Institución Universitaria

Detección conjunta de *malware* entre usuarios y dispositivos a partir de la validación de firmas digitales y/o la correlación de eventos en dispositivos *Android*

Juan Fernando Grajales Puerta

Instituto Tecnológico Metropolitano
Facultad de Ingenierías
Medellín, Colombia
2018

Detección conjunta de *malware* entre usuarios y dispositivos a partir de la validación de firmas digitales y/o la correlación de eventos en dispositivos *Android*

Juan Fernando Grajales Puerta

Tesis o trabajo de investigación presentada(o) como requisito parcial para optar al título de:
Magister en Seguridad Informática

Director:
Magister en Seguridad de las tecnologías de la Información y las Comunicaciones, Fernando Alonso Quintero Londoño

Director:
Magister en Seguridad Informática, Miguel Mariano Manosalva Pinedo

Línea de Investigación:
Seguridad de la información
Grupo de Investigación:
Automática, electrónica y ciencias computacionales

Instituto Tecnológico Metropolitano
Facultad de Ingenierías
Medellín, Colombia
2018

Dedicatoria

A mi familia, aquellas personan que con su confianza y apoyo me han ayudado a realizar el logro y satisfacción obtenidos en este trabajo y en el título profesional.

Agradecimientos

A mi familia y seres amados que siempre me apoyado en toda mi vida

A los Magister Fernando Alonso Quintero Londoño y Miguel Mariano Manosalva Pinedo, quienes como directores de esta tesis de Maestría pusieron sus conocimientos y experiencia en seguridad informática y metodología al servicio de este trabajo

Al Magister Héctor Fernando Vargas Montoya, quien como coordinador de la maestría puso todo su conocimiento y disponibilidad para ayudarme en todos los trámites correspondientes

A los docentes de la Maestría en Seguridad Informática del ITM, por la experiencia y conocimientos transmitidos, así como por su compromiso y esfuerzo en formar magister de calidad

A la Institución Universitaria ITM, por brindarme la posibilidad de formarme como profesional y poder optar por el título de Magister en seguridad informática

Resumen

Para la detección de *software* malicioso que compromete aplicaciones en teléfonos inteligentes con sistema operativo *Android*, los controles convencionales, utilizados entre el año 2012 y hasta el primer semestre del año 2018, requieren de una muestra de *malware* para realizar la detección. Estos controles de seguridad ejecutan el análisis de aplicaciones en la nube y no localmente en el dispositivo. La mayoría de los controles se limitan a las aplicaciones ofrecidas en la tienda de *Google* (*Play Store*) y, para que la neutralización sea efectiva, la mayoría de ellos requiere de habilidades especiales que no todo usuario final de *Android* posee. En este proyecto se hizo un análisis de estas técnicas, se compararon sus formas de detección y se registraron sus falencias. Con la información obtenida, se diseñó e implementó una aplicación para sistemas operativos *Android* en dispositivos móviles llamada *CAM (Control de Aplicaciones Móviles)*, para asegurar la integridad de las aplicaciones y revisar si han sido intervenidas con *malware*, por medio de la validación de firmas digitales y la correlación de eventos. *CAM* propone una estrategia de corresponsabilidad entre los desarrolladores de aplicaciones para móviles y la comunidad de usuarios del sistema operativo, basada en defensa activa para que la seguridad se convierta en un atributo del sistema y no complemento. La estrategia de corresponsabilidad busca que los desarrolladores y usuarios publiquen bases de datos de listas blancas de los principales eventos operativos de sus aplicaciones, para contrastarlas con la información que generen dichos eventos, con el fin de detectar y mitigar amenazas cibernéticas como el espionaje, la fuga de información, la suplantación de identidad, el robo de contraseñas y el control remoto del dispositivo por medio de troyanos (*bots*). También pretende brindar educación en materia de ciberseguridad a los usuarios, apoyándose en la entrega de alertas eficientes.

Para el desarrollo de este trabajo de grado, se utilizaron los registros estadísticos de dispositivos móviles con sistema operativo *Android* más usado entre el 2015 y el 2018 y se realizó un laboratorio de máquinas virtuales para simular dichas distribuciones de *Android*, se examinaron sus principales características y eventos operativos tales como: permisos, firmas y tráfico, se intervino las aplicaciones seleccionadas con el paquete de *Meterpreter* para *Android* del *framework Metasploit*. Para la detección de los indicadores de compromiso en las aplicaciones infectadas se usaron aplicaciones como: *Package Info*, *RL Permissions* y *Network Connections* entre otras. Estos resultados hicieron posible el desarrollo de la plataforma *CAM* para *Android* con arquitectura cliente-servidor. La plataforma *CAM* se encarga de almacenar y correlacionar los eventos operativos validos de las aplicaciones legítimas, en una lista blanca y posteriormente brindar al usuario un informe eficiente que le permita evitar e identificar cuando una aplicación móvil genera una ciber-amenaza en un teléfono inteligente.

Palabras claves: *Malware*, Firmas digitales, Correlación de eventos, Ciberseguridad, Defensa activa, Eventos operativos, Indicadores de compromiso, *Metasploit framework*

Abstract

For malicious *software* detection that compromises applications in *smartphones* with *Android* Operating *System*, conventional controls, used between 2012 the first half of 2018, require a sample of *malware* to perform the detection. Most *security* controls run the applications analysis in the cloud, and not locally on the *device*. Other controls are limited to the applications offered in the *Google Play Store*. In addition, for the neutralization to be effective, most controls require special abilities most end user of *Android* doesn't have. In this project, an analysis of these techniques is made, their forms of detection are compared, and their shortcomings are recorded. With the information obtained from these analyses, an application for *Android* operating systems is designed and implemented on mobile devices: *CAM (Control for Mobile Applications). T*o ensure the applications integrity, it is checked if they have been intervened with *malware*, through the digital signatures' validation and events' correlation. *CAM* proposes a strategy of co-responsibility between the mobile application developers and the operating *system* community users, based on active defense, so that *security* becomes systems attribute instead of just being a complementary service. The co-responsibility strategy aims for developers and users publishing their application main operational events white-list databases, to contrast them with the information generated by those applications' events. That way, the collected information may improve detection and mitigation of cyber threats such as espionage, information leakage, identity theft, password stealing, and remote *device* control through trojans (*bots*). The co-responsibility strategy also aims to provide education on cyber *security* to users, based on the delivery of efficient alerts.

For the development of this degree work, statistical records of mobile devices with the most used *Android* operating *system* between 2015 and 2018, and a laboratory of virtual machines, were made to simulate said distributions of *Android*, its main features and operational events such as: permits, signatures, and traffic were examined. The selected applications were intervened with the *Meterpreter* for *Android Package* of the *Metasploit framework*. For the commitment indicators detection in the infected applications, applications such as *Package Info*, *RL Permissions* and *Network Connections,* among others were used. The results of these experiments made possible the development of the *CAM* platform for *Android* with client-server architecture development. The *CAM* platform is responsible for storing and correlating the legitimate applications valid operational events in a white list. This white list is used to provide efficient reports to users, so they are able to identify and avoid when a mobile application generates a cyber-threat on a *smartphone*.

Keywords: *Malware*, Digital Signatures, Correlation Events, Cyber *Security*, Active Defense, Operative events, Compromise Indicators, *Metasploit Framework*.

Contenido

Lista de figuras

Lista de Tablas

1 Introducción

En la actualidad, el ser humano ha automatizado y sistematizado muchos procesos. Para esto ha diseñado dispositivos móviles inteligentes que le permiten vivir conectado en todo momento, generando la posibilidad de tener acceso a información de forma casi ilimitada. Esto se ha logrado gracias a dispositivos que son usados día a día, conocidos como teléfonos inteligentes *o smartphones.* Estos dispositivos son cada día más potentes y robustos en la capacidad de almacenar diferentes tipos de información. Actualmente, en la industria de los dispositivos móviles se ha desarrollado gran variedad de aplicaciones que ofrecen todo tipo de servicios. Sin embargo, de manera paralela también se ha abierto la posibilidad a personas mal intencionadas de desarrollar aplicaciones maliciosas que permiten tener acceso a la información confidencial que se maneja en los teléfonos inteligentes de los propietarios de dichos dispositivos, como su ubicación en tiempo real, acceso a los dispositivos de captura de imágenes o intervención del micrófono.

La conectividad de todos los dispositivos móviles y plataformas que conforman el ciberespacio ha maximizado los riesgos para la población con pocos conocimientos en temas de ciberseguridad. Por este motivo, el objetivo de este trabajo es brindar a los usuarios una herramienta que permita generar conocimiento de ciberseguridad. La herramienta diseñada permite detectar, informar y educar al usuario acerca de los eventos operativos de las aplicaciones legitimas de la industria, contenidas en una lista blanca con las aplicaciones instaladas en los dispositivos de los usuarios que ejecuten el sistema operativo *Android* de *Google Inc*. La herramienta diseñada permite a los usuarios avanzados y a los administradores de la base de datos definir una lista blanca o firma digital de los eventos operativos de las aplicaciones usadas por los usuarios propietarios de los dispositivos. Esta lista se correlaciona con las aplicaciones instaladas en los teléfonos inteligentes, con el fin de detectar e informar de manera amigable los indicadores de compromiso maliciosos de las aplicaciones suplantadas que fueron instaladas por el usuario final. Así, el usuario final podrá contener, mitigar, reportar las detecciones en formato de reglas, permitiendo compartir con otras plataformas que hagan detecciones a partir de este tipo de reglas generadas directamente desde el dispositivo móvil y remediar la intervención de su dispositivo. La plataforma diseñada, llamada *Control de Aplicaciones Móviles (CAM),* está limitada a detectar eventos operativos maliciosos en las aplicaciones configuradas en la lista blanca del administrador. En trabajos futuros se podrán agregar módulos con funciones como *antimalware* que permitan escanear indicadores de compromiso en las aplicaciones no configuradas en la lista blanca y hacer detecciones con las reglas *YARA* para generarlas a partir del análisis realizado en los mismos dispositivos móviles, además de módulos como *firewalls* que permitan bloquear las conexiones a las direcciones *IP* no detectadas a las pertenecientes a los desarrolladores de las aplicaciones.

1.1 Hipótesis

Los controles convencionales para dispositivos móviles, tales como *antimalware* y *firewalls* entre otros, son insuficientes para determinar cuándo un *Smartphone* está comprometido o presenta comportamientos anormales.

Se puede detectar *malware* en teléfonos inteligentes con sistema operativo *Android* a través de la verificación de firmas digitales y la correlación de eventos.

1.2 Objetivos

1.2.1 Objetivo General

Diseñar una herramienta de detección de *malware* mediante la interacción entre usuarios y aplicaciones a partir de la validación de firmas digitales y correlación de eventos en las principales aplicaciones de los sistemas operativos *Android* para teléfonos inteligentes.

1.2.2 Objetivos específicos

- Identificar técnicas y métodos de detección de *malware* en sistemas operativos *Android* para teléfonos inteligentes.
- Analizar y seleccionar herramientas para monitorear la actividad de las aplicaciones que se ejecutan en los sistemas operativos *Android* para teléfonos inteligentes.
- Elaborar un agente que ayude a los usuarios a detectar los comportamientos inusuales de las principales aplicaciones de los sistemas operativos *Android* para teléfonos inteligentes para que se familiaricen con la importancia de la seguridad de la informática.

2 Marco teórico y estado del arte

2.1 Antecedentes

La detección de *malware* en el sistema operativo *Android* es un campo bastante estudiado, ya que este es el sistema operativo más usado en los dispositivos móviles principalmente en los teléfonos inteligentes. A pesar de que este cuenta con un modelo de seguridad de multi capas muy robusto, y se han desarrollado diferentes controles por fabricantes de *software* de seguridad, los incidentes de seguridad de *Android* siguen aumentando y los atacantes son cada día más hábiles para vulnerar los sistemas. Gran parte de la responsabilidad del funcionamiento de este modelo de seguridad de *Android* se deja al usuario final. Esta situación genera inconvenientes de seguridad ya que el usuario puede otorgar permisos a las aplicaciones sin verificar. Esto sumado a los problemas de detección de las herramientas convencionales

Se han realizado bastantes estudios que proponen diferentes herramientas de seguridad para ayudar al usuario en la detección de *malware* en *Android*. Sin embargo, la oferta de aplicaciones sigue en aumento y siempre existe el riesgo para el usuario de instalar aplicaciones maliciosas, al no tener claridad de lo que realmente instala. *Android* es un espacio muy conveniente para los desarrolladores que pueden publicar fácilmente sus aplicaciones. Esta facilidad hace que los controles de *malware* sean insuficientes. Se puede optar por instalar *software* antivirus y escanear todas estas aplicaciones antes de descargar, “Sin Embargo la naturaleza reactiva de los antivirus existentes para móviles es inadecuada para la detección de aplicaciones nuevas intervenidas con mutaciones de virus.

Existen desarrollos de técnicas de detección de *malware* y herramientas que necesitan de una muestra de *malware* para aplicar sus métodos de aprendizaje de máquina o que dependen de la generación de firmas para realizar una detección efectiva. En otras técnicas, una interacción efectiva solo se logra si el usuario posee conocimientos avanzados en el manejo del sistema operativo. La *metadata* de las aplicaciones en la tienda de *Google* es la información que el usuario puede conocer antes de realizar la instalación de la aplicación en el dispositivo, como los permisos del sistema operativo, la información del desarrollador y el puesto que la aplicación ocupa en la tienda, que es información relevante para realizar la detección de *malware*. En ocasiones la *metadata* no es totalmente publicada no brindando al usuario final información necesaria para que este decida si instala o no la aplicación

Para la realización de este trabajo se han analizado los estudios y proyectos que previamente han tratado de dar solución al problema del *malware* en dispositivos móviles. De esta manera se puede dar una visión general de las utilidades necesarias para llevar a cabo esta labor y conocer de forma concisa cómo se puede implementar el proceso de detección de *malware* para dispositivos móviles, disminuir los ataques de día cero y bloquear comportamientos indeseables de aplicaciones relacionados a los permisos que se le otorgan.

(Kabakus & Dogru, 2018) desarrollaron *mad4a. E*ste es un sistema de detección y análisis de *malware* basado en técnicas de análisis dinámico y estático, técnicas que tienen sus ventajas y desventajas. Este modelo utiliza grandes bancos de datos para obtener un mejor contexto de las acciones realizadas por el *malware*. Los autores concluyeron que el modelo de permisos de *Android* no es lo suficientemente robusto como para dar una idea de qué aplicación puede ser benigna o maligna. Otro de los inconvenientes encontrados en dicho modelo es que las aplicaciones malignas tienden a desconectar el intercambio de datos durante su instalación y por ende la cantidad de información compartida por la aplicación es limitada si se compara con la de aplicaciones benignas, que a su vez presentan una mayor complejidad en la solicitud de permisos para poder proveer de más funcionalidades a la aplicación en proceso de instalación a diferencia de las malignas que se enfocan en la forma de realizar daño y presentan exageración en sus solicitudes de permisos.

*(*Elisth et al, 2018) realizaron un estudio comparativo basado en el comportamiento del *malware* y las aplicaciones benignas usando sus características estáticas y dinámicas, en el análisis estático, se consideran los permisos necesarios para una aplicación. Usaron la herramienta *Droidbox* la cual es una *Sandbox* que se usa con el fin de monitorear algunas acciones de la aplicación, como actividades de red, actividades del sistema de archivos, actividades criptográficas, fuga de información, etc, los autores consideran estas acciones, así como las llamadas de *API* dinámicas de las aplicaciones. En ese trabajo se propone implementar una herramienta *antimalware* para *Android* que pueda detectar si una aplicación es un *malware* o no, antes de la instalación.

(John & Vinod, 2018) desarrollaron un sistema de análisis y caracterización de *malware*, utilizando diferentes características como aprendizaje de máquina y análisis estático, haciendo uso de características como: componentes de *hardware*, solicitud de permisos, componentes de las aplicaciones. El experimento usa el conjunto de datos *Drebin* con más de 3000 muestras de *malware* y 1631 muestras benignas recolectadas de la tienda de *Google Play Store*. consiguiendo un sistema funcional que permite detectar, clasificar y mejorar el rendimiento, concluyendo que el enfoque puede ser usado para dar soporte a antivirus comerciales.

(Singh & Hofmann, 2017) buscaron extraer el comportamiento de las llamadas del sistema de 216 aplicaciones maliciosas y 278 aplicaciones normales para construir un vector de características y así entrenar un clasificador de lenguaje de máquina para realizar la detección de *malware* en *Android*. Incluyen entre otras técnicas de clasificación y modelos de predicción las siguientes: árbol de decisión, bosques aleatorios, potenciación de gradiente, algoritmo k-NN, redes artificiales neuronales, máquina de vectores de soporte y aprendizaje profundo. Por medio de técnicas de clasificación de características se seleccionaron las más apropiadas entre un grupo de 337 atributos (llamadas de sistema) y se determinó el peso de estas, descartando las de menor rango para medir el desempeño de los clasificadores, con eficacia y experiencia. El experimento muestra que las máquinas de vectores de soporte, después de seleccionar las características a través del análisis de correlación, superaron a otras técnicas en las que se logró una precisión del 97.16% con recuperación del 99.54% para aplicaciones maliciosas. El estudio también contribuye al identificar el conjunto de llamadas a los sistemas que son cruciales para identificar la intención maliciosa de las aplicaciones de *Android*.

(Leeds, Keffeler, & Atkison, 2017) examinaron dos tipos de características de *Android*, las solicitudes de permisos y los llamados al sistema aplicando aprendizaje de máquina, como una forma de detección de *malware* para *Android*. Los resultados mostraron que los datos referentes a la solicitud de permisos son mejores en la detección de *malware* que los datos de llamadas del sistema. Se logró una tasa promedio de precisión de clasificación del 80% cuando se utilizan datos de permisos para determinar la actividad maliciosa en dispositivos *Android*, por lo tanto, es una forma confiable de detectar *malware.*

2.2 Técnicas de detección de *malware* en *Android*

Para la realización de este trabajo se han analizado los estudios y proyectos que se relacionan en los antecedentes en los que previamente se ha tratado de dar solución al problema del *malware* en dispositivos móviles. De esta manera se puede dar una visión general de las utilidades necesarias para llevar a cabo esta labor y conocer de forma concisa como se puede implementar el proceso de detección de *malware* para dispositivos móviles.

En esta sección se estudian los modos de tratar los datos una vez han sido recopilados y extraídos del dispositivo que se va a estudiar, así como los tipos de estrategias de detección de *malware* en dispositivos móviles desde cuatro perspectivas, dependiendo de la naturaleza de los datos. Estas son: análisis estático, análisis dinámico, análisis mixto y análisis de metadatos. También se detallan los conjuntos de *malware* más utilizados para realizar pruebas.

2.2.1 Estrategias de detección

Para poder extraer los datos de la aplicación o sistema y hacerlos útiles para obtener conclusiones sobre su legitimidad, hay que elegir qué subconjunto de información es el que se va a analizar. Hay múltiples elementos que pueden ser monitorizados en un dispositivo móvil. Por ejemplo, componentes *hardware*, las comunicaciones, los sensores y el sistema (que incluye los procesos, la memoria, el planificador y el almacenamiento) o la actividad del usuario.
A continuación, se detallan las cuatro estrategias de análisis utilizadas en los trabajos relacionados a esta temática, en cada una de las cuales se monitorear a uno o varios de los elementos anteriores.

2.2.2 Análisis Estático

El análisis estático se centra en el estudio del archivo *APK* que contiene la aplicación móvil que se desea analizar. Un archivo *APK* viene definido por tres aspectos: el fichero *AndroidManifest* donde se declaran los permisos que requerirá la aplicación, el código fuente *Java* que posee la funcionalidad y, por último, los recursos necesarios para un correcto funcionamiento como pueden ser bases de datos, imágenes o diseños.

De esta forma el análisis estático investiga acerca de los permisos concedidos a la aplicación, su código fuente, sus filtros de intención, los sensores *hardware* que utilizara (ya que suelen sugerir su funcionalidad) y, por último, las direcciones de red asignadas evitando un posible ataque de comando y control.

Una primera aproximación a este método es proporcionada por *Androguard*, que permite descompilar el código *Dalvik*, y dados dos archivos *APK*, compara la similitud de su código fuente en *Java*.

(Desnos & Gueguen, 2011) una forma de concluir el estudio del código fuente es identificar el *malware* mediante una firma. Esta firma supone haber encontrado una secuencia de llamadas en el código que se ha demostrado maliciosa. Sin embargo, este método es altamente vulnerable a la ofuscación, pues una nueva aplicación con un cambio de orden en esta secuencia será indetectable.

(Arp, Munyaneza, Crosslin, Trumble, & Bextine, 2014; Dong-Jie, Ching-Hao, Hahn-Ming, Kuo-Ping, & I, 2012) *DroidMat*, en cambio, centra su estudio en los filtros de intención recogidos en el *AndroidManifest*. Para ello procesa los datos mediante la fórmula de la probabilidad condicionada de Bayes y los algoritmos de agrupamiento *k-means* y vecino más próximo. Los autores también utilizan el *AndroidManifest* para descubrir los recursos *hardware* que la aplicación usará, si por ejemplo necesita *3G* y *GPS* puede tratarse de un *malware* que registra la localización del usuario. El inconveniente de estas propuestas es difícil conseguir una gran precisión basándose en sólo un aspecto concreto de los que conforman el análisis estático como es el *AndroidManifest*.

(Zhibo, 2012) Otro método que estudia más características es el proporcionado por *TrustDroid*. En él se describe una estrategia que consiste en aislar y clasificar las aplicaciones en distintas categorías en función de su procedencia, la firma del desarrollador, las certificaciones que posea, los permisos requeridos o la información de la que se disponga sobre la aplicación en una base de datos común que se iría creando manera colaborativa. De esta forma se puede regir el comportamiento de cada categoría y la interacción entre ellas. Cabe decir que este método es sencillo, pero, aunque permite identificar *malware* previamente conocido gracias a la información de la base de datos, sería incapaz de detectar una muestra maliciosa desconocida o de día cero.

El análisis estático presenta la ventaja de la sencillez del proceso de extracción de los datos necesarios, residentes en el archivo *APK* (Heras. C & Sierra. L, 2015). Además, dado que *Android* funciona bajo un núcleo *Linux*, el estudio de los permisos concedidos a la aplicación supone la primera línea de defensa. Sin embargo, un estudio centrado en los permisos no consigue una alta precisión, pues no permite definir las actividades que realmente se están llevando a cabo. Por lo tanto, se hace necesario estudiar el código, ya que es la fuente de las acciones maliciosas. Sin embargo, esta es una labor compleja, pues es vulnerable a técnicas de ofuscación o cifrado para evadir los sistemas de detección. De esta forma un estudio completo de las características estáticas se convierte en un trabajo con cada vez más dificultad, que requiere de un desarrollo agresivo de nuevos métodos.

2.2.3 Análisis Dinámico

El análisis dinámico consiste en monitorizar el comportamiento del sistema para extraer la actividad de las aplicaciones instaladas a través de los siguientes identificadores: llamadas al sistema, interacción con el usuario, uso de los componentes *hardware* o tráfico de la red (Heras. C & Sierra. L, 2015). Posteriormente, se realiza una fase de entrenamiento en la que se aprende cual es el comportamiento adecuado para poder clasificar de forma correcta los datos obtenidos y, de este modo, poder distinguir posteriormente las actividades legitimas de las maliciosas.

(Feizollah, Anuar, Salleh, & Amalina, 2015) el modo de análisis dinámico más extendido es el estudio de las llamadas al sistema. Esto es debido a que el estudio completo de las características del tráfico de red supone investigar la gran cantidad de registros donde se almacenan los cambios, lo que puede llegar a ser muy costoso computacionalmente.

(Shabtai et al., 2016) Dado que el comportamiento del *malware* debe verse reflejado en varios ámbitos es importante poder seleccionar las características más importantes. Esto permite hacer el análisis más sencillo y efectivo. En este sentido, *Andromaly* propone una plataforma de trabajo dedicada a la detección ligera de *malware*, llevando a cabo la recopilación de datos concretos tales como el consumo de batería, número de paquetes enviados a través de *Wi-FI*, el número de procesos en ejecución o el consumo de *CPU*, más tarde, la información es procesada según el tipo de amenaza conocida, concluyendo con la emisión de alertas al usuario y la toma de acciones automáticas que mitiguen la amenaza.

(Saracino, Sgandurra, Dini, & Martinelli, 2016) Otro ejemplo de este método es *MADAM*. En él se mide la frecuencia de ejecución de cada llamada al sistema en T periodos, seleccionando las 12 llamadas que se han considerado como más representativas (*open, ioctl, brk, read, write, exit, close, sendto, recvfrom, recvmdg*). También se registra el número de *SMS* enviados, así como si el usuario ha estado usando el dispositivo o, en cambio, ha estado en modo inactivo.

Por lo tanto, la información a tratar es representada mediante un vector de datos de 14 posiciones, donde las 12 primeras corresponden a las llamadas y las dos últimas indican si el dispositivo ha estado activo y cuantos *SMS* se han enviado. Este vector es analizado y comparado con los resultados aprendidos anteriormente, para así determinar si el comportamiento del dispositivo está siendo adecuado (estándar) o malicioso (sospechoso). El mayor inconveniente de esta aproximación es la incapacidad de determinar posteriormente, de forma sencilla, cual es la aplicación que está provocando el comportamiento malicioso.

(Burguera, Zurutuza, & Nadjm-Tehrani, 2011) En *Crowdroid* se proponen registrar las llamadas al sistema realizadas por una aplicación durante la interacción del usuario en su totalidad. A partir de ello es posible la generación de un vector cuya longitud sea el número de llamadas que existen en *Android* (dependiendo de su versión puede variar entre 250 y 380). Para cada uso de una aplicación, se registra el número de veces que se ha producido cada una de ellas. Después, se envían los datos a un servidor remoto. En él se agrupan los datos obtenidos en las diversas ejecuciones mediante el algoritmo *K-means* y se componen los resultados del uso legítimo de cada aplicación. De esta forma, una vez construido un esquema de cómo debería ser su uso normal, este se podrá comparar con los datos extraídos de ejecuciones desconocidas de las aplicaciones para las cuales existan datos en el sistema.

Esta propuesta presenta la ventaja de que, al realizarse el análisis en un servidor externo al dispositivo, no se consume batería ni recursos del mismo. Además, permite comparar una ingente cantidad de datos, ya que podría haber muchos dispositivos enviando los suyos. Sin embargo, la idea de utilizar vectores de llamadas conlleva un claro inconveniente, y es que no resulta portable entre diferentes versiones de *Android*.

(Liu, Yan, Zhang, & Chen, 2009) En *VirusMeter* la detección mediante el análisis del uso de recursos se centra en el análisis de la batería del dispositivo. Los autores asumen la premisa de que cualquier actividad maliciosa debe consumir energía. Basándose en un modelo de potencia predefinido para el gasto de la batería en una actividad normal, se supervisan las actividades del sistema, y si estas han producido un gasto mayor a un umbral especificado, se advierte al usuario de la situación de riesgo.

En general, esta estrategia de detección se hace compleja y costosa en dispositivos con recursos limitados, como son los dispositivos móviles. Además, deben de ser capaces de ignorar su propia actividad, centrándose únicamente en las características auditadas provenientes de la actividad externa; Sin embargo, han demostrado ser los más efectivos, pues permiten identificar el inevitable rastro que el *malware* deja en la actividad del sistema puesto que enmascarar la actividad maliciosa durante la ejecución conlleva un mayor grado de dificultad que hacerlo en las características estáticas, susceptibles a la ofuscación y cifrado.

2.2.4 Análisis Mixto

Los entornos más complejos y más escasos para la detección de *malware* utilizan conjuntamente técnicas de análisis estático y dinámico.

En *AASandbox* se extraen los permisos y el código de *Java* desde el propio archivo *APK*. Estos son utilizados como características estáticas. A continuación, se instala la aplicación estudiando el sistema de registro de llamadas. Así son consideradas como características dinámicas. (Bläsing, Batyuk, Schmidt, Camtepe, & Albayrak, 2010).

La propuesta de *ProfileDroid* consiste en examinar el *AndroidManifest* y el código *Java* como características estáticas. Además, se estudia la interacción del usuario, el sistema de registro de llamadas y el tráfico de red como características dinámicas. (X. Wei & Gurkok, 2012).

Estos sistemas, aunque completos, son muy costosos de desarrollar y mantener debido a la gran cantidad de trabajo que abarcan.

2.2.5 Análisis por Metadatos

Para detectar muestras maliciosas unas pocas propuestas se centran en la información de la aplicación a la que un usuario puede acceder antes de su descarga. Entre ellos destacan la descripción de la aplicación y su valoración en el mercado de descargas, la identificación del creador, la categoría a la que pertenece, los permisos que solicita, nombre del paquete, video promocional, sitio web de contacto o precio.

(Teufl et al., 2006)(Corporation, 2012) En *WHYPER* se recogen únicamente los metadatos relacionados con los permisos solicitados. En cambio, en se recopilan además una gran cantidad de datos respecto a la última vez que se modificó la aplicación, tales como el precio o el número de descargas.

El inconveniente de estas técnicas es que requieren de nuevos sistemas que permitan la interpretación de los datos, como es el caso de sistemas de procesamiento de lenguaje natural, con el fin de identificar elementos disonantes en las opiniones o en la descripción de una aplicación.

2.3 *Android*

En esta sección se presentarán las generalidades del sistema operativo *Android*, características relevantes, arquitectura, modelo de seguridad y *software* de seguridad.

2.3.1 Generalidades de *Android*

Sistema operativo basado en *Linux*, diseñado principalmente para móviles e inicialmente desarrollado por *Android Inc*, una compañía que más tarde fuera adquirida por *Google* en julio de 2005.(**Camargo, Galindo, & Vega, 2013**). La mayoría del código fuente de este sistema, se encuentra bajo licencia Apache, licencia de *software* libre y código abierto. En este, se encuentran los drivers necesarios para el acceso al *hardware*, en concreto, para la gestión de sus dispositivos. En principio, el desarrollador no accederá directamente a esta capa, sino que utilizará una serie de librerías que están en un nivel superior. Estas librerías se encuentran programadas en C; sin embargo, el programardor accede a estas desde un *API* (*Application Programming Interface*) de *Java*, lenguaje que se usa para el desarrollo de *Android*. Para esto, el sistema incluye una máquina virtual *java* (JVM), *Dalvik*, la cual ejecuta archivos con extensión ".dex" en lugar de los ".class" clásicos de *java*.

2.3.2 Características relevantes de *Android*

- El *framework* de aplicaciones que permite el reemplazo y la reutilización de los componentes.
- El navegador integrado el cual está basado en los motores *open source Webkit*; *SQlite* base de datos para almacenamiento estructurado que se integra directamente con las aplicaciones.
- El soporte para medios con formatos comunes de audio, video e imágenes planas; máquina virtual *Dalvik* entre otras.

2.3.3 Arquitectura *Android*

La arquitectura interna de *Android* está formada básicamente por cinco (5) componentes:

Aplicaciones: las cuales se incluye como base un cliente email, programa de *SMS*, calendario, mapas, navegador, contactos y otros. Todas escritas en lenguaje de programación *Java*.

***Framework* de aplicaciones**: los desarrolladores de *Android* tienen acceso al código fuente usado en aquellas que son base, esto para que no se generen más componentes distintos, que respondan a la misma acción (reutilización de componentes).

Librerías: *Android* incluye en su base de datos un set de librerías C/C++, que son presentadas a todos los desarrolladores a través del *framework* de las aplicaciones. Algunas son: *System* C library, bibliotecas de medios, bibliotecas de gráficos, 3D y SQLite, entre otras.

Runtime* de *Android: incorpora un set de librerías que aportan la mayor parte de las funcionalidades disponibles en las librerías base del lenguaje de programación *Java*. Cada aplicación *Android* corre su propio proceso, con su propia instancia de la máquina virtual *Dalvik*. *Dalvik* ejecuta archivos en el formato *Dalvik Executable* (.dex), el cual está optimizado para memoria mínima.

Núcleo *Linux*: *Android* depende de *Linux* para los servicios base del sistema como seguridad, gestión de memoria, gestión de procesos, pila de red y modelo de controladores. Este, además actúa como una capa de abstracción entre el *hardware* y el resto de la pila de *software*. (Villanova-Pascual, 2016, p13)

La arquitectura *Android* se ilustra en *Figura 2-1*

Figura 2-1. Arquitectura de *Android*. (Fuente: Villanova-Pascual, 2016).

2.3.4 Modelo de seguridad *Android*

En el libro *Android Security Internals***,** (Elenkov, 2015)**,** el autor explica cómo funciona el modelo de seguridad del sistema operativo *Android* dando una breve descripción de su arquitectura y funcionamiento, en este se puede apreciar que *Android* está diseñado basado en un modelo de capas las cuales se comunican a través de una interface llamada *Binder* encargada de realizar la comunicación entre los respectivos procesos del sistema para evitar que cualquier otro pueda acceder a los espacios de memoria reservados y causar problemas de seguridad o de inestabilidad del sistema. El modelo de seguridad de *Android* se ilustra en la *Figura* 2-2.

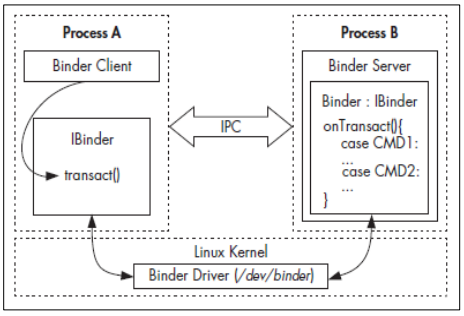


Figura 2-2*. Binder* IPC. (Fuente: Elenkov, 2015).

Para reforzar la seguridad durante el año 2018 *Google* tiene como tecnología *Bouncer* (Hou, 2012), la cual analiza las aplicaciones de *Google Play* en busca de *malware* y con la verificación de aplicaciones ya ha reforzado su política para desarrolladores.

2.3.4.1 Ejecución en Sandbox heredada de Linux

Cada vez que se instala una aplicación en *Android* se crea un usuario *Linux* para esta, de forma que una aplicación solo tiene acceso a sus recursos y no puede interferir directamente en el *hardware*, es lo que se conoce como caja de arena o *sandbox*. Cualquier dato que la aplicación almacene no podrá ser leído o modificado por otras aplicaciones a no ser que se declare como *MODE_WORLD_READABLE y/o MODE_WORLD_WRITEABLE*, es decir, que otras aplicaciones puedan leer los datos y/o escribir en ellos. Aunque otra aplicación reescriba completamente el contenido de otra, el propietario siempre será el usuario asignado a la aplicación que lo creó (Elenkov, 2015).

Gracias a la *sandbox*, *Linux* garantiza un nivel seguridad a cada proceso. Cada aplicación se ejecuta en un proceso diferente con un usuario de *Linux* distinto, evitando en gran medida las interferencias entre aplicaciones. Para que dos aplicaciones se ejecuten en el mismo proceso, deberán tener el mismo usuario asignado y el mismo certificado digital, y serán tratadas como una sola. (Simón, 2014).

Las aplicaciones deben estar firmadas con un certificado digital que identifique al autor. Cada vez que se modifique la aplicación esta deberá ser firmada de nuevo y solo podrá hacerlo el propietario de la clave privada.

Se han encontrado diversos bugs que permitían modificar la aplicación saltándose este certificado, y las diversas compañías lanzaron actualizaciones para sus terminales.

2.3.4.2 Modelo de permisos

El más importante pilar de seguridad en *Android* son los permisos. Si una aplicación desea realizar alguna acción que pueda comprometer la seguridad de nuestro dispositivo debe utilizar los permisos, de forma que el usuario está completamente informado de los riesgos que puede llevar instalar esa aplicación y decidir si instalarla o no. Si una aplicación intenta hacer uso de un permiso no declarado se creará una excepción de permiso y la aplicación se detendrá. (Simón, 2014).

2.3.5 Controles de seguridad para *Android*

En esta sección se muestran algunos controles de seguridad para *Android* desarrollados por la comunidad científica y fabricantes de *software* de seguridad, un análisis más detallado de estos trabajos se encuentra en la Tabla 8-1 y Tabla 8-2 del Anexo A

En el artículo “Riesgos de seguridad en *Android* del año 2015”, de la revista Seguridad. Defensa digital de la Universidad Autónoma de México, se revisan varias amenazas enfocadas específicamente en el sistema operativo *Android*, con el propósito de conocer los riesgos a los que está expuesto un usuario cuando utiliza esta tecnología planteando esto como un primer paso en el camino para proteger la información (Mendoza López, 2015).

El objetivo del proyecto de fin de carrera, ingeniería de telecomunicaciones de la Universidad Carlos III de Madrid, “Gestión de riesgo en dispositivos *Android* basada en eliminación de vulnerabilidades y detección de contextos del año 2013”, era proporcionar un mayor control sobre los riesgos de seguridad en *Android*. En este se pretendía incrementar el conocimiento y el control sobre las posibles vulnerabilidades presentes en las aplicaciones, además de contribuir a la adaptabilidad automática de la seguridad del dispositivo en función de su entorno. El sistema implementado cuenta con un gestor de vulnerabilidades y un módulo de seguridad por contexto, repositorio público de vulnerabilidades *software* estadounidense y la *API* de *Android*, tras la evaluación de varias alternativas como posibles fuentes de información. (de Matías García, 2013).

En lo que respecta a seguridad en la configuración de dispositivos se encuentra la “Guía de seguridad para *Smartphone*: ¿cómo configurar tu *Android* de la forma más segura?” elaborada en el año 2014, por la empresa de seguridad informática *ESET*. En esta guía se observan las cifras del mercado de plataformas móviles, evidenciando que el sistema operativo *Android* es el más utilizado en todo el mundo, por lo cual la guía apunta a explicar los aspectos más importantes a tener en cuenta a la hora de hacer las configuraciones de seguridad pertinentes en esta plataforma, de modo que se puedan prevenir incidentes como ataques informáticos y robo de información. (*ESET*, 2014) .

Otra investigación que analiza las debilidades que pueden presentarse en el sistema operativo *Android* y presenta recomendaciones a los usuarios de estos dispositivos con el objetivo de prevenir ataques que puedan permitir el robo, manipulación o pérdida de la información que con estos dispositivos se pueda manejar, es la monografía titulada “Seguridad en dispositivos móviles *Android*” realizada en la Universidad Nacional Abierta y a Distancia UNAD Escuela de Ciencias Básicas, Tecnología e Ingeniería especialización en seguridad informática, en el año 2015. Las pruebas de este trabajo fueron realizadas en la versión de *Android* 4.4 Llamada *Kit Kat*. (Betancur & Eraso, 2015)

Aplicación de políticas de seguridad basadas en el estándar *ISO* 17799 | 18 (ISO/IEC, 2005) Una vez revisada la literatura se ha evidenciado que los proyectos realizados se enfocan en explicar los riesgos que presentan los dispositivos móviles, analizando aspectos técnicos de los mismos, detallando el funcionamiento de dichos riesgos sin tener en cuenta la responsabilidad del usuario en el manejo de la información mediante el dispositivo móvil.

2.3.6 *Android GO*

Mediante *Android GO*, anunciada en la conferencia *Google* I/O de 2017 y centrada en el *software* para complementar la antigua iniciativa *Android ONE* del año 2014, más centrada en el *hardware* se pretende ofrecer actualizaciones de manera más ágil y continuada para dispositivos móviles de bajo coste, incluso pese a la existencia de personalizaciones de los fabricantes, adicionales a otras características. *Android GO* proporciona las capacidades de seguridad y protección de *Google Play Protect* (GPP) a los dispositivos móviles bajo su ámbito. Durante 2017, *Google* ha llevado a cabo la integración de múltiples de sus servicios y plataformas de seguridad bajo un mismo “paraguas”. *Google Play Protect* aún a múltiples capacidades y servicios previamente disponibles de manera individual en *Android*, cómo *Verify* Apps*, Find My Device, Google Bouncer, Android Device Manager, SafetyNet, Google Safe Browsing (en Chrome), el servicio de reCAPTCHA*, etc.

El objetivo es proporcionar un conjunto de servicios comunes para los dos billones de usuarios que hacen uso de *Android* a diario y proteger los dispositivos móviles, aplicaciones y los datos de los usuarios frente a las últimas amenazas de seguridad. Entre estos servicios se incluye, por ejemplo, el análisis y escaneo de más de 50 billones de aplicaciones diariamente. Estos números permiten hacerse una idea del volumen y complejidad del ecosistema móvil actual, en este caso, sólo de *Android* (CCN, 2013).

3 Malware en *Android*

(Malicious Mobile Threats Report., 2011) (Feizollah, Anuar, Salleh, & Wahab, 2015) *malware* es cualquier tipo de *software* o código de programa hostil, intrusivo o diseñado para usar un dispositivo sin el conocimiento del propietario. La evolución y proliferación del *malware* en dispositivos móviles está íntimamente ligada al aumento de sus capacidades de red y recursos informáticos. Así, se tienen evidencias de que el primer *malware* para móviles fue desarrollado en 2004 para atacar a dispositivos *Symbian*. Sin embargo, es a partir de 2010 cuando empieza a crecer significativamente, estando dedicado principalmente a los sistemas *Android* u iOS.

