Universidad de Alcalá
Escuela Politécnica Superior

**Grado en Ingeniería de Computadores**

**Trabajo Fin de Grado**

# Extracción de datos en Android para un análisis forense



**Autor:** Nicolas Logghe Barbini

**Tutor:** José Javier Martínez Herrainz

2018

# UNIVERSIDAD DE ALCALÁ

ESCUELA POLITÉCNICA SUPERIOR

**Grado en Ingeniería de Computadores**

**Trabajo Fin de Grado**

**Extracción de datos en Android para un análisis forense**

Autor: Nicolas Logghe Barbini

Director: José Javier Martínez Herrainz

**Tribunal:**

**Presidente:** Dª Carmen Pagés Arevalo

**Vocal 1º:** D. Manuel Sánchez Rubio

**Vocal 2º:** D. José Javier Martínez Herráiz

Calificación: .....

Fecha: .....

**Dedicado a mis padres por su paciencia infinita y haberme apoyado en todo momento...**

*"I have not failed, I just found 10,000 ways that do not work"*
(Tomas Alva Edison)

*"Focus is a matter of deciding what things you're not going to do."*
(John Carmack)

# Agradecimientos

Gracias a mis compañeros de la Universidad, en especial a todos los de la Cátedra, por la ayuda, la compañía y por haber hecho mas ameno mi trabajo. A los profesores que han estado siempre ahí. Y por encima de todo a mi familia, por haberme apoyado hasta el final.

# Resumen

A día de hoy se está viviendo una revolución tecnológica como nunca antes se había visto. Este crecimiento es casi exponencial en el campo de los dispositivos móviles. Actualmente se predice que habrá más de 9 billones de dispositivos móviles conectados en el mundo para el 2020. Esta estadística por sí sola refleja el crecimiento sin precedentes de los dispositivos móviles. Los teléfonos móviles no solo han aumentado en número sino que también se han vuelto más sofisticados en términos de funcionalidad.

El análisis forense móvil es una rama del análisis forense digital que está evolucionando muchísimo en la actualidad. Un forense se ocupa de extraer, recuperar y analizar los datos presentes en un dispositivo móvil a través de una variedad de técnicas. Sin embargo, es importante tener una comprensión clara de la plataforma y otros fundamentos antes de sumergirse a descubrir cómo extraer datos.

En este trabajo se explicara todo el conocimiento necesario para poder realizar una extracción de datos para un análisis forense de un dispositivo Android. Veremos como esta formado el sistema operativo Android su funcionamiento y el funcionamiento de las aplicaciones para poder hacer una extracción de la máxima cantidad de datos y lo mas completa posible. Con estos conocimientos adquiridos se hará una demostración creando un programa para la extracción de datos de la aplicación de Telegram.

**Palabras clave:** Telegram, Reversing, Forense, Mensajería, Andorid.

# Abstract

Today a technological revolution is being experienced like no other. This growth is almost exponential in the field of mobile devices. It is currently predicted that there will be more than 9 billion mobile devices connected in the world by 2020. This statistic alone shows the unprecedented growth of mobile devices. Mobile phones Not only have they increased in number but they have also become more sophisticated in terms of functionality.

Mobile forensics is a branch of digital forensics that is evolving in huge leaps. A forensic analist is in charge of extracting, recovering and analyzing the data present in a mobile device through a variety of complex techniques. However, it is important to have a clear understanding of the platform and other fundamentals before diving and discovering how to extract data.

In this paper we will explain all the necessary knowledge that is needed to be able to perform a data extraction for a forensic analysis of an Android device. We will see how the Android operating system works review all of its functioning and how andorid Apps work internaly in order to extract the maximum amount of data and recover as much information as possible. With the knowladege aquired in this document we will create as an example a program for data extraction of the Telegram app.

**Keywords:** Telegram, Reversing, Forensics, Messeging, Android.

# Resumen extendido

Actualmente hay varias herramientas que permiten a peritos o investigadores extraer la información que está contenida en dispositivos móviles con sistema operativo Android, pero no hay herramientas o muy pocas para poder descifrar la información extraída de aplicaciones particulares que en algunos casos, por ejemplo con la aplicación de Telegram sin estas herramientas la información extraída es totalmente inútil sin un posterior análisis.

La mensajería instantánea es una de los usos mas extendidos de los dispositivos móviles. Permiteque los usuarios puedan comunicarse con cualquier persona de una manera interactiva y fácil. Estasaplicaciones permiten el envió de texto voz imágenes y archivos. Existen varias aplicaciones de este tipopero suelen haber unos 3 que son los que mas se utilizan, dentro de estas tres se encuentra la aplicaciónde Telegram que es la que vamos a estudiar en profundidad

Telegram prácticamente no tiene ninguna herramienta de extracción de datos. Esto es a causa de que los datos son difíciles de visualizar y ofuscan por así decir sus datos y están de una manera que son difícil de extraer y recuperar. A partir de la situación de falta de herramientas para esta aplicación, se procedió a la realización de una herramienta para la extracción y visualización de los datos contenidos en la aplicación de Telegram.

Para realizar este trabajo tocaremos los siguientes puntos.

1. Estudio sobre el Análisis forense de un dispositivo movil.
2. Estudio sobre el sistema operativo.
3. Estudio sobre los pasos a seguir en una extracción.
4. Estudio sobre los principales Herramientas de extracción existentes.
5. Estudio sobre la estructura de la aplicación movil y de Telegram.
6. Estudio sobre como extraer los datos de la aplicación de telegram.
7. Desarrollo de herramienta para extracción de los conocimientos anteriores
8. Elaborar un Dashboard para la representación de los resultados.

# Índice general

# Índice de figuras

# Índice de tablas

# Índice de listados de código fuente

# Capítulo 1

# Introducción

## 1.1 Justificación

A día de hoy se esta viviendo un avance tecnológico como nunca se había visto antes en la que los dispositivos móviles están siendo imprescindibles en la vida cotidiana de cualquier persona. Este crecimiento es casi exponencial en el campo de los dispositivos móviles. En la actualidad se predice que habrá más de 9 billones de usuarios con móviles conectados para el 2020. Esta estadística por sí sola refleja el crecimiento sin precedentes de los dispositivos móviles. Los teléfonos móviles no solo han aumentado en número sino que también se han vuelto más sofisticados en términos de funcionalidad, últimamente los dispositivos móviles han permitido que se puedan tomar fotografías grabar vídeos, enviar mensajes, chatas, Emails, localización por GPS, por lo que puede ser una fuente importante para un caso de investigación. Teniendo en cuenta que un usuario medio utiliza su móvil 147 minutos al día se puede acumular mucha información sobre los hábitos y comportamientos de la persona a investigar. ¿Con tanta cantidad de datos existe una metodología para conseguir estos datos? ¿Como se analizan estos datos? y ¿Como se buscan y extraen en el caso de un análisis forense? ¿Que información obtenemos de ellos? Estas son las preguntas que intentaremos responder este trabajo.

## 1.2 Estado del arte

Actualmente hay varias herramientas que permiten a peritos o investigadores extraer la información que está contenida en dispositivos móviles con sistema operativo Android, pero no hay herramientas o muy pocas para poder descifrar la información extraída de aplicaciones particulares que en algunos casos, por ejemplo con la aplicación de Telegram sin estas herramientas la información extraída es totalmente inútil sin un posterior análisis.

Nuestros Smartphones permiten la comunicación constante e instantánea con nuestra familia, amigos e incluso compañeros de trabajo. Los mensajes de texto se encuentran entre las formas más populares de comunicarse en todo el mundo. Las aplicaciones de mensajería han crecido para ofrecer servicios completos que ofrecen todo tipo de interacciones con los demás, desde mensajes de texto gratuitos hasta llamadas de voz y vídeo, y uso compartido de fotos y archivos con el beneficio adicional de el cifrado. Aquí esta una lista de las aplicaciones mas usadas de mensajería en Android.[1]

Figura 1.1: Logos aplicaciones

- **Telegram** [2]
- **WhatsApp** [3]
- **WeChat** [4]
- **Snapchat** [5]
- **Facebook Messenger** [6]
- **Line** [7]

Una de las aplicaciones que mas herramientas tiene para el análisis forense son para la App de Whatsapp en esta podemos encontrar una lista infinita entre las mas conocidas

- **Whapa**[8]
- **WhatsApp Viewer**[9]

Telegram en cambio es de las segundas mas utilizadas pero no tiene ninguna herramienta. Esto es a causa de que en la aplicación de Whatsapp la información es fácil de extraer, simplemente abriendo la base de datos tenemos toda la información en claro de la aplicación entera, por el otro lado tenemos Telegram que ofusca por así decir sus datos y los pone de una manera que son difícil de extraer y recuperar.

Esta fue la inquietud que motivó la realización de este trabajo. A partir de la situación de falta de herramientas para esta aplicación, se procedió a la realización de una herramienta para la extracción y visualización de los datos contenidos en la aplicación de Telegram, con esto veremos todos los pasos que se han tomado para poder realizar esta herramienta y esta información podría servir para desarrollar una metodología que permitirá desarrollar herramientas especificas para la extracción de estos datos en distintas aplicaciones Andorid desde la extracción de los datos de el dispositivo hasta la extracción de los datos de la propia aplicación. Al mismo tiempo, se demuestra como con la extracción es posible realizar un análisis de la actividad y comportamiento del usuario que no era información intencionadamente almacenada.

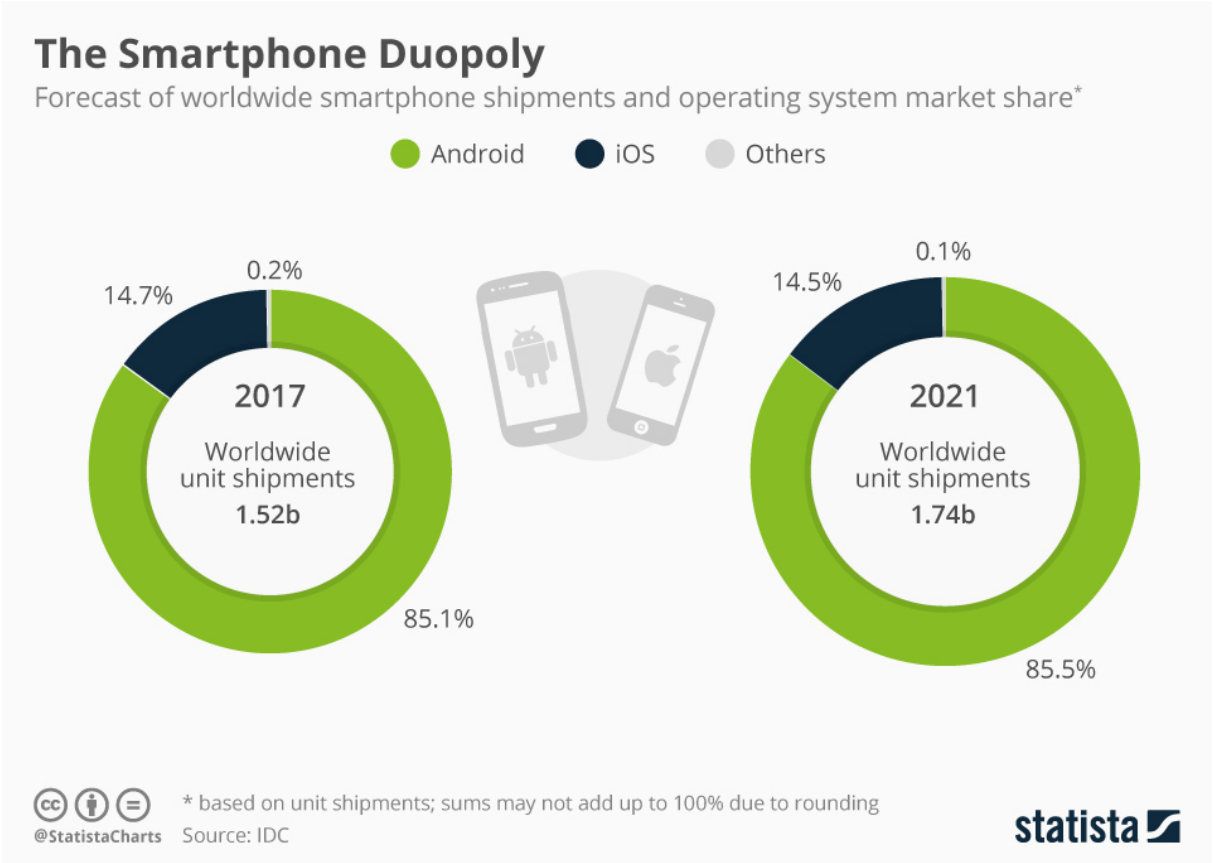



Figura 1.2: Comparativa Android iOS
https://www.statista.com/chart/9628/smartphone-platform-market-share-forecast/

## 1.3 Que es Android

En este articulo trabajaremos sobre el sistema operativo Andorid exclusivamente, existen otros como pueden ser iPhoneOS Symbian webOS Bada.. Pero al ser Android uno de los mas populares en España seguido de IOS nos concentraremos solo en Android. ¿Pero que es Android?

