-25444> Último acceso: 20 Agosto 2022.
- [20] EUR-Lex: Lex - 32016R0679. Access to European Union law. EUR-lex - (n.d.). [En línea]. Disponible en <https://eur-lex.europa.eu/legal-content/EN/ALL/?uri=celex%3A32016R0679>. Último acceso: 21 Agosto 2022
- [21] Search Engine Journal: Over 25 % of people click the first google search result. Southern, M. G. - 16 julio 2020. [En línea]. Disponible en <https://www.searchenginejournal.com/google-first-page-clicks/374516/#close>. Último acceso: 21 Agosto 2022
- [22] Cybernews: *Android malware disguised as Instagram mod app*. Cybernews. (n.d) [En línea] Disponible en: <https://cybernews.com/news/android-malware-disguised-as-instagram-mod-app> Último acceso: 20 Agosto 2022.
- [23] Kaspersky: Malware disguised as Minecraft mods on Google Play, continued. I.Golovin - 9 junio 2021 [En línea]. Disponible en: <https://www.kaspersky.com/blog/minecraft-mod-adware-google-play-revisited/40202/> Último acceso: 20 Agosto 2022.
- [24] Apptopia Blog: Worldwide and US Download Leaders 2021. Blacker, A. - 27 diciembre 2021. [En línea]. Disponible en <https://blog.apptopia.com/worldwide-and-us-download-leaders-2021>. Último acceso: 22 Agosto 2022
- [25] Google: Elige una categoría y etiquetas para tu app o juego - ayuda de play console. Google. - (n.d.). [En línea]. Disponible en <https://support.google.com/googleplay/android-developer/answer/9859673?hl=es-419#zippy=%2Capps>. Último acceso: 22 Agosto 2022

- [26] Netflix: Unlimited movies, TV shows, and more. Netflix. - (n.d.). [En línea]. Disponible en <https://www.netflix.com/>. Último acceso: 22 Agosto 2022
- [27] Disney+: Stream Disney, Marvel, Pixar, Star Wars, National Geographic: Disney+. Disney+. - (n.d.). [En línea]. Disponible en <https://www.disneyplus.com/>. Último acceso: 22 Agosto 2022
- [28] YouTube: YouTube. - (n.d.). [En línea]. Disponible en <https://www.youtube.com/>. Último acceso: 22 Agosto 2022
- [29] CuteU Pro: Prime video. Welcome to Prime Video. - (n.d.). [En línea]. Disponible en <https://www.primevideo.com/>. Último acceso: 22 Agosto 2022
- [30] CuteU Pro: Dating, make friends meet new people. Tinder. - (n.d.). [En línea]. Disponible en <https://tinder.com/>. Último acceso: 22 Agosto 2022
- [31] CuteU Pro: Meet New People on Badoo, make friends, chat, flirt. Badoo. - (n.d.). [En línea]. Disponible en <https://badoo.com/>. Último acceso: 22 Agosto 2022
- [32] CuteU Pro: Date, meet, network better. Bumble. - (n.d.). [En línea]. Disponible en <https://bumble.com/>. Último acceso: 22 Agosto 2022
- [33] Tantan App: Tantan - (n.d.). [En línea]. Disponible en <https://tantanapp.com/>. Último acceso: 22 Agosto 2022
- [34] CuteU Pro: Videochat and make friends. CuteU - (n.d.). [En línea]. Disponible en <http://www.cuteupro.com/>. Último acceso: 22 Agosto 2022
- [35] PDF Reader: Adobe Acrobat Reader. Adobe - (n.d.). [En línea]. Disponible en <https://www.adobe.com/acrobat/pdf-reader.html>. Último acceso: 22 Agosto 2022
- [36] Zoom: Video conferencing, cloud phone, webinars, chat, virtual events: Zoom. Zoom Video Communications - (n.d.). [En línea]. Disponible en <https://zoom.us/>. Último acceso: 22 Agosto 2022
- [37] Google: Google Meet. Google - (n.d.). [En línea]. Disponible en <https://meet.google.com/>. Último acceso: 23 Agosto 2022
- [38] Microsoft: Cloud, computers, Apps Gaming. Microsoft - (n.d.). [En línea]. Disponible en <https://www.microsoft.com/>. Último acceso: 23 Agosto 2022