La empresa de seguridad *G DATA* asegura que en 2017 fueron detectados más de 3 millones de *malware* lo que corresponde a 343 amenazas por hora (Lueg, 2018). La *Figura* 3-1 muestra el crecimiento de amenazas de *malware* para *Android* por año

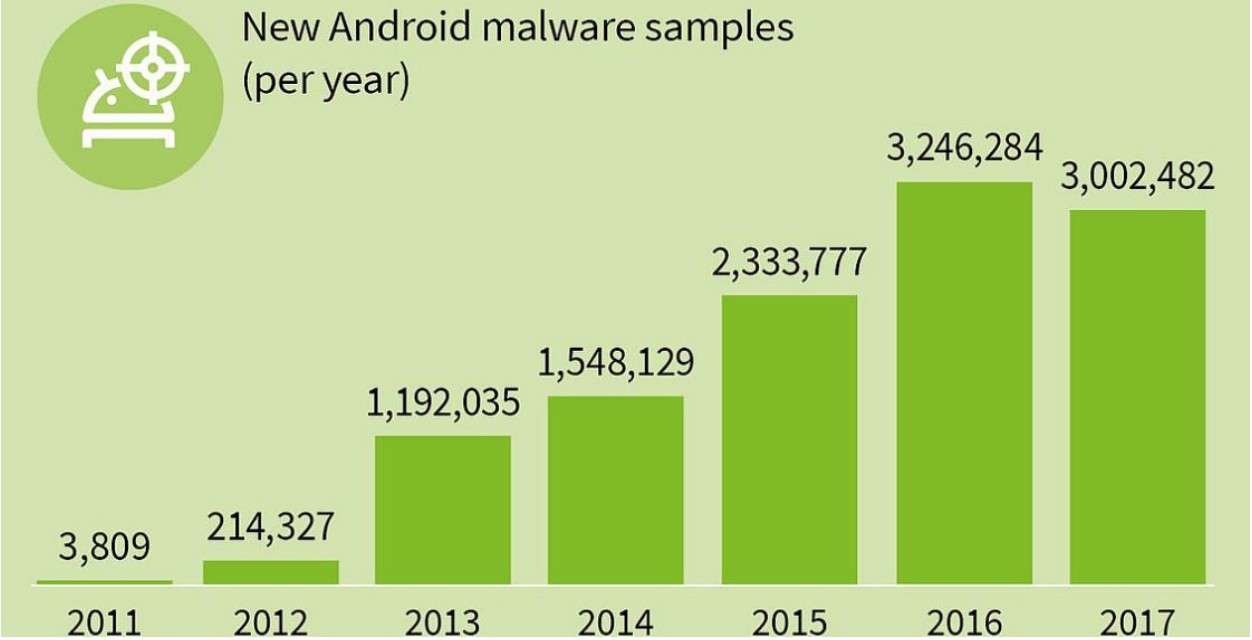


Figura 3-1. Amenazas de *malware* para *Android* detectadas por año. (Fuente: G DATA *Security*, 2018).

Para analizar los ataques y sus tipos, se deben tener en cuenta tres perspectivas: comportamiento y objetivos, formas de distribución del *malware* y métodos de adquisición de privilegios.

3.1 Comportamiento y objetivos del ataque

En la mayoría de los casos el *malware* persigue varios objetivos y además su propósito y comportamiento puede variar por medio de una orden remota emitida por quienes lo controlan. El más común de sus objetivos es el beneficio económico. En este sentido se pueden encontrar ejemplos varios, como en el caso del fraude u *overbilling*. Este ataque consiste en cargar gastos a la cuenta de la víctima (que son transferidos a la del atacante) por medio de llamadas o *SMS* enviados a números de tarificación adicional sin el consentimiento del usuario. Habitualmente, el usuario relaciona este tipo de ataques con un conflicto con la compañía proveedora del servicio, pasando por alto su verdadera naturaleza (Teufl et al., 2006).

(Burguera et al., 2011; Racic, Ma, & Chen, 2006) Otro tipo de ataque es la denegación de servicio o sabotaje, el llamado ataque *DoS* (del inglés Denial of Service) que trata de consumir la batería del dispositivo limitando su tiempo de operación, o bien de negarle al usuario el acceso a determinados recursos o redes. Cuando se ejecuta de forma distribuida (*DDoS*), puede ocasionar perjuicios no sólo a uno o varios usuarios, sino incluso a diversas organizaciones. Por ejemplo, se puede llegar a colapsar la conexión a Internet de una zona o servidor si se empiezan a enviar paquetes masivos a una red.

Por último, se puede tratar de comprometer la privacidad del usuario por medio de la técnica llamada *sniffing*, la cual aprovecha los datos que recogen los sensores y las redes de las que hace uso el dispositivo, pudiendo obtener imágenes, grabaciones telefónicas, contraseñas, mensajes de correo electrónico, datos bancarios o cualquier tipo de información que maneje, envíe o reciba el dispositivo.

Esta es una de las amenazas más preocupantes para los usuarios u organizaciones, puesto que en los dispositivos móviles se almacenan y se tratan datos privados y clasificados, tanto personales como empresariales. *La Figura 3-2* muestra tipos de ataques por sistema operativo

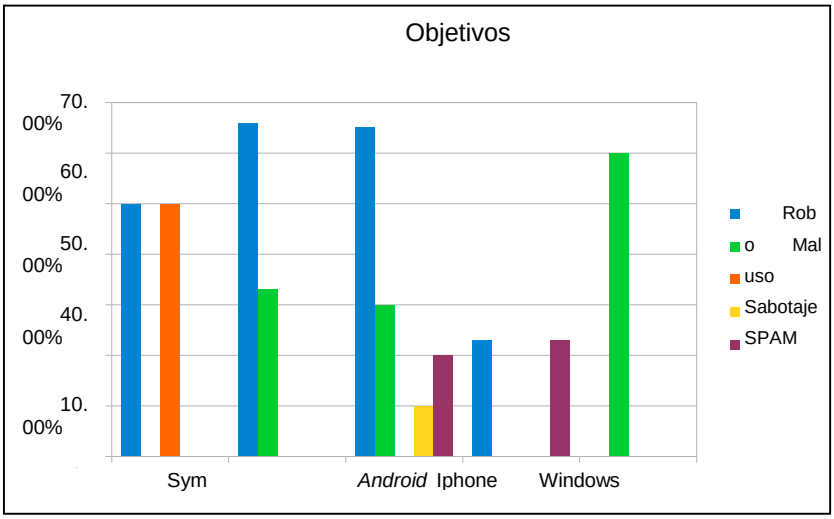


Figura 3-2. Tipos de Ataques por sistema operativo. (Fuente: Heras. C & Sierra. L, 2015).

3.2 Adquisición de Privilegios

(Shabtai et al., 2016) Para que el *malware* sea efectivo no es suficiente con que se propague al dispositivo, también tiene que conseguir una serie de privilegios para poder acceder a las partes críticas del sistema y realizar las acciones necesarias para ejecutar su labor. Habitualmente, estos privilegios son directamente concedidos por los usuarios al instalar aplicaciones en apariencia legítimas pero que esconden algún comportamiento malicioso. Los usuarios no son conscientes de la repercusión de transferir determinados permisos a programas desconocidos y, en ocasiones, ni tan siquiera prestan atención a las peticiones para su autorización.

(Davi, Dmitrienko, Sadeghi, & Winandy, 2011) Otro procedimiento basado en la tecnología consiste en la explotación de las vulnerabilidades o errores en la configuración de la plataforma. Se utilizan *rootkits* para detectar de qué forma se pueden aprovechar dichos errores. Este tipo de *malware* infecta el sistema operativo, por lo que puede ser considerablemente peligroso y, además, deja la puerta abierta a futuras infecciones

Por lo tanto, este método es altamente eficaz, tal y como se muestra en la *Figura 3-3* es posible adquirir privilegios por error del usuario o del sistema operativo

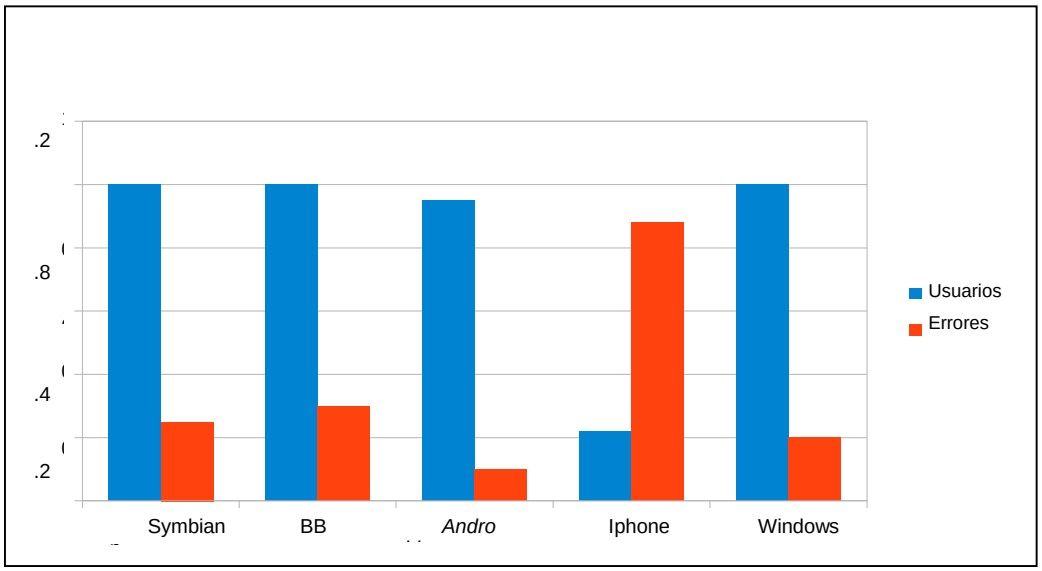


Figura 3-3. Formas de Adquisición de Privilegios por usuario o error del sistema por sistema operativo. (Fuente: Heras. C & Sierra. L, 2015).

3.3 Virus y *Malware* en Móviles *Android*

(*Kaspersky-LABS*, n.d.) Con el aumento del número de personas que utilizan sus *Smartphones* y *tablets* para navegar por la Web, actualizar los sitios de redes sociales y comprar y realizar transacciones bancarias online, los cibercriminales y el *malware* centran cada vez más sus ataques en los dispositivos móviles, con la consecuente aparición de nuevas amenazas para *Smartphones* y amenazas móviles.

Los dispositivos *Android* "están siendo atacados"

Durante el año 2012, el 99 % de todo el *malware* de móviles detectado por *Kaspersky Lab* se había diseñado con la plataforma *Android* como objetivo. Durante ese año, los expertos en seguridad en Internet de *Kaspersky* identificaron más de 35 000 programas malintencionados dirigidos a *Android*.

Las razones del gran aumento del *malware* dirigido a *Android* son las siguientes:

- La plataforma *Android* se ha convertido en el sistema operativo más utilizado en los *Smartphones*, con una cuota de mercado del 85,9% en el primer cuarto del 2018 (Gartner, 2018).

- La naturaleza de plataforma abierta del sistema operativo *Android*, la facilidad con la que se pueden crear las aplicaciones y la amplia variedad de mercados de aplicaciones no oficiales que existen influyen de manera significativa en la seguridad.

¿Qué tipos de amenazas afectan actualmente a los dispositivos *Android*?

(*Kaspersky-LABS*, n.d.) Los objetos maliciosos más comunes detectados en *Smartphones Android* pueden dividirse en tres grupos principales:

Virus troyanos por *SMS*, Módulos de publicidad y *exploits* para obtener acceso de usuario administrador a *Smartphones*.

Se ha detectado la existencia de *malware* en las tiendas de aplicaciones. Durante el año 2012, *Kaspersky* detectó programas maliciosos en *Google Play*, la tienda de aplicaciones de *Amazon* y otras tiendas de aplicaciones de terceros.

Riesgos de seguridad asociados a las operaciones bancarias online para los usuarios de *Android*

(Santomá, 2004) Los bancos y sistemas de pago electrónico ofrecen varias formas de proteger las transacciones de los usuarios, entre las que se incluyen la utilización de tokens electrónicos y contraseñas de un solo uso, la confirmación de transacciones a través de códigos enviados al teléfono, etc.

No obstante, los cibercriminales están desarrollando programas que burlan estas medidas. Por ejemplo, la familia de programas *Zitmo* está diseñada para atacar el teléfono móvil de un usuario y burlar los sistemas de autenticación de dos factores utilizados por los bancos europeos. Estos programas maliciosos para móviles funcionan de forma conjunta con *Zbot (ZeuS)*:

- En primer lugar, *Zbot* roba el nombre de usuario y la contraseña para acceder al sistema bancario online desde el computador infectado.
- A continuación, durante una transferencia de dinero, *Zitmo* (el homólogo para móviles de *Zbot)* entra en acción y reenvía el código de autorización de la transacción (*TAN*, del inglés "*Transaction Authorisation Code*") a los cibercriminales.

3.3.1 Incidentes de Ciberseguridad en *Smartphone* con sistema operativo *Android*

Dada la popularidad del sistema operativo *Android* de manera informal se tiene noticias de incidentes de ciberseguridad registrados en foros de diferentes revistas y por fabricantes de *software* de seguridad. En el **Anexo A** se presentan algunas noticias relevantes de vulnerabilidades de la plataforma explotadas entre el 2013 y el 2017.

4 Detección de *malware*

Para el primer trimestre del 2018 *Android* es el sistema operativo más usado en dispositivos móviles incluidos teléfonos inteligentes como se ve en la *Figura 4-1*

Worldwide Smartphone Sales to End Users by Operating System in 1Q18 (Thousands of Units)

Operating System	1Q18 Units	1Q18 Market Share (%)	1Q17 Units	1Q17 Market Share (%)
Android	329,313.9	85.9	325,900.9	86.1
iOS	54,058.9	14.1	51,992.5	13.7
Other OS	131.1	0.0	607.3	0.2
Total	**383,503.9**	**100.0**	**378,500.6**	**100.0**

Source: Gartner (May 2018)

Gartner clients can read more in the Gartner report titled "Market Share: Final PCs, Ultramobiles and Mobile Phones, All Countries, 1Q18 Update."

Figura 4-1. Dispositivos por sistema operativo en el mundo. (Fuente: Gartner, 2018).

(Villanova-Pascual, 2016) Muestra en su tesis que la plataforma ha sido un foco para los desarrolladores de *Malware* ya que en ella tienen la posibilidad de llegar a un gran número de víctimas, presenta el modelo de seguridad del fabricante de *Android* y las tecnologías de detección desarrolladas por fabricantes reconocidos de *software* de seguridad, luego hace una selección y evaluación de herramientas de prevención existentes en el mercado, con un conjunto también seleccionado de muestras de *malware*. El autor busco determinar si los controles son eficientes para la detección de *malware* y verificar si afectan el rendimiento del equipo en cuanto a consumo de *CPU*, memoria y duración de la batería. Se destaca el resultado de que ninguna solución reconoció el 100% de las muestras de *malware*, por lo que en el estudio el autor concluye que no existe seguridad total, independientemente de si las herramientas son pagas o no. Finalmente concluye que el *malware* en *Android* no para de crecer, que hay muchos tipos de *malware* y que las múltiples medidas de seguridad de *Google* son adecuadas en muchos aspectos y en otros no mostrando sus ventajas y desventajas

4.1 Indicadores de compromiso *IOCs*

(*Trend-Micro*, n.d.) Los indicadores de compromiso son definidos como artefactos forenses que se utilizan como signos de que un sistema ha sido comprometido por un atacante o que ha sido infectado por un *software* malicioso en particular. Estos eventos pueden ser conexiones de red a direcciones no conocidas, procesos desconocidos, firmas de dudosa procedencia o cualquier cosa que pueda revelar un comportamiento anormal del sistema.

Atendiendo a las cifras, la situación actual en el ámbito de la seguridad informática, en lo que a amenazas se refiere, resulta bastante inquietante:

- Hay alrededor de 170 millones de muestras de *malware*, de las cuales cerca de 70 millones han surgido en 2013.
- En el año 2012 el número de ficheros correspondientes a *malware* firmados digitalmente alcanzó los 2 millones.
- El cibercrimen ha generado pérdidas de 87.000 millones de euros en 2013.
- *Kaspersky* en el tercer trimestre del 2013 ha catalogado más de 120.000 muestras de *malware* para móviles.
- *McAfee* cataloga diariamente más de 100.000 muestras e identificó 200 nuevos ciberataques por minuto en 2013.

Es por ello que el enorme volumen de amenazas correspondientes a *malware* que surgen diariamente y la sofisticación de las mismas, como es el caso de las "*Advanced Persistent Threats*" o *APTs*, ha provocado la búsqueda de nuevas vías para proteger los sistemas y para poder comprobar si estos se han visto comprometidos, al quedarse las protecciones convencionales obsoletas.

Los Indicadores de Compromiso, "*Indicators of Compromise*" o *IOCs*, hacen referencia a una tecnología estandarizada que consiste en definir las características técnicas de una amenaza por medio de las evidencias existentes en un equipo comprometido, es decir, se identifican diferentes acciones como ficheros creados, entradas de registro modificadas, procesos o servicios nuevos, etc. de manera que puedan servir para identificar otros equipos afectados por la misma amenaza o prevenirlos de la misma.

De este modo, es posible realizar un intercambio sencillo y práctico de información con otras personas y grupos de gestión de incidentes e implementar las firmas en diferentes herramientas como:

- Sistemas de detección de intrusiones, "*Intrusion Detection System*" o *IDS*.
- Sistemas de prevención de intrusiones, "*Intrusion Prevention System*" o *IPS*.
- Sistema de detección de intrusiones en un Host, "*Host-based intrusion detection system*" o *HIDS*.
- Sistema de prevención de intrusiones en un Host, "*Hostbased Intrusion Prevention System*" o *HIPS*.
- *Firewalls*

El valor real de la tecnología radica precisamente en ese intercambio de información ya que permite generar y compartir conocimiento a partir de análisis forenses, respuesta a incidentes o análisis de *malware*. *IOC Bucket* es uno de los repositorios de *IOCs* más conocidos que puede servir como fuente de conocimiento y por tanto utilizar los *IOCs* disponibles para proteger un sistema.

Existen varios sistemas estandarizados de intercambio de este tipo de información, es decir, varias definiciones sobre cómo deben documentarse los indicadores de compromiso. Algunos de los más conocidos son enumeración y caracterización de atributos de *software* malintencionado o "*Malware Attribute Enumeration and Characterization*" *MAEC*. (Martinez, 2014)

4.1.1 *IOCs* para *Android*

En el caso del *Malware* en *Android*, basado en el artículo "*A review on feature selection in mobile malware detection*" (Feizollah et al, 2015) los indicadores de compromiso potenciales se pueden ubicar en cuatro categorías: Estáticas, dinámicas, hibridas y metadatos. La *Tabla 4-1* muestra las cuatro categorías de *IOCs* para el sistema operativo *Android*

Tabla 4-1 Indicadores de Compromiso para *Android.*

Estáticas	Son las características pertenecientes al archivo *APK* de la aplicación, como son el manifiesto de permisos, las conexiones de red o el código en lenguaje *Java*.
Dinámicas	Son la representación del comportamiento de la aplicación luego de ser instalada y en el momento de ser ejecutada, incluye el comportamiento con el sistema operativo o con la red.
Hibridas	Son una combinación de las características estáticas y dinámicas. Son las características más comprensibles porque analizan las aplicaciones desde varios aspectos.
Metadatos	Consiste en los metadatos pertenecientes a la aplicación como puede ser la información en *Google Play*.

Fuente: (Feizollah, Anuar, Salleh, & Wahab, 2015).

describe los tipos de *IOCs* para teléfonos inteligentes que pertenecen a cada categoría descrita en la *Tabla 4-1*.

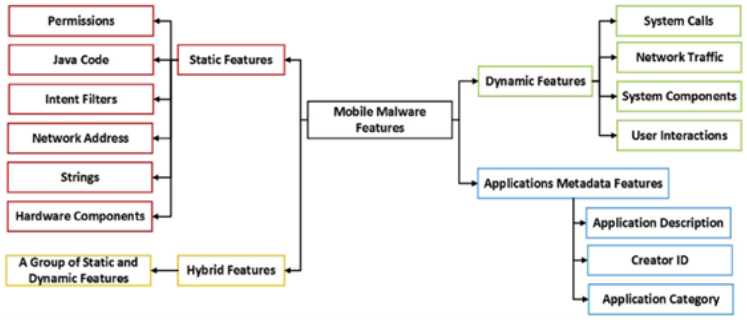


Figura 4-2. Taxonomía de las Características del *malware* para móviles. (Fuente: Feizolla et al, 2015).

4.2 Aplicaciones para detectar *IOCs*

En esta sección se describen las dos aplicaciones para *smartphone* con sistema operativo *Android* que se seleccionaron y probaron para monitorear y detectar *IOCs* para este proyecto

4.2.1 *Package Info*

Funciona en todos los dispositivos *Android*. Muestra los permisos y los metadatos de las aplicaciones instaladas en su teléfono inteligente.

Características principales:

- Muestra los detalles del paquete de cada aplicación, tales como el nombre del paquete, fecha de instalación y de actualización, el consumo de datos, tamaño de la aplicación, etc.
- Indica si esta aplicación aparece como programa de relleno de la comunidad de usuarios.
- Información detallada de versión de la aplicación, el código de versión, *SDK* de destino utilizado.
- Todos los permisos usados por el paquete están en la lista.
- Los servicios utilizados por la aplicación.
- Muestra receptores de *Android* y la información de biblioteca compartida.
- Nombre de paquete o el nombre de la aplicación.
- Capacidad para compartir la información detallada de un paquete a otros a través de correo electrónico, carbón, etc.
- Reúne más información acerca de la aplicación de búsqueda a través de *Google*.
- Capacidad para ejecutar y configurar aplicaciones.

Información de sistema

- Nombre del fabricante, modelo, número de serie, ID de dispositivo.
- versión de *Android*, la construcción y el estado ARRAIGADO.
- Información de la pantalla (resolución, densidad, *DPI*, píxeles / pulgada cuadradas)
- Información de Procesador (detalles de cada procesador, arquitectura de la *CPU*, variante, revisión).
- Información de red (*Wi-Fi MAC*, *IP* v4, la dirección *IP* v6, código de país, *IMEI*, número de serie *SIM*, *MCC*, *MNC*).
- Información de los sensores (lista de todos los sensores en el dispositivo).
- Memoria interna y los detalles de almacenamiento externo.
- Batería, porcentaje de batería, estado de carga, temperatura, voltaje, la capacidad, la tecnología de baterías. (*Google Play*, n.d.-b)

4.2.2 *Network Connections*

Herramienta que permite ver y monitorear toda conexión de entrada y salida desde y hacia su dispositivo *Android*. Con un módulo de conexiones de bajo nivel de captura asegura el mejor rendimiento con un mínimo uso de la batería. Funciona en teléfonos no *root*.

Características principales:

- Supervisa todas las conexiones a Internet desde y hacia su dispositivo *Android* notificando cuando aplicaciones ocultas se conectan a servidores remotos.
- Muestra información detallada *IP* para cada conexión - la dirección *IP*, la resolución inversa (*PTR*), número de *AS*, de abuso de datos *RBL*, información administrativa, etc.
- Obtiene información detallada acerca de cada aplicación que se conecta a los servidores remotos y conexiones de red utilizadas por ella - permisos concedidos a cada paquete capturado y todas las direcciones *IP* utilizadas por ella.
- En tiempo real ordena por frecuencia de actividad, las últimas conexiones o por nombres de paquetes.
- Registra y exportar los datos capturados.

La versión gratuita limita algunas características incluyendo periodo de conexiones, exportar 10 conexiones capturadas y el número de direcciones *IP* se limitan a cierta cantidad de intentos. (*Google Play*, n.d.-a)

4.3 Firmas digitales de aplicación

Una firma digital es un esquema matemático para la representación de la autenticidad de mensajes o documentos digitales. Una firma digital válida le da al receptor la confianza de que el mensaje fue creado por un remitente conocido (Autenticación) y que el mensaje no fue alterado en el camino de entrega (Integridad) (Katz, 2010)

El sistema operativo *Android* utiliza un mecanismo de firmas digitales que le permite al usuario final identificar quién es el desarrollador de las aplicaciones que instala. Este mecanismo funciona utilizando una clave privada perteneciente a cada desarrollador con la que a través de cálculos matemáticos se genera un certificado y una llave digital los cuales proveen una firma encriptada, única e inalterable.

(Elenkov, 2015) Al concluir el proceso de firmado de las aplicaciones, dentro del archivo de la aplicación (*APK*) sé almacena el directorio *META-INF* en donde se encuentran los archivos *MANIFEST.MF* y *CERT.SF.* El primer archivo almacena el *SHA-1 Digest* y la ruta de cada fichero contenido dentro del *APK* excluyendo todo lo contenido en el directorio *META-INF.* El segundo archivo *CERT.SF* o comúnmente conocido como el archivo de firma contiene al igual que *MANIFEST.MF* el *SHA1 Digest* de cada fichero, pero la diferencia es que incluye el *Digest* perteneciente al archivo *MANIFEST.MF y CERT.RSA* que es el *SHA-1 Digest* del archivo *CERT.SF* pero generado con la clave privada del desarrollador. La *Figura 4-3* muestra el Procedimiento de firma de aplicaciones de *Google Play*

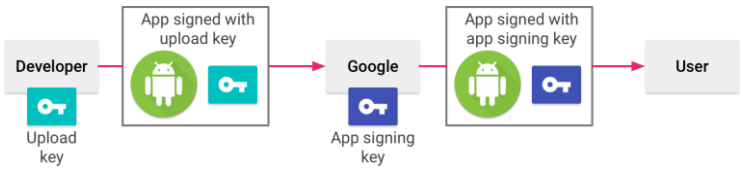


Figura 4-3. Procedimiento de firma de aplicaciones de *Google Play*. (Fuente: *Android* Developers, 2018).

La firma electrónica móvil permite incorporar una firma electrónica con certificados sobre documentos electrónicos desde una *tablet* (*iPad, tablets Android* como *Samsung Galaxy Tab* o *Galaxy Note o BlackBerry PlayBook*) o *Smartphone* (*iPhone, Android, Windows Phone, BlackBerry*) con plena validez legal y cumpliendo los requisitos de la Ley de Firma electrónica.

El certificado digital en dispositivos *Android* reside sobre la tarjeta *SD* del dispositivo en versiones anteriores a *Android* 4.0. A partir de la versión 4.0, *Android* implementa *Keychain* (almacén de certificados central).

4.4 Reglas *YARA*

(Alvarez, 2014) *YARA* es una herramienta orientada a (pero no limitada a) ayudar a los investigadores de *malware* a identificar y clasificar muestras de *malware*. Con *YARA* se pueden crear descripciones de las familias de *malware* (o lo que sea que se quiera describir) basado en los patrones textuales o binarios. Cada descripción de la regla consiste en un conjunto de cadenas o de expresiones booleanas las cuales determinan su lógica. *YARA* provee un lenguaje robusto basado en un reensamblaje de la sintaxis de C por lo cual es de fácil uso y fácil aprendizaje, está desarrollado con arquitectura multiplataforma, pero es comúnmente usado como una librería integrable con el lenguaje *Python*.

Aunque *YARA* parezca una herramienta bastante sofisticada, posee varias limitaciones, esto debido a que está basado a través de la coincidencia de firmas, patrones y cadenas por lo que se hace inútil en el momento de realizar detecciones que requieran de análisis comportamental.

4.5 *Koodous*

Koodous es una plataforma colaborativa para la investigación de *malware* de *Android* que combina la capacidad de las herramientas de análisis en línea con las interacciones sociales entre los analistas en un gran repositorio de *APK*. Permite a los usuarios escribir sus propias reglas de detección con lenguaje *YARA* e implementa varias librerías como: *Androguard*, una herramienta para realizar análisis estático a las aplicaciones de *Android*, permitiendo obtener la *metadata* más relevante como son, nombre del paquete, nombre de la aplicación, permisos, actividad, servicios, *SHA-1* del certificado, entre otras; *Droidbox*, una herramienta que permite hacer análisis dinámico a las aplicaciones de *Android*, permitiendo verificar si las aplicaciones envían mensajes de texto, realizan llamadas a través de la red, acceden a librerías o leen y escriben archivos; y file, un módulo que permite realizar coincidencias con el *MD5, SHA-1* y *SHA 256* de los *APK* analizados.

4.6 Correlación de eventos

Uno de los pilares en los que se basa la gestión de riesgos de seguridad de la información es, sin duda, el análisis y la gestión de logs y la correlación de eventos, lo que se entiende por *SIEM*. La confluencia de este pilar de la seguridad con el de la gestión de identidades y accesos a sistemas, redes y aplicaciones, la gestión de documentos y la gestión de evidencias, permitiría al responsable de seguridad *TIC* alcanzar su objetivo específico en la cadena de protección: saber en tiempo real qué está pasando en los sistemas tecnológicos que pueda ser relevante para su seguridad, para la de la información que tratan, y, en definitiva, para el negocio y actividades de su entidad. Las plataformas *SIEM* han tenido un desarrollo rápido en los últimos seis años, y constituyen hoy la base de proyectos de obligado emprendimiento para cualquier organización de cierta complejidad y bien gobernada. Sin embargo, los múltiples canales de las acciones fraudulentas y de los comportamientos no acordes con las políticas de seguridad, y el enfoque de la protección como un proceso integrado que afecta a la seguridad *TIC* y a la seguridad física, obliga a analizar con trazo fino cuando no a replantearse qué se quiere y puede monitorizar, a qué nivel, con qué reglas de correlación y para qué fines. Tal reflexión obliga a la industria y a los desarrolladores a refinar sus herramientas para que se ajusten en lo posible a las necesidades crecientes de los usuarios y a los requisitos legales. Conviene tener en cuenta, al respecto, la tendencia hacia un escenario marcado por la centralización de procesos esenciales de seguridad en *SOC y CERT*, y hacia servicios con base en *TIC* fundamentados en el uso de técnicas de virtualización y en *XaaS* y a ello se suma el uso intensivo de las tecnologías de la información en la vigilancia tradicional. (Rey & Carlos, 2010, p2)

5 Metodología

Para alcanzar los objetivos planteados se propone una metodología de búsqueda, experimentación, desarrollo y documentación. Las fases propuestas son:

5.1 Investigar técnicas de detección de *malware*

A partir del estado del arte, en el **Anexo A** se construye una tabla comparativa con los diferentes resultados de acuerdo con las técnicas de detección de *malware*. La **Tabla 8-1,** Técnicas de detección en teléfonos inteligentes con *Android*, muestra ocho técnicas de detección desarrolladas desde el año 2012 hasta el primer semestre del año 2018, lo que implica que el campo de detección de *malware* en *Android* es bastante estudiado, ya que esta es la plataforma más usada a nivel mundial para teléfonos inteligentes, por lo que también es la más atacada por los ciberdelincuentes que siempre están buscando diferentes formas para materializar sus ataques. La

Figura 5-1 muestra, técnicas de detección de *malware* para *Android* desarrolladas entre los años 2012 y 2018-1

Técnicas de detección de *malware* en Teléfonos inteligentes con Android entre el 2013 y el 2018-1									
Artículos	***Machine Learning***	**Análisis de *malware* dinámico en servidores**	**Análisis de *malware* Estático en servidores**	**Análisis de *malware* Estático y/o dinámico en móviles**	**Análisis semántico**	**Análisis Forense**	**Análisis basado medidas de similitud**	**Análisis remoto de aplicaciones en *Google Play***	**Análisis basado en llamados a la API**
(T. E. Wei et al., 2012) Android Malware Detection via a Latent Network Behavior Analysis							X		
(Y. Zhang et al., 2013) Vetting undesirable behaviors in android apps with permission use analysis						X			
(Teufl et al., 2006) Malware detection by applying knowledge discovery processes to application metadata on the Android Market (Google Play)	X							X	

Figura 5-1. Fragmento de la Tabla 8-1 investigación del Estado del Arte de técnicas de detección de *malware* en *Android* para el periodo 2013-2018-1. (Fuente: Propia.)

También en el **Anexo A,** en la **Tabla 8-2**, se muestran ocho debilidades de las técnicas de detección en teléfonos inteligentes. Se pueden ver las debilidades de las técnicas de detección expuestas en la tabla anterior, las cuales pueden ser utilizadas por los atacantes para comprometer los dispositivos móviles. La *Figura 5-2* muestra un fragmento de la **Tabla 8-2**.

Debilidades de las Técnicas de detección de malware en Teléfonos inteligentes con Android entre el 2013 y el 2018-1								
Artículos	**Requiere por lo menos de una muestra de malware para generar una detección**	**Requieren de conocimientos avanzados**	**Se requiere de un administrador**	**No se realiza detección en tiempo real**	**El análisis se hace entornos ficticios**	**Depende de grandes cantidades de información e infraestructura robusta**	**Solamente funciona con aplicaciones de la tienda de Google (Google Play)**	**No incluye análisis dinámico**
(T. E. Wei et al., 2012) Android Malware Detection via a Latent Network Behavior Analysis	X							
(Y. Zhang et al., 2013) Vetting undesirable behaviors in android apps with permission use analysis		X		X				
(Teufl Peter, Ferk Michaela, Fitzek Andreas, Hein Daniel, Kraxberger Stefan, 2014) Malware detection by applying knowledge discovery processes to application metadata on the Android Market (Google Play)	X						X	

Figura 5-2. Fragmento de la **Tabla 8-2**, investigación del Estado del Arte de debilidades de las técnicas de *malware* en *Android* para el 2013-2018-1. (Fuente: Propia).

5.2 Analizar y seleccionar herramientas para monitorear *IOCs* en *Smartphone* con sistema operativo *Android*

En esta sección se analizan y seleccionan los componentes del entorno de pruebas para el desarrollo de la plataforma *CAM*: Dispositivos móviles, *software* de emulación, aplicaciones entre otros

5.2.1 Selección de dispositivos móviles o *software* de emulación para la implementación del laboratorio de pruebas

Se buscaron estudios estadísticos de los dispositivos móviles más usados, las versiones del sistema operativo *Android* más usadas, las aplicaciones más populares y se compararon tres softwares de emulación

Dispositivos móviles *Android* más usados

Según (Delloite, 2017) en el reporte de consumo móvil en Colombia las 3 marcas de dispositivos móviles con sistema operativo *Android* más usados en el año 2015 y 2016 son: 1. *Samsung*, 2. *Huawei* y 3. *Motorola*, como se muestra en la *Figura 5-3.*

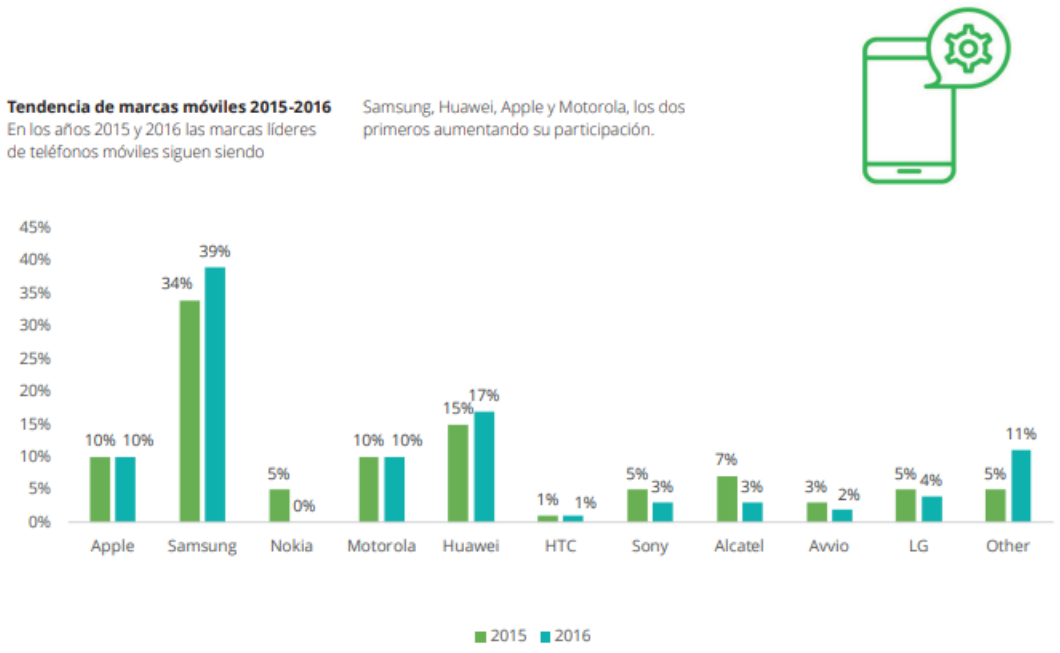


Figura 5-3. Tendencia de marcas móviles 2015-2016. (Fuente: Delloite, 2017).

Versiones del sistema operativo *Android* más usados

Según *Android Studio*, *software* de desarrollo para el sistema operativo *Android* las 3 versiones más usadas son: 4, 5 y 6, como se en la *Figura 5-4*, por lo que en esta investigación se implementarán las tres versiones más usadas, aunque se harán pruebas de compatibilidad de aplicación desarrollada con las versiones 7 y 8 del sistema operativo. (*Google*, 2018, p2).

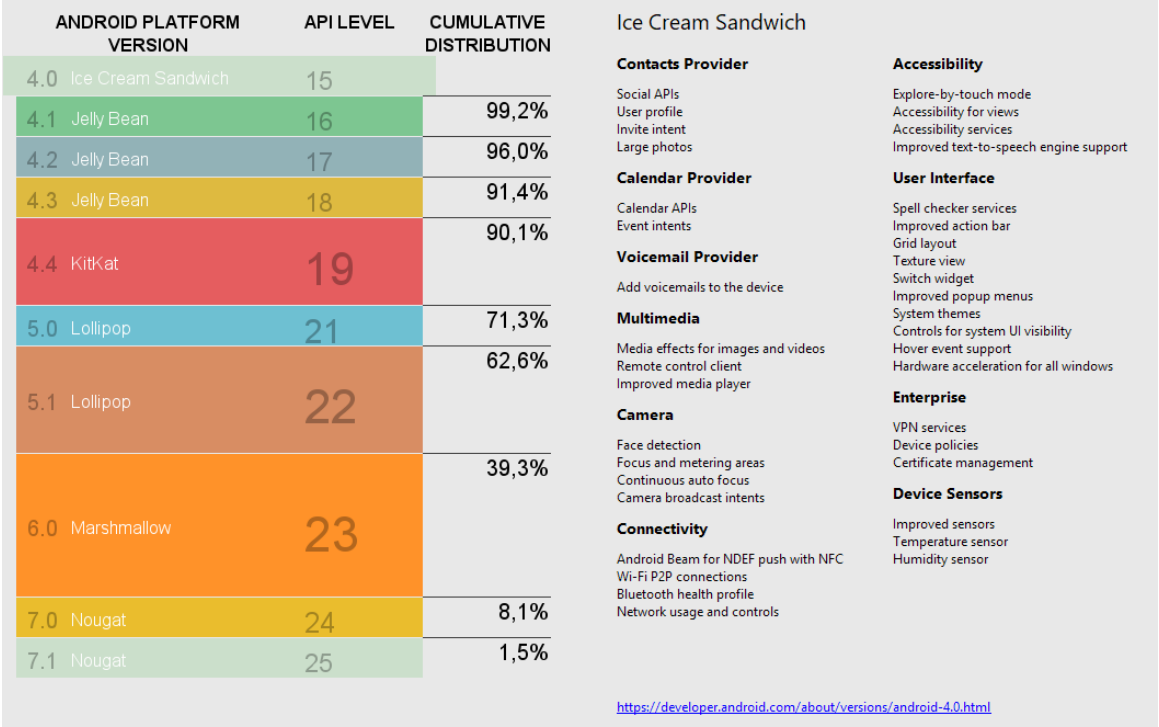


Figura 5-4. Plataforma *Android*/Distribución de la Versión *API*. (Fuente: *Android* Studio).

Emuladores de dispositivos móviles *Android*

En este trabajo se buscaron emuladores de aplicaciones para dispositivos móviles *Android* y se encontró que las plataformas más usadas son *Genymotion, AVD* y *BlueStacks*. Además, se buscó información que contrastará estas plataformas para seleccionar la más adecuada para el entorno del laboratorio de análisis de *malware.*

(Gupta, 2017) realizó pruebas comparando las plataformas *AVD* y *Genymotion* tomando como criterio la eficiencia de cada uno en el momento de realizar la emulación de máquinas virtuales con sistema operativo *Android*.

En el informe sé evidencia que *AVD* posee un rendimiento mayor cuando se corre en sistemas con procesadores basados en arquitectura de 64bits, pero comprueba que *Genymotion* es mucho más eficiente al ejecutarlo en máquinas con procesadores multi-núcleo, por lo que usaremos *Genymotion* ya que la máquina en la que se implementó el entorno de laboratorio es una maquina *Asus* multi-núcleo, con procesador *Intel® Core i5*, con una memoria *RAM* de 8GB y para el almacenamiento, una unidad de estado sólido de 240GB. La *Figura 5-5* muestra el rendimiento de los emuladores por arquitectura utilizando el *software Geekbench 4.*

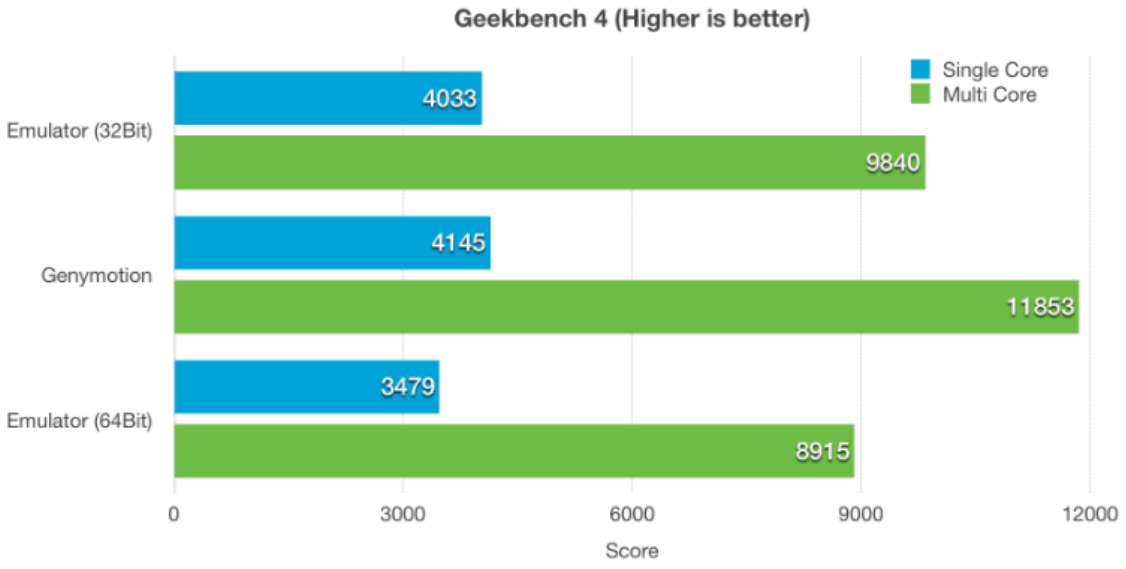


Figura 5-5. Comparación del rendimiento de los emuladores para *Android* usando *Geekbench* 4 (Más alto es mejor). (Fuente: Gupta, 2017).