[10]Android es un sistema operativo basado en el núcleo Linux. Fue diseñado principalmente para dispositivos móviles con pantalla táctil, como teléfonos inteligentes, tabletas y también para relojes inteligentes, televisores y automóviles. Inicialmente fue desarrollado por Android Inc., empresa que Google respaldó económicamente y más tarde, en 2005, compró. Android fue presentado en 2007 junto la fundación del Open Handset Alliance (un consorcio de compañías de hardware, software y telecomunicaciones) para avanzar en los estándares abiertos de los dispositivos móviles.


Figura 1.3: Logotipo de Andorid

La versión básica de Android es conocida como Android Open Source Project (AOSP) Este sistema operativo se utiliza no solo en smartphone pero en varios dispositiovs como pueden ser vehículos neveras tablets ordenadores radios...

## 1.4 Análisis Forense Móvil

El análisis forense móvil es una rama del análisis forense digital que está evolucionando muchísimo en la actualidad. Un forense se ocupa de extraer, recuperar y analizar los datos presentes en un dispositivo para una investigación a través de una variedad de técnicas. Para esto es importante tener una comprensión clara de la plataforma su estructura y funcionamiento tanto del dispositivo como las aplicaciones.

La mensajería instantánea es una de los usos mas extendidos de los dispositivos móviles. Permite que los usuarios puedan comunicarse con cualquier persona de una manera interactiva y fácil. Estas aplicaciones permiten el envió de texto voz imágenes y archivos. Existen varias aplicaciones de este tipo pero suelen haber unos 3 que son los que mas se utilizan, dentro de estas tres se encuentra la aplicación de Telegram que es la que vamos a estudiar en profundidad exclusivamente en un capitulo.

La velocidad a la que se crean nuevos modelos y actualización es bastante elevada por lo que seguir un procedimiento igual para todos los modelos y aplicaciones es bastante complicado por lo tanto lo que se aplica en esta investigación puede que sea distinta en otras versiones y aplicaciones pero la metodología se mantiene.

## 1.5 Breve descripción de la metodología de la investigación

En este trabajo se explicara todo el conocimiento necesario para poder realizar una extracción de datos para un análisis forense de un dispositivo Andorid. Veremos como esta formado el sistema operativo Android y su funcionamiento para poder hacer una extracción lo mas completa posible.

El problema que nos encontramos es que una vez tenemos los datos no siempre podemos utilizarlos ya que cada aplicación es distinta una de otra. Al haber millones de aplicaciones en el mercado para Android nos centraremos en recuperar datos para la aplicación Telegram específicamente que es una aplicación de uso muy popular, no tiene casi ninguna herramientas desarrollada para este fin y es un reto compleja que nos permitirá tocar varios puntos de la extracción de datos. Explicaremos todos los pasos que se han tomado para poder extraer los datos de una manera que puedan ser utilizados para un análisis Forense. Con estos conocimientos adquiridos los los utilizaremos para crear una aplicación de extracción de datos para un análisis forense y podremos aplicar esta misma metodología en otras aplicaciones.

## 1.6 Esquema del Trabajo de Fin de Grado

Para realizar este trabajo tocaremos los siguientes puntos.

1. Estudio sobre el Análisis forense de un dispositivo movil.
2. Estudio sobre el sistema operativo.
3. Estudio sobre los pasos a seguir en una extracción.
4. Estudio sobre los principales Herramientas de extracción existentes.
5. Estudio sobre la estructura de la aplicación movil y de Telegram.
6. Estudio sobre como extraer los datos de la aplicación de telegram.
7. Desarrollo de herramienta para extracción de los conocimientos anteriores
8. Elaborar un Dashboard para la representación de los resultados.

# Capítulo 2

# Estudio teórico

En esta sección se explicara la base de conocimientos que debemos tener para poder hacer una análisis forense y lo necesario para conocer el funcionamiento de el sistema operativo Android y las técnicas que necesitamos para poder hacer una extracción, donde y como buscar estos datos.

## 2.1 Proceso de análisis Forense

### 2.1.1 Confiscación y aislamiento

A la hora de confiscar el dispositivo se deberá tener cuidado de no modificar ni que sean modificados los datos datos durante la manipulación. Lo primero que debemos de tener en cuenta a la hora de adquirir un dispositivo móvil para un análisis forense es que los datos sean alterado lo mínimo posible durante todo el proceso de manipulación y análisis.

Cuando se confiscó el dispositivo si es posible hay que hacerse con el dispositivo ya desbloqueado ya que es un paso muy importante si se puede realizar ya que esto nos dará un acceso sin problemas a móvil sin tener que preocuparnos por saltarnos el bloqueo.

Una vez con el dispositivo es importante que este no se bloque por lo tanto debemos de poner el móvil de una manera para que no se nos bloquee y es importante que este no se apague por lo tanto inmediatamente poner a cargar en caso de que la batería esté baja y desactivar el bloqueo automático. Para hacer que no se nos apague el dispositivo y se bloque se puede hacer del siguiente modo Activando el modo Desarrollador en opciones de desarrollo y activando la opción de “Pantalla activa” y también poniendo el tiempo de suspensión al máximo

Es importante también aislar el dispositivo de comunicaciones ya que se podría alterar los datos por actualizaciones conexiones o programas ejecutándose o se puede borrar o bloquear el dispositivo remotamente por el usuario mediante los programas anti-robo, estos pueden ser por el Android Device Manager que permite el borrado remoto por internet o el Mobile Device Management que permite el borrado y bloqueo mediante un SMS por lo tanto es importante que no tenga conexión y esté completamente aislado. Esto se puede hacer activando el modo avión o metiendo el dispositivo en una Jaula de Faraday que impedirá que tenga ninguna señal. También aprovechar para activar el modo de depuración Android ADB (Android Debug Bridge) que nos dará acceso a la interfaz de desarrollo del dispositivo móvil.

### 2.1.2 Adquisición

La fase de adquisición es en la que mas detalle entraremos es donde se extraen los datos de el dispositivo. Dependiendo de el modelo versión y marca del dispositivo se tendrá distintas maneras disponibles por lo que siempre se debe elegir la mejor para cada caso.

- **Adquisición manual**, Extracción de datos mediante la interfaz de el propio dispositivo
- **Adquisición lógica**, Extracción de datos mediante las herramientas de el propio dispositivo como backups de el dispositivo mensajes archivos fotos vídeos aplicaciones...
- **Adquisición de sistema de archivos**, Extracción de sistema de archivos completo de este modo podemos recuperar archivos que han sido eliminados o información que no es accesible por la extracción lógica
- **Adquisición física**, Adquisición bit a bit de la memoria flash de un dispositivo en la que se tiene un clonado perfecto de el dispositivo que asegura la originalidad de la evidencia y que no va a ser alterada durante la investigación

En algunas de estas fases son necesarias técnicas ofensivas como crackeo de contraseñas y rooting. Hay que tener en cuenta que una vez obtenidos los datos también tendremos el problema de la extracción de datos de cada aplicación en la que unas sera mas fáciles que otras en la que se almacenan los datos de una manera distinta por cada aplicación. Tendremos un capitulo entera dedicado a sacar datos en detalle, como extraer la aplicación de Telegram, y veremos donde están almacenados los datos, ver como sacarle un sentido a estos datos para poder reconstruir los mensajes de la aplicación.

### 2.1.3 Análisis

En esta fase se extraen y analizan los datos utilizando distintas herramientas. Estos datos pueden ser desde archivos creados por el propio usuario como imágenes musica o datos que quedan tras el uso de aplicaciones como logs caches bases de datos. Estos datos nos pueden no solo dar información directa como fotos, contactos si no que podemos encontrar apartar de ellos información no intencionada como pueden ser los patrones de uso o de actividades diarias a partir de logs de tiempos de carga, Posicionamiento a partir de logs de conexiones de torres de telefonía móvil y así podemos conocer la actividades que suele hacer una persona diariamente, podemos averiguar sus gustos por logs de búsquedas. Para facilitar el trabajo en esta sección existen varias herramientas que podríamos utilizar para la búsqueda de estos datos como puede ser la aplicación de Autopsy. Una vez se obtienen estos datos se pasaría a un análisis en profundidad para poder obtener esta información.

### 2.1.4 Documentación

En esta fase se crea un documento claro de todos los pasos y datos encontrados durante el análisis apuntando: cada fase realizada

- **Fases de la extracción**: Detallando cada fase realizada en la extracción de los datos encontrados.
- **Tiempos**: Tiempos de los datos encontrados y de las extracciones.
- **Condiciones**: Condiciones de el dispositivo.

- **Marcas**: Marca y modelo de el dispositivo.
- **Imágenes y multimedia**: Fotos Vídeos Audios de el dispositivo.
- **Herramientas**: Lista de las Herramientas utilizadas.
- **Datos**: Datos documentados durante la examinacion de el dispositivo.
- **Evidencias**: Listado de las evidencias.

## 2.2 Android

Android es un sistema operativo creado por Open Handset Allience a comienzos del 2005 y luego fue adquirido por Google en el 2007. Este sistema operativo fue creado para una arquitectura ARM está basado en Linux y es de código abierto con licencia Apache 2.0. Es actualmente el sistema operativo más utilizado por los dispositivos móviles gracias a que cualquier fabricante puede adaptar Android para sus dispositivos. actualmente cuenta con más de 2 billones de dispositivos activos en todo el mundo.