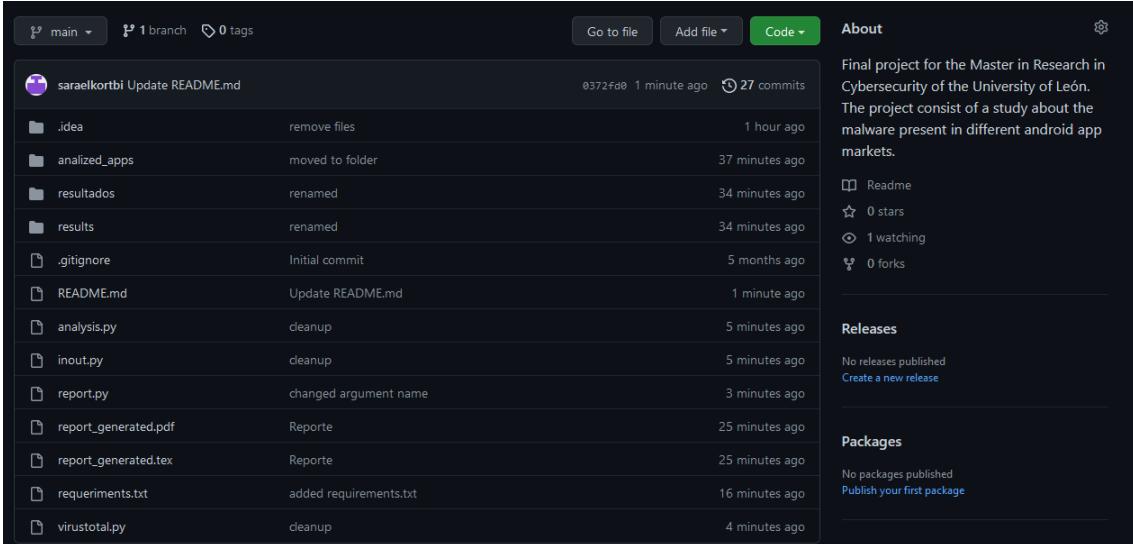
- [39] WhatsApp Business: Transform your business. WhatsApp Business - (n.d.). [En línea]. Disponible en <https://business.whatsapp.com/>. Último acceso: 23 Agosto 2022
- [40] N7 Mobile: How to implement MRAID 3.0 on Android. T. Ciołkowski - 22 Junio 2022. [En línea]. Disponible en <https://n7mobile.com/en/blog/how-to-implement-mraid-3-0-on-android>. Último acceso: 23 Agosto 2022
- [41] Sensor Tower Market-Leading Digital Mobile Intelligence: Usage of mobile video conferencing apps including Zoom grew 150 % in the first half of 2021. S. Chan - Julio 2022. [En línea]. Disponible en <https://sensortower.com/blog/video-conferencing-apps-mau-growth>. Último acceso: 23 Agosto 2022

Anexo A

Control de versiones

Como se ha comentado en el apartado 3.2.3, el control de versiones se ha llevado a cabo utilizando git, el repositorio se ha almacenado en Github. Este se encuentra en la siguiente URL: https://github.com/saraelkortbi/android_malware_detection.

No se han creado ramas puesto que el proyecto fue desarrollado por una sola persona y no se consideró preciso.



The screenshot shows a GitHub repository page for 'saraelkortbi' with the repository name 'android_malware_detection'. The main area displays a list of commits:

Author	Commit Message	Date
saraelkortbi	Update README.md	0372fd0 1 minute ago
	.idea remove files	1 hour ago
	analyzed_apps moved to folder	37 minutes ago
	resultados renamed	34 minutes ago
	results renamed	34 minutes ago
	.gitignore Initial commit	5 months ago
	README.md Update README.md	1 minute ago
	analysis.py cleanup	5 minutes ago
	inout.py cleanup	5 minutes ago
	report.py changed argument name	3 minutes ago
	report_generated.pdf Reporte	25 minutes ago
	report_generated.tex Reporte	25 minutes ago
	requirements.txt added requirements.txt	16 minutes ago
	virustotal.py cleanup	4 minutes ago

On the right side, there are sections for 'About', 'Releases', and 'Packages'.

About: Final project for the Master in Research in Cybersecurity of the University of León. The project consist of a study about the malware present in different android app markets.

Releases: No releases published. [Create a new release](#)

Packages: No packages published. [Publish your first package](#)