(Kearney, n.d.) Otra referencia de estudio se encuentra en la plataforma de *rankings online Slant* donde se realiza una comparativa entre el uso de *Genymotion* y *BlueStacks* encontrándose que *Genymotion* es la mejor opción en cuanto a emulación de máquinas virtuales con sistema operativo *Android* debido a que esta utiliza las librerías graficas de libre uso *OpenGL* permitiéndole tener un mayor rendimiento. Esta característica solo es comparable con el asistente de ejecución de *hardware* acelerado de *Intel* (*HAXM*) que posee *AVD*.

Además, se debe tener en cuenta que un requisito importante al trabajar en el análisis de *malware* es la capacidad de crear imágenes de estados anteriores de las máquinas de pruebas (*snapshots*) con el fin de poder restaurar las maquinas a un estado anterior luego de hacer pruebas con *malware* real.

Como se ha descrito en esta sección, es claro que los dispositivos móviles más usados son: *Samsung, Huawei* y *Motorola* y las versiones del sistema operativo *Android* más usadas son la 4, 5 y 6. Finalmente la plataforma de emulación con más ventajas y adecuada para el laboratorio es *Genymotion* ya que brinda mayor rendimiento y permite generar *snapshots* del dispositivo móvil emulado los cuales son fundamentales para el análisis de *malware.*

Selección de aplicaciones para infectar

Según la tienda de aplicaciones del sistema operativo *Android* (*Google Play*) las principales aplicaciones son las que se describen en la *Figura 5-6.*

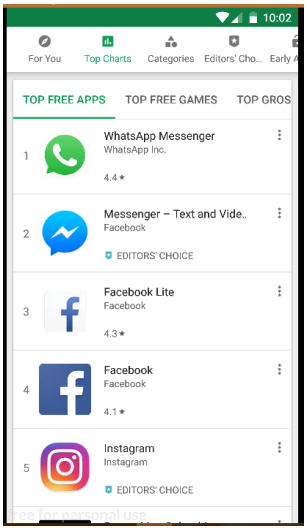


Figura 5-6. Top 5 de aplicaciones más populares de la *Play Store*. (Fuente: Propia).

Se seleccionó la aplicación ***WhatsApp* versión 2.18.223** y se infectó con *Meterpreter* para *Android* de *Metasploit* la *Figura 5-7* la aplicación de *WhatsApp* instalada en el emulador para hacer pruebas.

Figura 5-7. Versión de *WhatsApp* Messenger instalada para realizar las pruebas. (Fuente: Propia)

También se seleccionó e infectó la aplicación **Fábrica Dulce** versión 8.25.10.00 *Figura 5-8* para realizar pruebas de infección con una aplicación no tan popular, por lo que los fabricantes no la actualizan tan frecuentemente, reduciendo el impacto de las actualizaciones en las pruebas de laboratorio.

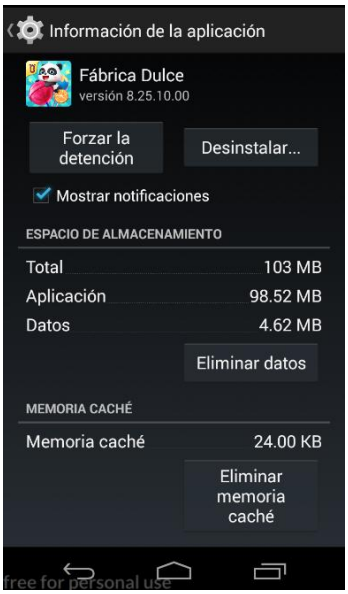


Figura 5-8. Información Fábrica de Dulce. (Fuente: Propia).

Finalmente, la *Figura 5-9* muestra el laboratorio que se implemento

Dispositivos móviles emulados en *Genymotion*

- Motorola *MotoX* con *Android* 4.4.4 *API* 19, que llamaremos: Dispositivo1
- Samsung *Galaxy* S6 con *Android* 5.1.0 *API* 22 que llamaremos: Dispositivo2
- Samsung *Galaxy* S6 con *Android* 6.0.0 *API* 23 que llamaremos: Dispositivo3

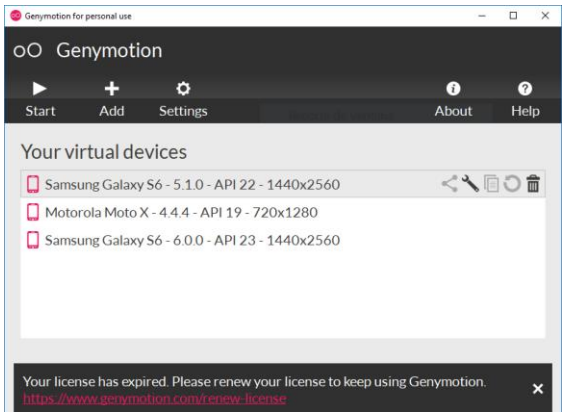


Figura 5-9. Interfaz de *Genymotion* con los teléfonos de prueba. (Fuente: Propia).

Dispositivo móvil

Se seleccionó el teléfono inteligente *Motorola MotoG 5 Plus* con *Android* 7 y parche de seguridad 1 de junio de 2018, que llamaremos: Dispositivo4 como se ilustra en la *Figura 5-10*

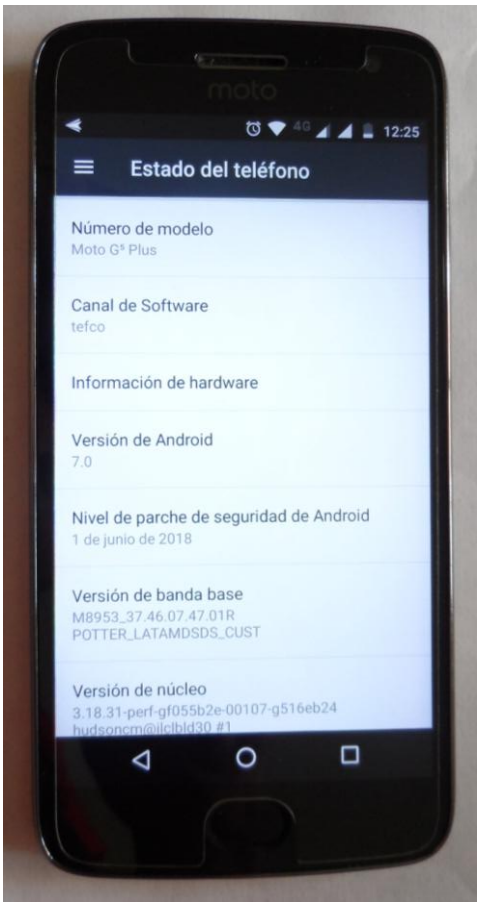


Figura 5-10. Dispositivo móvil físico *Moto G5 Plus.* (Fuente: Propia).

5.2.2 Selección de aplicaciones para detectar *IOCs* en *Android*

Para este estudio se infectaron dos de las tres máquinas emuladas con la aplicación infectada de ***WhatsApp*** y el teléfono físico con ***Fabrica DuIce.*** Se buscaron aplicaciones que pudieran monitorear, identificar y presentar los indicadores de compromiso de las aplicaciones comprometidas, las aplicaciones que se utilizaron para las pruebas de detección fueron las siguientes:

Aplicaciones para determinar *IOCs* estáticos, como son, permisos de las aplicaciones y del sistema: *Package Info* y *RL Permision.*

Package Info

Los detalles de la aplicación *Pakage Info* se ven en la *Figura 5-11.*

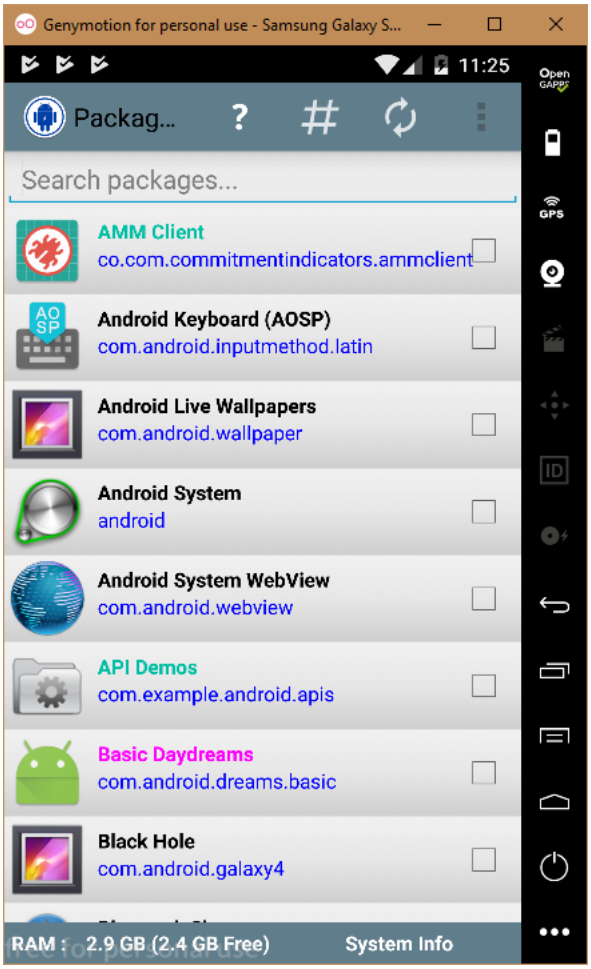


Figura 5-11. Pantalla principal de *Package Info*. (Fuente: Propia).

WhatsApp Infectado escaneo con *Package Info*

El resultado del escaneo de los permisos de la aplicación *WhatsApp* con *Package Info* se ve en la *Figura 5-12.*

Package Info

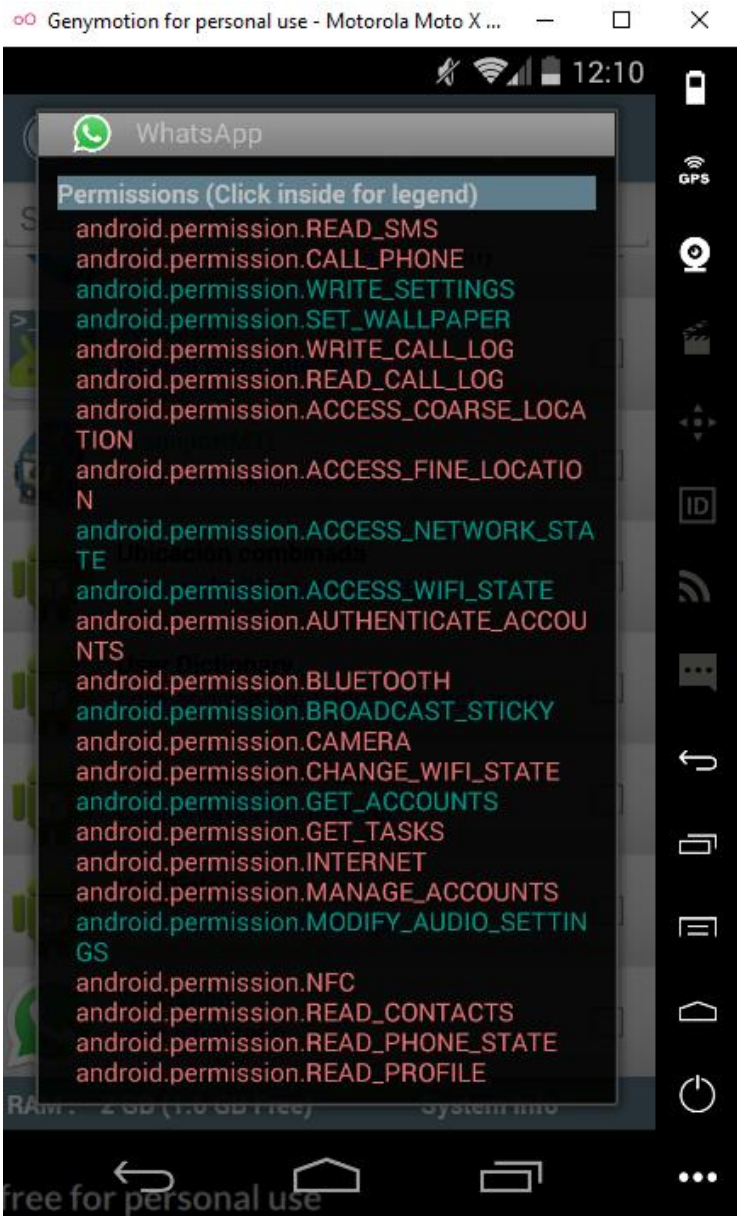


Figura 5-12. Permisos de *WhatsApp* infectado con *Package Info*. (Fuente: Propia).

WhatsApp infectado escaneado con *RL Permissions*

El resultado del escaneo de los permisos de la aplicación *WhatsApp* con *RL Permission* se ve en la *Figura 5-13.*

RL Permissions

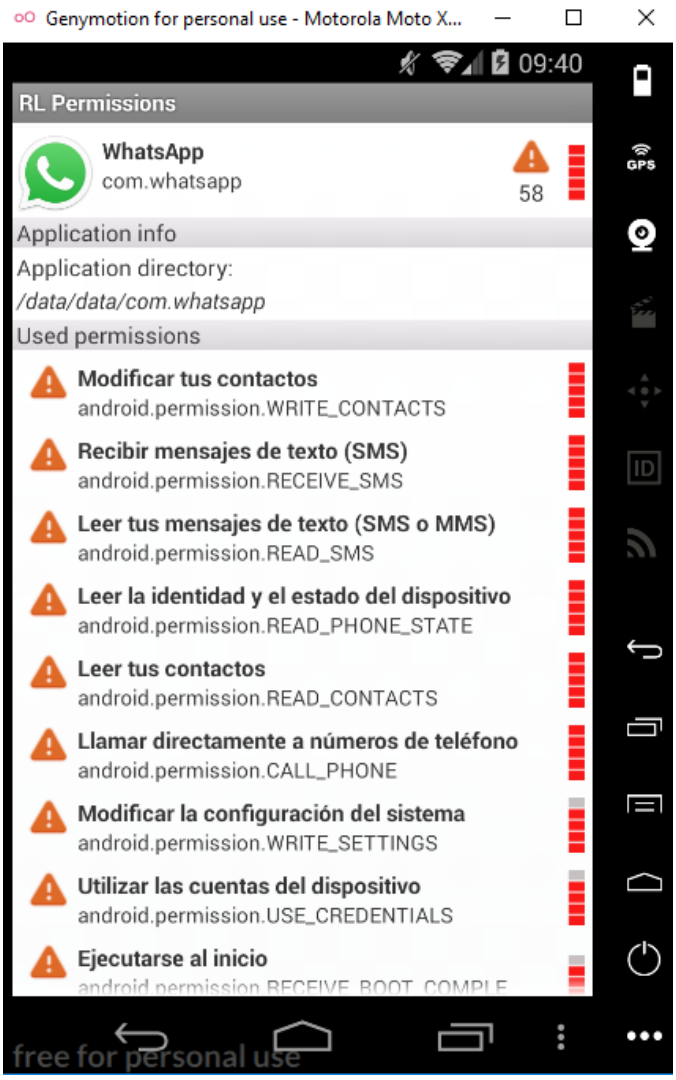


Figura 5-13. Permisos de *WhatsApp* Infectado en *RL* Permission. (Fuente: Propia).

Resultado: La aplicación infectada de *WhatsApp* tiene 58 Permisos

Aplicaciones para determinar *IOCs* dinámicos, tráfico de red por aplicación: *Network Connections* y *OS Monitor.*

WhatsApp infectado escaneado con *Network Connections*

La *Figura 5-14* muestra el resultado del escaneo de las conexiones de la aplicación *WhatsApp* con *Network Connections.*

Network Connections

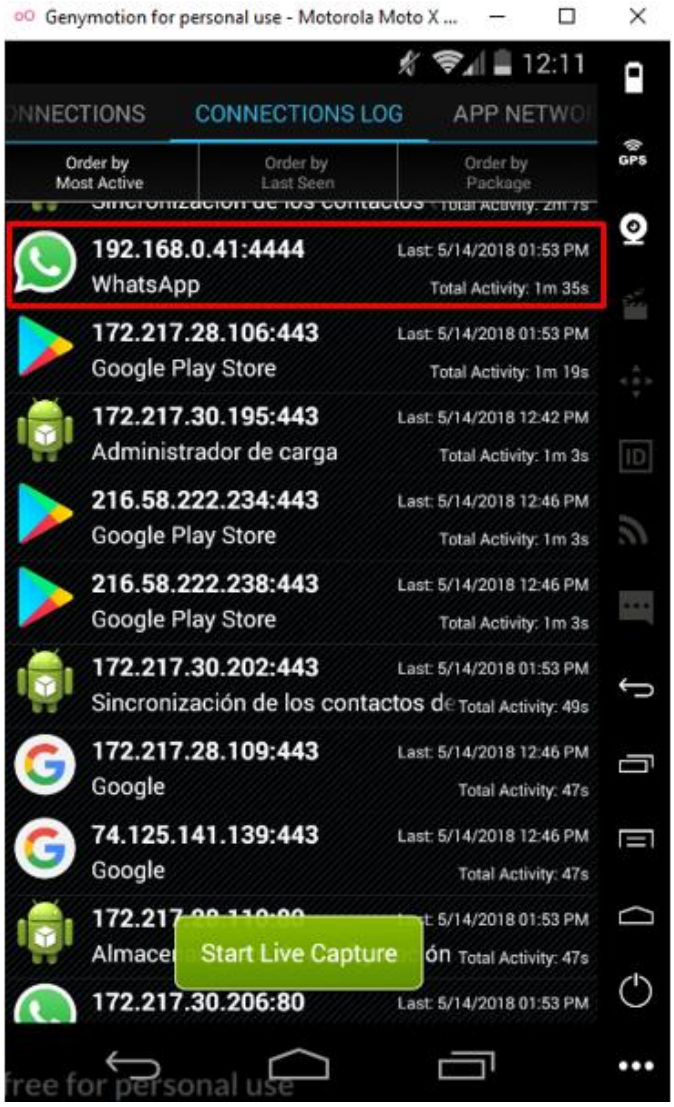


Figura 5-14. Captura de pantalla de análisis de tráfico de *Network Connections*. (Fuente: Propia).

Resultado: *WhatsApp* está solicitando conexiones a la dirección *IP* versión 4 privada 192.168.0.41 por el puerto 4444. Estas conexiones no corresponden a los dominios del fabricante.
La *Figura 5-15* muestra el detalle del resultado del escaneo de las conexiones de la aplicación *WhatsApp* con *Network Connections.*

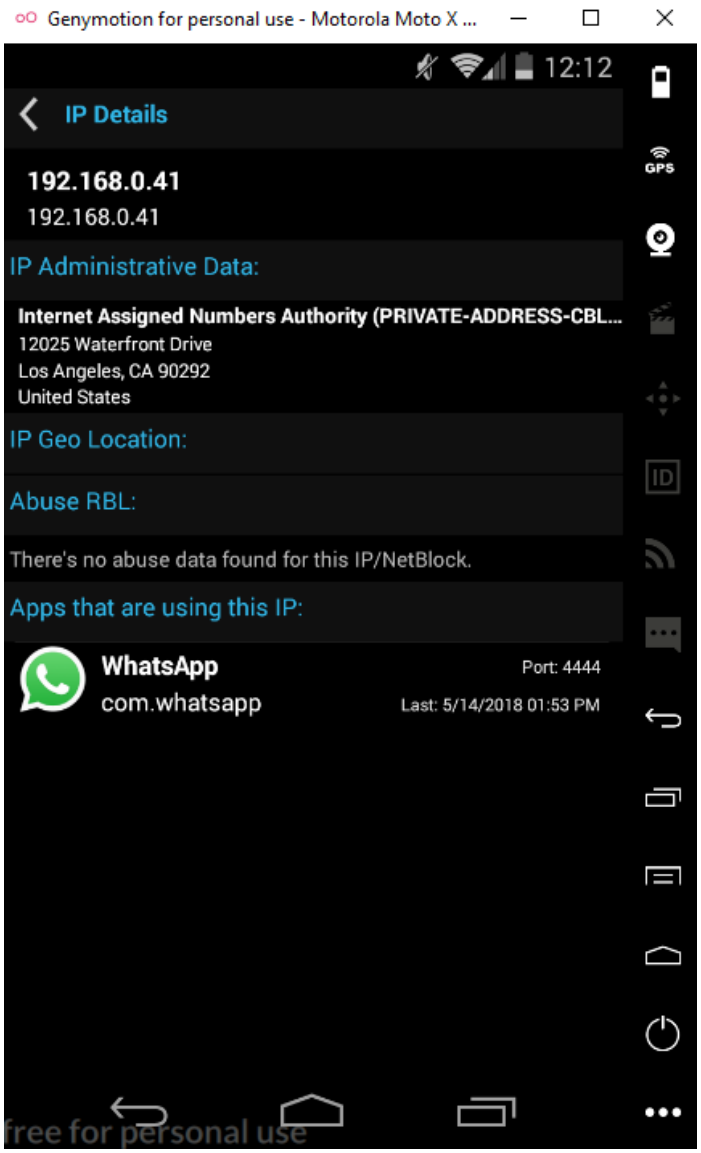


Figura 5-15. Dirección *IP* del centro de comando y control detectada por *Network Connections*. (Fuente: Propia).

Resultado: *Network Connections* detecta la *IP* 192.168.0.41 que es la *IP* de pruebas del comando y control.

WhatsApp* infectado escaneado con *OS Monitor

La *Figura 5-16* muestra el resultado del escaneo de las conexiones de la aplicación *WhatsApp* con *OS Monitor.*

OS Monitor

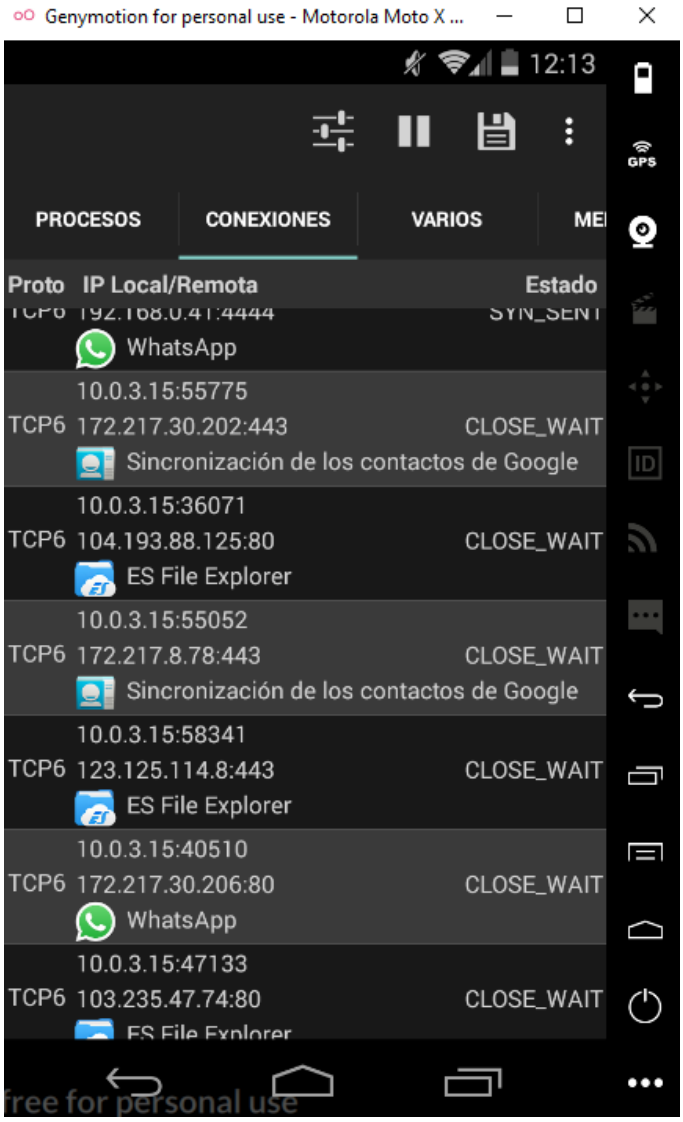


Figura 5-16. OS Monitor Escaneando el Tráfico incluido *WhatsApp* Messenger Infectado. (Fuente: Propia).

Resultados: se detecta una *IP* privada que no pertenece al dominio de *WhatsApp* y una *IP* pública.

Aplicaciones para determinar *IOCs* Metadatos, como son: firmas digitales *SHA1, MD5* y el tamaño de la aplicación: *Package Info*

La *Figura 5-17* muestra el resultado del escaneo de los metadatos de la aplicación *WhatsApp* con *Package Info*

Escaneando *WhatsApp* Infectado con ***Package Info***

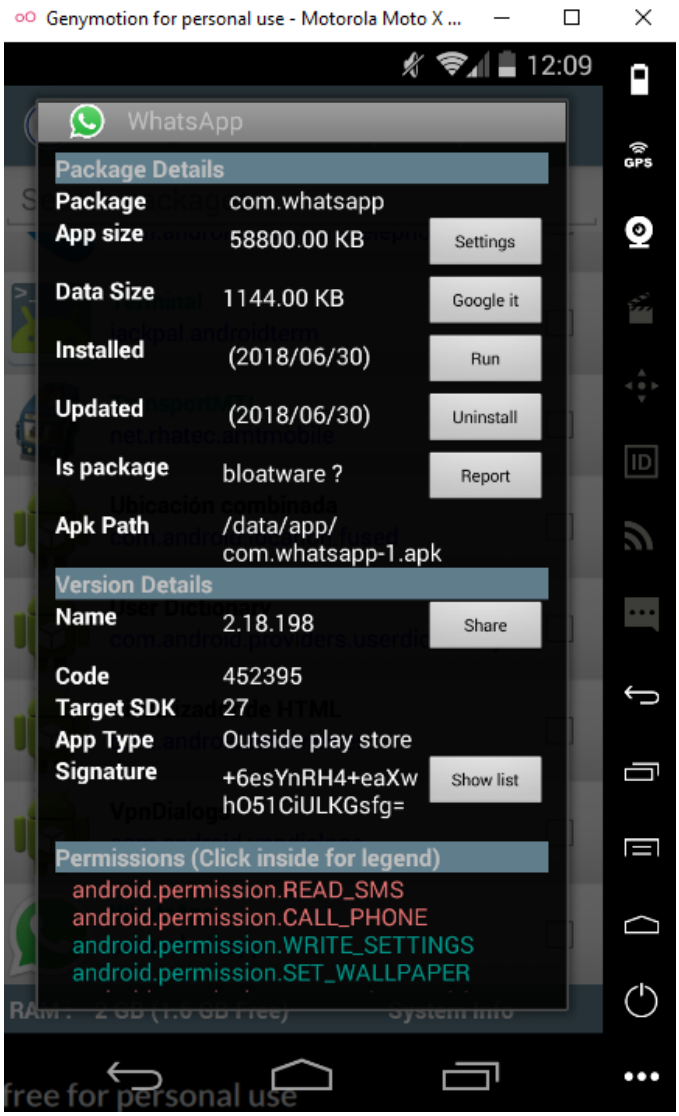


Figura 5-17. *Package Info* detectando la firma de *WhatsApp* Infectado. (Fuente: Propia).

Resultado: ***Package Info*** detecta la firma del desarrollador.

De estas cuatro herramientas se seleccionaron dos para trabajar en el proyecto debido a sus ventajas y desventajas:

Para monitorear *IOCs* estáticos y metadatos se seleccionó *Package Info* ya que tiene la capacidad de monitorear ambos *IOCs*, además *RL Permision* no se encuentra actualmente en la *Play Store.*

Para monitorear el tráfico de red por aplicación, se seleccionó *Network Connections*, por tener mayor compatibilidad con más versiones de *Android* ya que *Os Monitor* solo es compatible con las versiones *KitKa*t y *Lollipop* de *Android.*

5.2.3 Laboratorio seleccionado y uso

El objetivo de este laboratorio fue verificar la capacidad de detección de cinco softwares de seguridad y presentar las notificaciones de cada detección para determinar si pueden alertar y educar al usuario final de manera eficiente, los resultados del laboratorio están evidenciados en los capítulos de metodología, resultados y anexos.

- Dispositivo1: Se instaló las aplicaciones oficiales seleccionadas de la *Google Play* sin intervenir con *malware.* Además, la aplicación *Package Info* para presentar la *metadata* de cada aplicación de pruebas y *Network Connections* para monitorear el tráfico.
- Dispositivo2: Se instaló las aplicaciones oficiales seleccionadas de la *Google Play* e infectadas con *malware.* Además, la aplicación *Package Info* para presentar la *metadata* de cada aplicación de pruebas y *Network Connections* para monitorear el tráfico. A diferencia del Dispositivo1, en este se instaló cinco aplicaciones de *software* de seguridad de fabricantes reconocidos dos aplicaciones no pagas, dos aplicaciones pagas y por último el antivirus social *Koodous* para medir la capacidad de detección de cada uno de estos *softwares* y ver la eficiencia de sus alertas.
- Dispositivo3: Se instaló las mismas aplicaciones oficiales seleccionadas de la *Google Play* que el Dispositivo2 e infectadas con *malware*. También las mismas aplicaciones de monitoreo de *IOCs* metadatos, estáticos y dinámicos, pero solo tenía el *software* de seguridad desarrollado en este trabajo, la plataforma *CAM*, para presentar su capacidad de detección y notificaciones eficientes con el objetivo de alertar al usuario final.
- Dispositivo4: En este último se instaló la aplicación oficial de *WhatsApp* desde la *Google Play* y *Fabrica Dulce* infectada con *malware*. Se instaló las mismas aplicaciones de monitoreo de *IOCs,* metadatos, estáticos, dinámicos y el *software* de seguridad desarrollado en este trabajo la plataforma *CAM*, para presentar su capacidad de detección, funcionamiento en dispositivos físicos actuales y notificaciones eficientes con el objetivo de alertar al usuario final.

5.3 Desarrollo de la plataforma *CAM*

A partir del laboratorio seleccionado se desarrolla la plataforma *CAM* de la siguiente manera

5.3.1 Plataforma *CAM*

La plataforma *CAM* está compuesta de dos componentes: una consola de administración llamada *ACAM* (Administrador de Control de Aplicaciones Móviles) y un cliente o agente para dispositivo móviles llamado *CCAM* (Cliente de Control de Aplicaciones Móviles), que están desarrollados y trabajan de la siguiente manera:

Componentes de la plataforma *CAM*

- ***ACAM* Administrador de control de aplicaciones móviles**

ACAM es una plataforma web desarrollada en *Python Django* que se encarga de almacenar, administrar la base de datos de lista blanca de los Indicadores de compromiso de las principales aplicaciones oficiales para *Android* configuradas por el administrador y transferir las firmas de *IOCs* de las aplicaciones administradas. La *Figura 5-18* muestra el *main* de la interfaz de la consola *ACAM*

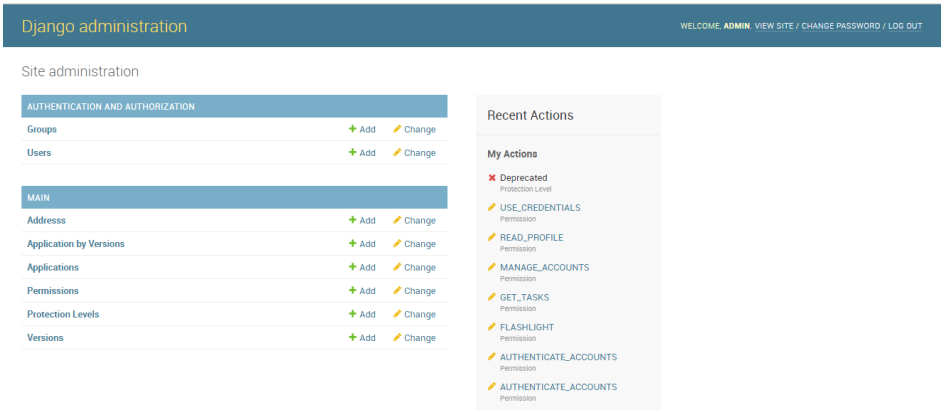


Figura 5-18*.* Plataforma *ACAM* basada en el *Framework* de Django. (Fuente: Propia).

Diagrama relacional de la Base de Datos de *ACAM*

La *Figura 5-19* muestra el modelo de la base de datos de la consola de administración *ACAM*

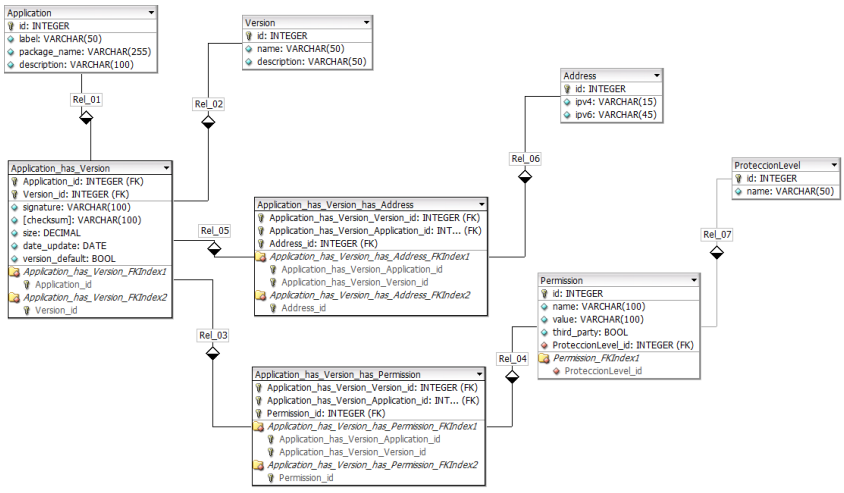


Figura 5-19. Diagrama relacional de la base de datos de la plataforma *CAM*. (Fuente: Propia).

La base de datos está diseñada por las siguientes tablas:

Proteccion_Level: en esta tabla se almacenan los niveles de protección con los que trabaja la aplicación. La *Figura 5-20* muestra cómo se ve la interfaz de nivel de protección de la consola *ACAM.*

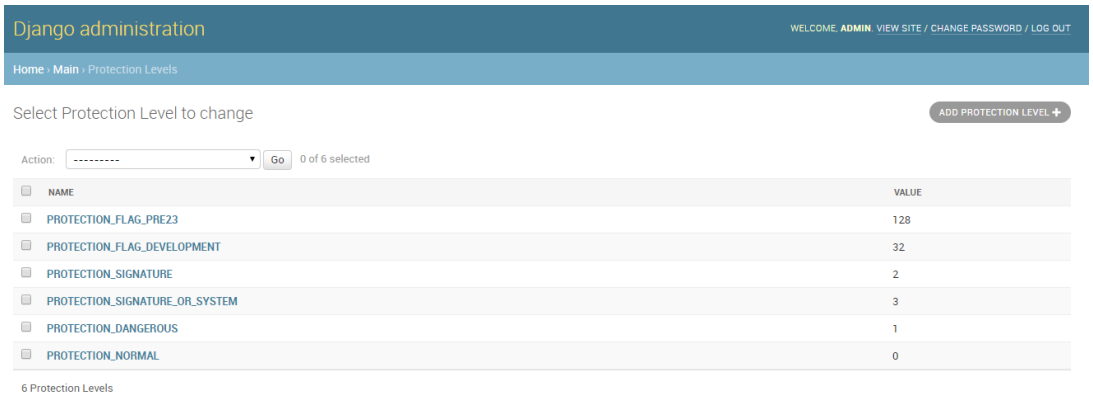


Figura 5-20. Campo del nivel de protección de permisos en *ACAM*. (Fuente: Propia).

En este desarrollo se seleccionaron 6 niveles de protección con base a la web de desarrolladores de *Android* y con un código de colores que se explica posteriormente en la página número 77:

- Permisos Peligrosos: La aplicación requiere datos o recursos que incluyen la información privada del usuario.

La *Figura 5-21* muestra el valor de protección peligros de la página del desarrollador de *Android.*

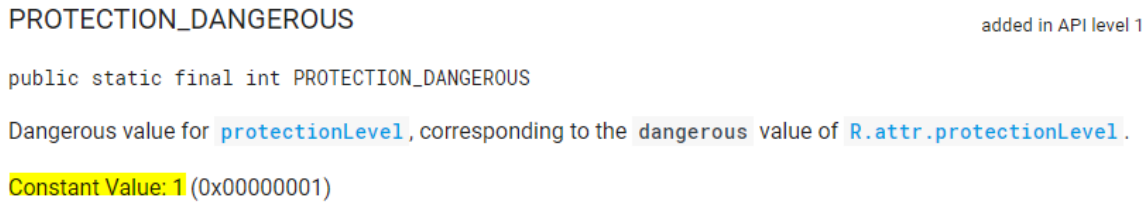


Figura 5-21. Nivel de protección peligrosa de los permisos de *Android*. (Fuente: *Android* Developers, 2018).

Normales: La aplicación tiene un riesgo mínimo para la privacidad del usuario o el funcionamiento de otras aplicaciones.

Fabricante: La aplicación tiene un permiso de tercero de riesgo moderado para la privacidad del usuario o el funcionamiento de otras aplicaciones.

La *Figura 5-22* muestra el valor de protección de fabricante de la página del desarrollador de *Android*.

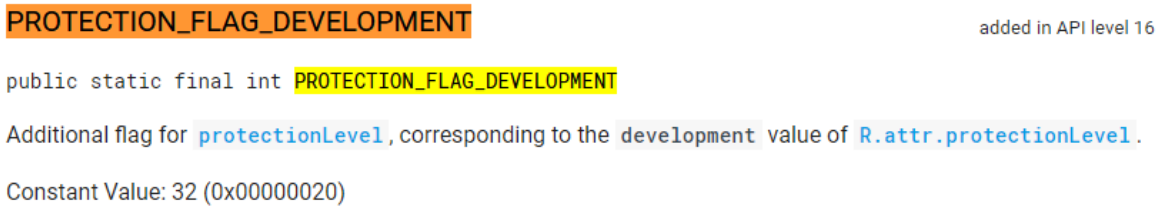


Figura 5-22. Nivel de protección del desarrollador de los permisos de *Android*. (Fuente: *Android* Developers, 2018).

- Firmas: La aplicación tiene un permiso de firma del fabricante, verificar que dicha firma no esté alterada.
- Firmas o sistema: La aplicación tiene un permiso de firma del fabricante, verificar que dicha firma no esté altera.
- Obsoleto: La aplicación tiene un permiso que se encuentra obsoleto, te recomendamos actualizarte a una versión reciente.

La *Figura 5-23* muestra el valor de protección de permiso obsoleto (Protection_Flag_23) de la página del desarrollador de *Android*.

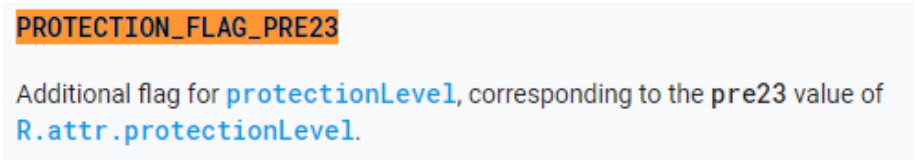


Figura 5-23. Nivel de protección obsoleto de los permisos de *Android*. (Fuente: *Android* Developers, 2018).

La *Figura* 5-24 muestra el resultado de la clasificación por color de los permisos de la aplicación *WhatsApp* con base a la clasificación de la página del desarrollador de *Android*.

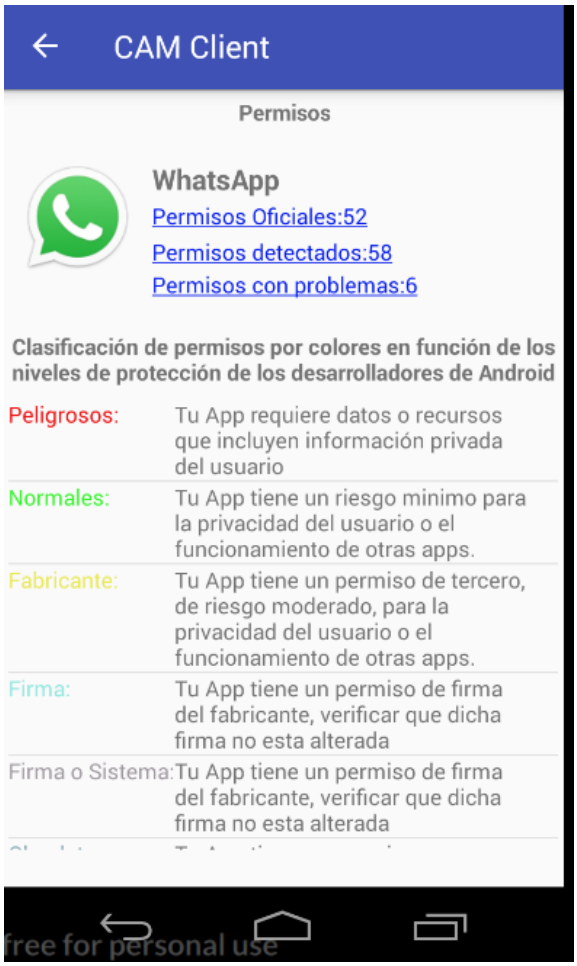


Figura 5-24. Clasificación de permisos *CAM* por colores. (Fuente: Propia).

Permission: en esta tabla se almacenan los permisos con los que trabajan las aplicaciones publicadas en *Google Play*.