### 2.2.1 Arquitectura

El sistema operativo es de código libre por lo tanto cualquier fabricante puede utilizar el código modificarlo a sus necesidades y aplicarlo a su dispositivo, Android está construido sobre un kernel Linux esta le permite una gran compatibilidad con una gran cantidad de hardware y hace que la adaptación distinto hardware sea muy sencillo. El sistema operativo funciona en varia capas conocidos como STACKS

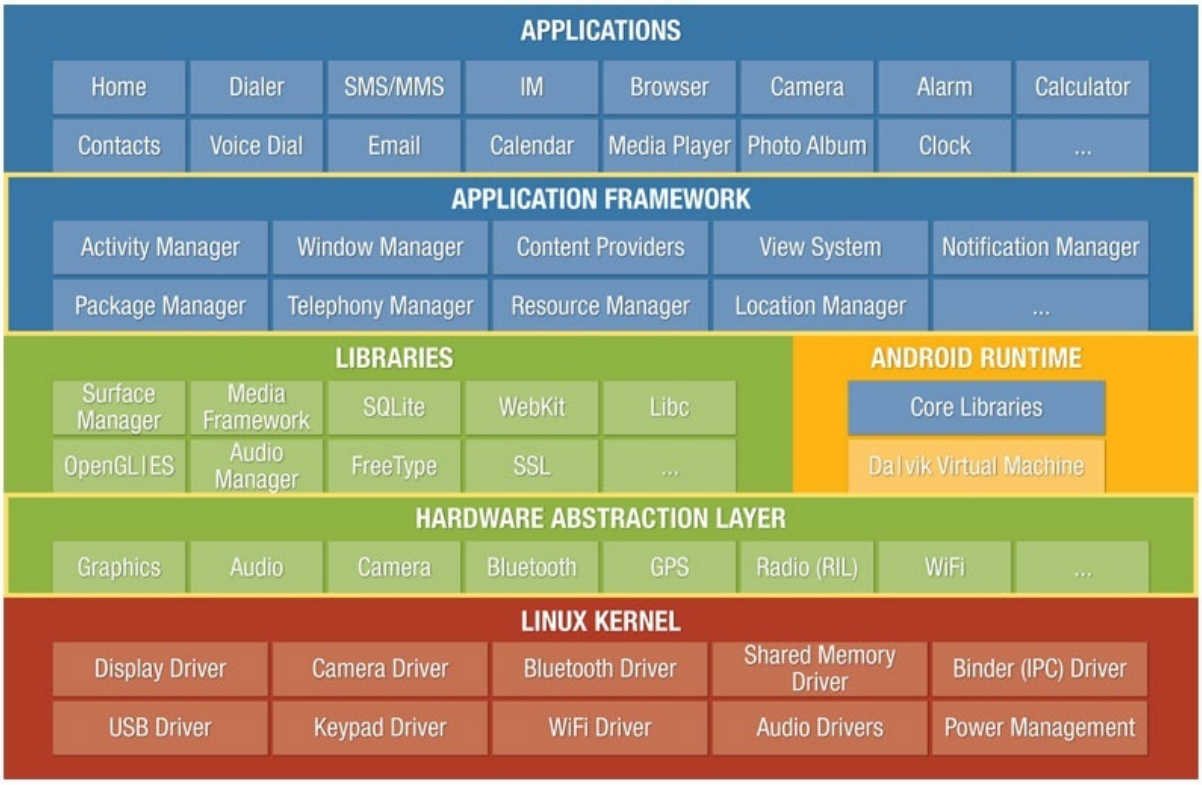



Figura 2.1: Arquitectura de el sistema operativo Android

#### 2.2.1.1 Kernel Linux

El kernel Linux es igual a cualquier otro kernel que puede estar incluida en cualquier distribución Linux y es la que da una capa de abstracción para el hardware del dispositivo como puede ser la pantalla la cámara Bluetooth Wifi audio etc. Esta también es la encargada de manejar la seguridad el manejo de la memoria protocolos de redes y los drivers de los dispositivos.

#### 2.2.1.2 Librerias

Son un conjunto de librerías nativas de Android programadas en C/C++ Están creadas por el fabricante para el hardware específico y son que se utilizan para manejar los datos como pueden ser multimedia OpenGL SQLite etc.

#### 2.2.1.3 Entorno de Ejecución de Android

https://www.theandroid-mania.com/android-architecture/ El Entorno de ejecución Android está compuesto por un capa de librerías en java con funcionalidades de específicas de Android como pueden ser las siguientes

- **android.app**Da acceso al modelo de la aplicación.
- **android.content**Facilita el acceso a contenido y comunicación entre aplicaciones y componentes.
- **android.database**Da acceso a datos con un sistema de interfaz con bases de datos sqlite.
- **android.opengl**Una interfaz para el API de OpenGL ES.
- **android.os**Acceso a servicios de el sistema operativo.
- **android.text**Para manipular texto en el dispositivo.
- **android.view**Servicios para creación de la interfaz gráfica.
- **android.widget**Componentes para la interfaz gráfica como botones labels listas etc.
- **android.webkit**Clases para interacción con la web

La parte más importante de esta es la llamada Dalvik Virtual Machine. Esta es una máquina virtual java optimizada para Android en la que se encarga de que cada aplicación Android este encerrada y funcione en su propia instancia virtual Dalvik por lo tanto haciendo que el sistema operativo sea muy seguro.

#### 2.2.1.4 Framework de Aplicaciones

Esta capa está formada por las clases y servicios que las aplicaciones utilizan durante la ejecución y son ejecutadas en la máquina virtual Dalvik para que los desarrolladores puedan utilizar los en sus aplicaciones. algunos de estos servicios son

- **Activity Manager** El encargado de controlar el ciclo de vida de la aplicación como puede ser la ejecución, sleep, segundo plano.
- **Content Providers** Permite que las aplicaciones compartan datos entre ellos.
- **Resource Manager** Da acceso a recursos como strings, colores y elementos de la interfaz graficas.

- **Notifications Manager** Permite que las aplicaciones muestran alertas al usuario.
- **View System** Para la creación de las interfaces de usuario.

#### 2.2.1.5 Capa de Aplicaciones

Aquí es donde se encuentra la ejecución de las aplicaciones en donde el desarrollador puede programar y crear las aplicaciones utilizando todos los recursos que da Android a su disposición. Estos pueden ser Navegadores Calendarios Juegos..

### 2.2.2 Inicialización del dispositivo

Es importante conocer cómo Android se inicializa para poder empezar a hacer un análisis forense de el dispositivo por lo tanto aquí hay una breve explicación del proceso

- **Boot ROM** Hay una dirección predefinida que al iniciar el microprocesador busca y encontrarás el bootloader por la cual procederá a cargar en memoria.
- **Bootloader**Este es el primer programa que se inicia al encender el dispositivo y es el encargado de encender los componentes hardware y manejar su funcionamiento y empezar la carga del arranque de el sistema.
- **Kernel** El kernel es inicializado en la que montará el sistema de archivos memoria drivers y al terminar buscará el programa INIT que será el que inicie el sistema operativo Android
- **El proceso Init**Una vez montado el sistema de archivos se ejecutara el proceso de init encargado de comenzar el sistema operativo Android cargando las librerías a el sistema para el funcionamiento correcto de esta.
- **Zygote y Dalvik** Init inicializa el servicio de Zygote que es el encargado de inicializar las librerías de clases de Android el encargado de que se inicialice las máquinas virtuales Dalvik que es la máquina virtual similar a las máquina virtual java donde se ejecutarán los procesos
- **Servicios del Sistema o Servicios** Zygote también inicializará los servicios de sistema como son los sensores el módem teclado batería etc..
- **Boot completado** Una vez completado el inicio de mandara un disparo de acción de tipo BROADCAST al sistema con el mensaje “ACTION_BOOT_COMPLETED” que indicará el fin de la carga del proceso y debería de ver ya por pantalla nuestro escritorio Android

### 2.2.3 Particionamiento de la memoria de Android

```
#cat /proc/mtd
dev:    size   erasesize  name
mtd0: 00040000 00020000 "misc"
mtd1: 00500000 00020000 "recovery"
mtd2: 00280000 00020000 "boot"
mtd3: 04380000 00020000 "system"
mtd4: 04380000 00020000 "cache"
mtd5: 04ac0000 00020000 "userdata"
```

Podemos distinguir seis particiones de gran interés estas son

- **Bootloader** Es donde encontramos el bootloader del teléfono y será lo primero que nuestro dispositivo iniciará.
- **Recovery** Desde el bootloader podemos elegir bootear desde esta partición y aquí tendremos herramientas de mantenimiento para el móvil. Al igual que la partición boot tiene su propio kernel y ramdisk.
- **Boot** Esta partición contiene el kernel y ramdisk de Android y es la que se inicia en un booteo normal.
- **System** Aquí encontramos el framework de Android.
- **Cache** Aquí se almacena los datos que el usuario utiliza con más frecuencia para agilizar la carga de ellos.

## 2.3 Extracción de datos

### 2.3.1 Confiscación y aislamiento

Antes que nada tener esto en cuenta.

- Si es posible hacerse con el dispositivo ya desbloqueado
- Poner el dispositivo en DebugMode lo antes posible.
- Importante que este no se bloque activando la opción de “Pantalla activa”
- Cargar el móvil y evitar que se apague
- Importante también aislar el dispositivo para evitar pérdidas.

**ESTO PASOS NOS FACILITARÁ MUCHÍSIMO EL TRABAJO

### 2.3.2 Android Debug Bridge y Fastboot

Android Debug Bridge (ADB)[11] es una herramienta de líneas de comandos versátil que te permite comunicarte con una instancia de un emulador o un dispositivo Android conectado. Esta herramienta proporciona diferentes acciones en dispositivos, como la instalación y la depuración de Apps, y proporciona acceso a un Shell Unix que puedes usar para ejecutar varios comandos en un emulador o un dispositivo conectado. Con esta Herramienta accederemos al dispositivo para poder extraer los datos o hacer un clonado de el sistema. Para poder Utilizar esta herramienta debemos de conectar el dispositivo por USB o por red y debemos conectarnos a el con el cliente que nos proporciona Android SDK[12]. Se debe activar desde el menú de configuración y poner el Dispositivo en modo Debug. Otro método seria iniciando un programa que ya lo lleve iniciado por defecto como por ejemplo un Custom Recovery de TWRP[13].

Fastboot es otra herramienta que no nos proporciona una consola ni manera de copiar archivos pero al entrar en modo Fastboot en el dispositivo nos da la posibilidad de poder flashear particiones de el dispositivo y nos permitirá desbloquear el bootloader en caso de que lo este bloqueado.

### 2.3.3 ¿Extracción física o lógica?

Cuando haya que hacer una extracción de datos de un dispositivo móvil para un análisis forense hay dos maneras en la que se pueden extraer los datos esta es la física y la lógica. Estos métodos de extracción dependerá de el acceso que tengamos al dispositivo, fabricante, modelo, etc.. Una nos dará mas información que otra y dependiendo de la situación estaremos obligados a utilizar una de las dos.

#### 2.3.3.1 Extracción Física

Si es posible, siempre es recomendable hacer una adquisición física, ya que se clona todo el dispositivo bit a bit. Con esta tendremos una copia exacta de el dispositivo no solo de los archivos de el usuario si no también de todo el sistema. Esto nos dará mas posibilidades a la hora de recuperar datos tantos los eliminados o que a lo mejor a plena vista no se sabia que estaban. Esta nos ocupara el tamaño que tenga el dispositivo de almacenamiento.

Para la extracción física podemos utilizar los siguientes métodos.

- **ADB + ROOT** Consiste en tener acceso root en el sistema y hacer una copia de el sistema
- **Boot loader** Hacer una adquisición desde el propio bootloader de el dispositivo con las herramintas especifivas para cada fabricante
- **Chip Off** Sacar el chip eMMC y hacer una lectura directa y dumpeado de esta misma.
- **JTAG ISP** En algunos dispositivos tenemos los puertos de JTAG o ISP que vienen en los SOC (Sistem On Chip) de los dispositivos y se utilizan para diagnostico y fabricación, Estas suelen ser accesibles desde la placa base o se puede modificar la placa base para conectarse a este puerto directamente en el chip. Con una herramienta conectada a este puerto se puede hacer un dumpeado de la memoria de el dispositivo.
- **Custom Recovery** Consiste en instalar un Recovery sobre el original que nos de acceso al sistema de archivos y nos permita hacer un clonado de la misma. Al iniciar como recovery tendremos acceso a todo el dispositivo sin necesidad de rootear el Sistema operativo Andorid. Aveces incluso podemos bootear este recovery desde memoria sin flashear.

#### 2.3.3.2 Extracción Lógica

Esta extracción consiste solo en copiar los archivos en las que el usuario tiene acceso. Esto incluye archivos multimedia, backups de aplicaciones descargas y algunos datos que las aplicaciones van guardando dentro de el espacio de el usuario. Nos ocupara solo el tamaño que tengamos en archivos de usuario.

Para la extracción lógica podemos utilizar los siguientes métodos.