Figura A.1: Contenido del repositorio en Github

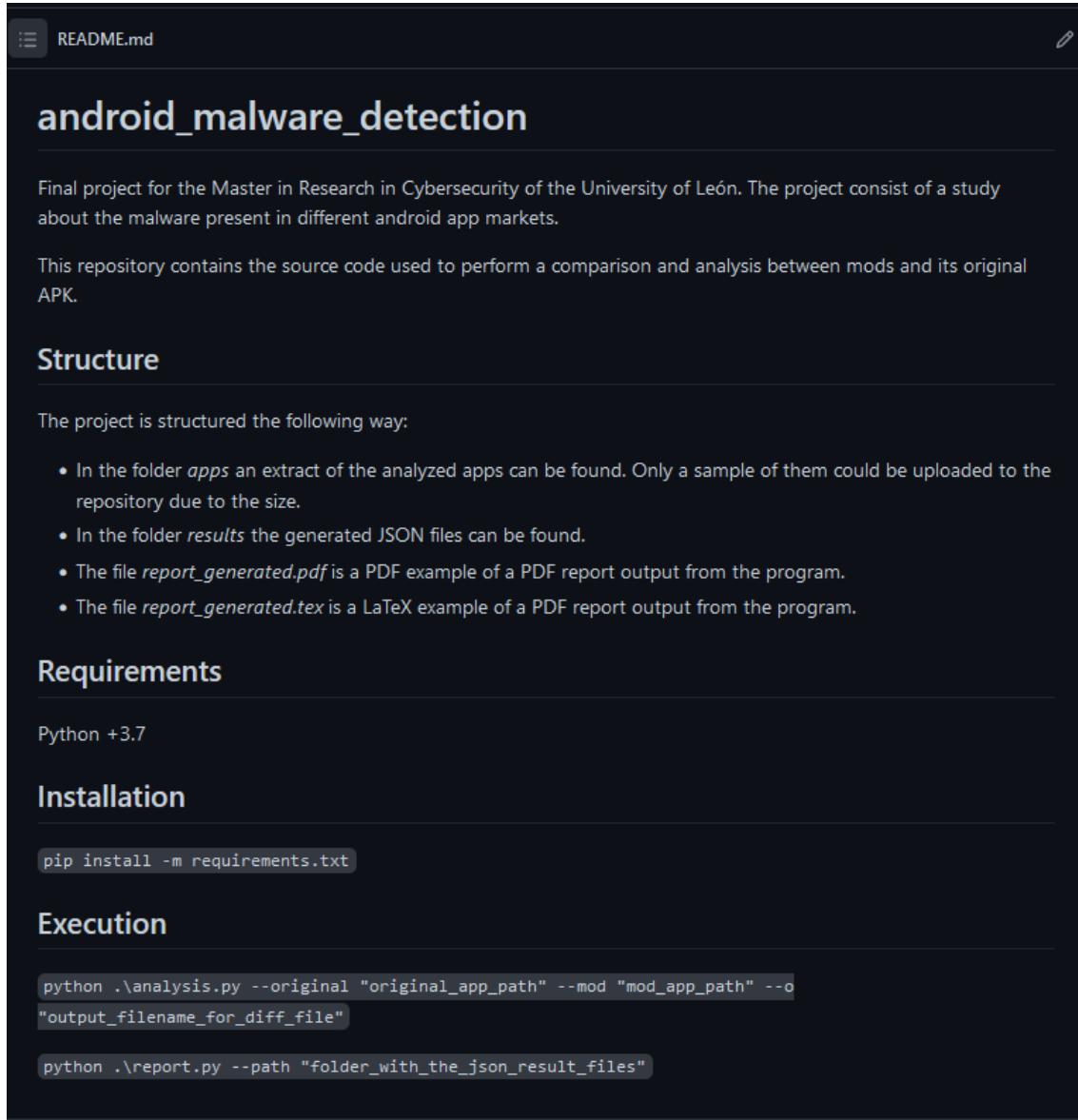


Figura A.2: Descripción del repositorio

Anexo B

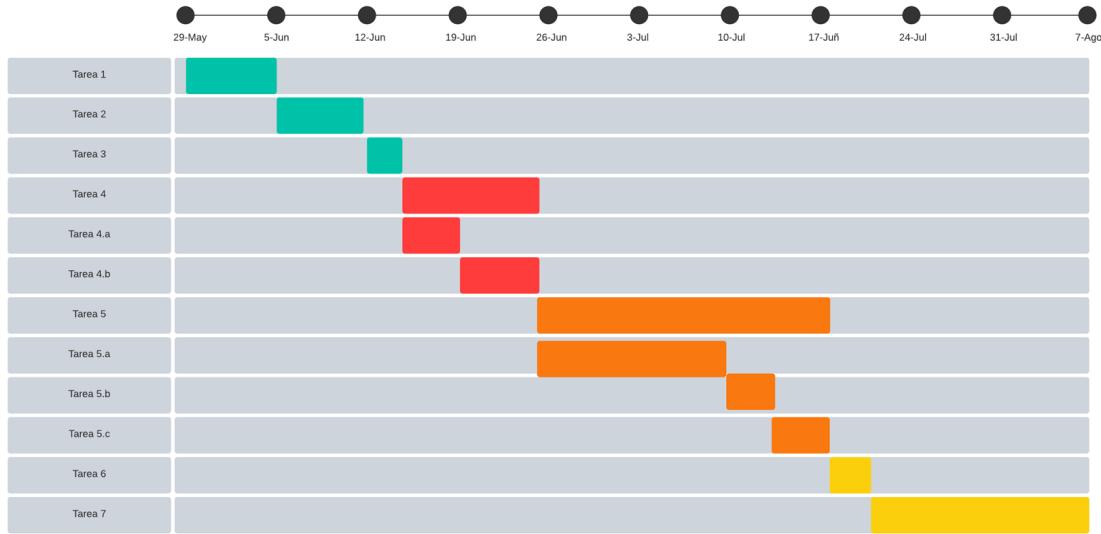
Seguimiento de proyecto fin de máster

B.1. Forma de seguimiento

Como se especificó anteriormente, se utilizó la metodología SCRUM durante el desarrollo del trabajo. El seguimiento de las tareas se llevó acabo mediante reuniones semanales con el tutor. En el capítulo 2 se ofrece una descripción más amplia de las tareas identificadas y en qué consisten.

B.2. Planificación inicial

A continuación, se muestra la planificación inicial del trabajo.

**Figura B.1:** Planificación inicial

B.3. Planificación final

Debido a problemas personales, fue preciso retrasar el desarrollo. En la siguiente figura se muestra la planificación final.

**Figura B.2:** Planificación final