La *Figura 5-25* muestra la interfaz de asignación de permisos de la consola *ACAM.*

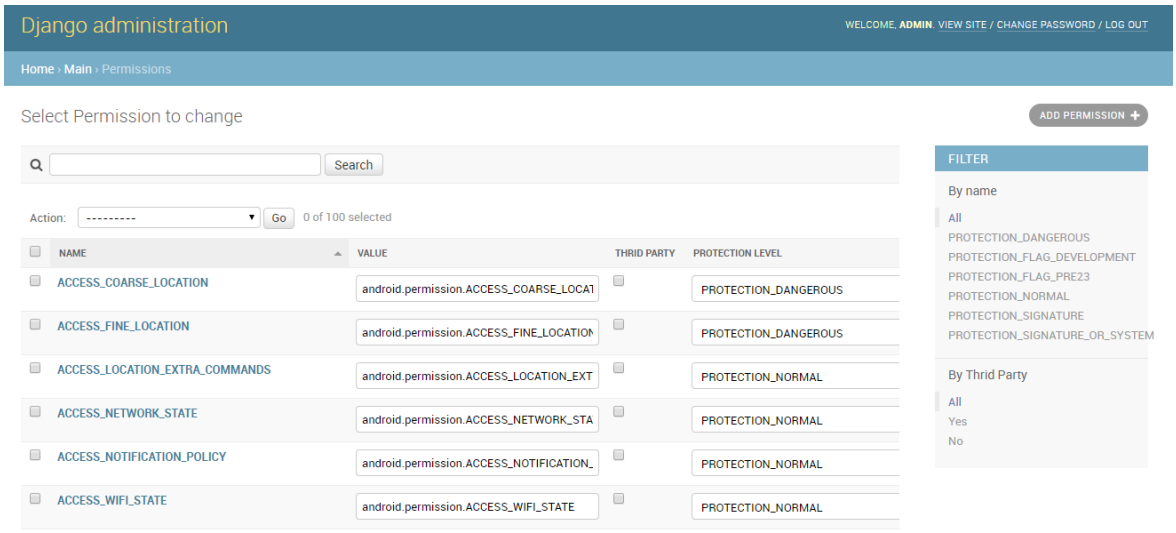


Figura 5-25. Campo de Asignación de Permisos en *ACAM*. (Fuente: Propia).

Application: en esta tabla se almacenan las aplicaciones que son administradas en el dispositivo móvil.

La *Figura 5-26* muestra la interfaz de configuración de aplicaciones de la consola *ACAM*

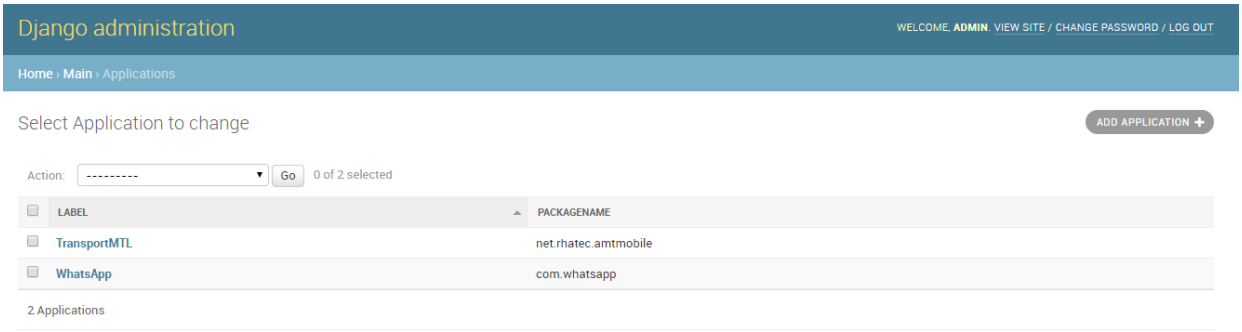


Figura 5-26. Campo de Aplicaciones en *ACAM*. (Fuente: Propia).

Versión: En esta tabla se almacenan las versiones de las diferentes aplicaciones que son administradas.

La *Figura 5-27* muestra la interfaz de configuración de versión de aplicaciones de la consola *ACAM.*

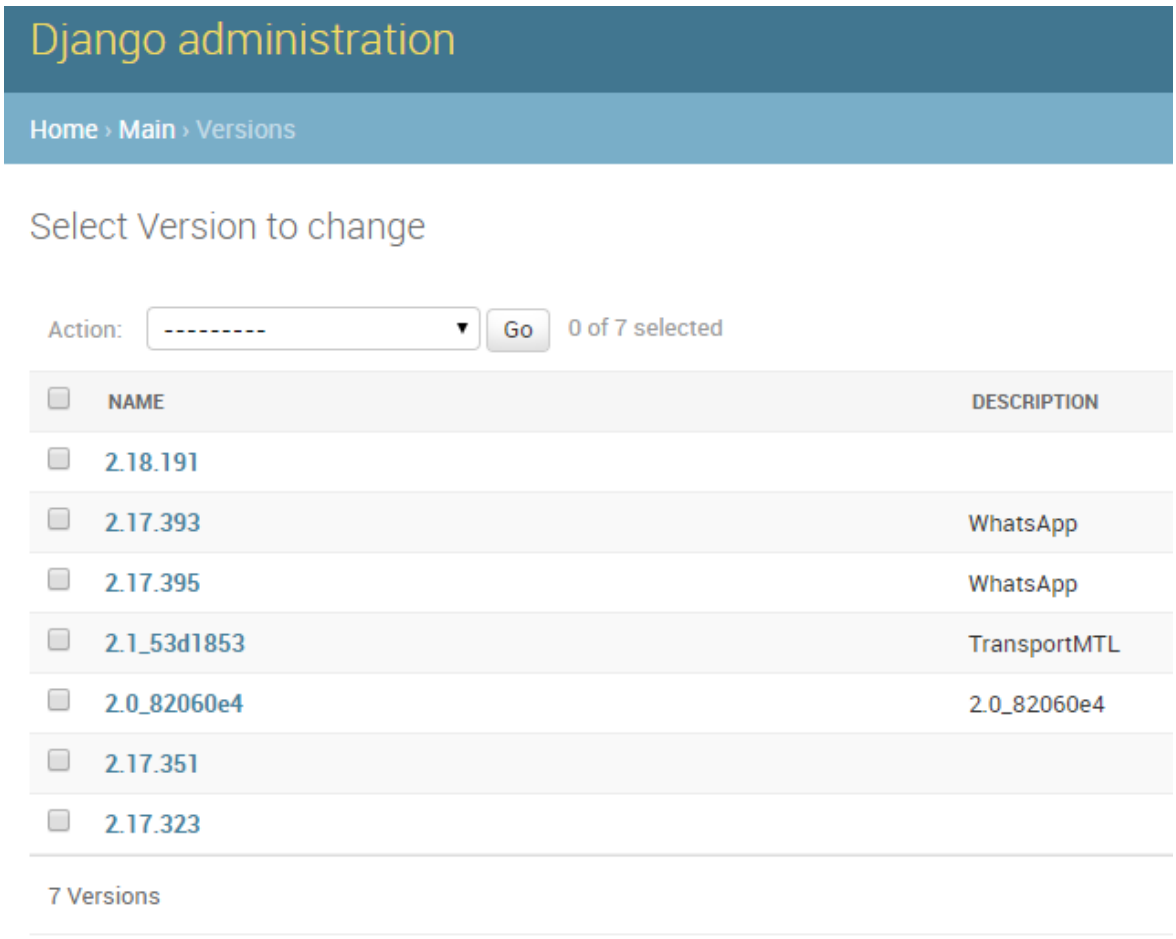


Figura 5-27. Campo de asignación de versión de *ACAM*. (Fuente: Propia).

Application_has_version: En esta tabla se almacenan las relaciones entre las aplicaciones y sus respectivas versiones.

La *Figura 5-28* muestra la interfaz de configuración de versiones de la consola *ACAM*.

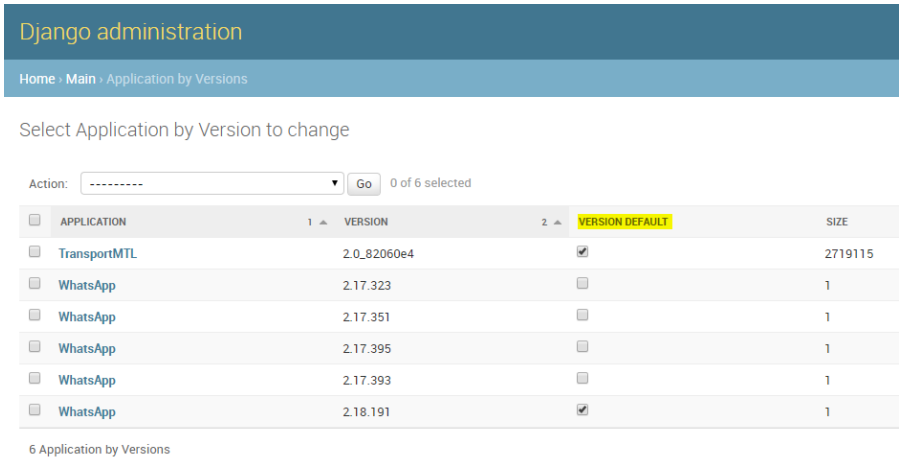


Figura 5-28. Campo de asignación de aplicación por versión de *ACAM.* (Fuente: Propia).

Application_has_version_has_permison: En esta tabla se almacenan las relaciones entre las aplicaciones por versión y sus respectivos permisos.

La *Figura 5-29* muestra la interfaz de configuración de aplicación por versión de la consola *ACAM*.

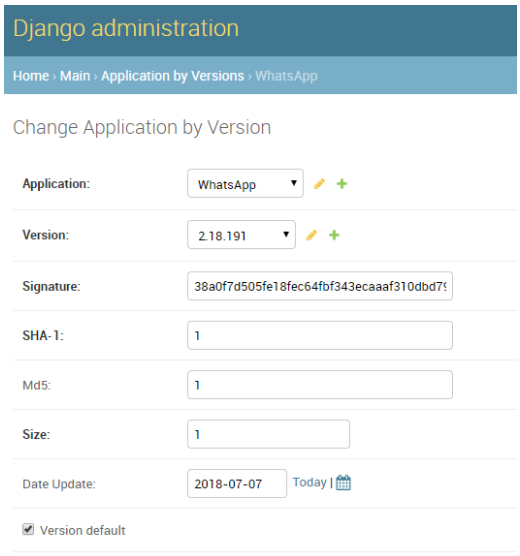


Figura 5-29. Parámetros de configuración de la aplicación por versión en *ACAM*. (Fuente: Propia).

La *Figura 5-30* muestra la interfaz de configuración de aplicación por versión por permisos de la consola *ACAM*.

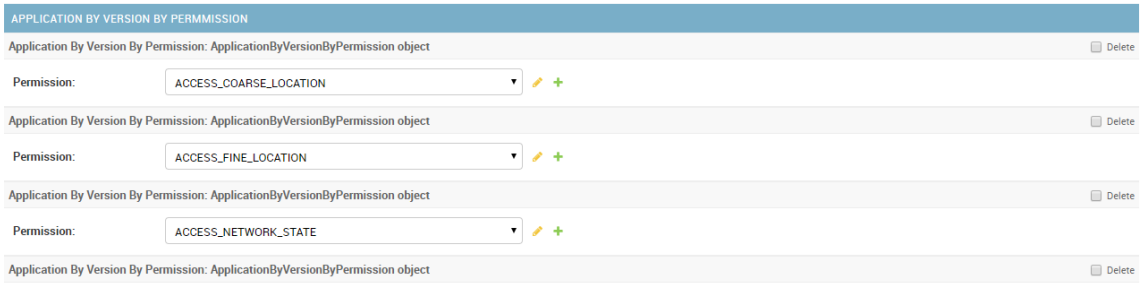


Figura 5-30. Configuración de permisos de la aplicación por versión por permisos de *ACAM*. (Fuente: Propia).

Address: En esta tabla se almacenan las direcciones IPV4 e IPV6 oficiales de cada aplicación.

La *Figura 5-31* muestra la interfaz de configuración de direcciones *IP* de la consola *ACAM*.

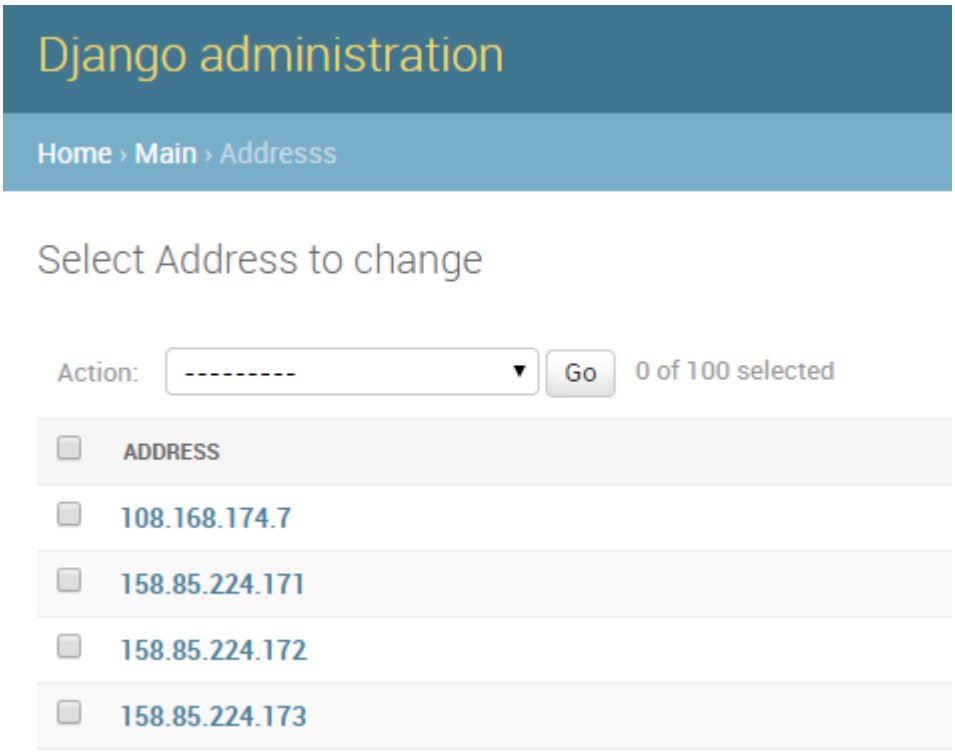


Figura 5-31. Configuración de las direcciones *IP* de *ACAM*. (Fuente: Propia).

Application_has_version_has_address: En esta tabla se almacenan las relaciones entre las aplicaciones por versión y sus respectivas direcciones.

La *Figura 5-32* muestra la interfaz de configuración de aplicación por versión por dirección de la consola *ACAM*.

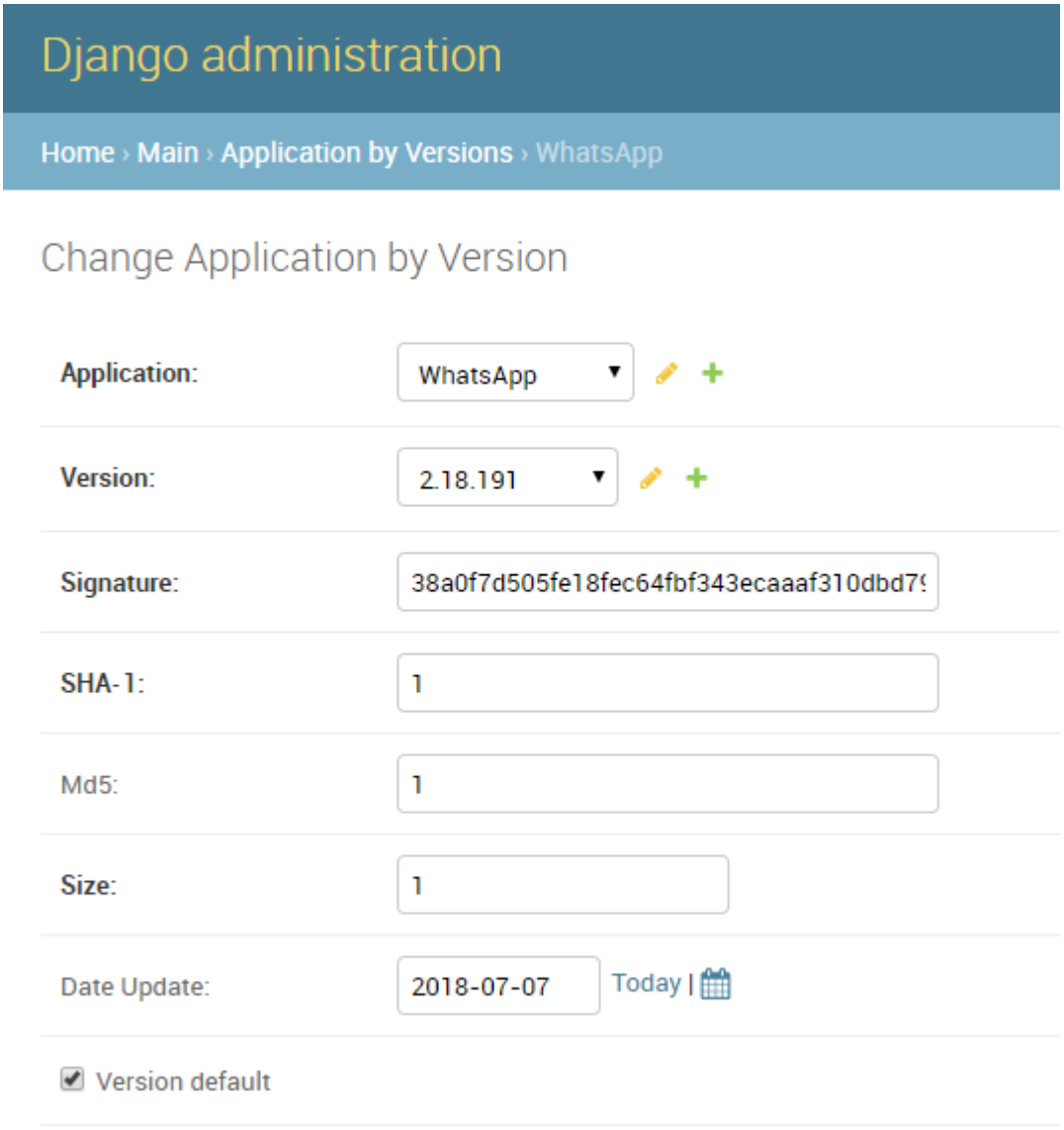


Figura 5-32. Configuración de los parámetros de la aplicación por versión en *ACAM*. (Fuente: Propia).

La *Figura 5-33* muestra la interfaz de configuración de direcciones *IP* a las aplicaciones por versión de la consola *ACAM*.

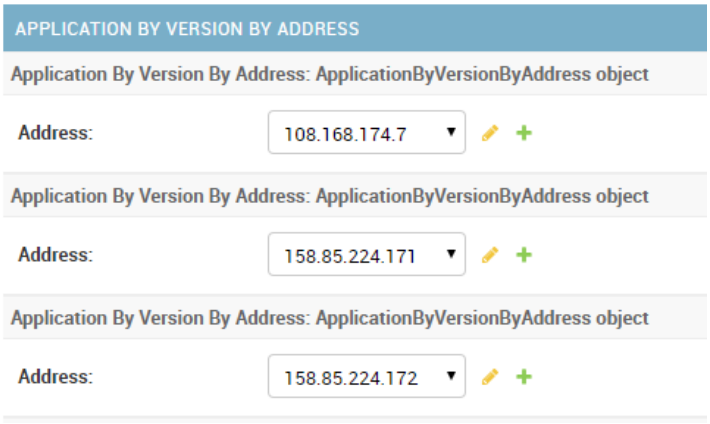


Figura 5-33. Asignación de las direcciones *IP* a las aplicaciones por versión. (Fuente: Propia).

5.3.2 Funciones de *ACAM*

Transferencia de firmas oficiales

Permite transferir las firmas de las aplicaciones oficiales. La *Figura 5-34* muestra el código que se encarga de comprimir las firmas de las aplicaciones oficiales y enviarlas desde la consola *ACAM* al cliente *CCAM*.

```
from django.shortcuts import render
from rest_framework import serializers, viewsets
from rest_framework.response import Response
from django.core.files import File
from django.http import HttpResponse
from django.conf import settings
# from io import StringIO
import io
import zipfile
import os

class DownloadFileSQLiteViewSet(viewsets.ViewSet):

    db_path = settings.BASE_DIR + '/bdci.sqlite3'

    def list(self, request):
        print ("Running list on DownloadFileSQLite...")
        compressFile,zip_filename = self._compressFile()
        compressFile.seek(0,os.SEEK_END)
        sizeFile = compressFile.tell()
        print ("Tamano del archivo:" + str(sizeFile))
        response = HttpResponse(compressFile.getvalue(), content_type = "application/x-zip-compressed")
        response['Content-Disposition'] = 'attachment; filename=%s' % zip_filename
        response['Content-Length'] = sizeFile
        return response

    def _compressFile(self):
        zip_subdir = "."
        zip_filename = "bdci.zip"
        s = io.BytesIO() #StringIO.StringIO()
        zf = zipfile.ZipFile(s, mode="w", compression=zipfile.ZIP_DEFLATED)
        fdir, fname = os.path.split(self.db_path)
        zip_path = os.path.join(zip_subdir, 'bdci')
        zf.write(self.db_path, zip_path)
        zf.close()
        return s,zip_filename
```

Figura 5-34. Compresión de la base de datos y envió de firmas a *ACAM*. (Fuente: Propia).

La *Figura 5-35* muestra la interfaz del código, encargada de comprimir y enviar la base de datos de firmas de las aplicaciones oficiales.

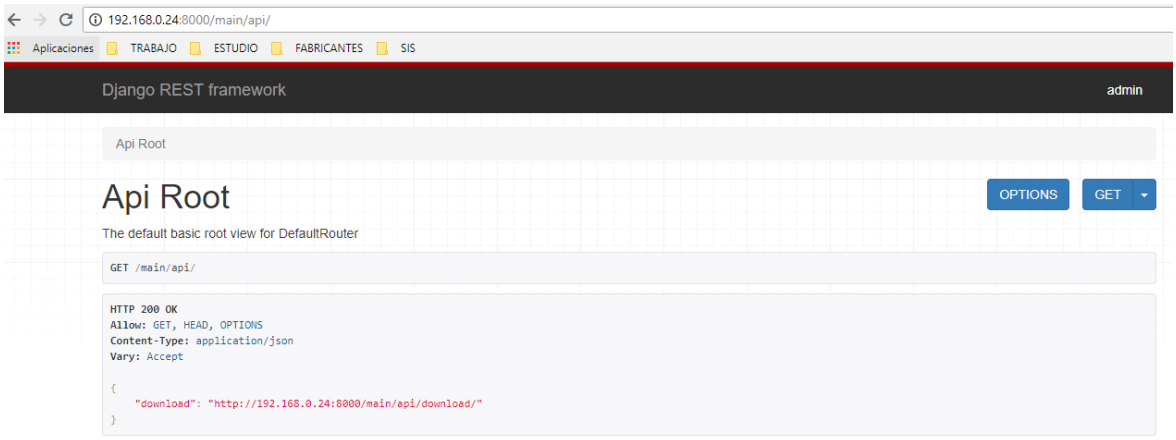


Figura 5-35. *API* de Django. (Fuente: Propia).

La *Figura 5-36* muestra el servicio de descarga en *Django Rest framework.* La imagen muestra la dirección web donde se encuentra el recurso *URL* para descargar la *BD*.

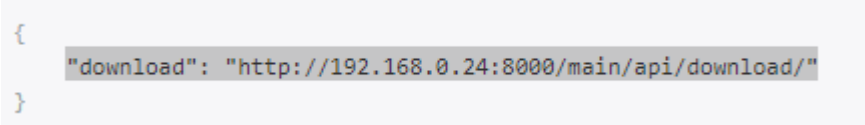


Figura 5-36*.* Dirección web del servicio de descarga de la base de datos. (Fuente: Propia).

Django rest-framework

Es un *framework* que permite desarrollar *APIs* Web http://www.django-rest-*framework*.org/

Selección de versión por defecto: En la tabla **Application_has_version** se permite seleccionar la versión por defecto que para casos en los que la aplicación instalada en el móvil sea diferente a la versión configurada en el administrador.

La *Figura 5-37.* Asignación de la versión por defecto de la aplicación por versión. Muestra la interfaz de selección de la versión por defecto de la aplicación, en este caso se selecciona la última versión configurada en la consola ACAM.

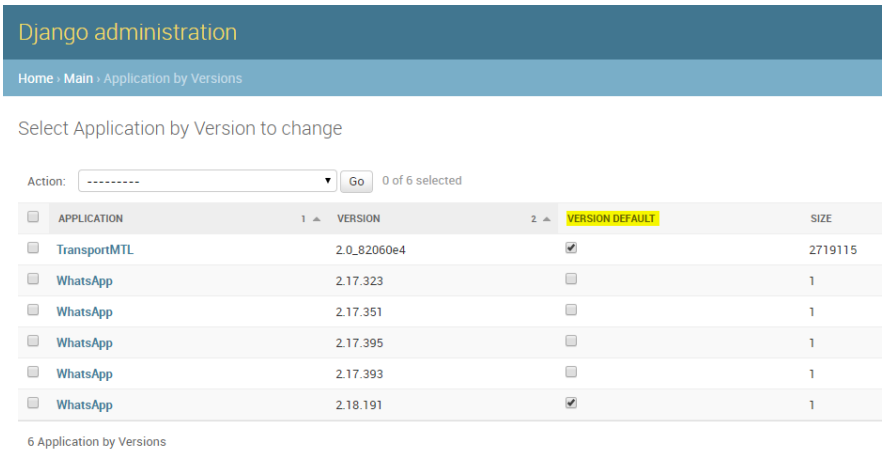


Figura 5-37. Asignación de la versión por defecto de la aplicación por versión. (Fuente: Propia).

Copiar Aplicaciones

Esta función permite copiar aplicaciones ya configuradas para simplificar la creación de nuevas firmas.

La *Figura 5-38* muestra la interfaz de la consola *ACAM* que se puede usar para copiar las aplicaciones configuradas y configurar versiones más recientes.

Figura 5-38. Copia de la aplicación por versión. (Fuente: Propia)

Pasos para habilitar la consola de administración *ACAM*

Abrir una consola de consola de comandos.

La *Figura 5-39* muestra la consola de comandos de Windows con la que se habilita la consola *ACAM*.

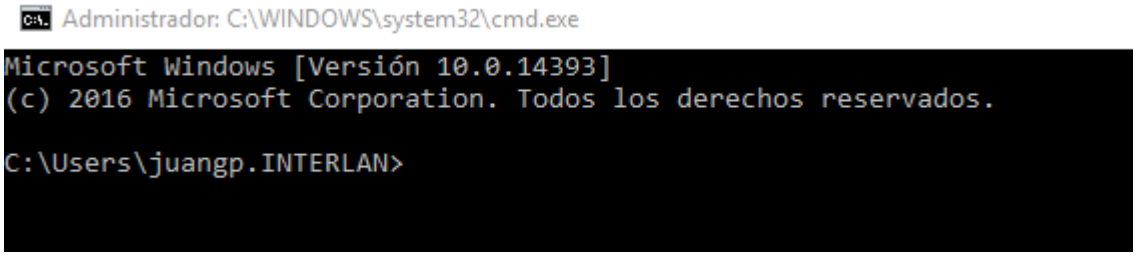


Figura 5-39. Ejecución de la Consola de Comandos de Windows. (Fuente: Propia).

En la ruta donde está el proyecto *ACAM*, se debe ejecutar el comando "**.\scripts\activate"** para activar el entorno virtual del proyecto y se muestra (*CCAM*) al inicio de la ruta, luego se vuelve a cambiar al directorio del proyecto con "**cd ruta del proyecto"** y se ejecuta el comando "**Python manage.py runserver 0.0.0.0:8000"**

La *Figura 5-40* muestra los comandos que se deben ejecutar para activar la consola *ACAM*

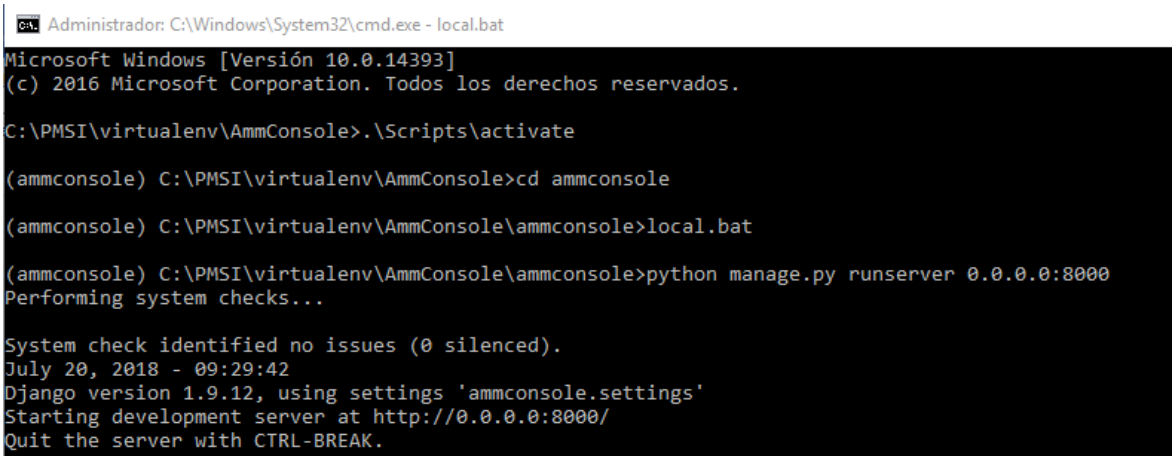


Figura 5-40. Ejecución de los comandos necesarios para iniciar *ACAM*. (Fuente: Propia).

La *Figura 5-41* muestra la interfaz web de la consola *ACAM*, *URL* 127.0.0.1:8000/admin

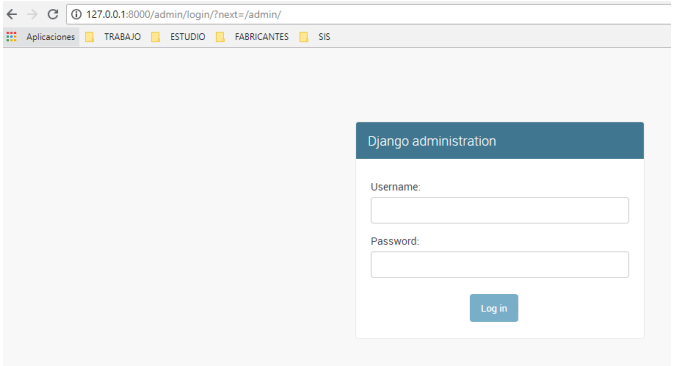


Figura 5-41. Interfaz de inicio de sesión de *ACAM***.** (Fuente: Propia).

5.3.3 *CCAM*

Es el cliente para los dispositivos móviles, que se encarga de descargar y/o almacenar las firmas de la base de datos del administrador *ACAM* de aplicaciones oficiales, escanea y contrasta tres indicadores de compromiso de las aplicaciones instaladas en el dispositivo: metadatos, estáticos y dinámicos, para detectar los eventos operativos maliciosos de las aplicaciones instaladas y genera reportes eficientes que muestran los eventos generados por el escaneo del *malware* permitiendo tomar acciones como: ver detalles de *IOCs*, ignorar, reportar por correo los incidentes detectados al administrador en un formato en regla *Yara* que se puede compartir con la comunidad de analistas de *malware* para *smartphone* con sistema operativo *Android* y desinstalar la aplicación.

CCAM utiliza el *framework Greendao* para realizar las consultas a la base de datos *sqlite*, se creó la clase *DatabaseService*, con las principales consultas: *getAppByVerWork*, g*etAppByVerAddressHashMap*, entre otras. La *Figura 5-42* muestra el código de la clase *DatabaseService.*

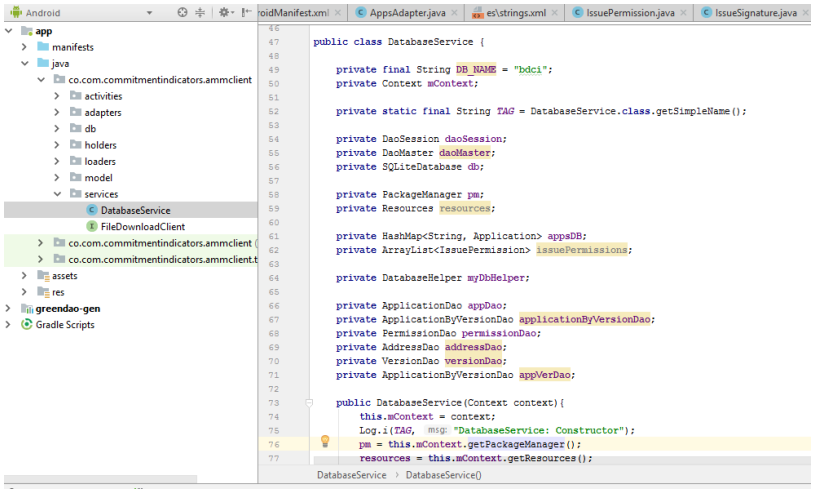


Figura 5-42. Clase DatabaseService. (Fuente: Propia).

Hilos de *CCAM*

Para la creación de los hilos se utilizó la clase *AsyncTaskLoader.* La *Figura 5-43* muestra el código de esta clase.

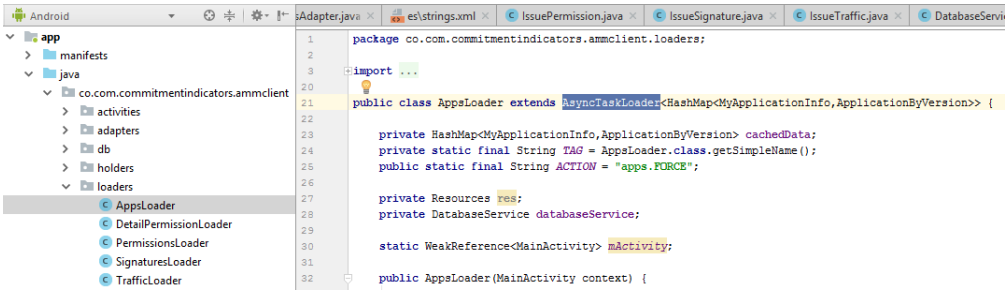


Figura 5-43. AsyncTaskLoader para la creación de hilos. (Fuente: Propia).

Función de los cuatro hilos

Escanear: ejecuta el escaneo de los cuatro hilos. La *Figura 5-44* ilustra el escaneo de los cuatro hilos del cliente *CCAM*.

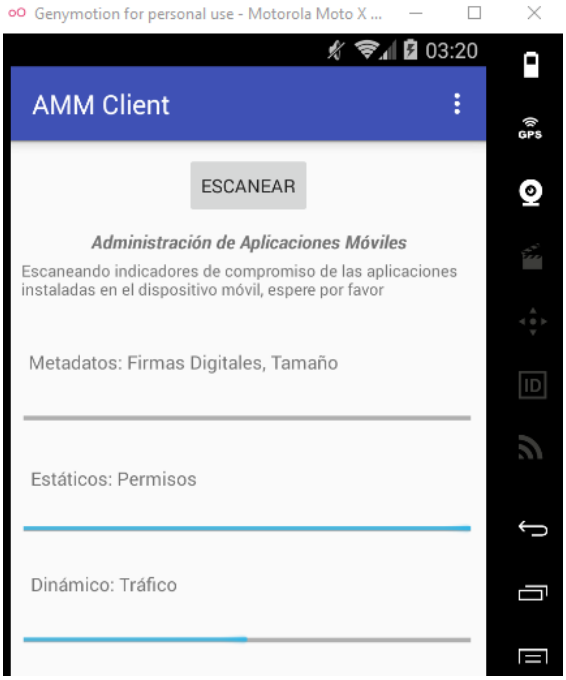


Figura 5-44. Ejecución de los cuatro hilos. (Fuente: Propia).

Descripción de los hilos

Hilo 1: Recupera las aplicaciones por defecto configuradas en la base de datos, para luego ser desplegadas en la actividad de lista de aplicaciones. La *Figura* **5-45** muestra el código de la función *loadInBackground.*

Figura 5-45. Función loadInBackground, donde se recuperan las aplicaciones por versión. (Fuente: Propia).

Hilo 2: Recupera los *IOCs* estáticos, permisos de las aplicaciones por defecto configuradas en la base de datos y las compara con los permisos de la aplicación instalada. Cuando se encuentran diferencias se crea un objeto *IssuePermissons* que contiene los permisos oficiales de aplicación y los permisos no encontrados en la aplicación instalada. La *Figura 5-46* muestra el código de la función *LoadInBackground* que crea el objeto *IssuePermissons*, donde se recuperan los permisos de la base de datos y luego se compara con los permisos por aplicación.

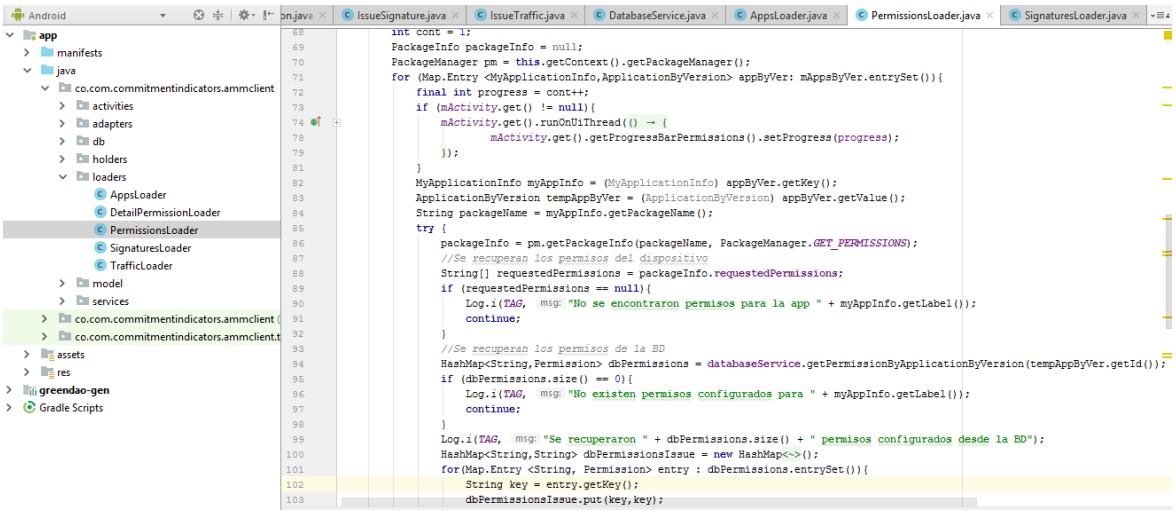


Figura 5-46. Función LoadInBackground que crea el objeto IssuePermissons. (Fuente: Propia).

Hilo 3: Recupera los *IOCs* metadatos: firmas digitales *SHA1, MD5* y tamaño de las aplicaciones por defecto configuradas en la base de datos y las compara con los metadatos de la aplicación instalada; cuando se encuentran diferencias se crea un objeto *IssueSignatures* que contiene los metadatos oficiales de aplicación y los metadatos no encontrados en la aplicación instalada.

La *Figura 5-47* muestra el código de la función *loadInBackground* que crea el objeto *IssueSignatures,* donde se recuperan las firmas y tamaño de la base de datos y luego se compara con las firmas y tamaño por aplicación.

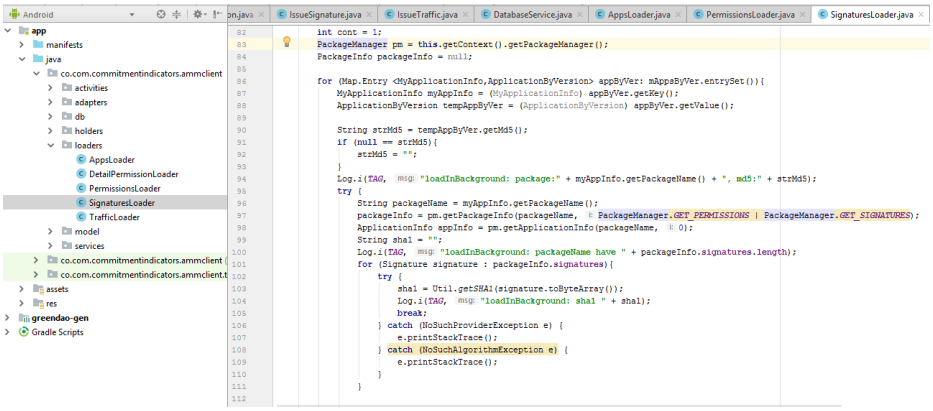


Figura 5-47. Función loadInBackground que crea el objeto IssueSignatures. (Fuente: Propia).

La *Figura 5-48* muestra el código que se encarga de comparar el valor de la firma *SHA-1* oficial configurada en la base de datos de la consola *ACAM* con la firma *SHA-1* detectada de la aplicación instalada en el teléfono inteligente.

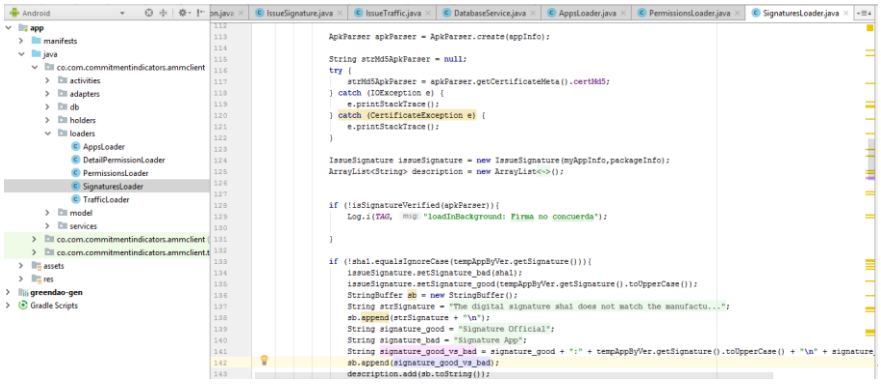


Figura 5-48. Comparación del *SHA-1.* (Fuente: Propia).

Hilo 4: Recupera los *IOCs* dinámicos: direcciones de red *IPv4* e *IPv6* de las aplicaciones configuradas en la base de datos y las compara con las direcciones solicitadas de la aplicación instalada; cuando se encuentran diferencias se crea un objeto *IssueTraffic* que contiene las direcciones oficiales de la aplicación y las direcciones no encontradas en la aplicación instalada.