- **AFLogical** AFLogical es una herramienta que nos automatiza el trabajo de conectarse por adb y copiar los archivos.
- **De forma Manual** Podemos extraer los datos uno a uno abriendo una consola con ADB utilizando .ªDB SHELL.ª una sd externa o compilando los archivos directamente con .ªDB PULL"

#### 2.3.3.3 Desafíos

Por lo tanto ¿Cual de los dos métodos utilizamos y por que tenemos que elegir? En la extracción lógica sera en el caso de que no tengamos acceso de superusuario (ROOT) en el sistema o no haya alguna manera de poder extraer la memoria completa por medio de hardware u otros métodos así que la única opción que nos queda es hacer la exacción lógica. Siempre que sea posible y no estemos limitados por espacio de almacenamiento ni herramientas.

### 2.3.4 Rooteo de Android

Root o superusuario es el usuario con máximos privilegios de un sistema por lo que rootear un dispositivo es intentar conseguir de alguna manera ser el usuario root para poder tener todos los privilegios de un sistema desde acceso a carpetas o funcionalidades del sistema. En un sistema Android por seguridad se tiene acceso a una cuenta de usuario con privilegios limitados y no tenemos acceso a el dispositivo entero por lo tanto esto es muy necesario a la hora de hacer una análisis forense si queremos tener acceso y poder hacer una copia lo más completa del dispositivo para su análisis.

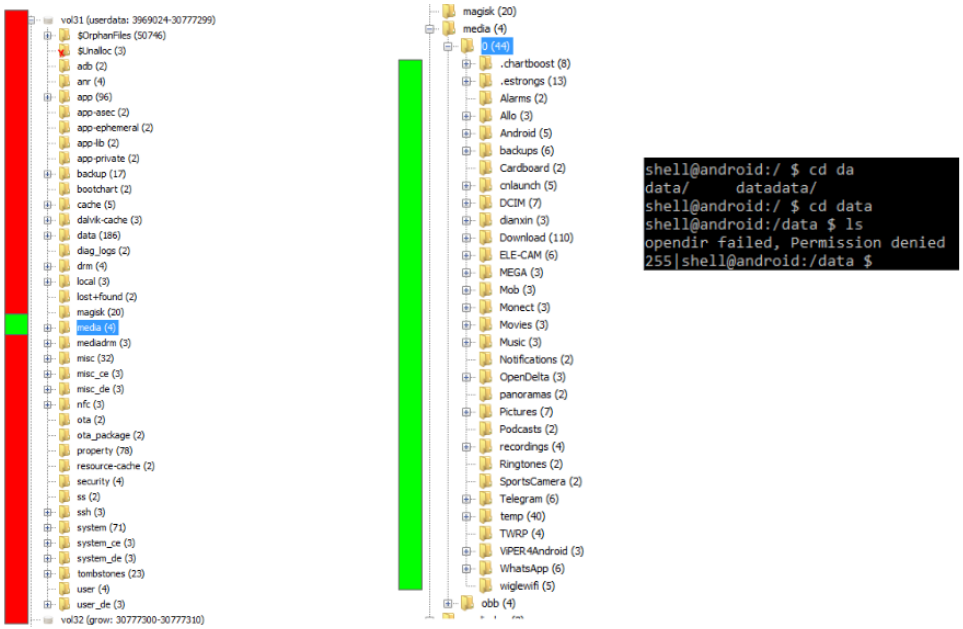



Figura 2.2: Sistema de archivos Android accesos 'SIN' ROOT

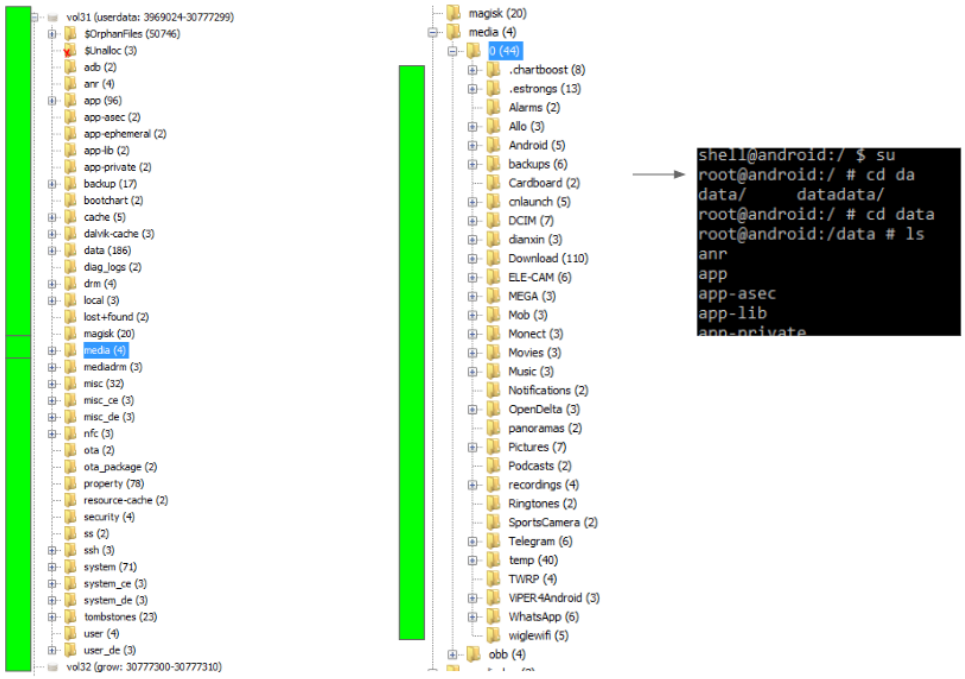



Figura 2.3: Sistema de archivos Android accesos 'CON' ROOT

#### 2.3.4.1 Recovery y Fastboot

Antes de rootear debemos de tener en cuenta esto dos modos de bootear un dispositivo android Para poder entrar en estos modos lo que hacemos en iniciar mediante una serie de combinaciones de botones al encender que dependerá de el fabricante de el dispositivo.Estos son pequeños sistemas operativos que se inician en la que podemos administrar y hacer un servicio a nuestro dispositivo.

Dependiendo de el dispositivo entrara en el modo Recovery o Fastboot

- El modo Recovery es un sistema operativo que nos permite recuperar archivos instalar actualizaciones, hacer pruebas o restaurar el sistema a fabrica. Hay que tener en cuenta que desde este sistemas operativos tenemos acceso al dispositivo completo. Si podemos instalar un Custom Recovery que nos permitira utilizar adb o alguna herramienta podremos hacer un clonado físico de el dispositivo.

- En modo Fastboot esta creado para poder tener una manera de flashear y actualizar el sistema completo para poder recuperar el sistema en caso de que falle y os permitirá elegir a donde bootear si al sistema operativo a Recovery o incluso desde memoria. Este es útil ya que podremos utilizarlo para poder instalar un Recovery Custom que nos permitirá utilizar ADB o alguna herramienta para clonar el dispositivo y fastboot nos facilita ya que no necesitamos software especifico para tener que flashaer el dispositivo.

#### 2.3.4.2 Como Rootear

Para poder conseguir ser ROOT en un dispositivo Android tenemos varios caminos. Estos dependerán de el dispositivo el fabricante la versión de Android que tengan, si existen exploits recientes sin parchear si esta bloqueado... y varios otros factores que se tendrán que estudiar a la hora de conseguir rootear el dispositivo. No entraremos en mucho detalle ya que estos datos varían muchísimo con el tiempo y es cuestión de buscar la información en el momento. Aquí se deja algunos de los métodos actuales a día de la escritura de esta trabajo y algunos métodos comunes para poder conseguir esto. Estos pueden ser

- Roots universales
  - DIRTY COW
  - KING ROOT
  - MAGESTIC
- Flashear Custom Recovery

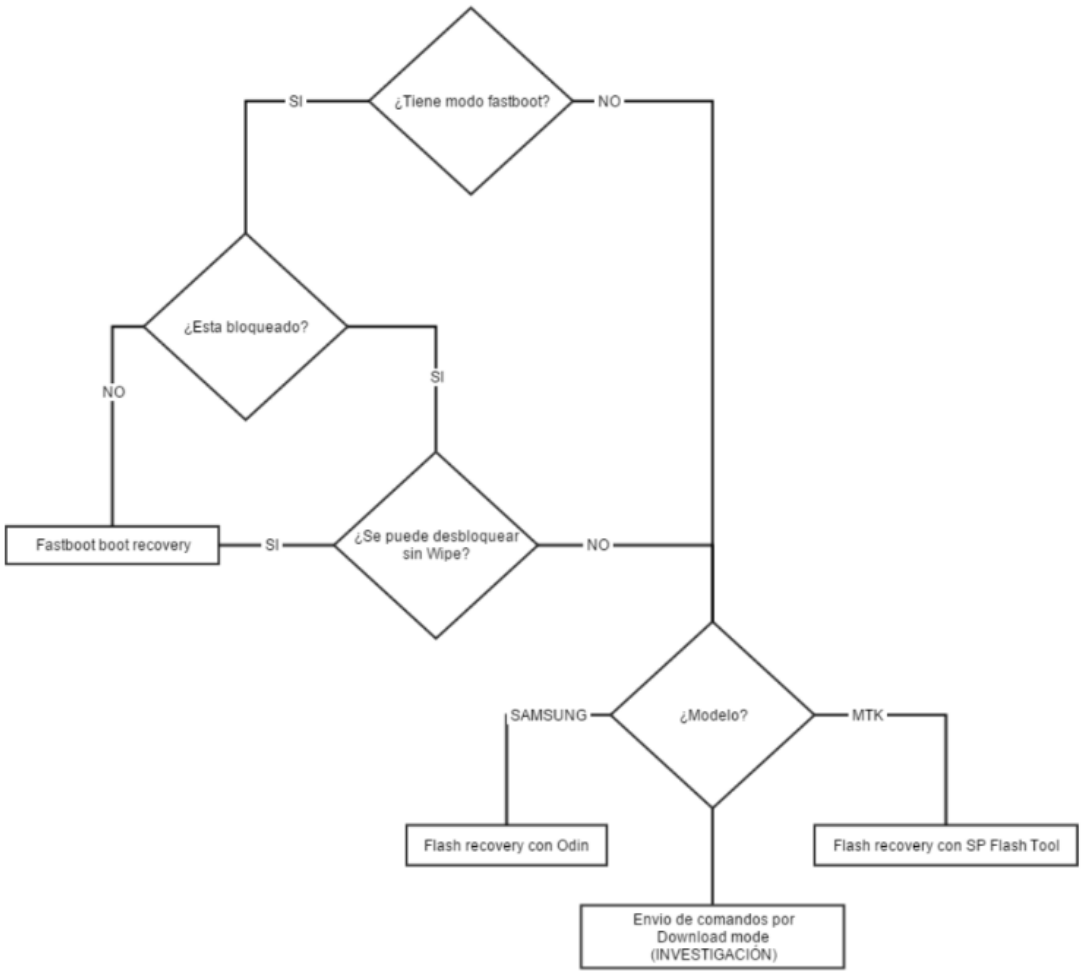



Figura 2.4: Flujograma flasheo de Custom Recovery

#### 2.3.4.3 Bootloader Bloqueado

Si nos encontramos con un Bootloader Bloquado se puede desbloquear desde Fastboot con el comando .ºEM Unlock"pero esto tiene un gran problema, por seguridad los dispositivos borrando toda la memoria de le dispositivo poniendo lo de fabrica al desbloquear el bootloader por lo tanto esta es una opción que tendríamos que descartar e intentar flashear de otro modo como seria con las aplicaciones especificas de cada fabricante.

#### 2.3.4.4 Custom Recovery

Un Custom Recovery nos permitirá poder tener acceso root de nuestro dispositivo y nos permitirá hacer una extracción física de el dispositivo. Este recovery puede ser uno programado o modificado por nosotros

mismos basado en el original, que venga con nuestra herramienta forense o podemos buscar una de las muchas disponibles creadas por la comunidad. Algunas de las mas populares son

- TWRP https://twrp.me/
- ClockworkMod Recovery https://www.xda-developers.com/how-to-install-clockworkmod/
- Philz Recovery https://forum.xda-developers.com/showthread.php?t=2201860

### 2.3.5 Extracción de memoria volátil

También es de gran interés hacer una extracción de la memoria volátil. Esta puede contener información de las aplicaciones que se están ejecutando claves de cifrados contraseñas en claro y puede haber información de mucho valor en ella. No entraremos en esto ya que daría para un trabajo completo por si solo pero si es importante que se conozca. Esta se podría hacer con una herramienta llamada lime y un kernel customizado para Android. https://www.youtube.com/watch?v=KN9kxn6htMU https://github.com/504ensicslabs/lime

## 2.4 Estructura de las aplicaciones móviles

Las aplicaciones en el sistema operativo Android se instalan y siguen una estructura igual en casi todas las versiones de Android. A la hora de extraer una aplicación de Android nos interesaran las siguiente s carpetas.