La *Figura 5-49* muestra el código de la función *loadInBackground* que crea el objeto *IssueTraffic* donde se recuperan las direcciones *IPv4* e *IPv6* de la base de datos y luego se compara con las direcciones *IP* leídas del *log* de *Network Connections.*

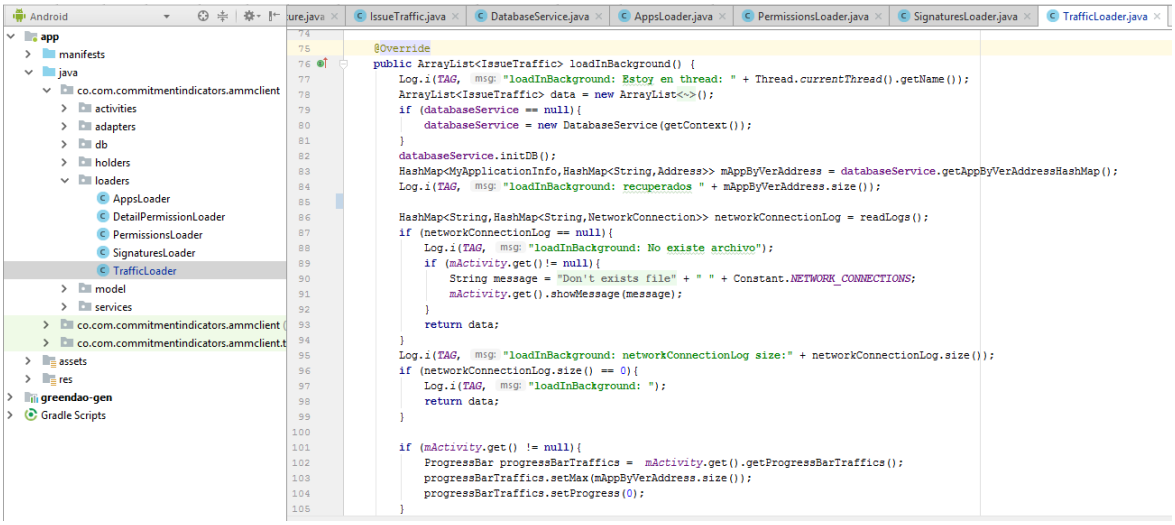


Figura 5-49. Función LoadInBackground que crea el objeto IssueTraffic. (Fuente: Propia).

muestra el código de la función *readlogs* que permite leer el *log* de *Network Connections.*

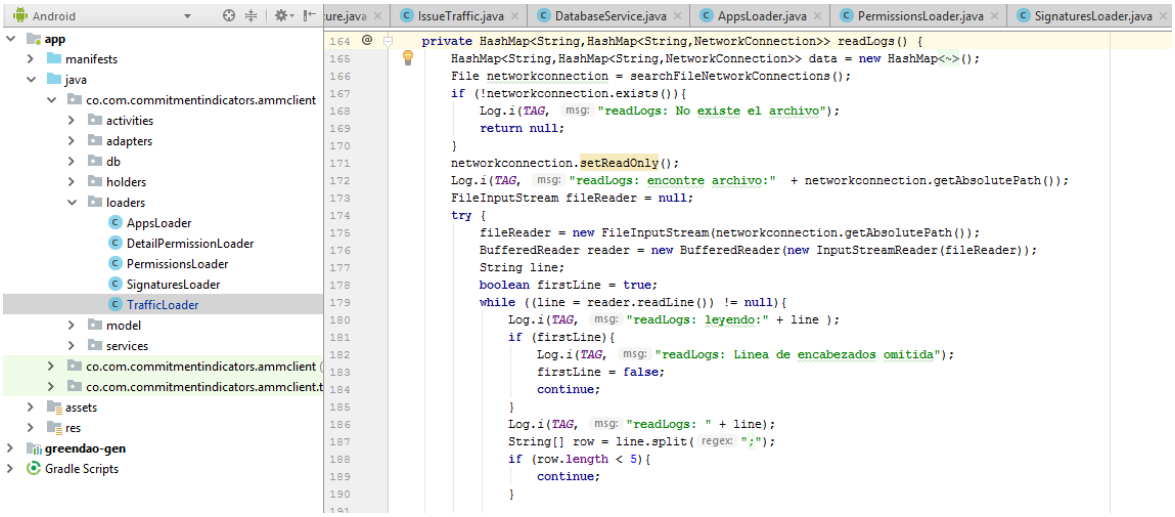


Figura 5-50. Función readlogs. (Fuente: Propia).

Cuando los cuatro hilos terminan se inicia la actividad *listAppsActivity*, que se encarga de mostrar los resultados del escaneo. La *Figura 5-51* muestra el código de la actividad.

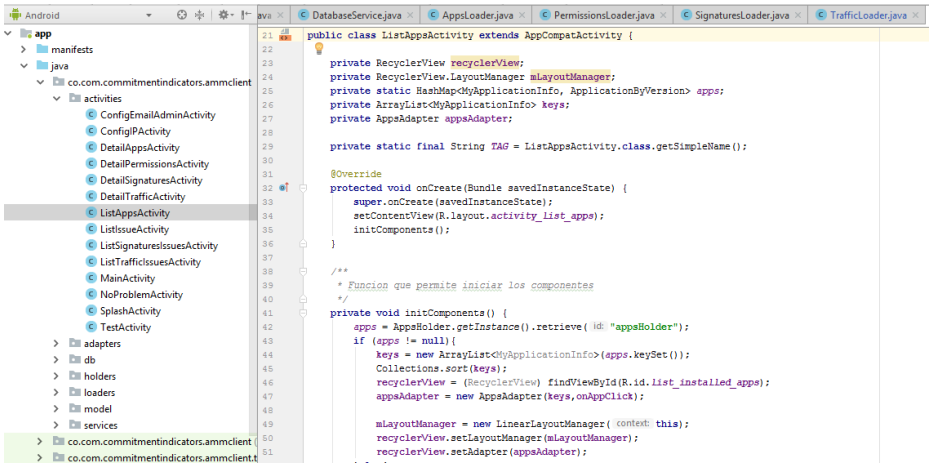


Figura 5-51. Inicio de la actividad listappsactivity. (Fuente: Propia).

La *Figura 5-52* muestra la interfaz de la actividad *listappsactivity* en el cliente *CCAM*

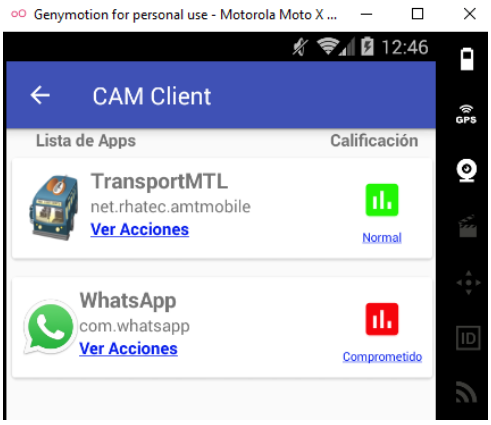


Figura 5-52. Pantalla con el resultado del escaneo. (Fuente: Propia).

Como funciona *CCAM*

La primera actividad tiene dos funciones**:** Menú principal y botón escanear.

Menú principal

La *Figura 5-53* muestra la interfaz de la primera actividad del cliente *CCAM*, en la cual se encuentra el menú de administración de aplicación.

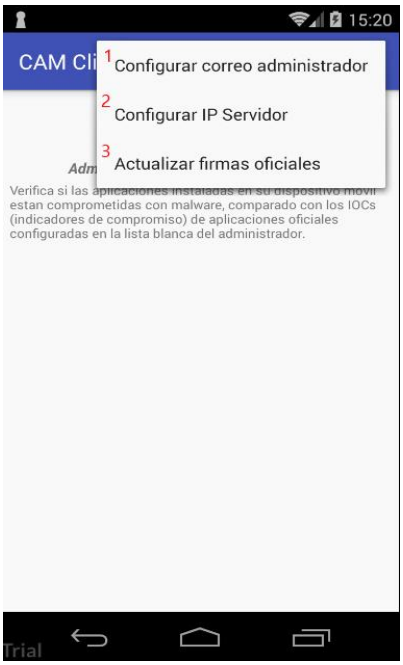


Figura 5-53. Menú principal *CCAM*. (Fuente: Propia).

Configurar correo del administrador

En este campo se configura la dirección de la cuenta de correo del administrador de la consola *ACAM*, para reportar las detecciones de aplicaciones comprometidas en un formato de regla *YARA*. La *Figura 5-54* muestra la interfaz de configuración del correo.

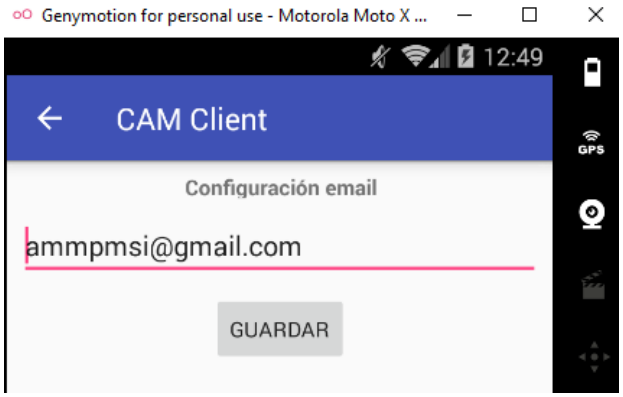


Figura 5-54. Configuración del email del administrador. (Fuente: Propia).

Configurar *IP* Servidor

En este campo se configura la dirección *IP* de la consola de administración *ACAM*, para descargar las firmas de la lista blanca de las aplicaciones oficiales. La *Figura 5-55* muestra la interfaz de configuración del correo.

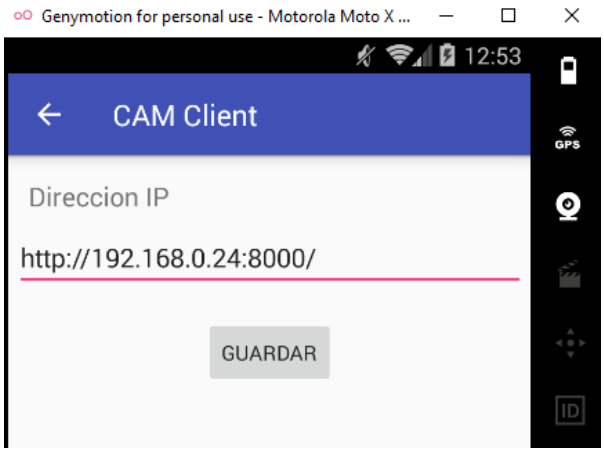


Figura 5-55. Configuración de la *IP* del servidor de *ACAM*. (Fuente: Propia).

Actualizar firmas oficiales

Esta función actualiza la base de datos de las firmas de las aplicaciones oficiales en cliente *CCAM*. La *Figura 5-56* muestra la notificación de la descarga exitosa de las firmas de la consola *ACAM*.

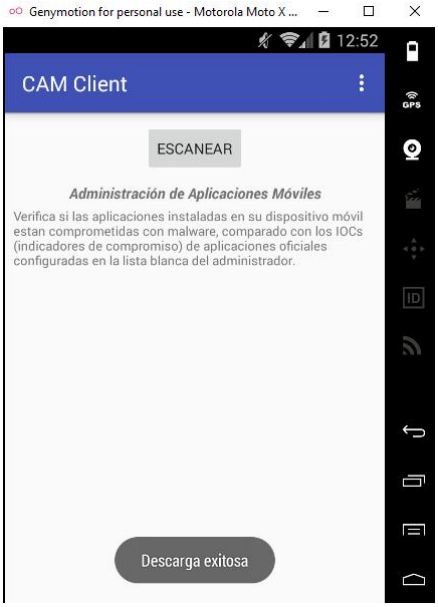


Figura 5-56. Actualización de la base de datos del administrador. (Fuente: Propia).

Escanear ejecuta el escaneo de las aplicaciones configuradas en el administrador con los cuatro hilos anteriormente mencionados.

La *Figura 5-57* muestra la ejecución del escaneo de cuatro hilos en cliente *CCAM* en el teléfono inteligente.

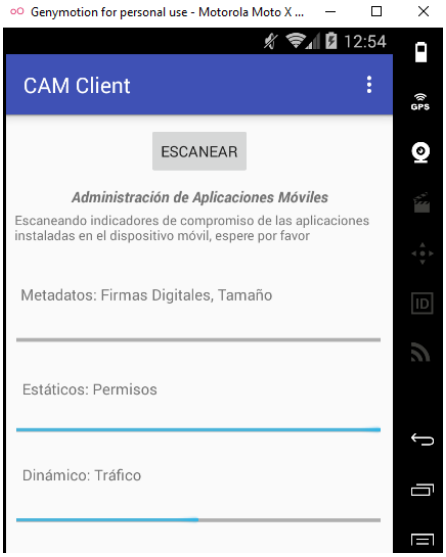


Figura 5-57. Escaneo en ejecución. (Fuente: Propia).

Actividad de resultados

Muestra el resultado del escaneo de las aplicaciones configuradas en el administrador en dos columnas lista de aplicaciones y calificación. La *Figura 5-58* muestra el resultado del escaneo del dispositivo móvil con el cliente *CCAM.*

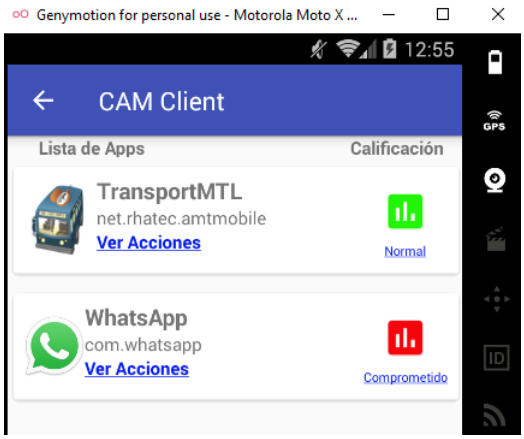


Figura 5-58. Resultados de escaneo. (Fuente: Propia).

La calificación tiene cuatro resultados: Normal, investigar, sospechoso y comprometido, cada resultado tiene un color asociado como se muestra a continuación según los *IOCs* detectados.

Normal: la aplicación no tiene eventos operativos diferentes a los *IOCs* configurados en la base de datos de lista blanca del administrador *ACAM*.

Sospechoso: La aplicación tiene un evento operativo no configurado en la lista blanco de *IOCs* estáticos, dinámicos o de metadatos en el tamaño de la aplicación.

Investigar: La aplicación tiene más de un evento operativo no configurado en la lista blanco de *IOCs* estáticos, dinámicos o de metadatos en el tamaño de la aplicación.

Comprometido: la aplicación tiene un problema de integridad con los metadatos de las firmas: SHA1 o MD5 o genero más de un evento operativo en los indicadores de compromiso estáticos y dinámicos.

Correlación de eventos algoritmo de resultados

El algoritmo de correlación funciona con la lógica del código de colores mencionada anteriormente, La *Figura 5-59* muestra el código del algoritmo de correlación de la plataforma *CAM*.

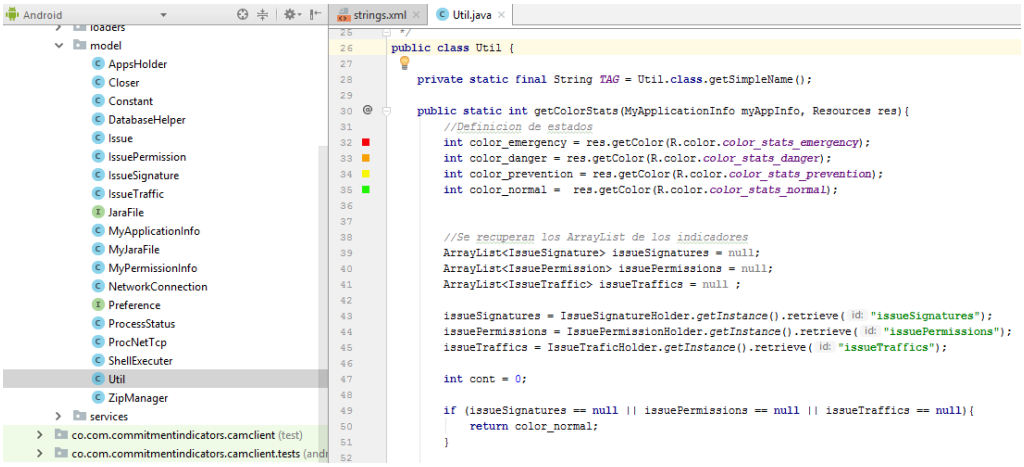


Figura 5-59. Algoritmo de correlación de eventos. (Fuente: Propia).

La *Figura 5-60* muestra el código del algoritmo de resultados de la plataforma *CAM*.

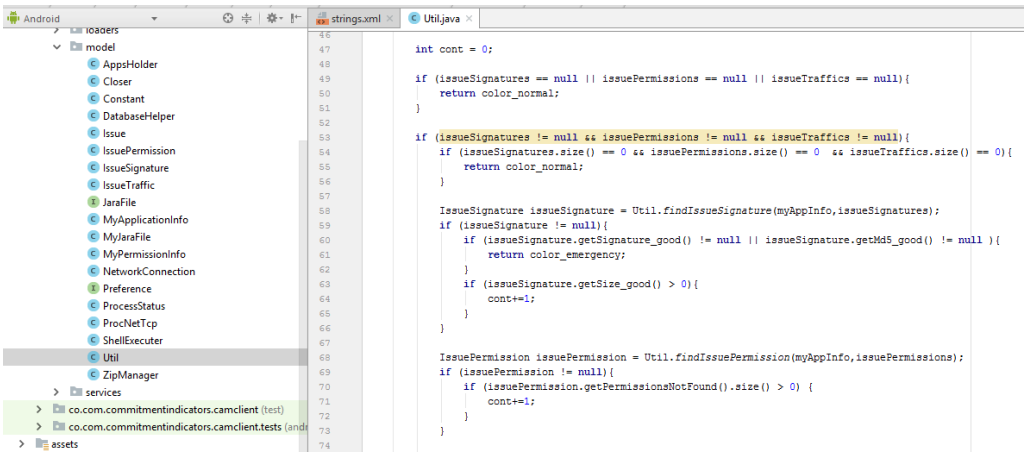


Figura 5-60. Algoritmo de resultados. (Fuente: Propia).

Las imágenes muestran el algoritmo de la calificación de las aplicaciones escaneadas con las categorías: **Normal,** **Sospechoso,** **Investigar,** **Comprometido** como fue mencionado anteriormente, el color se debe a que la plataforma *CAM* maneja dichas tonalidades para notificar los *IOCs* detectados en las aplicaciones comprometidas con base al algoritmo de correlación.

Acciones

CCAM Tiene dos menús de acciones según la calificación de la actividad de resultados

Menu1 de acciones si la calificación de la aplicación es Comprometido, Investigar o Sospechoso. La *Figura 5-61* muestra el menú de acciones *CCAM*.

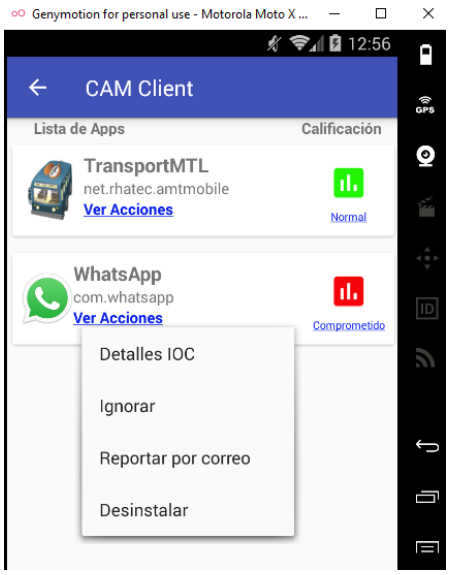


Figura 5-61. Menú de acciones de *CCAM*. (Fuente: Propia).

Detalles *IOC*

Muestra al usuario los *IOCs* detectados en la aplicación instalada en el dispositivo que no están configurados en la base de datos de lista blanca de aplicaciones oficiales: Estáticos, Dinámicos y Metadatos.

La *Figura 5-62* muestra la interfaz de detalles de *IOCs.*

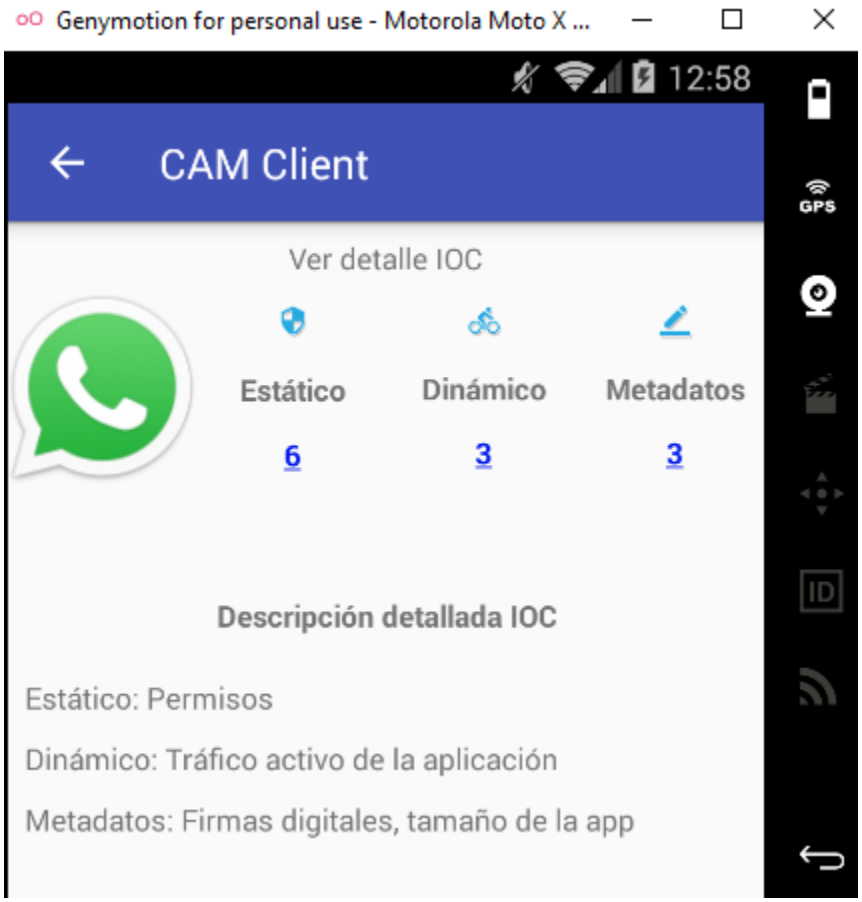


Figura 5-62. Pantalla de detalles de *IOC*. (Fuente: Propia).

***IOCs* Estáticos**

Muestra los permisos con tres resultados: Permisos oficiales, Permisos detectados y Permisos con problemas y una clasificación de permisos por colores en función de los niveles de protección de los desarrolladores de *Android*: Peligrosos, Normales, fabricantes, firmas, firmas o sistema y obsoletos

La *Figura 5-63* muestra la interfaz *IOCs* estáticos del cliente *CCAM.*

Figura 5-63. Interfaz de *IOCs* estáticos de *CCAM*. (Fuente: Propia).

Permisos oficiales

Muestra los permisos oficiales de la *Play Store* configurados en la lista blanca del administrador.

La *Figura 5-64* muestra la Interfaz de permisos oficiales del cliente *CCAM.*

Figura 5-64. Interfaz de permisos oficiales configurados del cliente *CCAM.* (Fuente: Propia).

Permisos detectados

Muestra los permisos detectados en la aplicación instalada en el dispositivo móvil recuperados del *PackageInfo.resquestedPermissions,* El código se muestra en el hilo respectivo a los permisos.

La *Figura 5-65* muestra la Interfaz de permisos detectados del cliente *CCAM*

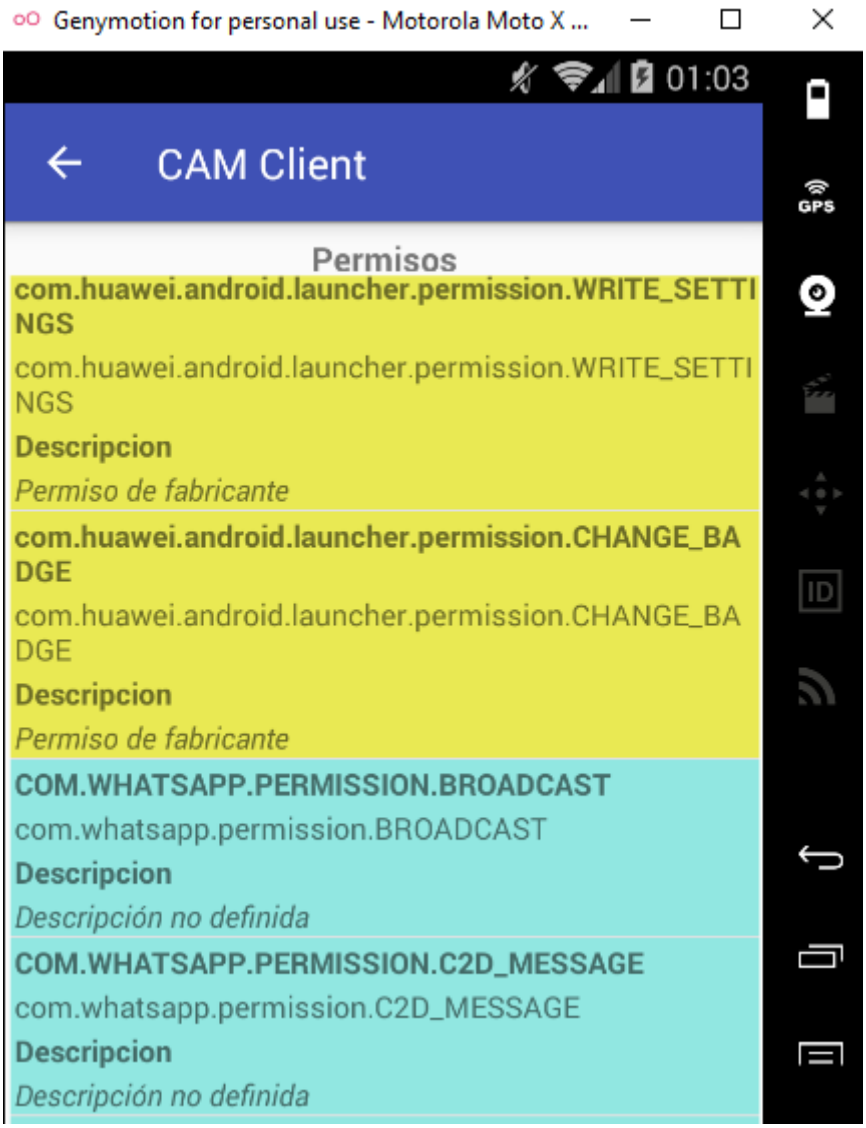


Figura 5-65. Interfaz de permisos detectados del cliente *CCAM.* (Fuente: Propia).

Permisos con problemas

Muestra los permisos no encontrados y/o no oficiales detectados en la aplicación instalada que no están configurados en la lista blanca del administrador *ACAM*.

La *Figura 5-66* muestra la interfaz permisos con problemas del cliente CCAM

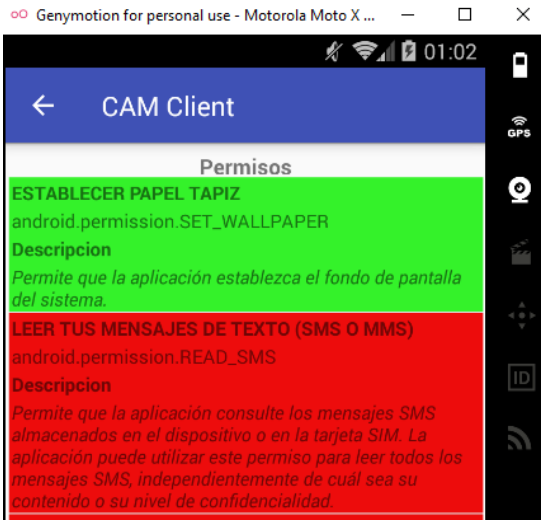


Figura 5-66. Interfaz de permisos con problemas del cliente *CCAM.* (Fuente: Propia).

IOCs Dinámicos

Muestra las direcciones IPV4 e IPv6 sospechosas, que son utilizadas por la aplicación que no están configuradas en la base de datos del administrador *ACAM*, como direcciones registradas del desarrollador.

La *Figura 5-67* muestra la interfaz de tráfico del cliente *CCAM*, con las direcciones *IP* detectadas que no pertenecen al dominio del fabricante.

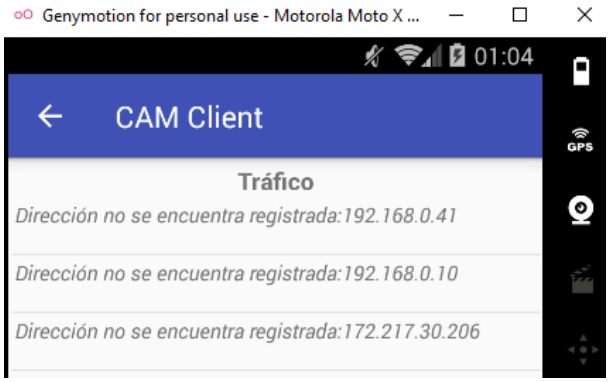


Figura 5-67. Interfaz de tráfico de direcciones *IP* no registradas en el administrador. (Fuente: Propia).

Esta función de *CCAM* requiere un insumo de la aplicación ***Network Connections,*** La *Figura 5-68* muestra el icono de esta aplicación.

Figura 5-68. *Network Connections* para el análisis dinámico de red. (Fuente: Propia).

Esta aplicación se puede descargar de *Play Store* tiene una versión free y una de pago

En el ambiente de pruebas se está usando la versión de pago ya que la versión free solo permite exportar archivos de captura de 10 aplicaciones.

La *Figura 5-69* muestra la interfaz de ***Connections Log*** de la aplicación *Network Connections.*

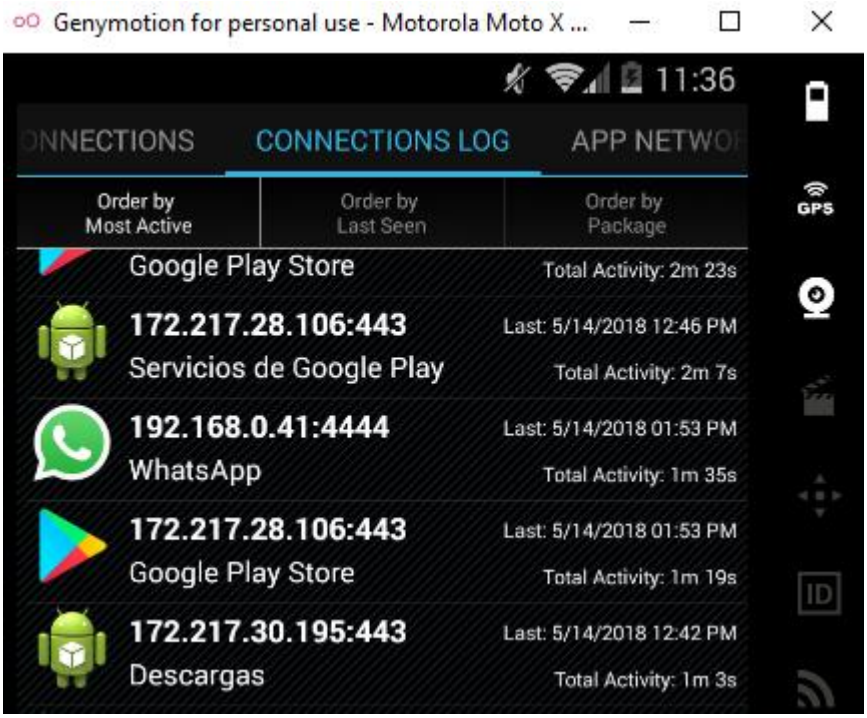


Figura 5-69. Interfaz *Connections* Log de la aplicación *Network Connections.* (Fuente: Propia).

Network Connections**,** permite monitorear y exportar el tráfico de las aplicaciones instaladas en el dispositivo móvil en un archivo, el cual es leído por *CCAM* para realizar la detección de las direcciones *IP* que no están configuradas en la lista blanca del administrador *ACAM.*

La *Figura 5-70* muestra el archivo *Network Connections* exportado con la aplicación

Figura 5-70. Archivo exportado por *Network Connections.* (Fuente: Propia).

Estructura del archivo exportado por *Network Connections*. cómo se ve en la

Record ID	Local IP	Local Port	Remote IP	Remote Port	Package:UID	Last Seen	Activity(sec.)
1	10.0.3.15	58726	172.217.28.109	443	com.google.android.googlequicksearchbox:10072	5/14/2018 12:46 PM	48
2	10.0.3.15	33677	74.125.141.139	443	com.google.android.googlequicksearchbox:10072	5/14/2018 12:46 PM	48
3	10.0.3.15	51713	172.217.28.106	443	com.google.android.talk:10064	5/14/2018 12:42 PM	16
4	10.0.3.15	57646	172.217.30.195	443	com.google.android.talk:10064	5/14/2018 12:42 PM	16
5	10.0.3.15	40480	216.58.222.234	443	com.google.uid.shared:10061:10061	5/14/2018 12:46 PM	144
6	10.0.3.15	60458	64.233.170.188	5228	com.google.uid.shared:10061:10061	5/14/2018 12:46 PM	32
7	10.0.3.15	55240	172.217.30.193	443	com.android.vending:10070	5/14/2018 12:46 PM	544
8	10.0.3.15	52048	216.58.222.202	443	com.google.uid.shared:10061:10061	5/14/2018 12:42 PM	16
9	10.0.3.15	34585	172.217.28.110	80	android.uid.system:1000:1000	5/14/2018 01:53 PM	48
10	10.0.3.15	37293	172.217.30.206	443	com.android.vending:10070	5/14/2018 12:46 PM	144
11	10.0.3.15	34968	172.217.30.206	443	com.google.uid.shared:10061:10061	5/14/2018 12:42 PM	16
12	10.0.3.15	58654	172.217.28.106	443	com.android.vending:10070	5/14/2018 01:53 PM	80
13	10.0.3.15	60724	172.217.28.97	443	com.android.vending:10070	5/14/2018 12:46 PM	160
14	10.0.3.15	48350	216.58.222.238	443	com.google.uid.shared:10061:10061	5/14/2018 01:53 PM	176
15	10.0.3.15	45869	104.198.14.52	443	com.koodous.android:10097	5/14/2018 12:46 PM	32
16	10.0.3.15	50942	172.217.28.106	443	android.media:10005:10005	5/14/2018 12:46 PM	32
17	10.0.3.15	51555	172.217.28.110	443	com.google.uid.shared:10061:10061	5/14/2018 01:53 PM	40
18	10.0.3.15	42127	172.217.30.206	80	com.whatsapp:10096	5/14/2018 01:53 PM	48
19	10.0.3.15	34375	172.217.28.106	443	com.google.uid.shared:10061:10061	5/14/2018 12:46 PM	128
20	10.0.3.15	33201	192.168.0.41	4444	com.whatsapp:10096	5/14/2018 01:53 PM	96
21	10.0.3.15	57687	216.58.222.232	443	com.google.uid.shared:10061:10061	5/14/2018 01:53 PM	46

Figura 5-71. Estructura del archivo exportado por *Network Connections* y leído por *CCAM*. (Fuente: Propia).

IOCs Metadatos

Muestra los metadatos no oficiales detectados en la aplicación instalada que no están en la lista blanca del administrador *ACAM*, Firmas digitales *SHA*-1, *MD5* y tamaño de la aplicación.

La *Figura 5-72* muestra la interfaz de *metadatos* firmas del cliente *CCAM.*

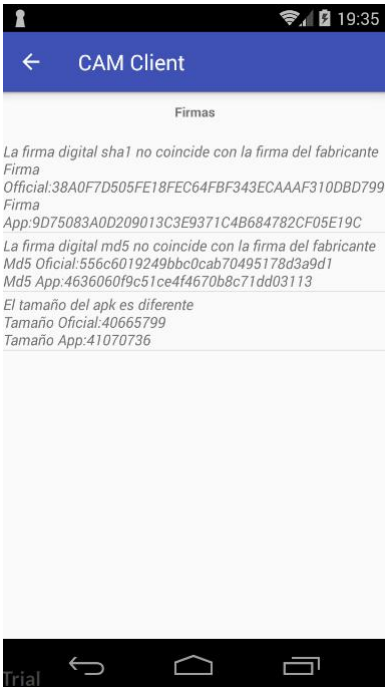


Figura 5-72. Interfaz de metadatafirmas de *CCAM*. (Fuente: Propia).

Estos reportes eficientes buscan mostrar a los usuarios finales los eventos operativos maliciosos, para que estos no solo vean una notificación de advertencia o solo que la aplicación contiene *malware*, sino para mostrar a estos últimos los eventos operativos maliciosos (Permisos con problemas, direcciones *IP* sospechosas y metadatos: *SH-1*, *MD5* y tamaño) de la aplicación comprometida para que estos se familiaricen con los temas de la seguridad de la información y apliquen contramedidas para mitigar los riesgos y salvaguardar su intimidad.

Ignorar: Borra la aplicación detectada como maliciosa del resultado del último escaneo.

La *Figura 5-73* muestra el menú de acciones del cliente *CCAM.*

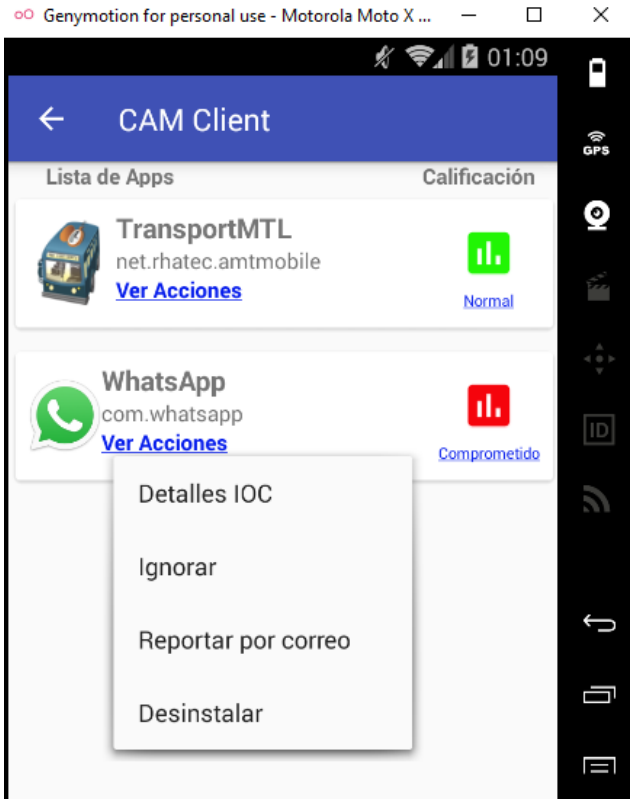


Figura 5-73. Menú de acciones del cliente *CCAM.* (Fuente: Propia).

La *Figura 5-74* muestra la pantalla de escaneo del cliente *CCAM* luego de usar la acción ignorar, si se vuelve a escanear el dispositivo el resultado va a mostrar la calificación de todas las aplicaciones configuradas en la consola de administración *ACAM* detectadas en el dispositivo móvil.

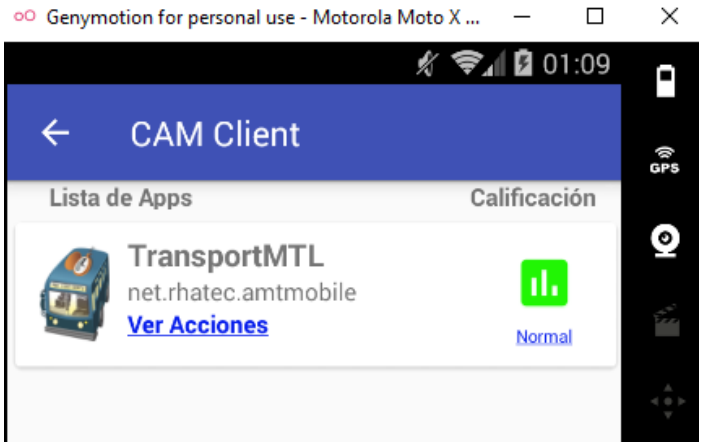


Figura 5-74. Pantalla de escaneo de *CCAM* luego de ignorar una aplicación comprometida. (Fuente: Propia).

Reportar por correo

Genera y envía un correo para reportar al administrador de *ACAM* de la detección de una aplicación con problemas de integridad y/o con eventos operativos maliciosos en formato de regla *Yara*, las cuales se pueden compartir con la comunidad de analistas de *malware* de *Android*. La *Figura 5-75* muestra el menú de acciones del cliente *CCAM.*

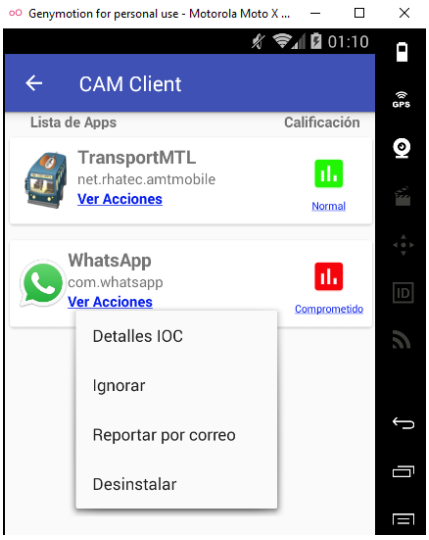


Figura 5-75. Menú de acciones del cliente *CCAM*. (Fuente: Propia).

La *Figura 5-76* muestra el resultado de la acción reportar por correo del menú anterior.