- **Core de las aplicaciones:** ((/data/xxx.xxxx.xxxx)) En esta encontramos todos los archivos propios de la aplicación, las imágenes de la propia aplicación las bases de datos, archivos temporales, logs..
- **APK de Instalación:** ((/data/app/xxx.xxxx.xxxx/base.apk)) Aquí encontramos el apk de la aplicación. Esta es muy útil ya que tenemos la versión exacta que tenemos instalada y nos servirá en el caso de que necesitemos hacer Reverse Engineering de la aplicación.
- **Multimedia:** ((/sdcard/XXXX)) Depende de la aplicación en la SD de el dispositivo puede que encontremos archivos de multimedia o archivos de la aplicación que son creadas por el usuario por lo tanto aquí podemos encontrar mucha información valiosa para un análisis forense. Para esta no es necesario ser ROOT por lo tanto esto es lo que buscaremos en el caso de adquisición lógica

### 2.4.1 Base de datos

SQLite es un sistema completo de bases de datos que soporta múltiples tablas, índices, triggers y vistas. No necesita un proceso separado funcionando como servidor ya que lee y escribe directamente sobre archivos que se encuentran en el disco duro. El formato de la base de datos es multiplataforma e indistintamente se puede utilizar el mismo archivo en sistemas de 32 y 64 bits. La base de datos se almacena en un único fichero a diferencia de otros DBMS que hacen uso de varios archivos. SQLite emplea registros de tamaño variable de forma tal que se utiliza el espacio en disco que es realmente necesario en cada momento.

La mayoría de Apps móviles almacenan información en SQLite. Suelen están almacenadas en la carpeta de la aplicación en /databases/ dentro de el Core de la aplicación. Los archivos SQLite suelen

tener la extensión .db aunque hay que tener en cuenta que no todos los archivos de SQLite van a tener esta extensión por esto hay que analizar cada uno de los archivos que encontremos mirando las cabeceras.

Ejemplos

- **Log de Llamadas** /com.android.providers.contacts/databases/contacts2.db (Calls)
- **Log de SMS** /com.android.providers.telephony/databases/mmssms.db (sms)
- **Log de Contactos** /com.android.providers.contacts/databases/contacts2.db (raw_contacts)
- **Log de Cuentas** de sincronización /com.android.providers.settings/databases/contacts2.db (accounts)
- **Log de Descargas** /com.android.providers.downloads/databases/downloads.db (downloads)

#### 2.4.1.1 SQLite Vacuum

Al borrarse información de las bases de datos sqlite no se eliminan realmente solo se marcan como libres. SQlite ha implementado Vacuum para eliminar los datos de una manera segura y se hacen poniendo a 0 estos bloques.En el caso de los dispositivos móviles con Andorid para poder ahorrar en procesado y consumo de batería esta no esta activada por defecto ni en iPhone ni Android. Si no se han sobrescrito estos datos borrados pueden ser recuperados fácilmente con un lector hexadecimal.

### 2.4.2 Ejemplos de datos que podemos encontrar

- **Logs de Providers**
  - **Log de Llamadas** /com.android.providers.contacts/databases/contacts2.db (Calls)
  - **Log de SMS** /com.android.providers.telephony/databases/mmssms.db (sms)
  - **Log de Contactos** /com.android.providers.contacts/databases/contacts2.db (raw_contacts)
  - **Log de Cuentas** de sincronización /com.android.providers.settings/databases/contacts2.db (accounts)
  - **Log de Descargas** /com.android.providers.downloads/databases/downloads.db (downloads)
- **Logs de Geolocalizacion** No solo necesitamos informacion gps para poder geolocalizar un dispositivo. viendo direciones ip datos de connecion con la wifi o datos de las torres moviles podriamos geolocalizar el dispositivo y tener un historial de las ubicaciones por las que ha estado. https://github.com/NoSuitsSecurity/GeoAndroid/blob/master/README.md
  - **misc/wifi/wpa_supplicant.conf** Log de Wifis Usadas GSM Cuando se usó Youtube, Google Maps, Google Play..
  - **/data/com.google.android.gms/databases/herrevad**
- **Logs de Bluetooth**
  - **com.android.settings/databases/search_index.db** Prefs_index
  - **misc/bluedroid/bt_config.conf**
- **Logs de Habitos**

  - **com.google.android.apps.fitness/databases/fit-0.db** (Prefs) (Sessions)
  - **com.google.android.apps.fitness/databases/fit-0.db-journal**
  - **com.google.android.deskclock/databases/alarms.db**

- **Logs de uso**
  - **Lista Apps Instaladas:** /system/packages.list Última vez que se ha utilizado una aplicación:
  - **/system/packages-usage.list**

# Capítulo 3

# Descripción experimental

## 3.1 Introducción

En este articulo se ha desarrollado una herramienta para la extracción de datos automatizada para la aplicación de Telegram. Esta aplicación es bastante compleja de extraer los datos por lo que iremos explicando todos los pasos que se han tomado. Muchas cosas quedaran sin poder ser resueltas ya que no existe ninguna documentación publica de esta aplicación por lo que se intentara extraer todo lo posible que podría ser de utilidad en un caso de Análisis Forense, estos datos siendo desde los mensajes, usuarios, archivos multimedia y también poder visualizar estos en un sentido temporal y a quien va dirigido los mensajes.

Con las técnicas explicadas anteriormente se extraerán los datos de la aplicación de Telegram y estos los vamos a analizar para ver como poder sacar los datos contenidos dentro de la propia aplicación para poder utilizarlos para un Análisis Forense ya que a diferencia de otras aplicaciones como Whatsapp los datos en Telegram no tienen una estructura que se puede ver a simple vista. Para realizar estas pruebas se ha utilizado un móvil limpio y preparado con un mensaje de cada tipo en la que estamos seguros y tenemos control de la información que tenemos, también se ha utilizado un móvil de uso real para verificar que funciona y para poder observar los datos y comparar con mayor cantidad de datos.

## 3.2 App Telegram

Es una aplicación creada por Pavel y Nikolai Durov fundadores de la red social VK Rusa. La aplicación de Telegram es una aplicación gratuita de mensajería instantánea que ha nacido como una alternativa a Whatsapp. Esta es de código abierto (aunque poco actualizado)

## 3.3 Preparación

Para realizar una herramienta es de gran ayuda estudiar bien la aplicación antes de comenzar y crear un entorno de pruebas con todos los posibles datos y combinaciones que podamos tener en la aplicación apuntando estos datos y sabiendo exactamente que son. Esto nos vendrá de gran ayuda a la hora de buscar la información en la aplicación.

En un dispositivo limpio con Telegram recién instalado y con una cuenta nueva. En este caso se ha utilizado un emulador para facilitar la extracción y con un numero de teléfono nuevo.

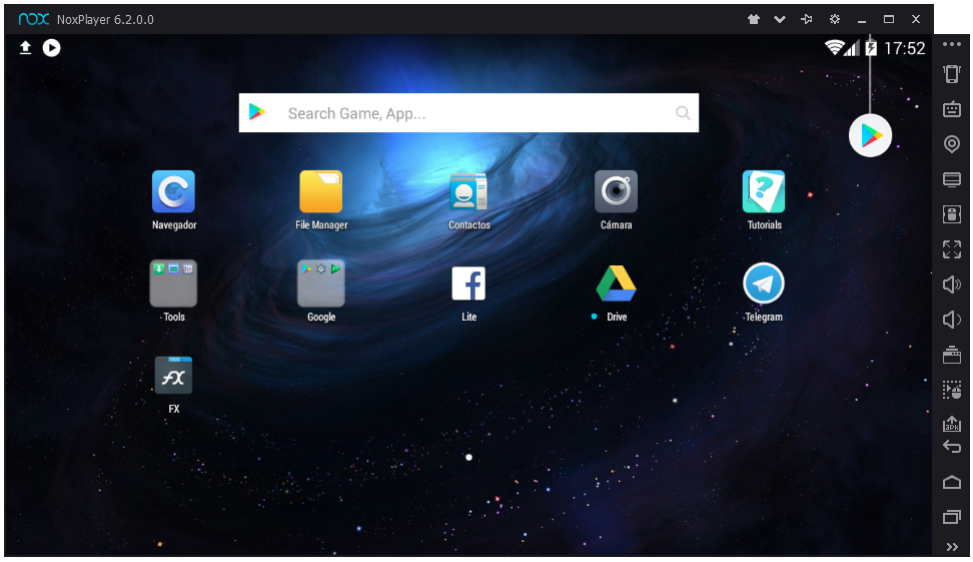



Figura 3.1: Pantalla de emulador NOX de Android

Se procede a enviar los siguientes mensajes para luego poder recuperar.

- —Recibidos—
  - Este es un mensaje normal mensajenormal
  - Este es un mensaje borrado mensajeborrado
  - Esto es un video mensajevideo
  - Esto es una foto mensajefoto
  - Un Audio nombre de archivo: Chewbacca descripcion: Starwars
  - Archivo: archivoenviado.txt contenido:esto es el contenido del archivo contenidodelarchivo
  - Contacto: Contactoenvaido Numero:123456789
- —Enviados—
  - Este es un mensaje normal enviado mensajenormalenviado
  - Archivo: extraccionteleg ram.zip

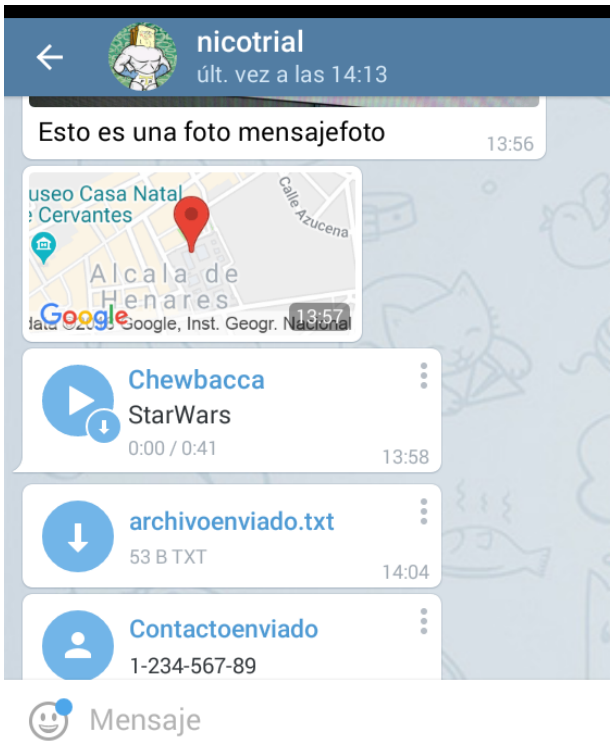



Figura 3.2: Pantalla Telegram mensajes pruebas

Es muy recomendable también tener un móvil con una aplicación que ha tenido un uso real, esto nos ayudara muchísimo al tener una gran cantidad de datos para poder comparar.

En el caso de Telegram sabemos que podemos enviar mensajes enviar multimedia crear chats secretos hacer llamadas de voz. También sabemos que la aplicación mantiene los mensajes se sincroniza con nuestros contactos, guarda archivos en el dispositivo.. Se han utilizado dos dispositivos Android uno real y otro limpio en la que se han creado una serie de mensajes de cada tipo, mensajes en distintos estados, mensajes borrados, mensajes multimedia, contactos añadidos y borrados. Ya con esto podemos continuar en la extracción de datos de el dispositivo y utilizaremos estas dos durante la extracción de datos de la aplicación.