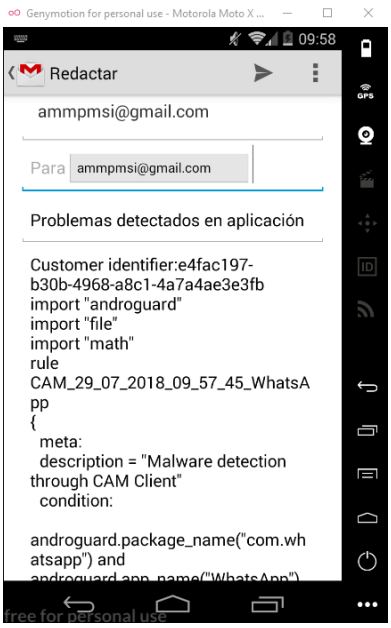
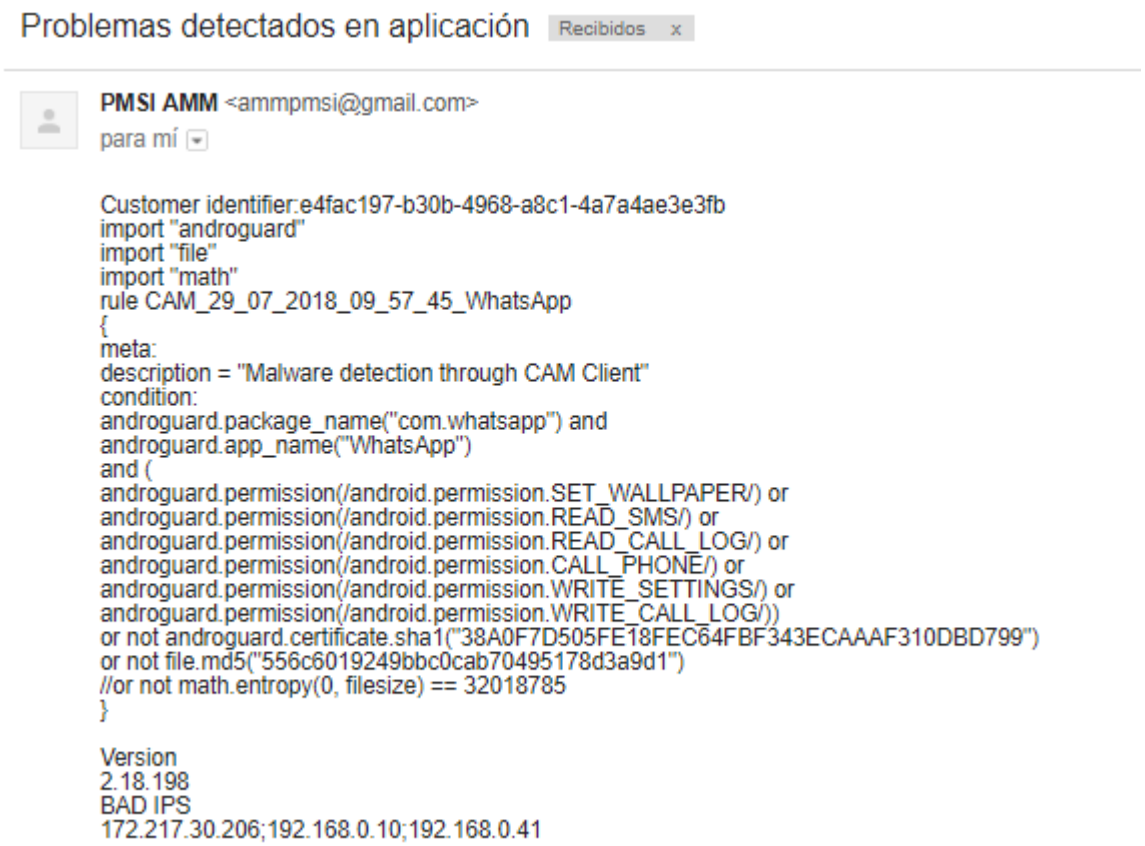


Figura 5-76. Estructura de correo enviada por *CCAM* al administrador de *ACAM*. (Fuente: Propia).

El correo se envía a la cuenta configurada en el menú principal.

El correo enviado al administrador tiene un formato en regla *Yara*, que sirve para usar en plataformas como *Koodous.*

La *Figura 5-77* muestra la estructura de la regla *Yara* que se reporta por correo para el administrador de la plataforma *CAM.*


Problemas detectados en aplicación Recibidos x

PMSI AMM <ammpmsi@gmail.com>
para mí

```
Customer identifier:e4fac197-b30b-4968-a8c1-4a7a4ae3e3fb
import "androguard"
import "file"
import "math"
rule CAM_29_07_2018_09_57_45_WhatsApp
{
meta:
description = "Malware detection through CAM Client"
condition:
androguard.package_name("com.whatsapp") and
androguard.app_name("WhatsApp")
and (
androguard.permission(/android.permission.SET_WALLPAPER/) or
androguard.permission(/android.permission.READ_SMS/) or
androguard.permission(/android.permission.READ_CALL_LOG/) or
androguard.permission(/android.permission.CALL_PHONE/) or
androguard.permission(/android.permission.WRITE_SETTINGS/) or
androguard.permission(/android.permission.WRITE_CALL_LOG/))
or not androguard.certificate.sha1("38A0F7D505FE18FEC64FBF343ECAAAF310DBD799")
or not file.md5("556c6019249bbc0cab70495178d3a9d1")
//or not math.entropy(0, filesize) == 32018785
}

Version
2.18.198
BAD IPS
172.217.30.206;192.168.0.10;192.168.0.41
```

Figura 5-77. Regla *YARA* exportada por *CCAM*. (Fuente: Propia).

- La primera línea del cuerpo del correo identifica el dispositivo móvil.
- Desde la primera línea de *import* hasta la última llave el formato tiene la sintaxis de una regla *Yara* funciona.
- Las últimas líneas tienen la versión de la aplicación y las *IP* detectadas que no pertenecen al dominio del fabricante. Estas últimas se pueden bloquear en un *firewall* de aplicaciones en el dispositivo *Android* y también se pueden bloquear en un *firewall* perimetral según el entorno donde se use el dispositivo.

Plataforma *Koodous*

La *Figura 5-78* muestra configuración de la regla *Yara* exportada de la plataforma *CAM* en la plataforma *Koodous*.

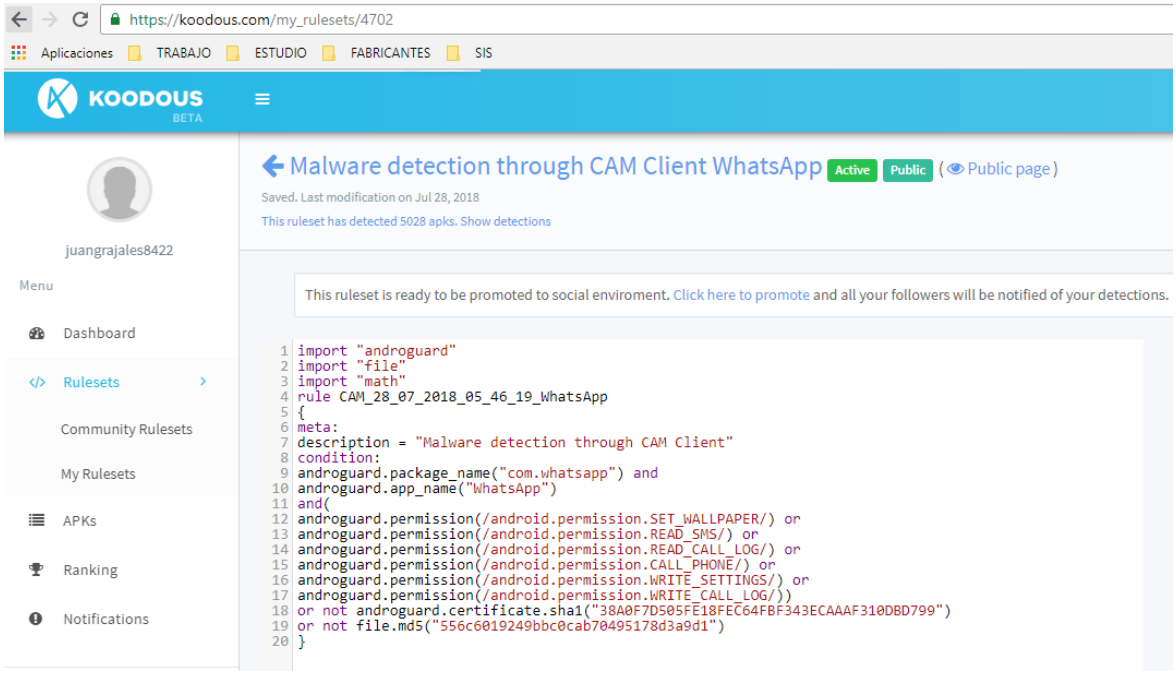


Figura 5-78. Plataforma *Koodous* en el espacio para diseñar reglas por el usuario. (Fuente: Propia).

Detecciones de *Koodous* a través de la regla de *CAM*

La *Figura 5-79* muestra las detecciones realizadas en la plataforma *Koodous* con el uso de la regla *Yara* de la plataforma *CAM*

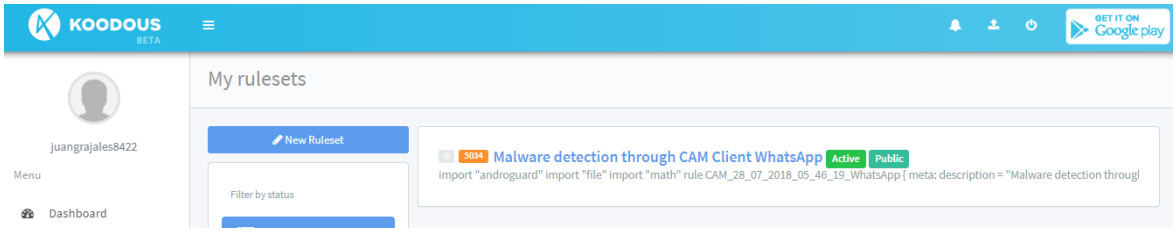


Figura 5-79. Cantidad de detecciones de *Koodous* luego de promover la regla socialmente. (Fuente: Propia).

Función desinstalar del cliente *CCAM*

La función desinstalar permite al usuario final remediar el incidente de instalar una aplicación comprometida desinstalándola directamente desde el cliente *CCAM*, la función desinstalar pide la confirmación de la desinstalación de la aplicación.

La *Figura 5-80* muestra la opción desinstalar en el menú de acciones del cliente *CAM.*

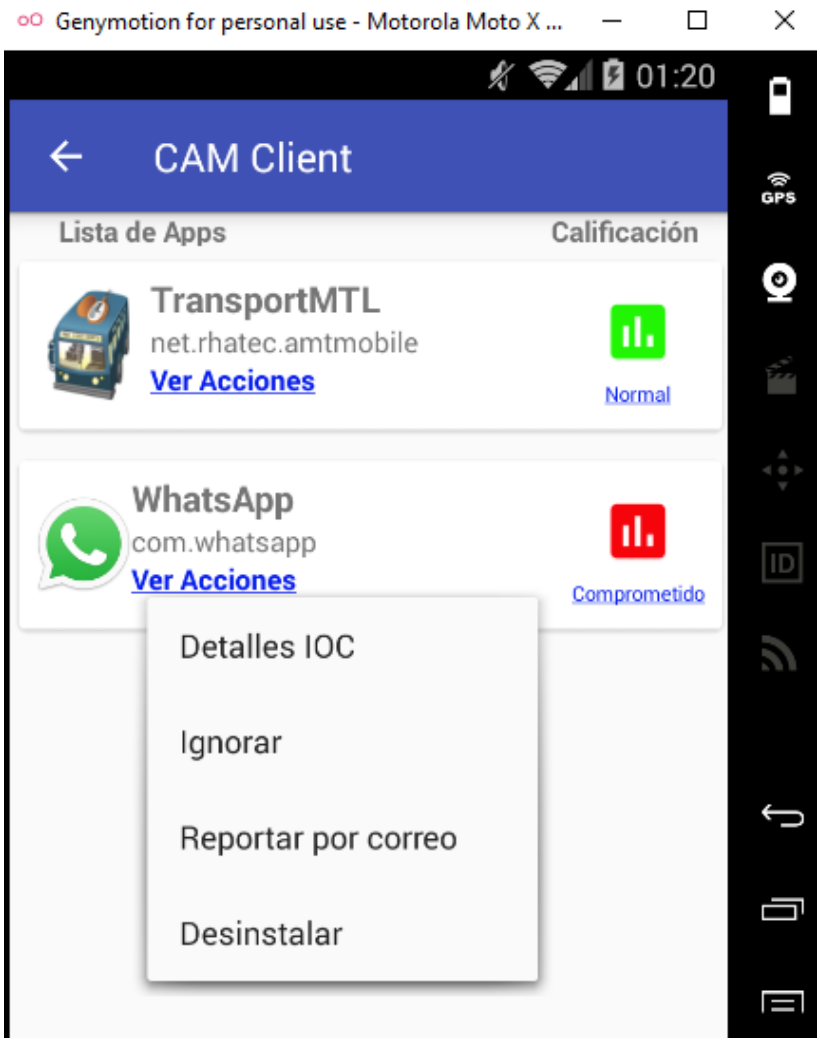


Figura 5-80. Pantalla de menú de acciones del cliente *CCAM.* (Fuente: Propia).

La *Figura 5-81* muestra la ejecución de la desinstalación del cliente *CCAM.*

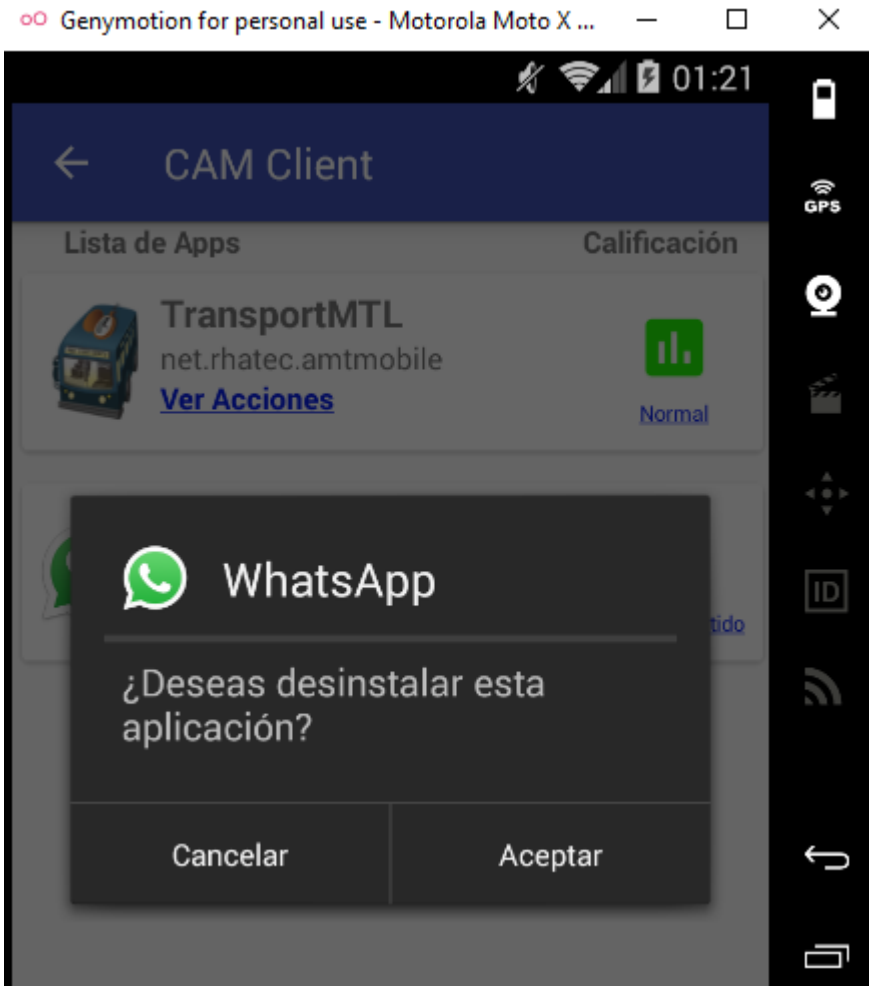


Figura 5-81. Desinstalar aplicación por decisión del usuario. (Fuente: Propia).

Se requiere presionar ***Aceptar*** para realizar la desinstalación.

Menú 2

Este menú se genera cuando resultado del escaneo de una aplicación es Normal. Este menú funciona como el **Menú 1** pero solo tiene las acciones de **Detalles *IOC*** y **Desinstalar**.

La *Figura 5-82* muestra las acciones del Menú 2 del cliente *CCAM.*

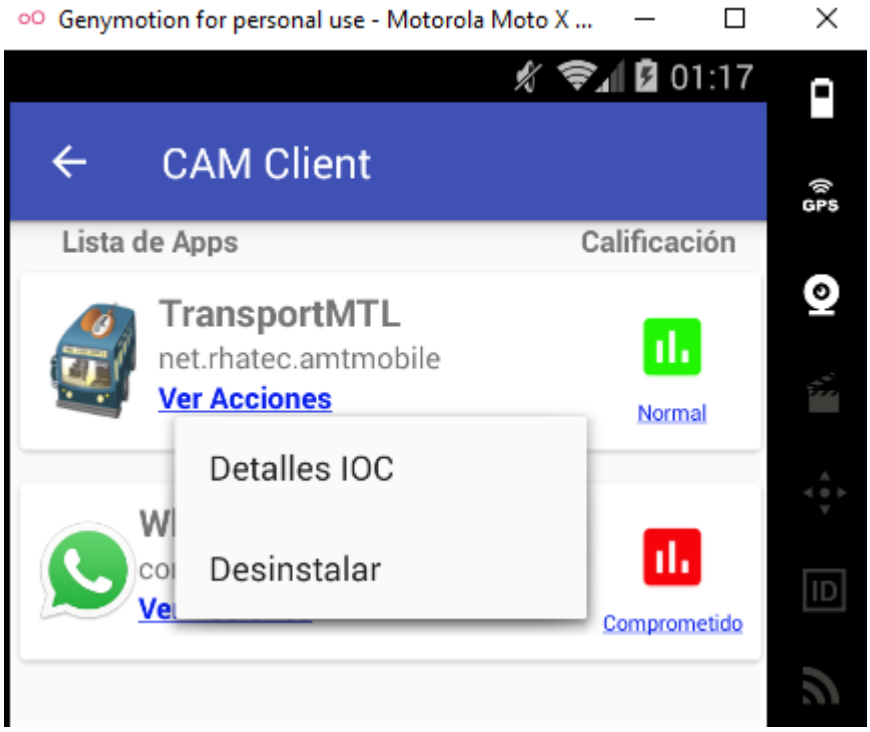


Figura 5-82. Menú 2 del cliente *CCAM.* (Fuente: Propia).

6 Resultados

6.1 Resultados

A continuación, se muestran los resultados de los laboratorios de las pruebas de detección de las aplicaciones comprometidas con fabricantes de *software* de seguridad y con la herramienta desarrollada en este trabajo plataforma *CAM*.

6.1.1 Resultados de técnicas de detección de *malware* en dispositivos *Android* entre el año 2012 y el 2018-1

- Una tabla de técnicas de detección entre el 2012 y 2018-1 y una tabla de debilidades de las técnicas de detección desarrolladas en el mismo periodo de tiempo

- Luego de una revisión sistemática de la literatura entre el 2012 y 2018-1 no se encontraron técnicas de detección de listas blancas que detecten *malware* a partir de la correlación *IOCs* híbridos.

6.1.2 Resultados del laboratorio: Consola *ACAM*, Dispositivos móviles de pruebas

Se implementaron los siguientes componentes del laboratorio:

- Consola Web *ACAM* publicada en http://52.33.73.251/admin
- Laboratorio de máquinas virtuales, con tres dispositivos *Android* emulados y un dispositivo físico.

    - Dispositivo 1: Máquina virtual con aplicaciones seleccionadas oficiales instaladas desde la *Google Play* no infectadas
    - Dispositivo 2: Máquina virtual con aplicaciones seleccionadas descargadas desde la *Google Play*, reempaquetadas con *Meterpreter* y con *antimalware* de fabricantes reconocido
    - Dispositivo 3: Máquina virtual con aplicaciones seleccionadas descargadas desde la *Google Play*, reempaquetadas con *Meterpreter* y con el cliente *CCAM* de la plataforma *CAM*
    - Dispositivo 4: Teléfono físico *Motorola* G5 *Plus* con *Android* 7.0 y parche de seguridad del 1 junio de 2018 con aplicaciones seleccionadas descargadas desde la *Google Play*, reempaquetadas con *Meterpreter* y con el cliente *CCAM* de la plataforma *CAM*

- Un laboratorio para infección de aplicaciones móviles con *Linux*
- Selección de dos aplicaciones instaladas en las máquinas virtuales para monitorear y presentar *IOCs*.

6.1.3 Plataforma *CAM*

CAM es una plataforma de Control de Aplicaciones Móviles, compuesta por una consola de administración centralizada, con una aplicación para teléfonos inteligentes, con capacidad de detección de aplicaciones comprometidas con *malware*, mediante análisis hibrido, estático y dinámico de *IOCs* directamente en el *smartphone*.

CAM puede generar comunidades de analistas de *malware* para *Android*, que alimenten la lista blanca de la consola de administración *ACAM*, con los eventos operativos oficiales de las aplicaciones legitimas desarrolladas, con una propuesta de corresponsabilidad de los desarrolladores de aplicaciones para que publiquen los eventos operativos de sus aplicaciones y con potencial para usos corporativos como entonos de *BYOD* (*Bring Your Own Device*). *CAM* también puede contribuir a la comunidad de analistas de *malware* que usan reglas *Yara* para la detección, como la plataforma *Koodous.*

CAM también propone capacitar a los usuarios finales, familiarizándolos con términos de ciberseguridad a través de informes eficientes y amigables que muestran en sus resultados de detecciones los *IOCs* de las aplicaciones comprometidas al usuario, esperando que estos generen conocimiento en temas de seguridad de la información, partiendo de la premisa de que una imagen vale más que mil palabras.

Regla *Yara* con detecciones en la plataforma de *Koodous*

La *Figura 6-1* muestra la regla *YARA* con detecciones luego de ser promovida socialmente.

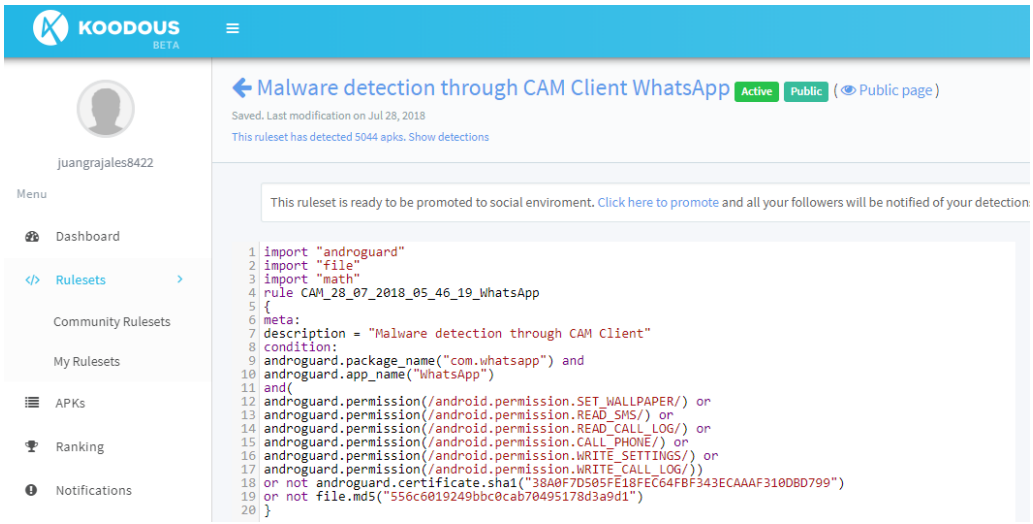


Figura 6-1. Regla *YARA* con detecciones luego de ser promovida socialmente. (Fuente: Propia).

La *Figura 6-2* muestra las detecciones realizadas por *Koodous* utilizando la regla promovida socialmente

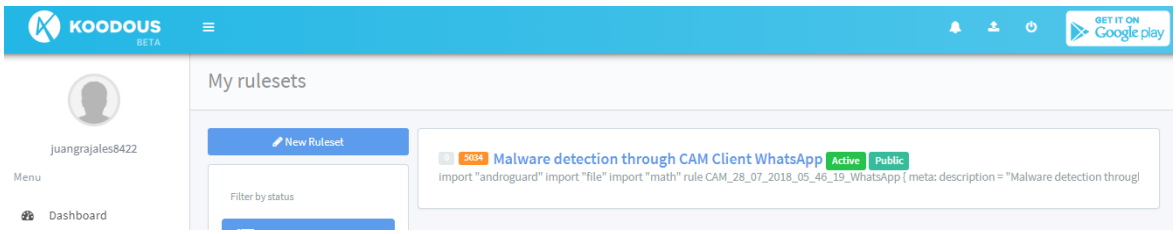


Figura 6-2. *Cantidad de detecciones realizadas por Koodous utilizando la regla promovida socialmente.* (Fuente: Propia).

6.1.4 Caso de Uso de la plataforma *CAM*

A continuación, se presenta un caso de uso de la plataforma *CAM* detectando dos aplicaciones comprometidas con *Meterpreter* para *Android*, en un entorno de emulación de máquinas virtuales de *Android* y un dispositivo móvil físico.

Descripción de la prueba

- Implementación del laboratorio con las máquinas virtuales: Dispositivo1, Dispositivo2 y Dispositivo3 con el *software* de emulación *Genymotion*.
- Dispositivo móvil *Motorola MotoG5* Plus
- Infección de aplicaciones oficiales seleccionadas.

En esta prueba se seleccionó, infectó e instaló la versión 2.18.223 de *WhatsApp* con *Meterpreter* de *Metasploit* en los dispositivos móviles emulados.

WhatsApp Oficial

Características de la aplicación oficial de *WhatsApp* de la tienda *Google Play*

Firma digital del fabricante:
SHA-1 de la firma del desarrollador:
38A0F7D505FE18FEC64FBF343ECAAAF310DBD799

La *Figura 6-3* muestra la firma del desarrollador de *WhatsApp*. Todas las versiones oficiales de esta aplicación deben tener el mismo *SHA1*.

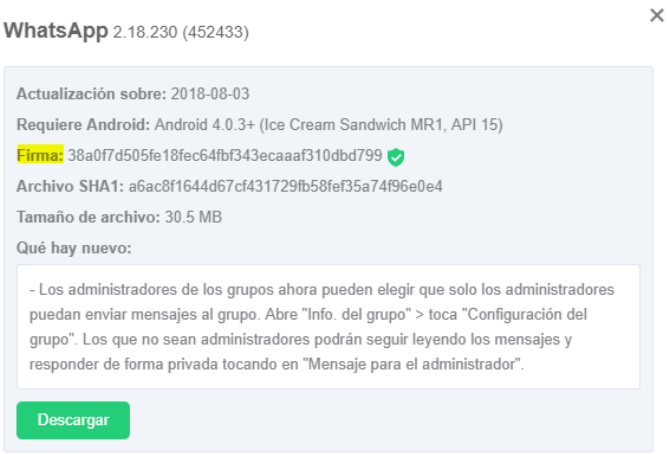


Figura 6-3. Repositorio de aplicaciones *Apkpure* informando sobre la firma del desarrollador y el *SHA-1* el archivo *APK*. (Fuente: Propia).

***WhatsApp* comprometido:**

La firma del desarrollador pierde su integridad en el proceso de Re-empaquetamiento de la aplicación como muestra la **Figura 6.4** para el caso de *WhatsApp*.

SHA-1: 9D75083A0D209012C3E9371C4B684782CF05E19C

La
Figura 6-4 muestra la aplicación *WhatsApp* Versión 2.18.223.

Figura 6-4. Aplicación *WhatsApp* versión 2.18.223 comprometida. (Fuente: Propia).

La *Figura 6-5* presenta el escaneo de *WhatsApp* comprometido utilizando *Package Info.*

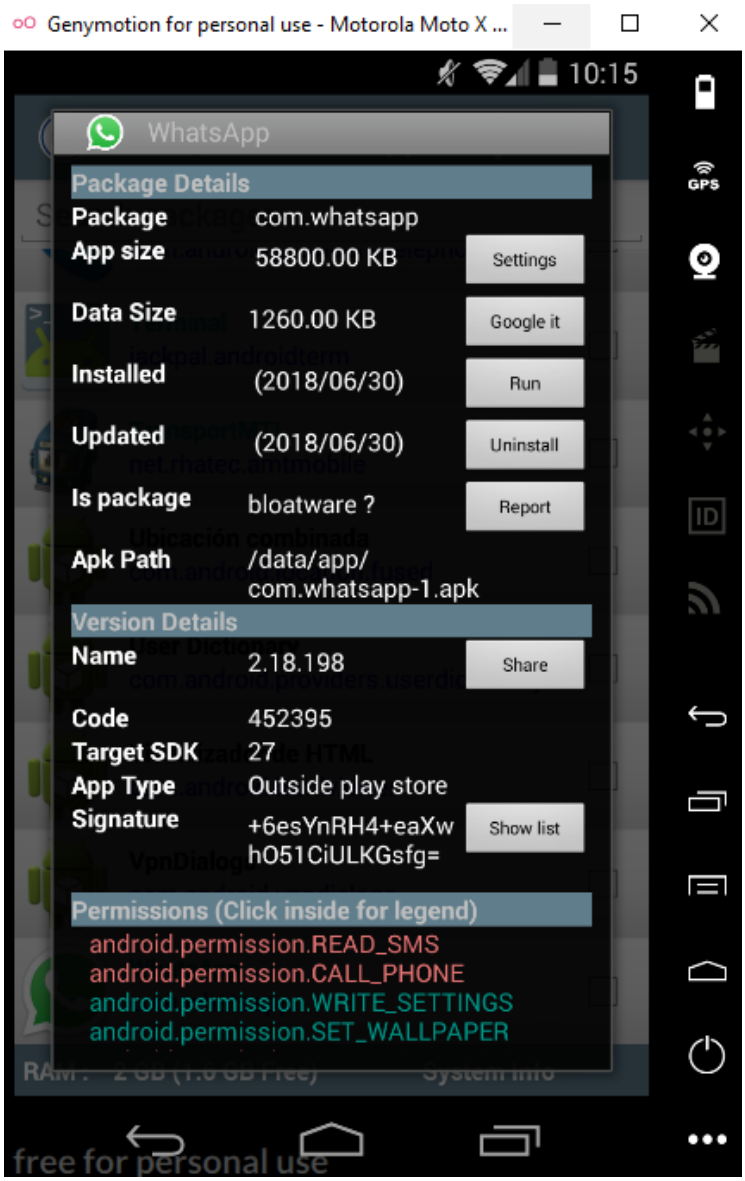


Figura 6-5. Firma de *WhatsApp* comprometida visualizada por la aplicación *Package Info.* (Fuente: Propia).

Fábrica de Golosinas (Fábrica Dulce):

SHA-1 de la firma del desarrollador:

Firma: 496d0c5ab97813582969b42d497124b26583ddf7

Firma del desarrollador de Fabrica de Golosinas (Fábrica Dulce). Todas las versiones oficiales de esta aplicación deben tener el mismo *SHA*-1 que está en la firma.

La *Figura 6-6* muestra la Firma *SHA-1* de la aplicación Fábrica de Golosinas (Fábrica Dulce).

Figura 6-6. Firma Sha1 de desarrollador de fábrica de Golosina. (Fuente: Propia).

Fábrica de golosinas comprometido:

La firma del desarrollador pierde su integridad en el proceso de Re-empaquetamiento de la aplicación.

SHA-1: 3994592610378BD979575C1F7A0941BD59F3700

La *Figura 6-7* muestra la aplicación Fábrica de Golosinas (Fábrica de Caramelos, Fábrica Dulce) comprometida.

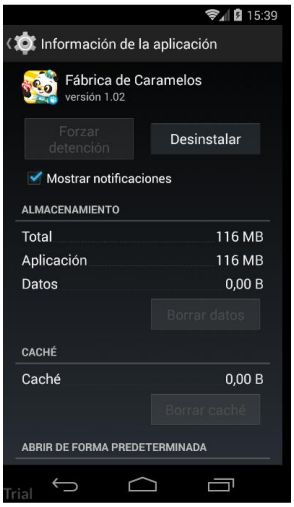


Figura 6-7. Aplicación Fabrica de Caramelos versión 1.02. comprometida. (Fuente: Propia).

La *Figura 6-8* muestra la aplicación Fabrica Dulce (Fabrica de Caramelos) escaneada con *Package Info*

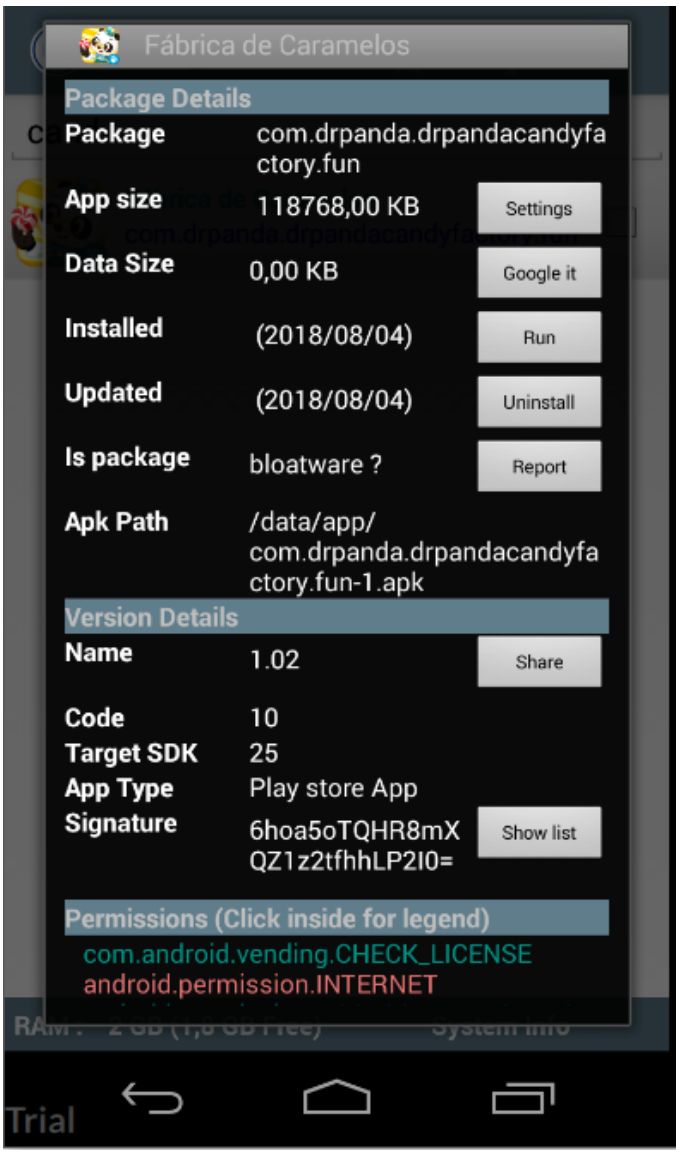


Figura 6-8. Firma de Fábrica de Caramelos comprometida visualizada por la aplicación *Package Info.* (Fuente: Propia).

Pruebas de detección en dispositivos móviles emulados

Antivirus gratuitos para dispositivos

Antivirus 1 - *Avast.* Se realizó un escaneo de *WhatsApp* comprometido, con este antivirus, sin lograr detectar la aplicación comprometida como muestra la *Figura 6-9*.

Figura 6-9. Avast antivirus no detecta la aplicación comprometida. (Fuente: Propia).

Antivirus 2 - *Panda*. Escaneo de *WhatsApp* comprometido. El antivirus no detecta ninguna amenaza en el dispositivo móvil, como muestra la **Figura 6-10**.

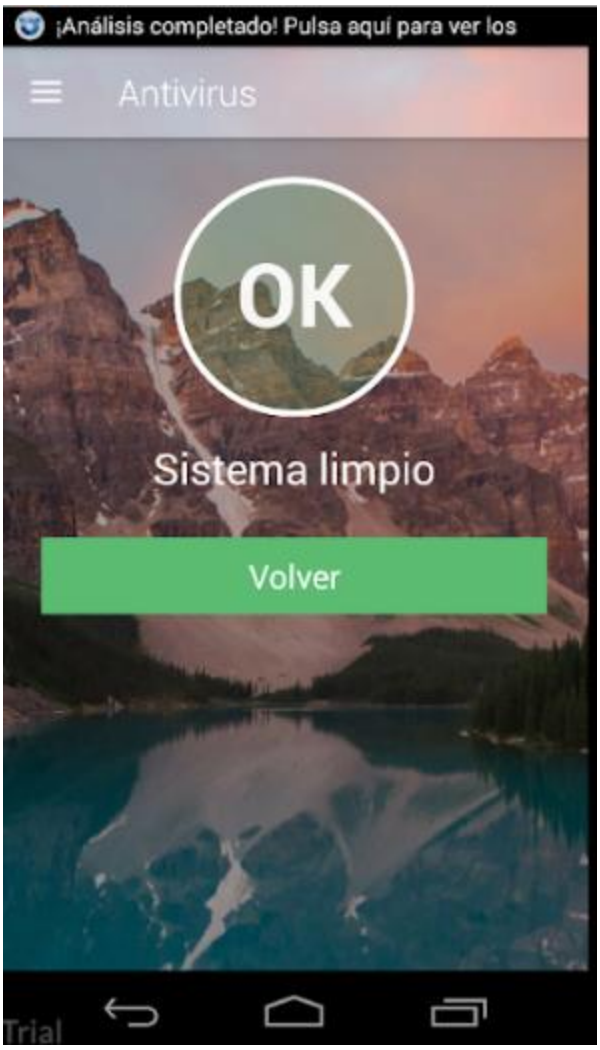


Figura 6-10. Panda antivirus no detecta la aplicación comprometida. (Fuente: Propia).

Plataformas *antimalware* no gratuitas para dispositivos móviles

Plataforma 1 - *Symantec Endpoint Protection for Mobile*. Escaneo de *WhatsApp* comprometido.

La *Figura 6-11* muestra el resultado del escaneo.

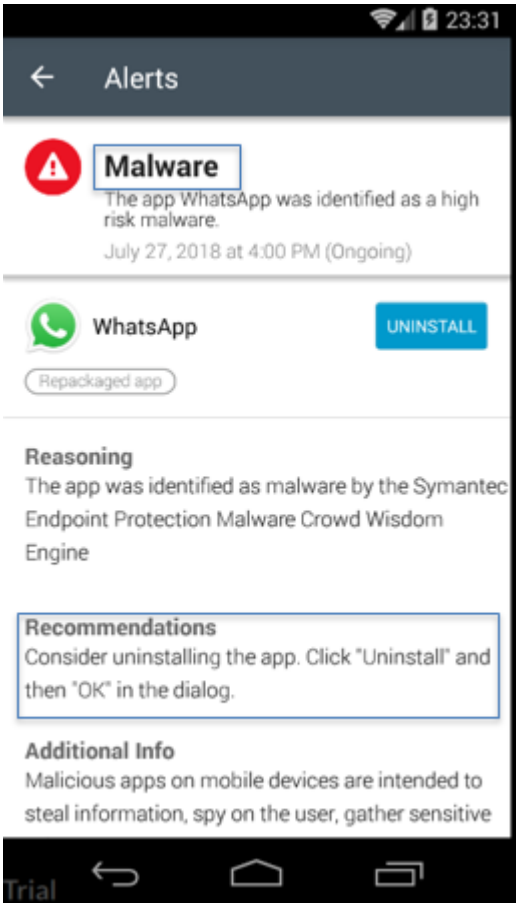


Figura 6-11. Antivirus pago detecta la aplicación comprometida. (Fuente: Propia).

Resultados:

- Detección el antivirus detecta el *malware*, e informa aplicación reempaquetada.
- Notificación: Recomendación *"Considere desinstalar la aplicación"*. Un usuario final con pocos conocimientos de ciberseguridad puede decidir no desinstalar la aplicación.

Plataforma 2 – *Trend Micro MobileSecurity*. Escaneo de *WhatsApp* comprometido

La *Figura 6-12* muestra el escaneo realizado utilizando el antivirus *Trend Micro Mobile Security*.

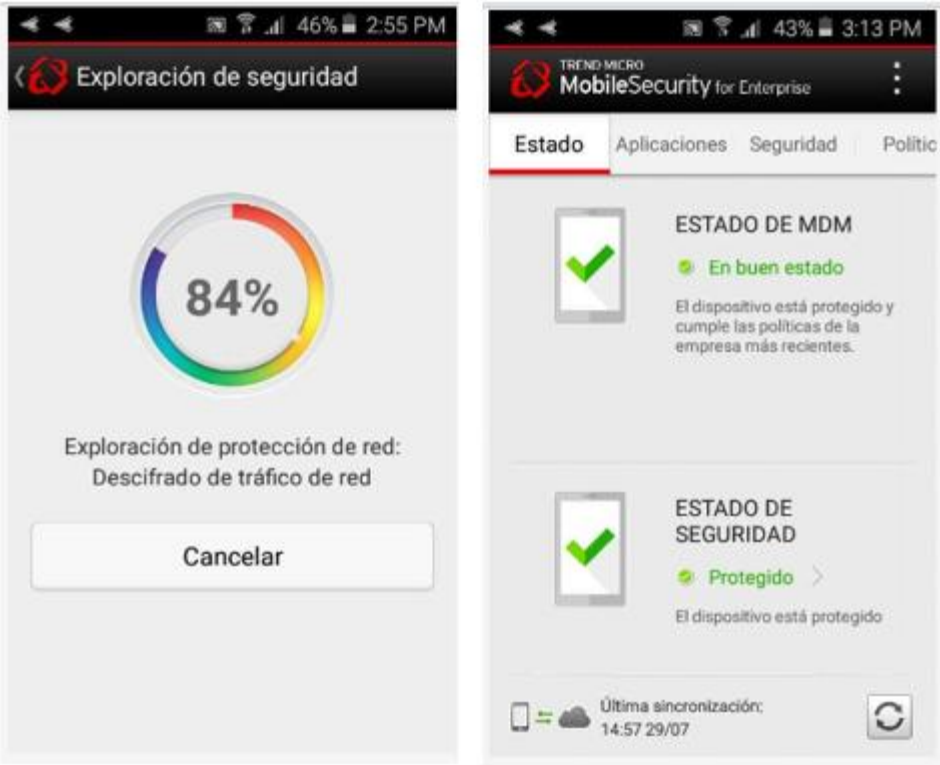


Figura 6-12. El Antivirus *Trend Micro* MobileSecurity no detecta la aplicación comprometida. (Fuente: Propia).

Resultado:

- Detección: La plataforma *antimalware* no detecta ninguna amenaza en el dispositivo móvil

Plataformas Antivirus social *Koodous*

Koodous, Escaneo de *WhatsApp* comprometido.

La *Figura 6-13* muestra el escaneo de la aplicación *WhatsApp* utilizando la plataforma *Koodous*.

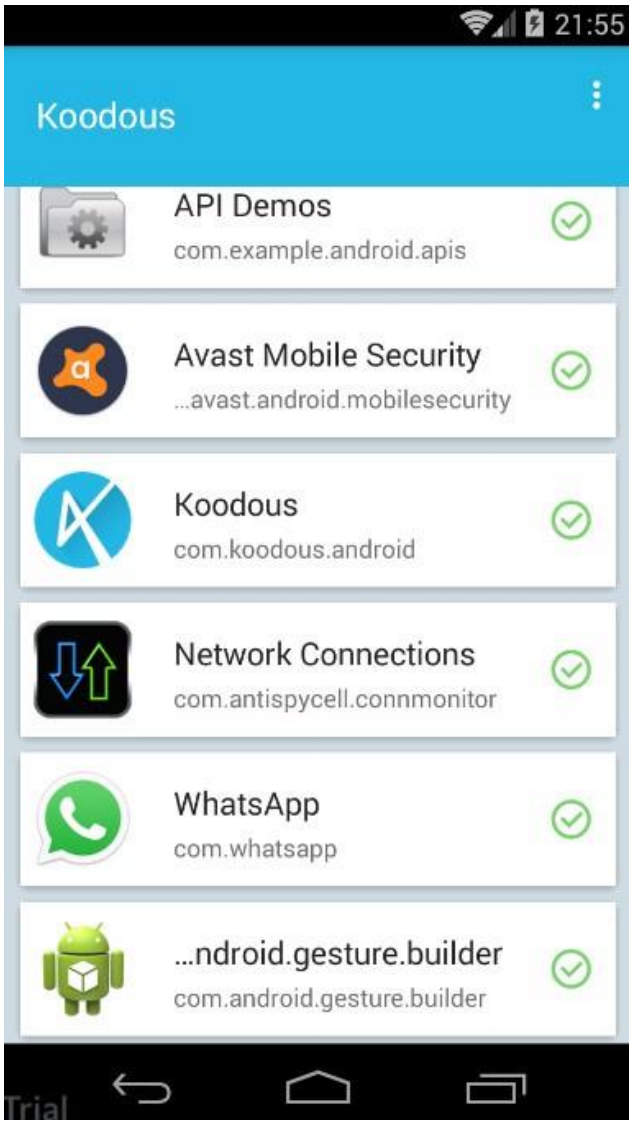


Figura 6-13. El antivirus Social *Koodous* no detecta la aplicación reempaquetada. (Fuente: Propia).

Resultados:

- Detección: El antivirus social *Koodous* no detecta ninguna amenaza en el dispositivo móvil

Plataforma desarrollada *CAM* (Control de Aplicaciones Móviles) en dispositivos móviles emulados

Escaneo de *WhatsApp* comprometida con *CCAM*

La *Figura 6-14* muestra el escaneo de la aplicación *WhatsApp* comprometida utilizando la plataforma desarrollada *CCAM*:

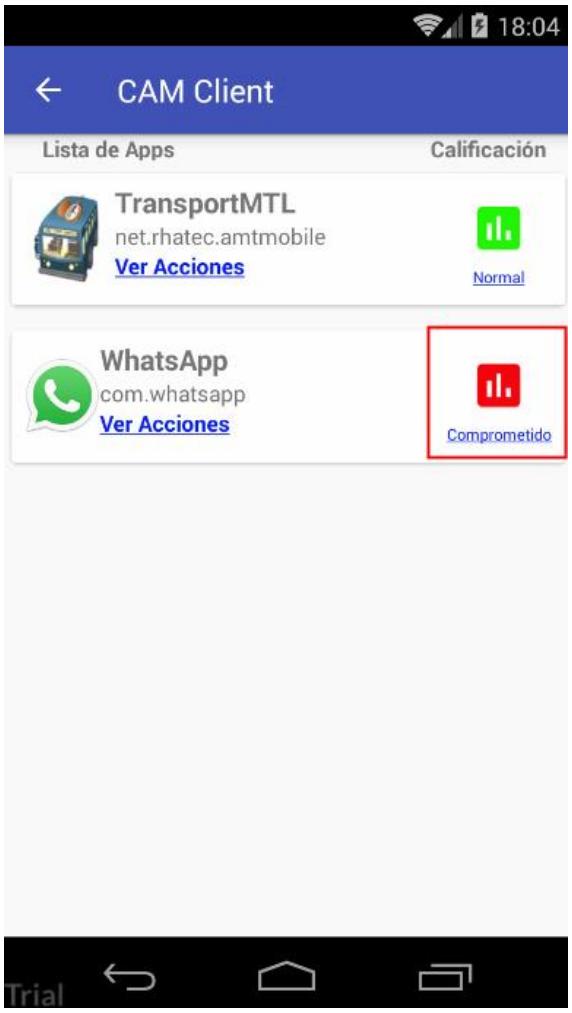


Figura 6-14. *CCAM* detecta la aplicación comprometida no detectada por las otras soluciones. (Fuente: Propia).

Resultados:

- Detección: Calificación de *WhatsApp* la aplicación está COMPROMETIDA.
- Notificación: Ver detalles de *IOCs.*
- Informes eficientes generados con *CAM, IOCs* detectados.

La *Figura 6-15* muestra la actividad de ver detalles de *IOCs* del cliente *CCAM.*

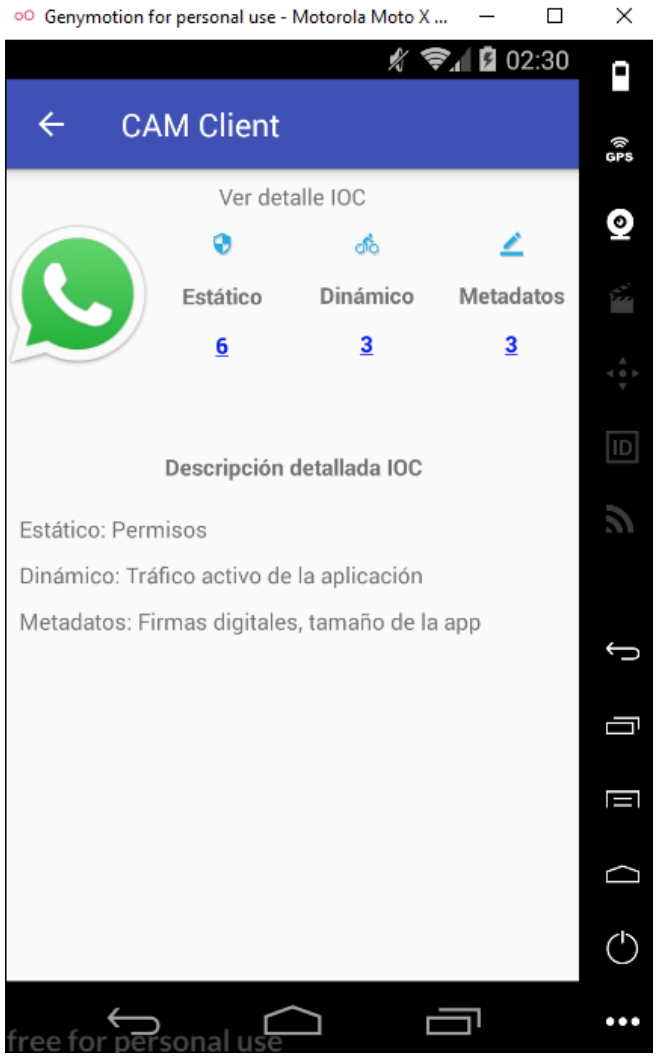


Figura 6-15. Informes eficientes generados por *CCAM.* (Fuente: Propia).

Resultado: detección de *IOCs*, Estáticos: 6, Dinámico: 3, Metadatos: 3

***IOCs* estáticos detectados**

Notificación:

- Permisos oficiales: Informa los eventos operativos oficiales que se configuraron en la base de datos de lista blanca del administrador.
- Permisos Detectados: Informa los eventos operativos detectados en la aplicación instalada.
- Permisos con problemas: Informa los *IOCs* detectados en la aplicación instalada al hacer la diferencia entre los eventos operativos oficiales y los *IOCs* de más detectados en la aplicación comprometida.

La *Figura 6-16* muestra el informe de *IOC* estáticos detectados y permisos con problemas.

Figura 6-16. Informe de *IOC* estáticos detectados y Permisos con problemas. (Fuente: propia).

Permisos con problemas clasificados con base a la clasificación de permisos de la página de desarrolladores de *Android*:

Android.permission.SET_WALLPAPER
Android.permission.READ_*SMS*
Android.permission.READ_CALL_LOG
Android.permission.CALL_PHONE
Android.permission.WRITE_SETTINGS
Android.permission.WRITE_CALL_LOG

IOCs dinámicos detectados

Notificación:

- Trafico: Informa las direcciones IPv4 e IPV6 solicitadas por la aplicación comprometida que no hacen parte del dominio del fabricante.

La *Figura 6-17* presenta el informe de *IOC* dinámicos con problemas.

Figura 6-17. Informe de *IOC* dinámicos con problemas. (Fuente: Propia).

La imagen muestra dos *IPs* privadas que no pertenecen al dominio de *WhatsApp*, se debe validar si la dirección *IP* pública corresponde a una nueva dirección del fabricante

***IOCs* Metadatos**

Notificación:

Firmas: Informa digitales y peso La *Figura 6-18* muestra el informe de *IOC* de metadatos con problemas.

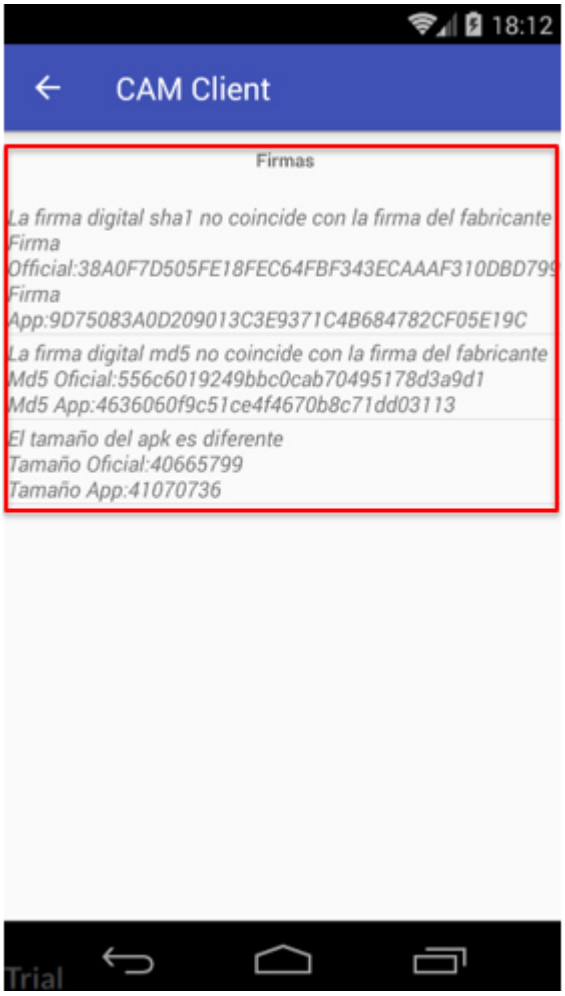


Figura 6-18. Informe de *IOC* de metadatos con problemas. (Fuente: Propia).

Se puede evidenciar que las firmas digitales y el peso de la aplicación comprometida no son iguales a los metadatos oficiales

Pruebas de detección en dispositivo móvil físico

Escaneo de Fabrica Dulce comprometido con *CCAM*

La *Figura 6-19* muestra el resultado del escaneo de la aplicación Fábrica dulce utilizando el cliente *CCAM*.

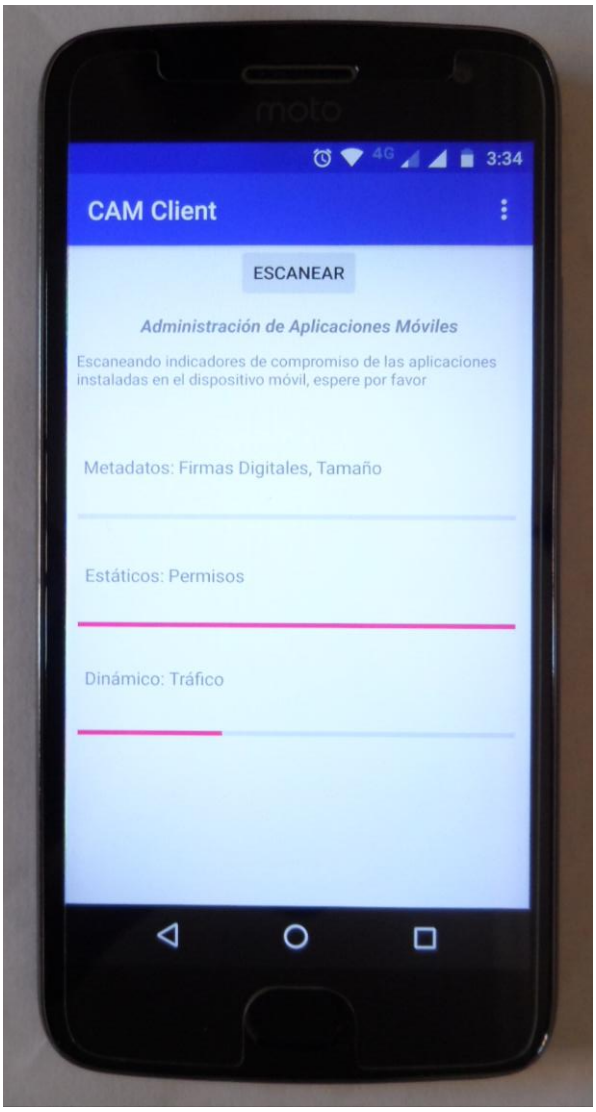


Figura 6-19. Escaneo de Fábrica Dulce con el cliente *CCAM.* (Fuente: Propia).

Resultado del escaneo de Fabrica Dulce comprometido con *CCAM*

La *Figura 6-20* muestra la detección de la aplicación comprometida Fabrica Dulce en el dispositivo físico

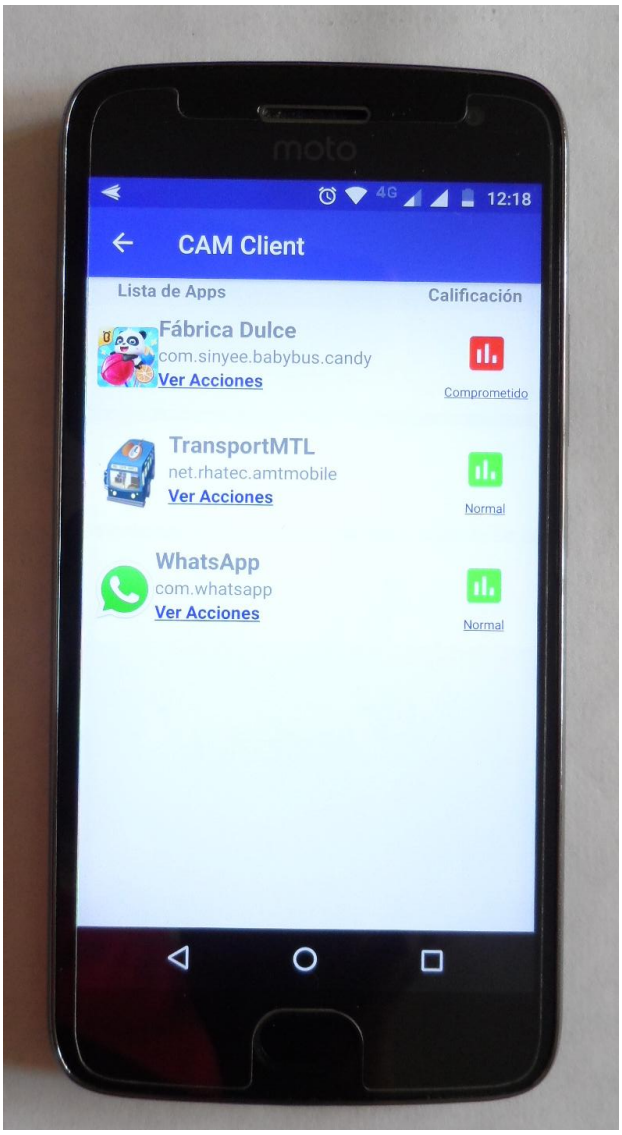


Figura 6-20. Detección de *CCAM* detecta la aplicación comprometida en el dispositivo físico. (Fuente: Propia).

Función de ver acciones

La *Figura 6-21* muestra el menú de acciones en el dispositivo.

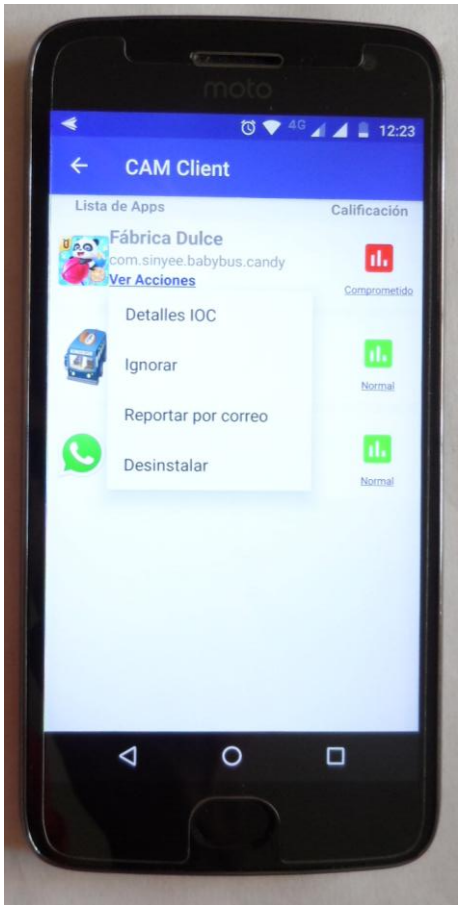


Figura 6-21. Menú de ver acciones en el dispositivo físico. (Fuente: Propia).

Resultados:

- Detección: Calificación de *WhatsApp* la aplicación está COMPROMETIDA.

Ver detalles de *IOCs*

Informes eficientes generados con *CAM*, *IOCs* detectados. La *Figura 6-22* muestra la interfaz de la función de **Ver detalle *IOC*** después de la detección de la aplicación *Fabrica Dulce* reempaquetada.

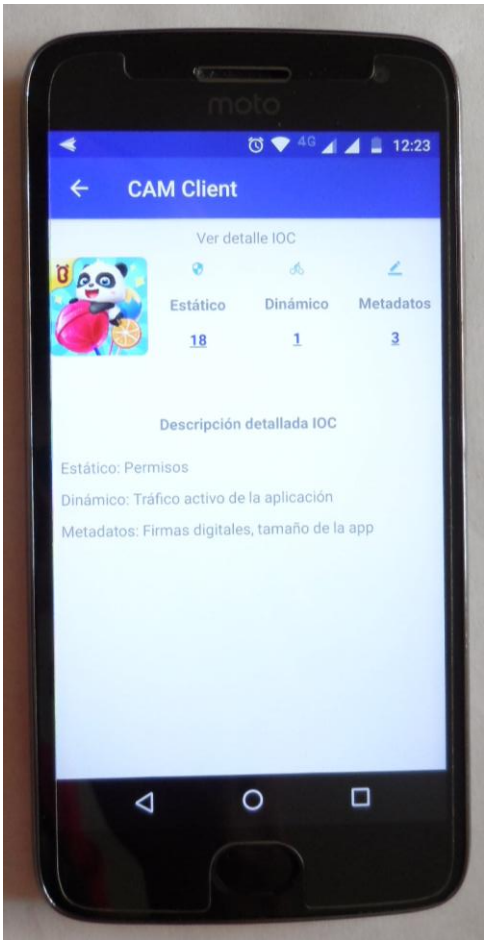


Figura 6-22. Interfaz Ver detalle *IOC* del cliente *CCAM* en el dispositivo físico. (Fuente: Propia).

Resultado: detección de *IOCs*, Estáticos: 6, Dinámico: 3, Metadatos: 3.

***IOCs* estáticos detectados**

Notificación:

- Permisos oficiales: Informa los eventos operativos oficiales que se configuraron en la base de datos de lista blanca del administrador.
- Permisos Detectados: Informa los eventos operativos detectados en la aplicación instalada.
- Permisos con problemas: Informa los *IOCs* detectados en la aplicación instalada al hacer la diferencia entre los eventos operativos oficiales y los *IOCs* de más detectados en la aplicación comprometida.

La *Figura 6-23* muestra las interfaces de *IOCs* Estaticos detectados y Permisos con problemas del cliente *CCAM* en el dispositivo fisico

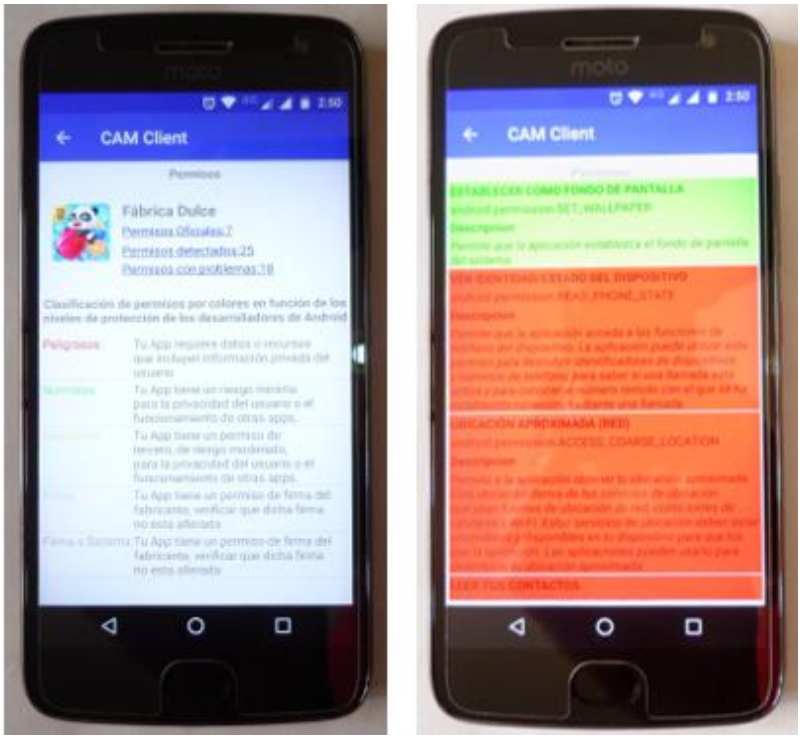


Figura 6-23. Interfaces de *IOC* estáticos detectados y Permisos con problemas en el dispositivo físico. (Fuente: Propia).

Permisos con problemas clasificados con base a la clasificación de permisos de la página de desarrolladores de *Android*.

Android.permission.SET_WALLPAPER
Android.permission.READ_PHONE_STAT
Android.permission.ACCESS_COARSE_LOCATION
Android.permission.READ_CONTACTS
Android.permission.WRITE_CALL_LOG
Android.permission.RECORD_AUDIO
Android.permission.RECEIVE_*SMS*
Android.permission.READ_*SMS*
Android.permission.WRITE_EXTERNAL_STORAGE
Android.permission.ACCESS_FINE_LOCATION
Android.permission.CAMERA
Android.permission.READ_CALL_LOG
Android.permission.SEND_*SMS*
Android.permission.RECEIVE_BOOT_COMPLETED
Android.permission.WRITE_CONTACTS
Android.permission.READ_EXTERNAL_STORAGE
Android.permission.CALL_PHONE

IOCs dinámicos detectados

Notificación:

- Trafico: Informa las direcciones *IPv4* e *IPV6* solicitadas por la aplicación comprometida que no hacen parte del dominio del fabricante.

La *Figura 6-24* muestra la interfaz de *IOC* dinámicos detectados en el dispositivo físico después de escanear Fabrica Dulce reempaquetada.

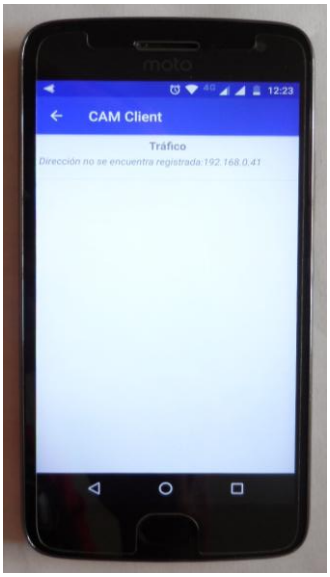


Figura 6-24. Informe de *IOC* dinámicos con problemas. (Fuente: Propia).

La imagen muestra una dirección *IP* privadas que no pertenecen al dominio de *WhatsApp*, se debe validar si la dirección *IP* pública corresponde a una nueva dirección del fabricante

IOCs Metadatos

Notificación:

- Firmas: Informa digitales y peso

La *Figura 6-25* muestra la pantalla de la actividad de *IOC* metadatos detectados en el dispositivo físico después de escanear Fabrica Dulce reempaquetada.

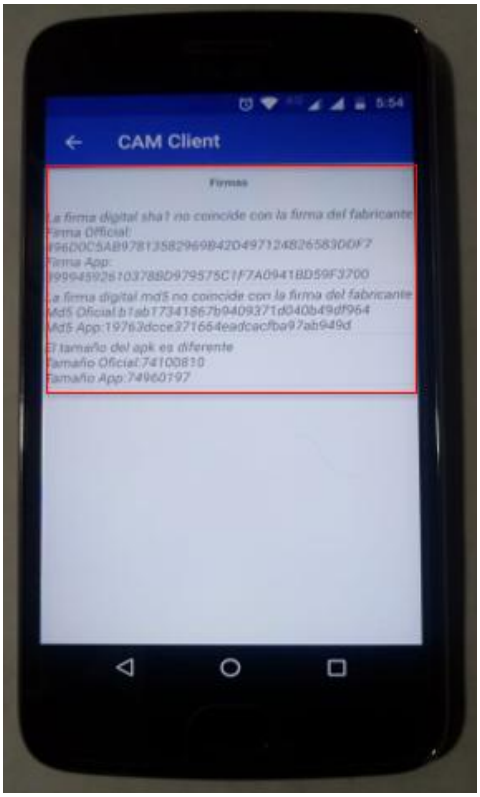


Figura 6-25*.* Informe de *IOC* de metadatos con problemas. (Fuente: Propia)*.*

Se puede evidenciar que las firmas digitales y el peso de la aplicación comprometida no son iguales a los metadatos oficiales

En las pruebas realizadas la plataforma *CAM* detectó los *IOCs* de las aplicaciones comprometidas instaladas y no generó falsos positivos con las otras aplicaciones

CAM también notificó informes eficientes que pueden alertar mejor a los usuarios finales, mostrándoles los permisos con problemas y su clasificación por colores según los desarrolladores de *Android*, las direcciones *IP* que no pertenecen al dominio del fabricante y las diferencias de las firmas digitales y su peso, así como el peso de la aplicación infectada instalada evidenciando su falta de integridad.

7 Conclusiones

La plataforma *CAM* permite detectar aplicaciones infectadas instaladas en el teléfono inteligente configuradas en la lista blanca del administrador, por medio de la correlación de eventos de *IOCs* híbridos de *Android* (Estáticos, Dinámicos y Firmas digitales).

La plataforma *CAM* desarrolla un sistema de notificaciones eficientes que pueden alertar, educar y mejorar la capacidad de respuesta del usuario final, quien usualmente no posee los conocimientos necesarios y por lo tanto es más vulnerable.

La ciberseguridad en dispositivos móviles es un campo bastante estudiado, pero en el que se puede innovar con nuevas técnicas de detección que mejoren las capas de seguridad de la plataforma para el usuario final.

Este proyecto propone la creación de comunidades y una estrategia de corresponsabilidad entre los desarrolladores de las aplicaciones móviles y los usuarios de la plataforma *CAM* - administradores y no administradores - para que los primeros publiquen los eventos operativos de las aplicaciones oficiales de las aplicaciones desarrollada, con el fin de que los usuarios administradores de la plataforma puedan alimentar la base de datos de lista blanca de la consola de administración *ACAM.* De esta manera, los usuarios no administradores pueden comprobar la integridad de sus aplicaciones con el cliente *CCAM* para dispositivos *Android* aprovechando los conocimientos de los administradores.

La plataforma *CAM* automatiza la creación de reglas *YARA* de los *IOCs* detectados en las aplicaciones comprometidas desde el dispositivo móvil del usuario final para que este las pueda compartir y reportar al administrador, de manera que los administradores puedan publicar las reglas creadas en plataformas similares que usa en este lenguaje, como por ejemplo *Koodous*.

8 Recomendaciones

Una primera recomendación es implementar la plataforma *CAM* como un proyecto de comunidad compuesta de administradores y usuarios finales no administradores de teléfonos inteligentes con sistema operativo *Android*, para que los administradores alimenten la base de datos de lista blanca con las aplicaciones más usadas, analicen las aplicaciones reportadas por los usuarios finales y que compartan conocimientos con estos últimos para que mejoren sus prácticas en temas de seguridad de la información y empiecen a dejar de ser el eslabón más débil de la cadena de seguridad.

Otra recomendación es implementar la plataforma *CAM* en un entorno corporativo que permita medir su alcance y determinar sus mejores usos.

También se recomienda levantar estadísticas que ayuden a medir el grado de conocimiento de los usuarios finales antes y después de usar la plataforma *CAM* con el fin de comprobar si las notificaciones eficientes cumplen con el objetivo de brindar conocimiento en seguridad de la información.

Durante las pruebas de detección con Antivirus gratuitos y no gratuitos diferentes de la plataforma *CAM*, se encontró que los a*ntimalware* no detectan la aplicación reempaquetada antes de ser instalada en el dispositivo móvil, por lo que finalmente el teléfono inteligente es infectado. En investigaciones futuras se puede desarrollar una tecnología de detección en tiempo real o que escanee las aplicaciones antes de ser instaladas.

Desarrollar módulos para la plataforma *CAM* que le permitan monitorear el tráfico de las aplicaciones instaladas en el dispositivo móvil, sin la dependencia de aplicaciones de terceros, como un módulo *VPN* que le permitiría capturar el tráfico, monitorearlo y bloquear direcciones *IP* que no pertenezcan al dominio de los fabricantes de la aplicación.

La técnica de detección de lista blanca de la plataforma *CAM* no debe generar falsos positivos si la base de datos de *ACAM* está bien administrada, pero en trabajos futuros se recomienda hacer un análisis de los falsos positivos que se pueden generar con las reglas *YARA* publicadas en otras plataformas similares como *Koodous*, para optimizar la sintaxis de las reglas que se van a intercambiar y así minimizar los falsos positivos.

También se recomienda realizar un análisis del uso de los recursos de los teléfonos inteligentes del cliente *CCAM*: memoria, procesador y uso de la batería, para contrastarlo con el uso de recursos de otros softwares de seguridad similares. Así será posible determinar las ventajas y desventajas de esta técnica detección con base al rendimiento de estos dispositivos móviles.

Por último, se recomienda aplicar metodologías de desarrollo seguro adecuadas para plataformas de seguridad a los componentes de la plataforma *CAM* e implementar buenas prácticas como comunicaciones cliente-servidor cifradas, desactivar la interface de administración de los usuarios finales, entre otras que le brinden a la plataforma una postura de seguridad adecuada para salvaguardar la información del dispositivo móvil.

A. Anexo Tabla 8-1 de Técnicas de detección de *malware* y Tabla 3 debilidades de las Técnicas de detección de *malware* en Teléfonos inteligentes con sistema operativo *Android* entre el 2013 y el 2018-1

La **Tabla 8-1.** Técnicas de detección de *malware* en *Android* entre el 2013 y 2018-1 resultado del estado del arte.

Tabla 8-1. Técnicas de detección de *malware* en *Android* entre el 2013 y 2018-1 resultado del estado del arte.
Fuente: Propia.

Técnicas de detección de *malware* en Teléfonos inteligentes con *Android* entre el 2013 y el 2018-1									
Artículos	***Machine Learning***	**Análisis de *malware* dinámico en servidores**	**Análisis de *malware* Estático en servidores**	**Análisis de *malware* Estático y/o dinámico en móviles**	**Análisis semántico**	**Análisis Forense**	**Análisis basado medidas de similitud**	**Análisis remoto de aplicaciones en *Google Play***	**Análisis basado en llamados a la *API***
(T. E. Wei et al., 2012) *Android Malware* Detection via a Latent *Network* Behavior Analysis							X		

(Y. Zhang et al., 2013) Vetting undesirable behaviors in *Android* aplicaciones with permission use analysis						X			
(Teufl et al., 2006) *Malware* detection by applying knowledge discovery processes to application *metadata* on the *Android* Market (*Google Play*)	X							X	
(Seo, Gupta, Sallam, Bertino, & Yim, 2014) Detecting mobile *malware* threats to homeland *security* through static analysis			X						
(M. Zhang, Duan, Yin, & Zhao, 2014) Semantics-Aware *Android Malware* Classification Using Weighted Contextual *API* Dependency Graphs	X				X				
(Du Yao, Wang Xiaoqing, 2014) A static *Android* malicious code detection method based on multi-*source* fusion			X						

(Wu, 2014) DroidDolphin: a Dynamic *Android Malware* Detection *Framework* Using Big Data and Machine Learning Wen-Chieh	X	X							
(Elish, Shu, Yao, Ryder, & Jiang, 2015) Profiling user-trigger dependence for *Android malware* detection					X				
(Jang, Kang, Woo, Mohaisen, & Kim, 2015) Andro-AutoPsy: Anti-*malware system* based on similarity matching of *malware* and *malware* creator-centric information							X		
(Kaushik & Jain, 2015) *Malware* Detection Techniques in *Android*	X	X	X						

(Laura Gheorghe, Bogdan Marin, Gary Gibson, Lucian Mogosanu, Razvan Deaconescu, Valentin-Gabriel Voiculescu, 2015) Smart *malware* detection on *Android*	X	X							
(Lin Ying-Dar, Lai Yuan-Cheng, Lu Chun-Nan, Hsu Peng-Kai, 2014) Three-phase behavior-based detection and classification of known and unknown *malware*	X								
(Talha, Alper, & Aydin, 2015) *APK* Auditor: Permission-based *Android malware* detection *system*			X				X		
(Liang, Might, & Van Horn, 2015) AnaDroid: *Malware* analysis of *Android* with user-supplied predicates					X				
(Han, Chen, Yan, Lizhi, & Zhang, 2015) A Real-time *Android Malware* Detection *System* Based on *Network* Traffic Analysis	X								

(Kate & Dhavale, 2015) Two Phase Static Analysis Technique for *Android Malware* Detection	X		X						
(Jung, Kim, & Cho, 2016) A study of *Android malware* detection techniques in virtual environment		X	X						
(Ali Alatwi, Oh, Fokoue, & Stackpole, 2016) *Android Malware* Detection Using Category-Based Machine Learning Classifiers	X	X							
(Somarriba, Zurutuza, Uribeetxeberria, Delosieres, & Nadjm-Tehrani, 2016) Detection and Visualization of *Android Malware* Behavior									X

(Z. Wang, Li, Yuan, Guan, & Xue, 2016) DroidChain: A novel *Android malware* detection method based on behavior chains					X				
(E. M. B. Karbab, Debbabi, & Mouheb, 2016) Fingerprinting *Android* packaging: Generating DNAs for *malware* detection					X				
(Malik & Kaushal, 2016) CREDROID: *Android Malware* Detection by *Network* Traffic Analysis							X		
(E. B. Karbab, Debbabi, Derhab, & Mouheb, 2016) Cypider: Building Community-Based Cyber-Defense Infrastructure for *Android Malware* Detection							X		
(Chen, Hou, & Ye, 2017) SecureDroid: Enhancing *Security* of Machine Learning-based Detection against Adversarial *Android Malware* Attacks (2017)	X								

(Singh & Hofmann, 2017) Dynamic *Permissions*-based *Android malware* detection using machine learning techniques	X								
(Leeds et al., 2017) A Comparison of Features for *Android Malware* Detection									X
(Singh & Hofmann, 2017) Dynamic Behavior Analysis of *Android* Applications for *Malware* Detection	X								X

(Tan Cam & Hong Phuoc, 2017) NeSeDroid—*Android Malware* Detection Based on *Network* Traffic and Sensitive Resource Accessing		X	X						
(X. Wang, Zhang, Su, & Li, 2017) Mlifdect: *Android Malware* Detection based on Parallel Machine Learning and Information Fusion	X		X						
(John & Vinod, 2018) Statistical approach using meta features for *Android malware* detection *system*	X		X						
(Sugunan, Kumar, & Dhanya, 2018) Static and Dynamic Analysis for *Android Malware* Detection (2018)		X	X						
(Zhu et al., 2017) DroidDet: Effective and robust detection of *Android malware* using static analysis along with rotation forest model			X						

(Kabakus & Dogru, 2018) An in-depth analysis of *Android malware* using hybrid techniques		X	X						

Debilidades de las Técnicas de detección de *malware* en Teléfonos inteligentes con *Android* entre el 2013 y el 2018-1

La **Tabla 8-2**. Debilidades de las Técnicas de detección de *malware* en *Android* entre el 2013 y 2018-1 resultado del estado del arte.

Tabla 8-2. Debilidades de las Técnicas de detección de *malware* en *Android* entre el 2013 y 2018-1 resultado del estado del arte.
Fuente Propia.

Debilidades de las Técnicas de detección de *malware* en Teléfonos inteligentes con *Android* entre el 2013 y el 2018-1								
Artículos	**Requiere por lo menos de una muestra de *malware* para generar una detección**	**Requieren de conocimientos avanzados**	**Se requiere de un administrador**	**No se realiza detección en tiempo real**	**El análisis se hace entornos ficticios**	**Depende de grandes cantidades de información e infraestructura robusta**	**Solamente funciona con aplicaciones de la tienda de *Google* (*Google Play*)**	**No incluye análisis dinámico**

(T. E. Wei et al., 2012) *Android Malware* Detection via a Latent *Network* Behavior Analysis	X							
(Y. Zhang et al., 2013) Vetting undesirable behaviors in *Android* aplicaciones with permission use analysis		X		X				
(Teufl Peter, Ferk Michaela, Fitzek Andreas, Hein Daniel, Kraxberger Stefan, 2014) *Malware* detection by applying knowledge discovery processes to application *metadata* on the *Android* Market (*Google Play*)	X						X	

(Seo et al., 2014) Detecting mobile *malware* threats to homeland *security* through static analysis	X	X						X
(M. Zhang et al., 2014) Semantics-Aware *Android Malware* Classification Using Weighted Contextual *API* Dependency Graphs		X						X
(Du Yao, Wang Xiaoqing, 2014) A static *Android* malicious code detection method based on multi-*source* fusion		X						X
(Wu, 2014) DroidDolphin: a Dynamic *Android Malware* Detection *Framework* Using Big Data and Machine Learning	X					X		

(Elish et al., 2015) Profiling user-trigger dependence for *Android malware* detection (2015)		X						X
(Jang et al., 2015) Andro-AutoPsy: Anti-*malware system* based on similarity matching of *malware* and *malware* creator-centric information (2015)						X	X	X
(Kaushik & Jain, 2015) *Malware* Detection Techniques in *Android*	X		X					

(Laura Gheorghe, Bogdan Marin, Gary Gibson, Lucian Mogosanu, Razvan Deaconescu, Valentin-Gabriel Voiculescu, 2015) Smart *malware* detection on *Android*	X	X				X		
(Lin Ying-Dar, Lai Yuan-Cheng, Lu Chun-Nan, Hsu Peng-Kai, 2014) Three-phase behavior-based detection and classification of known and unknown *malware*	X							X
(Talha et al., 2015) *APK* Auditor: Permission-based *Android malware* detection *system*		X	X			X		X
(Liang et al., 2015) AnaDroid: *Malware* analysis of *Android* with user-supplied predicates		X						X

(Han et al., 2015) A Real-time *Android Malware* Detection *System* Based on *Network* Traffic Analysis	X				X			
(Kate & Dhavale, 2015) Two Phase Static Analysis Technique for *Android Malware* Detection	X							X
(Jung et al., 2016) A study of *Android malware* detection techniques in virtual environment		X	X		X			
(Ali Alatwi et al., 2016) *Android Malware* Detection Using Category-Based Machine Learning Classifiers	X							

(Somarriba et al., 2016) Detection and Visualization of *Android Malware* Behavior		X						X
(Z. Wang et al., 2016) DroidChain: A novel *Android malware* detection method based on behavior chains		X						X
(E. M. B. Karbab et al., 2016) Fingerprinting *Android* packaging: Generating DNAs for *malware* detection								X

(Malik & Kaushal, 2016) CREDROID: *Android Malware* Detection by *Network* Traffic Analysis						X		
(E. B. Karbab et al., 2016) Cypider: Building Community-Based Cyber-Defense Infrastructure for *Android Malware* Detection						X		
(Chen et al., 2017) SecureDroid: Enhancing *Security* of Machine Learning-based Detection against Adversarial *Android Malware* Attacks	X							

(Mahindru & Singh, 2017) Dynamic *Permissions*-based *Android malware* detection using machine learning techniques	X	X						X
(Leeds et al., 2017) A Comparison of Features for *Android Malware* Detection		X						X
(Singh & Hofmann, 2017) Dynamic Behavior Analysis of *Android* Applications for *Malware* Detection	X	X						X
(Tan Cam & Hong Phuoc, 2017) NeSeDroid—*Android Malware* Detection Based on *Network* Traffic and Sensitive Resource Accessing		X	X					

(X. Wang et al., 2017) Mlifdect: *Android Malware* Detection based on Parallel Machine Learning and Information Fusion	X							X
(John & Vinod, 2018) Statistical approach using meta features for *Android malware* detection *system*	X	X				X		

(Sugunan et al., 2018) Static and Dynamic Analysis for *Android Malware* Detection		X			X			
(Zhu et al., 2017) DroidDet: Effective and robust detection of *Android malware* using static analysis along with rotation forest model								X
(Kabakus & Dogru, 2018) An in-depth analysis of *Android malware* using hybrid techniques					X	X		

B. Anexo Incidentes de *malware* en Teléfonos inteligentes con *Android* entre el 2013 y el 2018-1

(Pastor, 2013) Escribió en el Webblog *Xatakamovi*l a partir de la última gran vulnerabilidad detectada en *Android* y con base a las pruebas realizadas por *AV Comparatives, AV-Test* y *PC Security LABS*, publicó ¿Sirven de algo los antivirus para móviles?