## 3.4 Extracción

Para la extracción de la aplicación de Telegram se ha optado por una extracción Física. Para poder acceder a los archivos de los mensajes que nos interesan están dentro de la carpeta de el sistema de androide por lo tanto necesitamos poder tener acceso ROOT y conectar el dispositivo y copiar con DD una imagen de el sistema. Una vez teniendo ROOT para facilitar el trabajo podemos utilizar la aplicación Magnet Aquire [14] que nos hará el trabajo de extracción automáticamente y podremos visualizar los datos con la aplicación de AUTOPSY [15].

Ya que nuestra aplicación es solamente dedicada a la extracción de datos de la aplicación de Telgeram esta tendrá una opción para la carga de los archivos extraídos por métodos externos o lo hará automáticamente la extracción de solo los archivos de la aplicación activando una opción para esta en el programa. Para poder lograr esto el dispositivo móvil se conectara por USB reiniciara en modo FASTBOOT en la que indicaremos que boote una imagen de recovery TWRP que hemos descargado en RAM y esta al

iniciar tiene una sesión de ADB escuchando. Una vez esta ha bootead iniciara una sesión con ADB en la que tendremos acceso a todo el dispositivo como root y hacemos una copia

## 3.5 Extracción de artefactos en Telegram Messenger

Datos que podemos recuperar de la aplicación de Telegram son las siguientes

- Conversaciones.
- Multimedia Audios, Videos, Fotos.
- Contactos Números y Nombres
- Grupos y relación de personas
- Horas de Conexión
- Mensajes Borrados
- Ubicaciones

Los datos de la aplicación de Telegram están almacenadas en distintas partes de el dispositivos en la que vamos a encontrar los archivos propios de las apps imágenes archivos de usuario y bases de datos. Dependiendo de los archivos necesitaremos distintos privilegios para poder acceder a ellos por lo que es importante tener acceso root para poder hacer una extracción completa. Los directorios que nos interesan para esta aplicación están ubicadas en las siguientes direcciones

- Core de Telegram
  - /data/org.telegram.messenger/
    - files/cache4.db
- Apk de Instalacion
  - /data/app/org.telegram.messenger-2/base.apk
- Ficheros Multimedia y Archivos Transferidos
  - /sdcard/Telegram/

Un archivo de gran interés en este caso es el de cache4.db. Este archivo es una base de datos sqlite en la que se encuentran todos los datos de la aplicación Esta tiene la siguiente estructura

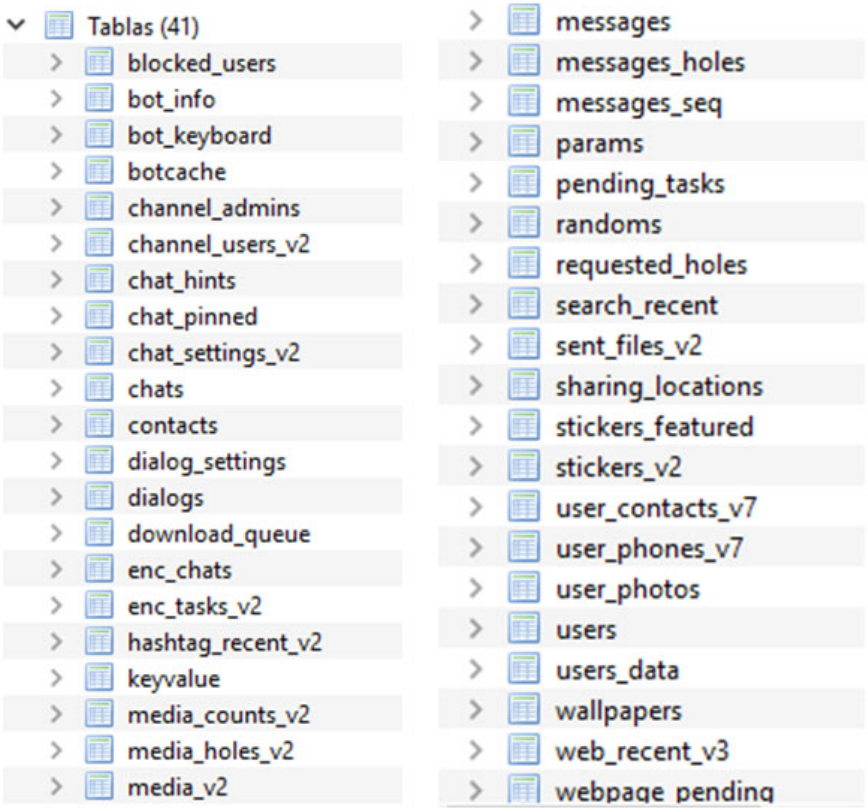



Figura 3.3: Base de datos cache4.db Telegram

### 3.5.1 Código

Para poder ayudarnos a la hora de interpretar los datos seria de buena ayuda tener el código fuente de la aplicación. Primero se ha optado por intentar reversear la aplicación para esto se ha pasado el apk que podemos encontrar en ((/data/app/org.telegram.messenger-2/base.apk)) por el programa APK TOOLS. Esta nos ha dado un código fuente interpretado por la herramienta que nos ayuda a la hora de descifrar los datos y ver como se tratan pero aun así es complicado para leer.

Tenemos la suerte de que podemos encontrar el código fuente de la aplicación de Telegram en Github [16] ya que es de código libre y esta públicamente abierta. El código fuente de Telegram es de una calidad bastante mala es muy lioso y muy difícil de leer pero con un poco de paciencia y habilidad para buscar nos puede ayudar muchísimo para ver como se estructuran los datos, como se leen y como se guardan.

### 3.5.2 Base de datos Telegram

Las bases de datos de Telegram y de muchas otras aplicaciones de Android utilizan SQLite.[17]. Para poder visualizar el contenido de una manera fácil y cómoda de los archivos .db de SQLite utilizaremos una aplicación llamad DBrowser [18] que es la que hemos utilizado para ir navegando los datos para su estudio. Luego estos datos son extraídos en nuestra programa mediante una librería de Python.Explorando las bases de datos que encontramos la que mas información contiene es cache4.db. Al abrir la base de datos nos encontraremos con un listado amplio de tablas 3.3. Dentro de cada una de estas tablas podemos extraer información de gran valor de la que iremos mostrando las mas interesantes.

- **Chats** En esta tabla podemos encontrar los grupos de Telegram el nombre y la descripción se encuentra con mas metadatos en el campo de data como un binario.


Figura 3.4: Tabla chats base de datos cache4.db

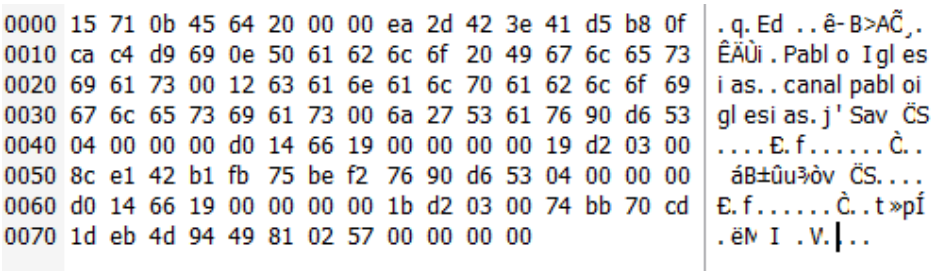



Figura 3.5: campo de datos en Tabla chats base de datos cache4.db

- **enc_chats** En esta tabla podemos encontrar los contactos que el usuario tiene agregados a la agenda y están en Telegram con su nombre en texto claro y UID.


Figura 3.6: Tabla enc_chats base de datos cache4.db

- **media_v2** En esta tabla podemos encontrar los datos guardados y referencias de la ubicación o el enlace de los mensaje de multimedia que se encuentran en binario en el campo data. También cabe destacar el campo MID que es el message id el UID que es el UID de el usuario que lo envió date que es la fecha en Epoch y type que nos indicara el tipo de mensaje enviado.

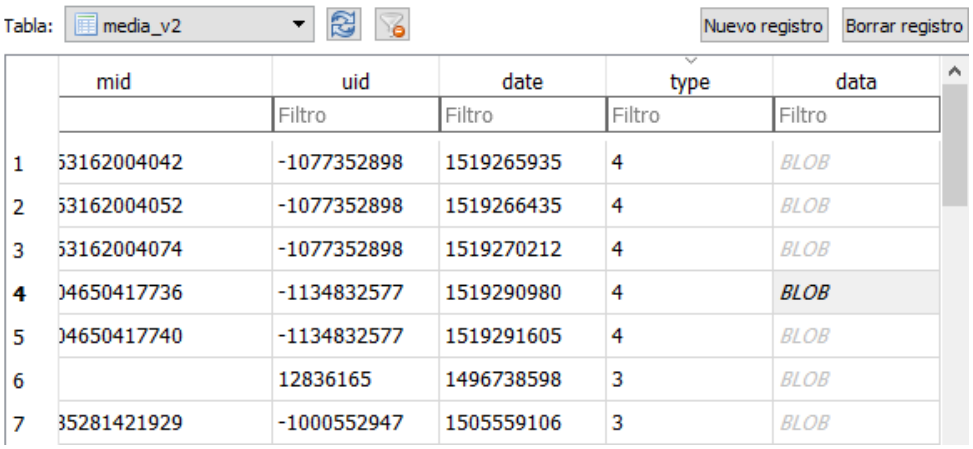


Tabla: media_v2 Nuevo registro Borrar registro

| | mid | uid | date | type | data |
|---|---|---|---|---|---|
| | Filtro | Filtro | Filtro | Filtro | Filtro |
| 1 | 53162004042 | -1077352898 | 1519265935 | 4 | BLOB |
| 2 | 53162004052 | -1077352898 | 1519266435 | 4 | BLOB |
| 3 | 53162004074 | -1077352898 | 1519270212 | 4 | BLOB |
| 4 | 04650417736 | -1134832577 | 1519290980 | 4 | BLOB |
| 5 | 04650417740 | -1134832577 | 1519291605 | 4 | BLOB |
| 6 | | 12836165 | 1496738598 | 3 | BLOB |
| 7 | 35281421929 | -1000552947 | 1505559106 | 3 | BLOB |

Figura 3.7: Tabla media_v2 base de datos cache4.db

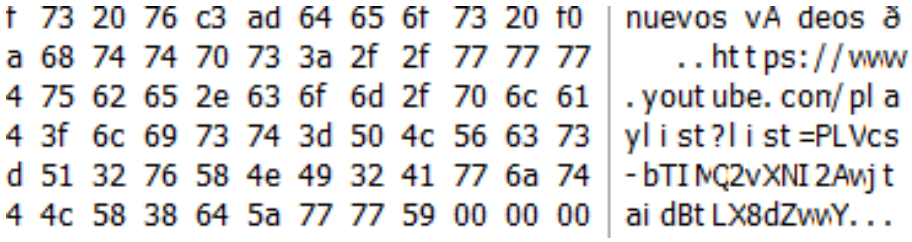



Figura 3.8: Tabla media_v2 campo data base de datos cache4.db

- **user_contacts_v7** En esta tabla podemos encontrar los nombres sacados de la lista de contactos de el dispositivo con una UID asociada.

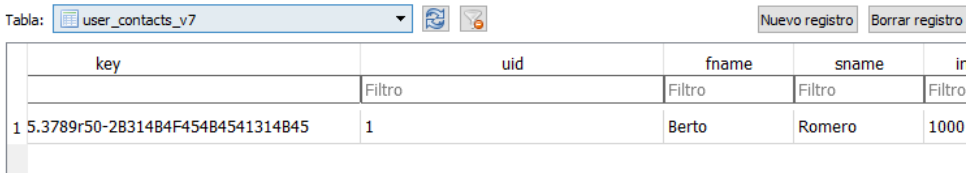


Tabla: user_contacts_v7 Nuevo registro Borrar registro

| | key | uid | fname | sname | ir |
|---|---|---|---|---|---|
| | | Filtro | Filtro | Filtro | Filtro |
| 1 | 5.3789r50-2B314B4F454B4541314B45 | 1 | Berto | Romero | 1000 |

Figura 3.9: Tabla user_contacts_v7 base de datos cache4.db

- **user_phones_v7** En esta tabla podemos encontrar los teléfonos sacados de la lista de contactos de el dispositivo y destacar que se puede ver que números ya no están en la agenda y fueron eliminados.