(Ludwig, 2015) Según informes de *Incibe*, *Hispasec*, *Sophos*, *McAfee*, *Kaspersky*, *Trend Micro*, *Symantec* y *Cisco* en otras entidades y fabricantes de *software antimalware* entre los años 2014 y 2017, *Android* se ha convertido en la plataforma móvil más atacada por parte de los ciber criminales.

(PandaLabs, 2015) En el año 2014 *Panda Security* predijo que en el año siguiente las amenazas para dispositivos móviles con sistema operativo *Android* iban a aumentar de manera significativa, siendo esta, una predicción bastante acertada de lo que sucede en la actualidad.

(Giusto Bilić, 2015) *WeLiveSecurity*, blog de noticias perteneciente al fabricante *ESET*, informa sobre el crecimiento en los tipos de amenazas para teléfonos móviles con sistema operativo *Android*, siendo los troyanos, el código malicioso de más popularidad en año 2015.

La *Figura 8-1* muestra las familias de *malware* para *Android* que más crecieron en 2015 que más crecieron en 2015.

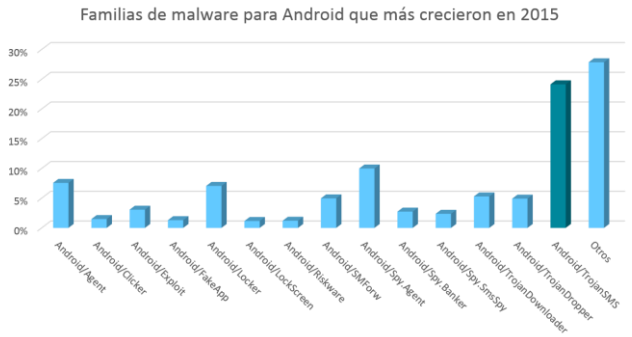


Figura 8-1 Familias de *malware* para *Android* que Más Crecieron en 2015. (Fuente : Giusto Bilić, 2015).

(Emm et al., 2016) escritores pertenecientes a *Securelist*, blog de noticias perteneciente al fabricante *Kaspersky Lab*, informan que en el año 2016 los troyanos aún tenían una gran cantidad de popularidad, superados por el código malicioso de la familia *Adware*.

La muestra la distribución de nuevo *malware* para móviles por familia en los dos primeros Q de 2016.

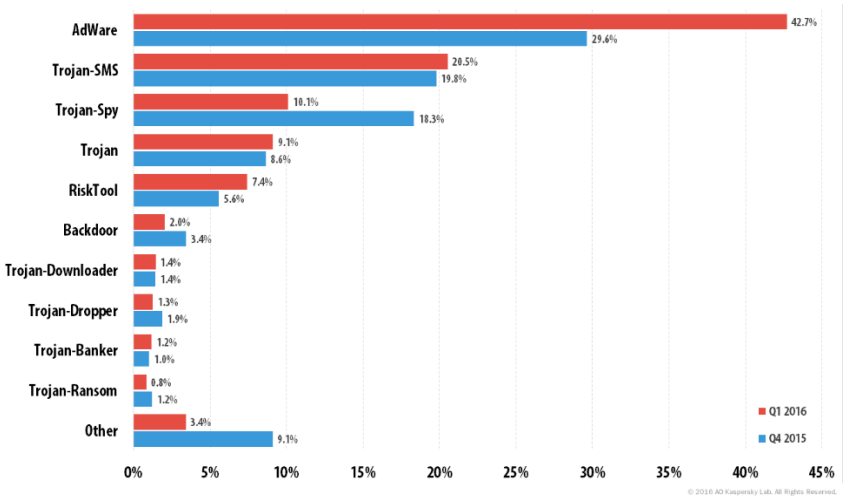


Figura 8-2. Distribución de nuevo malware para móviles por familia (Q1 2016 y Q2 2016). (Fuente: Emm et al., 2016)

Basado en estudios realizados por (Benzmüller, 2017) perteneciente al fabricante *G Data Security*, se puede evidenciar un aumento en la cantidad de especímenes de *malware* existentes entre el año 2015 y 2016 y aún más considerable en el año 2017 tal cual como lo predijo *Panda Security* en el año 2015.

La *Figura 8-3* muestra el numero nuevas variantes de *malware* entre el periodo 2007 y el 2017.

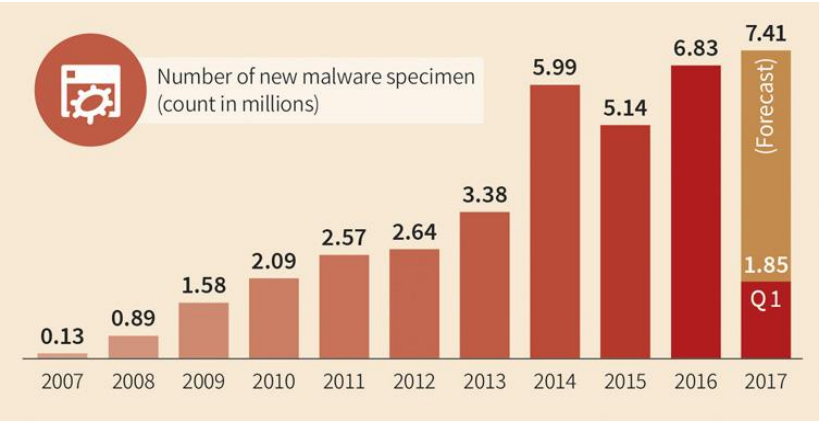


Figura 8-3. Número de nuevos especímenes de *malware* (Contar en Millones). (Fuente: Benzmüller, 2017).

(Emm et al., 2016) *Android* cerró con 735 fallos (error de *software*), un 40,5 por ciento más que en 2016, donde se registraron 523 vulnerabilidades (debilidad del sistema informático) reportadas. Estas cifras, según la firma de ciberseguridad *ESET*, posicionan a *Android* como el sistema operativo con más amenazas durante el 2017.

(Benzmüller, 2017) El 27 por ciento de las fallas que podría implicar la ejecución de código malicioso y el 45 por ciento de estos correspondieron a una criticidad alta, lo que implica que podría tener un grave impacto para la información almacenada en el equipo.

El *malware*, por su parte, se mantuvo durante los últimos dos años en aproximadamente 300 nuevas variantes mensuales. A su vez, la cantidad de detecciones de aplicaciones maliciosas decayó un 47,24 por ciento con respecto a 2016.

No obstante, el informe detalla que las amenazas en la *Play Store* no han dejado de ser frecuentes; por el contrario, son comunes los casos de troyanos disfrazados de *aplicaciones* benignas que logran saltar los controles de seguridad de *Google* para afectar a los usuarios.

Según (TECNÓSFERA, 2018) Los países de América Latina con mayor detección de '*malware*' fueron México (32 por ciento), Brasil (12 por ciento) y Colombia (9 por ciento). A escala mundial, dicha tendencia se concentró en Irán (16 por ciento), Rusia (11 por ciento) y Ucrania (7 por ciento).

C. Anexo: Laboratorio de infección de aplicaciones (*WhatsApp*)

1. Se descarga la aplicación oficial que se va a infectar, en este caso *WhatsApp* versión 2.18.223 desde la *PlayStore*.

La *Figura 8-4* muestra el archivo *WhatsApp* versión 2.18.223 oficial.

Figura 8-4 *APK* de *WhatsApp.* (Fuente: Propia).

2. procedemos a la instalación del entorno de pruebas para la infección de *WhatsApp y Fabrica Dulce* con *Meterpreter* de *Metasploit.*

Se instala *Ubuntu* 16.04 *LTS* en una máquina virtual bajo emulada con de *VirtualBox* Ver.5.2.16.

La *Figura 8-5* muestra la instalación de *Ubuntu* versión 16.04 *LTS.*

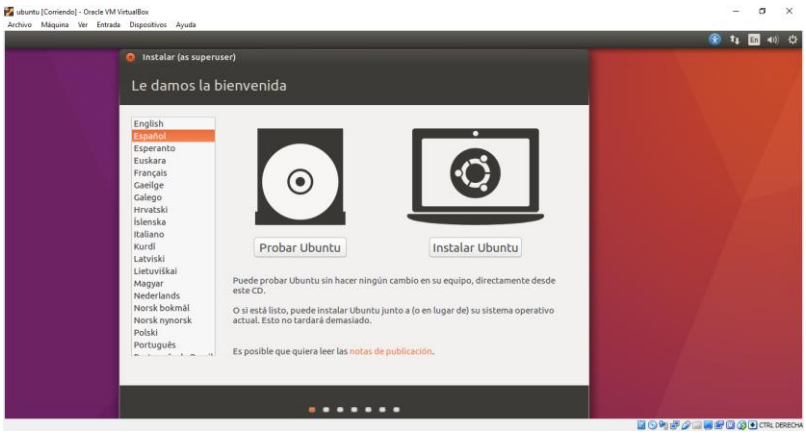


Figura 8-5. Instalación de *Ubuntu* en una máquina de *Virtualbox*, selección del lenguaje de instalación. (Fuente: Propia).

Se selecciona el idioma del sistema operativo en este caso es español

La *Figura 8-6* muestra los parámetros de instalación del sistema operativo *Ubuntu*

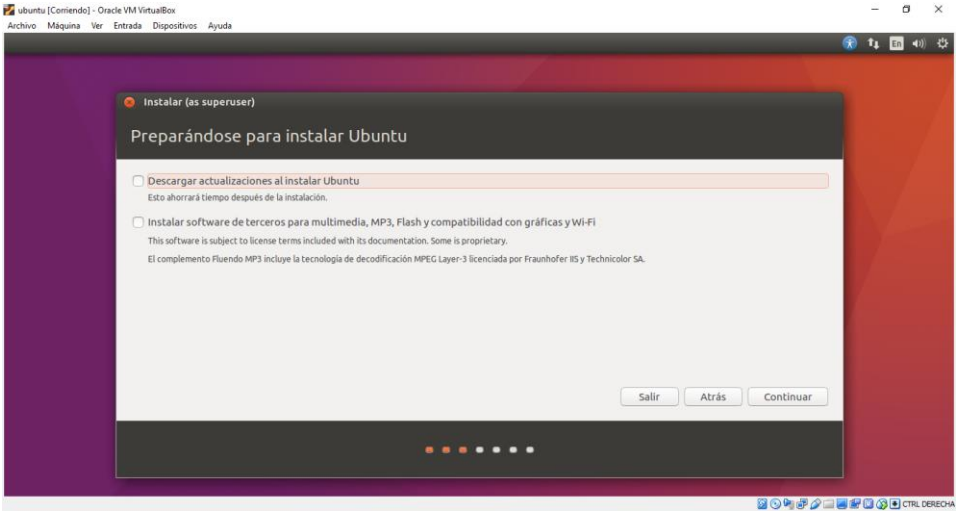


Figura 8-6. Selección de parámetros de instalación de sistema operativo *Ubuntu.* (Fuente: Propia).

3. Actualizar el sistema operativo, seleccionando la opción “Descarga actualizaciones” al instalar *Ubuntu*

La *Figura 8-7* muestra los parámetros de instalación del sistema operativo *Ubuntu*

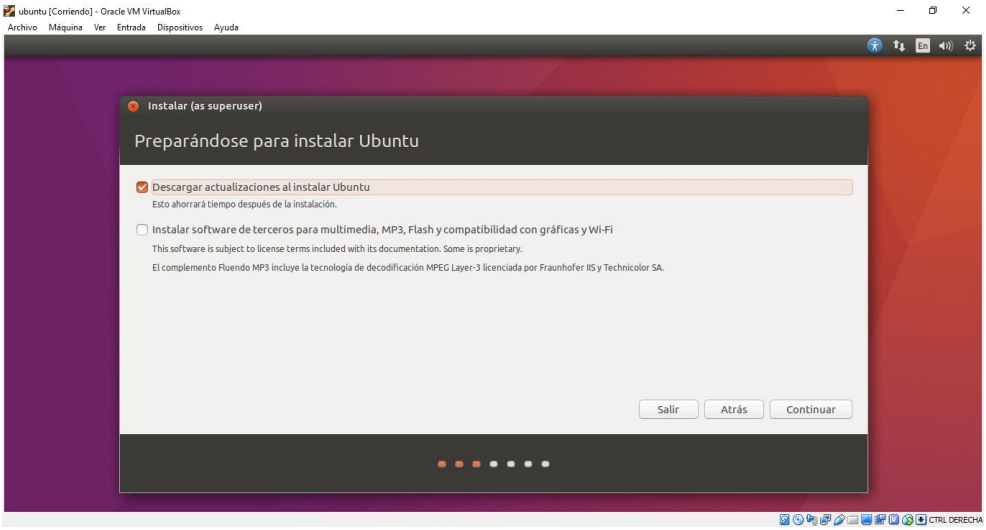


Figura 8-7. Parámetros seleccionados para esta instalación. (Fuente: Propia).

Clic en continuar

4. Una vez finalizada la instalación de *Ubuntu* 16.04 *LTS* se descarga el paquete *Apktool*, seguir el paso a paso de la *URL*

Desde la página: https://ibotpeaches.github.io/Apktool/install/

La *Figura 8-8* muestra la instalación de *Apktool.*

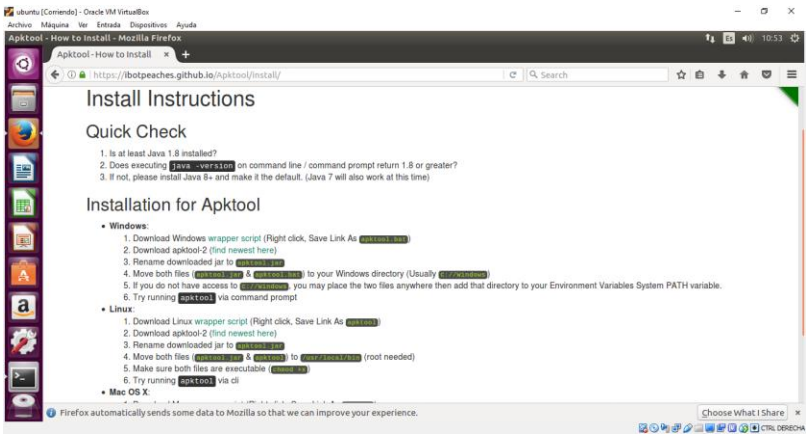


Figura 8-8. Instalación de Apktool. (Fuente: Propia).

Desde la consola de comandos instalamos todos los paquetes adicionales necesarios para la infección. Paquetes usados:

-Paquete *JDK* de *Java* por medio del comando: *sudo apt install open-jdk-headless*
-Paquete para descompilar y compilar la aplicación después de infectada por medio del comando:

sudo apt install I zipalign

La *Figura 8-9* muestra la instalación de *Zipaling* en *Ubuntu.*

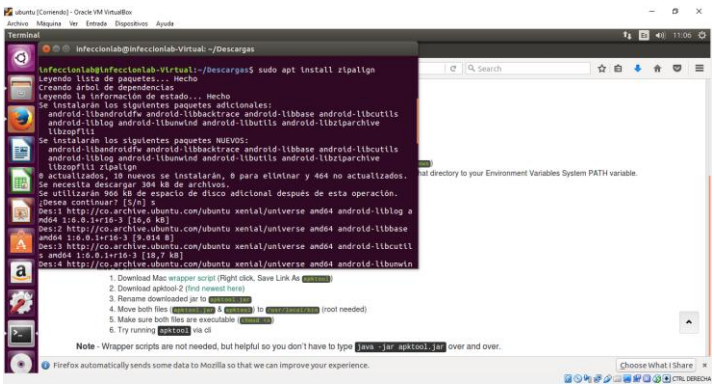


Figura 8-9. Instalación de *Zipalign.* (Fuente: Propia).

La *Figura 8-10* muestra la instalación de *OpenJava* en *Ubuntu.*

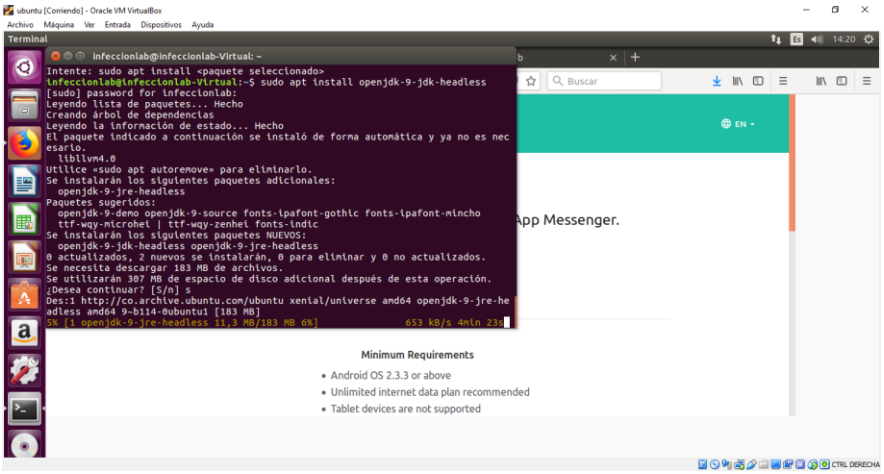


Figura 8-10. Instalación de *OpenJava.* (Fuente: Propia).

La imagen muestra cuando se ejecuta el comando sudo *install apt openjdk-9-jdk-headless*

5. Se ejecuta la siguiente línea de comandos desde la consola

Msfvenom -x /home/infeccionlab/Descargas/WhatsApp.APK -p Android/Meterpreter/reverse_tcp LHOST=192.168.0.14 LPORT=444 O-WhatsApp_infected.APK

La *Figura 8-11* muestra cuando se ejecuta el proceso de infección, en la línea de comandos se puede ver el directorio donde está el archivo *APK* para ser infectado la dirección *IP* a la cual se va a conectar el *Meterpreter* y el puerto, además como se va a llamar el nuevo archivo infectado

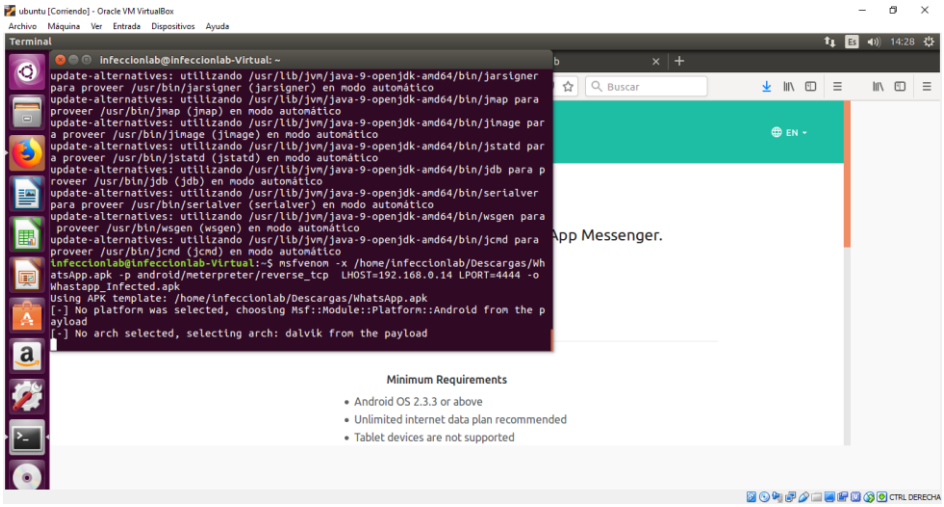


Figura 8-11. Infección de la *APK* con la carga útil del *malware.* (Fuente: Propia).

6. Una vez terminado el proceso de infección se creará un archivo nuevo en el directorio asignado anteriormente por medio de la línea de comandos.

La *Figura 8-12* muestra el archivo *WhatsApp_Infected.APK* reempaquetado con *Meterpreter* para *Android.*

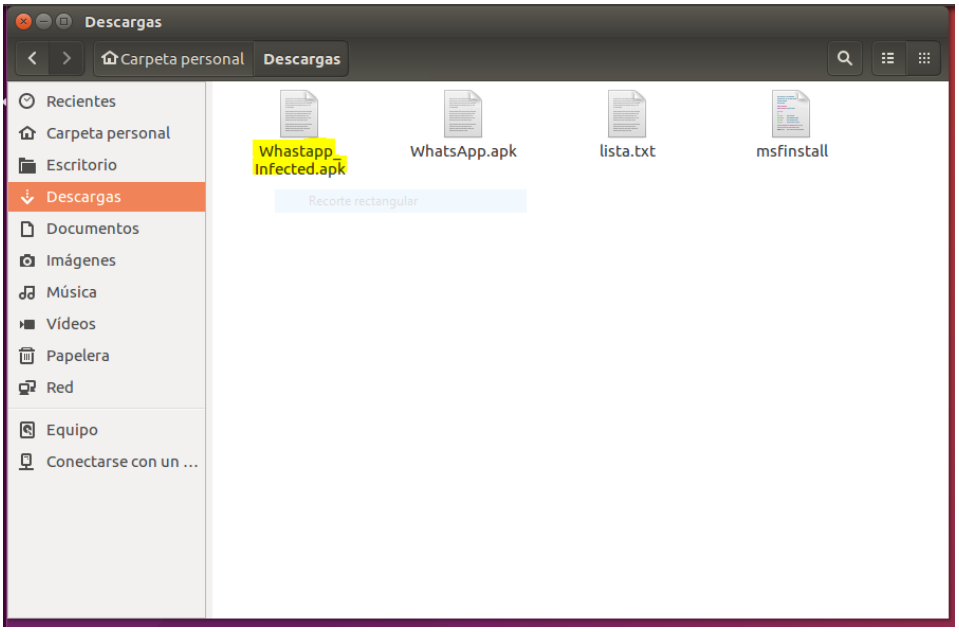


Figura 8-12. Archivo resultante luego de concluirse el proceso de infección. (Fuente: Propia).

Una vez instala la aplicación infectada con *Meterpreter* en el dispositivo móvil, el atacante puede ejecutar comandos, obtener información, manipular procesos del sistema y finalizarlos, los datos son trasmitidos vía *SSL*, lo cual significa que la información viaja encriptada y no es fácilmente detectada por los antivirus convencionales.

9 Referencias

Ali Alatwi, H., Oh, T., Fokoue, E., & Stackpole, B. (2016). Android Malware Detection Using Category-Based Machine Learning Classifiers. *Proceedings of the 17th Annual Conference on Information Technology Education - SIGITE '16*, 54–59. https://doi.org/10.1145/2978192.2978218

Alvarez, V. M. (2014). YARA Documentation. Retrieved from https://media.readthedocs.org/pdf/yara/latest/yara.pdf

Android Developers. (2018). Firmar tu aplicación. Retrieved from https://developer.android.com/studio/publish/app-signing?hl=es

Arp, A., Munyaneza, J. E., Crosslin, J. M., Trumble, J., & Bextine, B. (2014). A global comparison of Bactericera cockerelli (Hemiptera : Triozidae) microbial communities. *Environmental Entomology*, *43*(2), 344–352. https://doi.org/10.1603/EN13256

Benzmüller, R. (2017). Malware trends 2017. Retrieved from https://www.gdatasoftware.com/blog/2017/04/29666-malware-trends-2017

Betancur, O., & Eraso, S. (2015). *Seguridad en dispositivos móviles*. Universidad Veracruzana. Retrieved from http://stadium.unad.edu.co/preview/UNAD.php?url=/bitstream/10596/3614/1/59836994.pdf%0Ahttp://cdigital.uv.mx/handle/123456789/32348

Bläsing, T., Batyuk, L., Schmidt, A.-D., Camtepe, A., & Albayrak, S. (2010). An Android Application Sandbox System for Suspicious Software Detection. In *5th International Conference on Malicious and Unwanted Software* (pp. 55–62).

Burguera, I., Zurutuza, U., & Nadjm-Tehrani, S. (2011). Crowdroid: Behavior-Based Malware Detection System for Android. *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices - SPSM '11*, 15. https://doi.org/10.1145/2046614.2046619

Camargo, L. M. P., Galindo, J. C. A., & Vega, J. J. C. (2013). Seguridad En Dispositivos Móviles Con Sistemas Operativos Android Y Ios. *Tecnología Investigación y Academia*, *1*(2), 42–59. Retrieved from http://revistas.udistrital.edu.co/ojs/index.php/tia/article/view/4312

CCN, C. C. nacional. (2013). Indicadores de compromiso (IOC).

Chen, L., Hou, S., & Ye, Y. (2017). SecureDroid. *Proceedings of the 33rd Annual Computer Security Applications Conference on - ACSAC 2017*, 362–372. https://doi.org/10.1145/3134600.3134636

Corporation, M. (2012). Sophisticated New Android Trojan Found in Wild. *Geinimi*.

Davi, L., Dmitrienko, A., Sadeghi, A. R., & Winandy, M. (2011). Privilege Escalation Attacks on Android. *Springer Berlin Heidelberg*, *6531*, 346–360. Retrieved from http://link.springer.com/10.1007/978-3-642-18178-8_30

de Matías García, L. (2013). *Gestión de riesgo en dispositivos Android basada en eliminación de vulnerabilidades y detección de contextos*. Universidad Carlos III de Madrid. Retrieved from http://e-archivo.uc3m.es/handle/10016/18529

Delloite. (2017). Consumo móvil en Colombia. Los móviles prueban ser indispensables en un mundo "siempre" conectado. *Deloitte*, 1–78. Retrieved from https://www2.deloitte.com/content/dam/Deloitte/co/Documents/technology-media-telecommunications/Consumo movil(VF1).pdf

Desnos, A., & Gueguen, G. (2011). Android: From reversing to decompilation. *Proc. of Black Hat Abu Dhabi*, 1–24. Retrieved from http://media.blackhat.com/bh-ad-11/Desnos/bh-ad-11-DesnosGueguen-Andriod-Reversing_to_Decompilation_WP.pdf

Dong-Jie, W., Ching-Hao, M., Hahn-Ming, L., Kuo-Ping, W., & I, A. P. (2012). Droidmat: Android Malware Detection Through Manifest and Tracing. *In Proceedings of the Seventh Asia Joint Conference on Information Security*.

Du Yao, Wang Xiaoqing, W. J. (2014). A static Android malicious code detection method based on multi-source fusion. *International Journal of Applied Engineering Research*, *9*(22), 5968–5974. https://doi.org/10.1002/sec

Elenkov, N. (2015). *Android Security Internals*. *Network Security* (Vol. 2015). https://doi.org/10.1016/S1353-4858(15)30046-5

Elish, K. O., Shu, X., Yao, D., Ryder, B. G., & Jiang, X. (2015). Profiling user-trigger dependence for Android malware detection. *Computers and Security*, *49*(540), 255–273. https://doi.org/10.1016/j.cose.2014.11.001

Emm, D., Unuchek, R., Garnaeva, M., Ivanov, A., Makrushin, D., & Sinitsyn, F. (2016). IT THREAT EVOLUTION IN Q2 2016 IT threat evolution in Q2 2016. Overview. Retrieved from https://securelist.com/files/2016/08/Kaspersky_Q2_malware_report_ENG.pdf

ESET. (2014). *¿Cómo configurar tu Android de la forma más segura?* Retrieved from https://www.welivesecurity.com/wp-content/uploads/2014/07/guia_seguridad_android_eset.pdf

Feizollah, A., Anuar, N. B., Salleh, R., & Amalina, F. (2015). Comparative Evaluation of Ensemble Learning and Supervised Learning in Android Malwares Using Network-Based Analysis. *Springer International Publishing*, *315*, 1025–1035. Retrieved from http://link.springer.com/10.1007/978-3-319-07674-4_95

Feizollah, A., Anuar, N. B., Salleh, R., & Wahab, A. W. A. (2015). A review on feature selection in mobile malware detection. *Digital Investigation the International Journa...*, *13*(C), 22–37. Retrieved from http://dl.acm.org/citation.cfm?id=2795791

Gartner, I. (2018). Gartner Says Worldwide Sales of Smartphones Returned to Growth in First Quarter of 2018. Retrieved from https://www.gartner.com/newsroom/id/3876865

Giusto Bilić, D. (2015). ¿Cómo saber si tu teléfono ha sido comprometido? *WeLiveSecurity*, 1–5. Retrieved from https://www.welivesecurity.com/la-es/2015/11/23/como-saber-tu-telefono-ha-sido-comprometido/

Google. (2018). Paneles de control | Android Developers. Retrieved from https://developer.android.com/about/dashboards/index.html?hl=es-419

Google Play, G. I. (n.d.-a). Network connections. Retrieved from https://play.google.com/store/apps/details?id=com.antispycell.connmonitor

Google Play, G. I. (n.d.-b). Package info. Retrieved from https://play.google.com/store/apps/details?id=com.osp.pmanagerinfo

Gupta, S. (2017). Genymotion vs Android Emulator. *Plight of Byte*. Retrieved from https://www.plightofbyte.com/android/2017/09/03/genymotion-vs-android-emulator/

Han, H., Chen, Z., Yan, Q., Lizhi, P., & Zhang, L. (2015). A Real-time Android Malware Detection System Based on Network Traffic Analysis. *Algorithms and Architectures for Parallel Processing*, *9530*, 504–516. https://doi.org/10.1007/978-3-319-27137-8_37

Heras. C, I., & Sierra. L, D. (2015). Sistema de Deteccíon de Malware en Android.

Hou, O. (2012). A Look at Google Bouncer. *TrendLabs SECURITY INTELLIGENCE Blog*. Retrieved from https://blog.trendmicro.com/trendlabs-security-intelligence/a-look-at-google-bouncer/

ISO/IEC. (2005). Iso/Iec 17799. *Control*, 1–170.

Jang, J. W., Kang, H., Woo, J., Mohaisen, A., & Kim, H. K. (2015). Andro-AutoPsy: Anti-malware system based on similarity matching of malware and malware creator-centric information. *Digital Investigation*, *14*, 17–35. https://doi.org/10.1016/j.diin.2015.06.002

John, M. M., & Vinod, P. (2018). *Statistical approach using meta features for android malware detection system*. *Advances in Intelligent Systems and Computing* (Vol. 729). Springer Singapore. https://doi.org/10.1007/978-981-10-8536-9_27

Jung, H. M., Kim, K.-B., & Cho, H.-J. (2016). A study of android malware detection techniques in virtual environment. *Cluster Computing*, *19*(4), 2295–2304. https://doi.org/10.1007/s10586-016-0630-5

Kabakus, A. T., & Dogru, I. A. (2018). An in-depth analysis of Android malware using hybrid techniques. *Digital Investigation*, *24*, 25–33. https://doi.org/10.1016/j.diin.2018.01.001

Karbab, E. B., Debbabi, M., Derhab, A., & Mouheb, D. (2016). Cypider: Building Community-Based Cyber-Defense Infrastructure for Android Malware Detection. *ACSAC '16 (32nd Annual Computer Security Applications Conference)*, 348–362. https://doi.org/10.1145/2991079.2991124

Karbab, E. M. B., Debbabi, M., & Mouheb, D. (2016). Fingerprinting Android packaging: Generating DNAs for malware detection. *Digital Investigation*, *18*, S33--S45. https://doi.org/10.1016/j.diin.2016.04.013

Kaspersky-Labs. (n.d.). Virus y Malware en Moviles Android. Retrieved from https://www.kaspersky.es/resource-center/threats/mobile

Kate, P. M., & Dhavale, S. V. (2015). Two Phase Static Analysis Technique for Android Malware Detection. *Proceedings of the Third International Symposium on Women in Computing and Informatics*, 650–655. https://doi.org/10.1145/2791405.2791558

Katz, J. (2010). Digital Signature, 197. https://doi.org/10.1017/CBO9781107415324.004

Kaushik, P., & Jain, A. (2015). Malware Detection Techniques in Android. *International Journal of Computer Applications*, *122*(17), 22–26.

Kearney, S. (n.d.). No Title. Retrieved from https://www.slant.co/versus/4345/4348/~genymotion_vs_bluestacks

Laura Gheorghe, Bogdan Marin, Gary Gibson, Lucian Mogosanu, Razvan Deaconescu, Valentin-Gabriel Voiculescu, M. C. (2015). Smart malware detection on Android. *International Journal of Applied Engineering Research*, *9*(22), 5968–5974. https://doi.org/10.1002/sec

Leeds, M., Keffeler, M., & Atkison, T. (2017). A Comparison of Features for Android Malware Detection. *Proceedings of the SouthEast Conference on - ACM SE '17*, 63–68. https://doi.org/10.1145/3077286.3077288

Liang, S., Might, M., & Van Horn, D. (2015). AnaDroid: Malware analysis of Android with user-supplied predicates. *Electronic Notes in Theoretical Computer Science*, *311*, 3–14. https://doi.org/10.1016/j.entcs.2015.02.002

Lin Ying-Dar, Lai Yuan-Cheng, Lu Chun-Nan, Hsu Peng-Kai, L. C.-Y. (2014). Three-phase behavior-based detection and classification of known and unknown malware. *International Journal of Applied Engineering Research*, *9*(22), 5968–5974. https://doi.org/10.1002/sec

Liu, L., Yan, G., Zhang, X., & Chen, S. (2009). VirusMeter: Preventing your cellphone from spies. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *5758 LNCS*, 244–264. https://doi.org/10.1007/978-3-642-04342-0_13

Ludwig, A. (2015). Android Security 2014 Year in Review. *Google Report 2014*, 1–43. Retrieved from https://source.android.com/devices/tech/security/reports/Google_Android_Security_2014_Report_Final.pdf%5Cnhttp://googleonlinesecurity.blogspot.com/2015/04/android-security-state-of-union-2014.html

Lueg, C. (2018). Some 343 new Android malware samples every hour in 2017. Retrieved September 25, 2018, from https://www.gdatasoftware.com/blog/2018/02/30491-some-343-new-android-malware-samples-every-hour-in-2017

Mahindru, A., & Singh, P. (2017). Dynamic permissions based Android malware detection using machine learning techniques. *10th Innovations in Software Engineering Conference (ISEC)*, 202–210. https://doi.org/10.1145/3021460.3021485

Malicious Mobile Threats Report. (2011). *Juniper Networks Global Threat Center*.

Malik, J., & Kaushal, R. (2016). CREDROID: Android Malware Detection by Network Traffic Analysis. *Proceedings of the 1st ACM Workshop on Privacy-Aware Mobile Computing*, 28–36. https://doi.org/10.1145/2940343.2940348

Martinez, A. (2014). La «otra manera» de identificar malware (/blog/indicadores-de-compromiso). Retrieved from https://www.incibe-cert.es/blog/indicadores-de-compromiso

Mendoza López, M. Á. (2015). Riesgos de seguridad en Android. *Revista .Seguridad UNAM*. Retrieved from http://ru.tic.unam.mx:8080/tic/handle/123456789/1688

PandaLabs. (2015). Informe Especial : Predicciones Informe Especial : Predicciones 2015.

Pastor, J. (2013). ¿Sirven para algo los antivirus para teléfonos móviles?, 1–21. Retrieved from https://www.xatakamovil.com/seguridad/sirven-para-algo-los-antivirus-para-telefonos-moviles

Racic, R., Ma, D., & Chen, H. (2006). Exploiting MMSVulnerabilities to Stealthily Exhaust Mobile Phone's Battery. *2006 Securecomm and Workshops*. https://doi.org/10.1109/SECCOMW.2006.359550

Rey, H., & Carlos, J. (2010). La correlación de eventos con fines de seguridad, *2010*.

Santomá, J. (2004). Nuevos Medios De Pago Electrónicos: Hacia La Desintermediación Bancaria. *Ice*, *El comerci*(813), 101–114.

Saracino, A., Sgandurra, D., Dini, G., & Martinelli, F. (2016). MADAM: Effective and Efficient Behavior-based Android Malware Detection and Prevention. *IEEE Transactions on Dependable and Secure Computing*, *5971*(c), 1–1. https://doi.org/10.1109/TDSC.2016.2536605

Seo, S. H., Gupta, A., Sallam, A. M., Bertino, E., & Yim, K. (2014). Detecting mobile malware threats to homeland security through static analysis. *Journal of Network and Computer Applications*, *38*(1), 43–53. https://doi.org/10.1016/j.jnca.2013.05.008

Shabtai, A., Kanonov, U., Elovici, Y., Chanan, G., A, Suarez-Tangil, G., … IEEE. (2016). 2112). Andromaly: Malware Detection Framework for Android Devices. *Journal of Intelligent Information Systems Evolution Detection and Analysis of Malware for Smart Devices Surveys Tutorials*.

Simón, M. (2014). Cómo funciona la seguridad en Android. Retrieved from https://rootear.com/android/seguridad-en-android

Singh, L., & Hofmann, M. (2017). Dynamic Behavior Analysis of Android Applications for Malware Detection. *2017 International Conference on Intelligent Communication and Computational Techniques (ICCT)*, (2013), 1–7. https://doi.org/10.1109/INTELCCT.2017.8324010

Somarriba, O., Zurutuza, U., Uribeetxeberria, R., Delosieres, L., & Nadjm-Tehrani, S. (2016). Detection and Visualization of Android Malware Behavior. *Journal of Electrical and Computer Engineering*, *2016*(i). https://doi.org/10.1155/2016/8034967

Sugunan, K., Kumar, T. G., & Dhanya, K. A. (2018). Static and Dynamic Analysis for Android Malware Detection. In E. B. Rajsingh, J. Veerasamy, A. H. Alavi, & J. D. Peter (Eds.), *Advances in Big Data and Cloud Computing* (Vol. 645, pp. 147–155). Singapore: Springer Singapore. https://doi.org/10.1007/978-981-10-7200-0

Talha, K. A., Alper, D. I., & Aydin, C. (2015). APK Auditor: Permission-based Android malware detection system. *Digital Investigation*, *13*, 1–14. https://doi.org/10.1016/j.diin.2015.01.001

Tan Cam, N., & Hong Phuoc, N. C. (2017). NeSeDroid—Android Malware Detection Based on Network Traffic and Sensitive Resource Accessing. *Proceedings of the International Conference on Data Engineering and Communication Technology*, *469*, 19–30. https://doi.org/10.1007/978-981-10-1678-3

TECNÓSFERA. (2018). Android, el sistema operativo con más vulnerabilidades en 2017. Retrieved from http://www.eltiempo.com/tecnosfera/novedades-tecnologia/android-fue-el-sistema-operativo-con-mas-vulnerabilidades-en-2017-167314

Teufl, P., Ferk, M., Fitzek, A., Hein, D., Kraxberger, S., Orthacker, C., … GPRS. (2006). Malware Detection by Applying Knowledge Discovery Processes to Application Metadata on the Android Market (Google Play). Security and Communication Networks,. & Vulnerabilities and Possible Attacks Against the Network. *In Critical Information Infrastructures Security Springer*.

Teufl Peter, Ferk Michaela, Fitzek Andreas, Hein Daniel, Kraxberger Stefan, O. C. (2014). Malware detection by applying knowledge discovery processes to application metadata on the Android Market (Google Play). *International Journal of Applied Engineering Research*, *9*(22), 5968–5974. https://doi.org/10.1002/sec

Trend-Micro. (n.d.). Indicators of Compromise. Retrieved from https://www.trendmicro.com/vinfo/us/security/definition/indicators-of-compromise

Villanova-Pascual, O. (2016). Malware en Android y medidas de prevención. Retrieved from http://reunir.unir.net/handle/123456789/3622

Wang, X., Zhang, D., Su, X., & Li, W. (2017). Mlifdect : Android Malware Detection based on Parallel Machine Learning and Information Fusion, *2017*, 1–14. https://doi.org/10.1155/2017/6451260

Wang, Z., Li, C., Yuan, Z., Guan, Y., & Xue, Y. (2016). DroidChain: A novel Android malware detection method based on behavior chains. *Pervasive and Mobile Computing*, *32*, 3–14. https://doi.org/10.1016/j.pmcj.2016.06.018

Wei, T. E., Mao, C. H., Jeng, A. B., Lee, H. M., Wang, H. T., & Wu, D. J. (2012). Android malware detection via a latent network behavior analysis. *Proc. of the 11th IEEE Int. Conference on Trust, Security and Privacy in Computing and Communications, TrustCom-2012 - 11th IEEE Int. Conference on Ubiquitous Computing and Communications, IUCC-2012*, 1251–1258. https://doi.org/10.1109/TrustCom.2012.91

Wei, X., & Gurkok, C. (2012). ProfileDroid : Multi-layer Profiling of Android Applications Categories and Subject Descriptors. In ACM (Ed.), *MobiCom'12* (pp. 137–148). Istambul. https://doi.org/10.1016/B978-0-12-416689-9.00004-6

Wu, 吳文傑 T A - Wen-Chieh. (2014). 運用機器學習及巨量資料分析實現Android惡意程式動態偵測 TT - DroidDolphin: a Dynamic Android Malware Detection Framework Using Big Data and Machine Learning. 資訊工程學研究所, 碩士, 43 NI-臺灣博碩士論文知識加值系統 LG-英文. https://doi.org/10.1145/2663761.2664223

Zhang, M., Duan, Y., Yin, H., & Zhao, Z. (2014). Semantics-Aware Android Malware Classification Using Weighted Contextual API Dependency Graphs. *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security - CCS '14*, 1105–1116. https://doi.org/10.1145/2660267.2660359

Zhang, Y., Yang, M., Xu, B., Yang, Z., Gu, G., Ning, P., … Zang, B. (2013). Vetting undesirable behaviors in android apps with permission use analysis. *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security - CCS '13*, 611–622. https://doi.org/10.1145/2508859.2516689

Zhibo, Z. (2012). C. Colon Osono. *F TrustDroid Preventing the Use of SmartPhones for Information Leaking in Corporate Networks Through the Used of Static Analysis Taint Tracking In Proceedings of the 7th International Conference on Malicious and Unwanted Software MALWARE*.

Zhu, H. J., You, Z. H., Zhu, Z. X., Shi, W. L., Chen, X., & Cheng, L. (2017). DroidDet: Effective and robust detection of android malware using static analysis along with rotation forest model. *Neurocomputing*, *272*, 638–646. https://doi.org/10.1016/j.neucom.2017.07.030