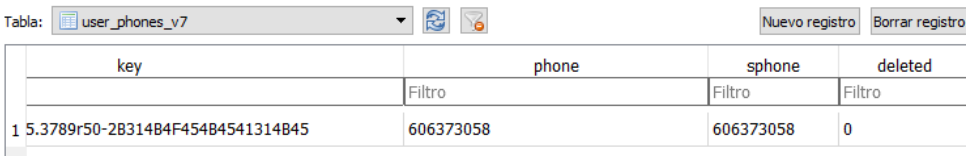


Tabla: user_phones_v7 Nuevo registro Borrar registro

| | key | phone | sphone | deleted |
|---|---|---|---|---|
| | | Filtro | Filtro | Filtro |
| 1 | 5.3789r50-2B314B4F454B4541314B45 | 606373058 | 606373058 | 0 |

Figura 3.10: Tabla user_phones_v7 base de datos cache4.db

- **users** En esta tabla podemos encontrar todos los usuarios de Telegram que tiene el usuario, tanto usuarios de su lista de contactos como los usuarios de los grupos que no tiene en su lista de contactos.

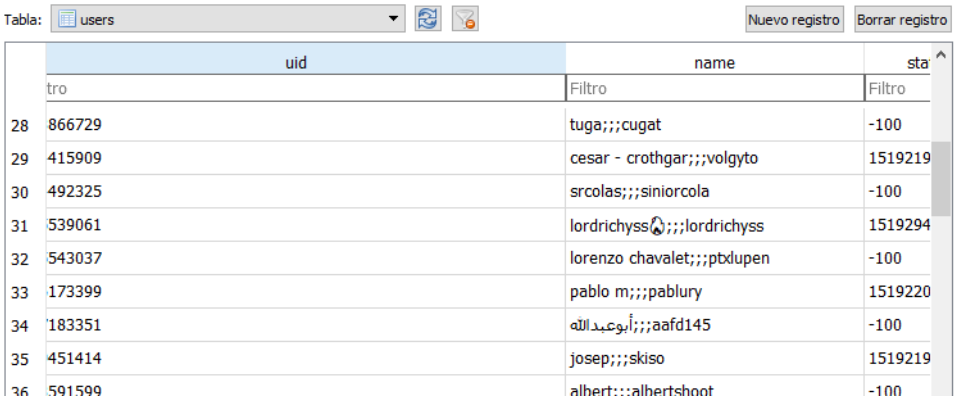



Figura 3.11: Tabla users base de datos cache4.db

- **messages** En esta tabla podemos encontrar todos los mensajes enviados desde los usuarios con su UID que nos hará referencia de que grupo viene, el texto en binario con otros meta datos y el id de el mensaje MID. Algunos datos que podemos interpretar son los siguientes.
  - read_state : 2= enviado y no leído, 3=enviado y leído
  - send_state: 0= enviado 1=pendiente de enviar
  - date: fecha Unix Epoch
  - data: Binario que contiene mensaje y varios meta datos
  - out: 1=Enviado 0=Recibido
  - media: -1=no es media
  - mention: Si ha sido mencionado en el mensaje

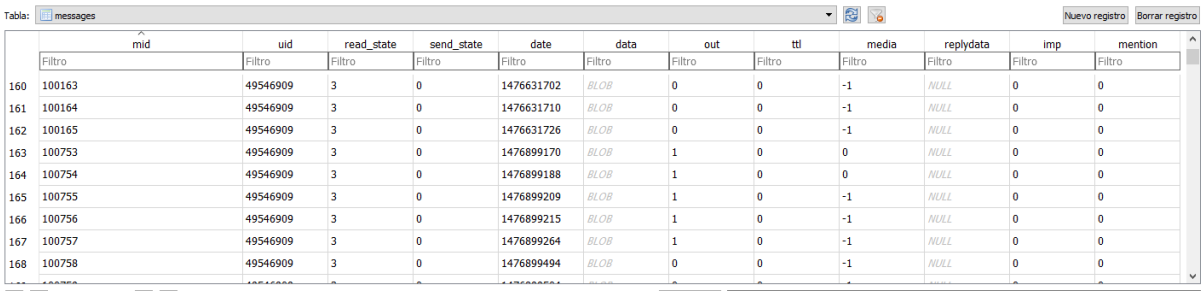



Figura 3.12: Tabla messages base de datos cache4.db

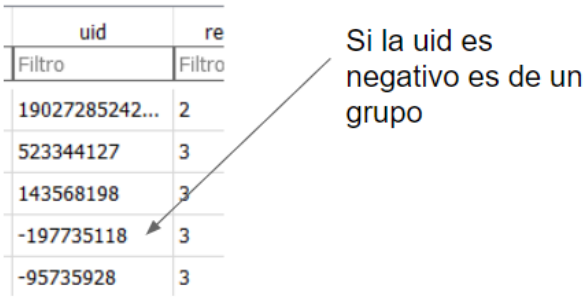



Figura 3.13: Tabla messages identificación grupos base de datos cache4.db

- **media_v2** En esta tabla podemos encontrar los mensajes media que se han enviado y recibido

- **sent_files_v2** En esta tabla podemos encontrar los archivos enviados y su ubicación en la memoria interna.

  Podemos ver que cada uno de las tablas necesita de otras para poder completar los datos por lo tanto es necesario hacer relaciones entre ellas para poder sacar información con sentido. En el siguiente apartado se hace demostración de esto haciendo querys a la base de datos.

### 3.5.3 Ver Lista de todos los usuarios

Podemos sacar un listado de todos los usuarios tanto los agregados por el usuario de el dispositivo como los agregados por los grupos de chats utilizando la siguiente query

```
SELECT * FROM users
```

### 3.5.4 Ver Contactos en contacts

Para ver los contactos que tiene en Telegram y son conocidos por el usuario es decir que los tiene en la agenda y no son de un grupo debemos de ver los usuarios que coinciden con la UID en la tabla de contactos

Se puede hacer con la siguiente query

```
SELECT * FROM users WHERE uid IN (SELECT uid FROM contacts)
```

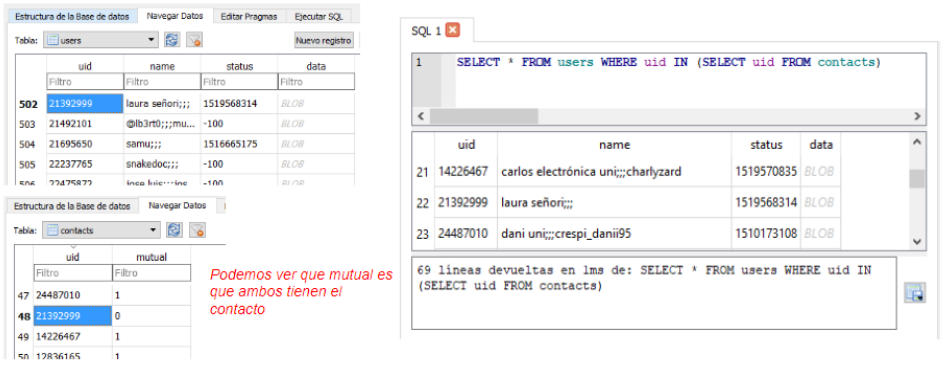



Figura 3.14: Relación de datos en la base de datos contacts y users

### 3.5.5 Ver Contactos Bloqueados

Podemos ver los contactos bloqueados como hemos visto previamente hay una tabla llamada blocked_users si en esta tabla aparece el UID de un contacto o usuario podemos ver los contactos que han sido bloqueados por el usuario u podemos sacar el nombre e información de estos contactos.

```
SELECT * FROM contacts WHERE uid IN (SELECT uid FROM blocked_users)
```

```
SELECT * FROM users WHERE uid IN (SELECT uid FROM blocked_users)
```

### 3.5.6 Ver mensajes de un contacto específico

Si queremos sacar todos los datos de un usuario específico sabiendo su uid podemos hacer la siguiente querys

```
SELECT * FROM messages WHERE uid = x
```

### 3.5.7 Ver todos los BOTS utilizados en los grupos

Telegram permite crear y utilizar BOTS en grupos de chats estas pueden ser para obtener noticias avisos y notificaciones de una manera automatizada. Cuando una de estas se agrega se almacenan en la tabla bot_info Podemos sacar un listado de los bots con la siguiente query

```
SELECT * FROM bot_info WHERE uid = x
```

### 3.5.8 Mensajes eliminados o secretos

Las bases de datos sqlite para mejorar el rendimiento hacen uso de un sistema llamado WriteAheadLog el la que se van almacenando las acciones a la base de datos temporalmente. En el caso de telegram para la base de datos cache4.db el archivo de esta es cache4.db-wal. De aquí podemos ir sacando mensajes borrados o de chat secreto que se van almacenando en este archivo [19]. Tirando el comando

```
strings cache4.db-wal | grep mensajeborrado
```

Podemos observar que el mensaje que habíamos creado y borrado lo podemos encontrar en este archivo mucho después de haberse borrado.

```
nicotrial@nicotrial-VirtualBox:~/Downloads/telegramextracted/org.telegram.messen
ger/files$ strings cache4.db-wal |grep borrado
5K()Esto es un mensaje borrado mensajeborrado
5K()Esto es un mensaje borrado mensajeborrado
5K()Esto es un mensaje borrado mensajeborrado
5K()Esto es un mensaje borrado mensajeborrado
```

Figura 3.15: Mensaje borrado en cache4.db-wal

Aquí podemos ver una comparativa de los dos archivos


Figura 3.16: Comparacion cache4.db con cache4.db-wal vemos el mensaje de chat secreto

### 3.5.9 Extracción de los mensajes

Como se ha mencionado anteriormente dentro de messages en la base de datos cache4.db encontramos los siguientes el campo data en la que tenemos datos en binario y en los que si hacemos un strings parece contener el texto de todos los mensajes de Telegram.

```
nicotrial@nicotrial-VirtualBox:~/Downloads/telegramextracted/org.telegram.messenger
/files$ strings cache4.db |grep mensaje
9K[9Este es un mensaje normal enviado por mi mensajenormalmio
Esto es una foto mensajefoto
deo mensajevideo
%.4K['Esto es un mensaje normal mensajenormal
Esto es una foto mensajefoto
deo mensajevideo
```

Figura 3.17: Texto de mensajes dentro de cache4.db con strings

Si sacamos solo el texto de estos datos tendremos los mensajes pero no sabemos a quien van dirigidos ni quien los ha enviado, si pertenece a un grupo, tipo de mensaje o cuando se ha enviado. Vemos que en los datos almacenados en la base de datos hay mas información binaria antes de el texto de el mensaje. Estos datos binarios nos interesa sacar para poder reconstruir el mensaje y decodificar y poder dar sentido a los mensajes. Viendo un ejemplo de mensaje lo podemos encontrar en el siguiente formato.

```
=´ùC. . . . . S« %
n½± ]T□. 2@K[ . . . .
×p°□. . . . Ç+#□3. . .
T. YR¯ . □hĈêë□1@K[
. appl i cat i on/ zi p
l µH. <â. . . . . .
. . . . . Äµ. . . . h. Y.
. t el egr anext r act
ed. zi p. . . . . .
```

```
3db4f944020300000c00000053aba0256dbcb19d5d548c094c494b5b00000000d770b09c01
000000c72b238736060000541c5952395803182c25af374c494b5b186170706c6963617469
6f6e2f6f637465742d73747265616d000000001000003ce2170e00000000040000000000000
015c4b51c0100000068005915096361636865342e646200003f2f73746f726167652f656d756
c617465642f302f446f776e6c6f61642f74656c656772616d6578747261637465642f73646361
72642f6361636865342e6462
```

Figura 3.18: Ejemplo de una trama de mensaje

Como podemos observar entre varios caracteres raros encontramos el texto claro del mensaje esto nos dice que no sólo contiene el mensaje de texto si no que hay mas meta datos antes de el mensaje y después de el mensaje.

Tenemos la suerte de que el código fuente de Telegram está en GitHub para poder descifrar el contenido [20].

Analizando el código nos damos cuenta de que los datos están serializados esto quiere decir que los datos están uno detrás de otro observando varios mensajes de la base de datos podemos ver que cada dato está categorizada con una cabecera de 32 bits que va cambiando dependiendo de el tipo de mensaje. Vemos que se van haciendo lecturas de 32 en 32 bit por lo tanto observemos los primero 32 bit de él data

Observando varios mensajes parece ser que los primeros 32 bit son cabeceras para identificar el tipo de mensaje ya que en todos se repiten. En el ejemplo anterior haciendo varias búsquedas de el código 3db4f94 en el código fuente de Telegram no aparece nada por lo tanto me da la pista de que puede estar en little endian.

Viendo la cabecera de 32bit "Esta está codificada en little endian" y lo podemos colocar como

```
3db4f944 en little endian nos queda 44 f9 b4 3d
(derecha izquierda de dos en dos)
```

en la búsqueda de esta misma coincide con

```
public static class TL_message extends Message {
    public static int constructor = 0x44f9b43d;
```

Este código pertenece la categoría "TL message" y es interpretado en el archivo TLRPC.java dentro de el código fuente de la aplicación de Telegram [21]. Aquí ya podemos identificar el tipo de mensaje y podemos ver cómo se decodifica el stream para poder extraer los datos que queremos

Después de la cabecera se encuentran otros 32 bit que son flags. Por cada flag se van leyendo distintos datos y se va actuando sobre ellos. Los flags que se han identificado en el caso de el TL_Message son los siguientes

- out
- mentioned
- media
- unread
- silent
- post
- from_id
- fwd_from
- via_bot_id
- replay_to_msg_id
- media, replay_markup
- magic
- views
- edit_date
- post_author
- grouped_id

Cada flag se activa en una posición distinta de el número binario de 32 bits, pueden estar varias flags activadas a la vez por lo que revisamos cada una de ellas si una está activa deberemos leer los datos que nos pide el flag y decodificarlos

Esto hace que el tamaño de el mensaje varía dependiendo de si unos flags están activos o no, por lo tanto complica la extracción ya que no están en una posición fija.

Con los flags vamos viendo que bit está a 1 y podemos ver que acción tomar Tomando el ejemplo puesto anteriormente tenemos como flags 0x02030000 en little endian es

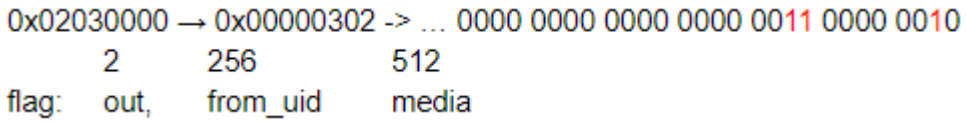



Figura 3.19: Transformación de little endian e interpretación de flags

En este caso vemos que las acciones que Telegram toma para este mensaje son las siguientes

```
flag 512     if ((flags & 512) != 0) {
               media = MessageMedia.TLdes

flag 256   if ((flags & 256) != 0) {
                 from_id = stream.readInt32(exception);


flag 2       out = (flags & 2) != 0;
```

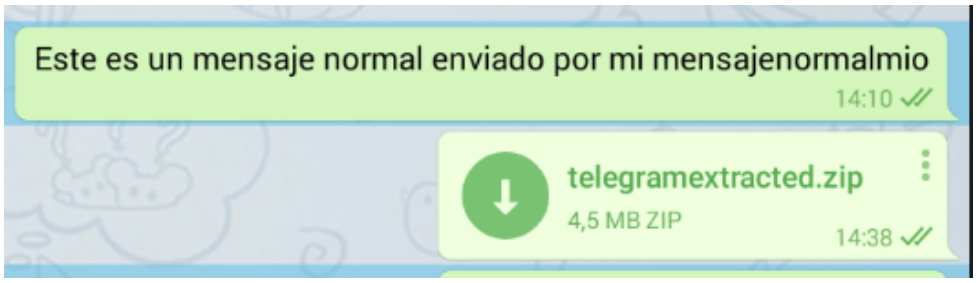


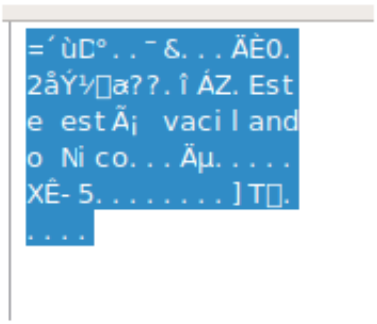

Figura 3.20: Captura de pantalla movil mensaje Telegram

vemos que hemos sido mencionados tiene un stream de texto y es un mensaje saliente.

### 3.5.10 Ejemplos

Ejemplo1:

El mensaje grupo Cátedra es de Carlos Catedra pone “Este esta vacilando Nico” 2 abril 10:47 Administrador




3db4f944b0010080 26100000c4c83000
32e5ddbd9be63f3f0aeec15a19457374
6520657374c3a120766163696c616e64
6f204e69636f000015c4b51c01000000
58ca2d3514000000040000005d548c09
00000000

Cabecera: 0x3db4f944 ----> little endian *0x44 f9 b4 3d*
Flags:0xb0010080 ----> little endian 80 00 01 b0

Figura 3.21: Ejemplo 1 Identificación de cabecera y flags

Ejemplo2:

Enviado por mi a Junquera “No entiendo por que va guardando de izq a derecho de cada valor”

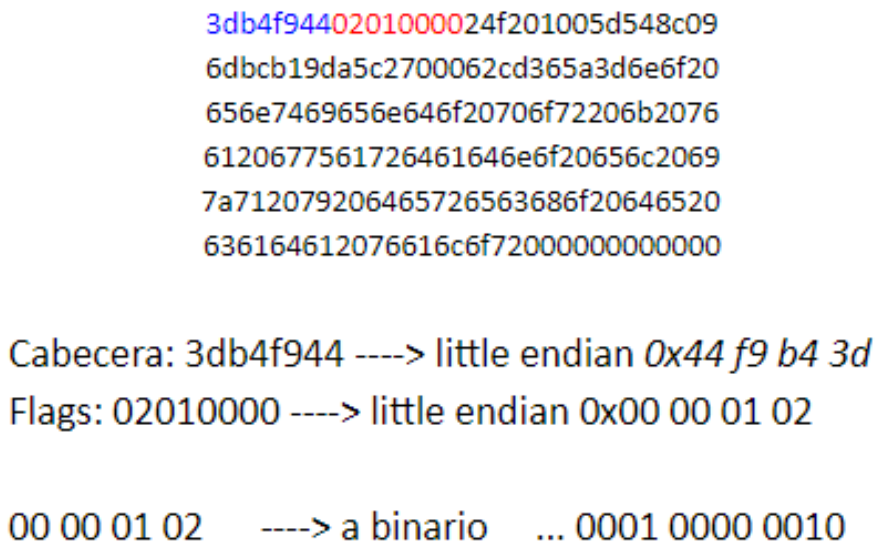



Figura 3.22: Ejemplo 2 Identificación de cabecera y flags

Estas flags activan este trozo de codigo

```
flag 2      out = (flags & 2) != 0;
flag 256      if ((flags & 256) != 0) {
                    from_id = stream.readInt32(exception);
```

Se puede observar que dependiendo de el flag que se active hace distintas cosas. En el ejemplo anterior vemos como el flag de la posición 256 activa otra lectura de 32 bits de la trama de datos.

Ejemplo3:

Mensaje de Grupo enviado por Enrique “Pues sì que hay contactos lejanos en la cátedra”

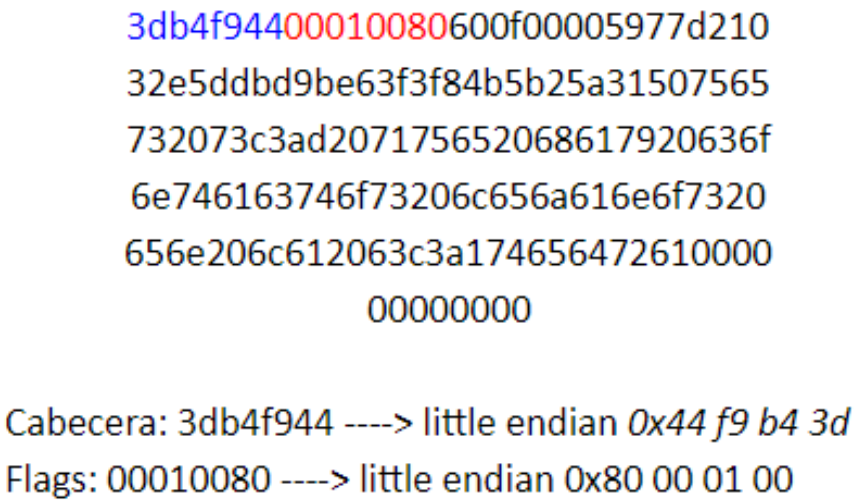



Figura 3.23: Ejemplo 3 Identificación de cabecera y flags

Ejemplo4: Mensaje enviado por Carlos “Es descargar un pdf”

3db4f94400010000dd000200c4c83000
6dbcb19dc4c8300075d0b05a13457320
6465736361726761722075 6e20706466
00000000

Cabecera: 3db4f944 ----> little endian *0x44 f9 b4 3d*
Flags: 00010000 ----> little endian 0x00 00 01 00

Figura 3.24: Ejemplo 4 identificación de cabecera y flags

En un TL_message entre los flags encontramos el mensaje en texto claro. Este está siempre en una posición distinta dependiendo de los flags anteriormente activas. Esta tampoco tiene un tamaño fijo por lo que tenemos que ver donde comienza y dónde termina.

El flag de media es el que nos variara la interpretación de el mensaje si el flag de media está activo tendrá más meta datos en el caso de que no esté activo el flag de media es un mensaje de texto normal.


Figura 3.25: Estructura de la trama de datos de mensaje

### 3.5.11 Datos extraídos de la trama mensaje

Utilizando la herramienta online de ascii to hex de la siguiente pagina web [22] podemos observar que 0x4e6f es la palabra “No” vemos que tiene datos delante en el mensaje puede ser un posible indicador de tamaño de el string, en el resto de mensajes no vemos ningún carácter que indique fin de mensaje que nos coincida en el resto de mensajes.

```
Jajajaja @carloslannister tienes que insistir así de bien con los paper
```

9be63f3fd333d25a484a616a616a616a6120406361726c6f736c616e6e69737465722074696
56e6573207175652069 6e73697374697220617 3c3ad206465206269656e20636f6e206c6f73
207061706572000001 5c4b51c010000009d5704fa0900000010000000000000000

Figura 3.26: Trama de data de los mensajes en la base de datos cache4.db

“Jajajaja” = 0x4a616a616a61

9be63f3fd333d25a484a616a616a616a6120406361726c6f736c616e6e69737465722074696
56e65732071756520696e73697374697206173c3ad206465206269656e20636f6e206c6f73
20706170657200000015c4b51c010000009d5704fa0900000010000000000000000

Figura 3.27: Localización de el texto de el mensaje

Primera palabra “Jajajaja” = 0x4a616a616a61
Última palabra “paper” = 0x7061706572

Tenemos en el texto 72 caracteres, En hexadecimal 74 = 0x48 esto coincide con el número detrás de el texto

Jajajaja @carloslannister tienes que insistir así de bien con los paper

9be63f3fd333d25a484a616a616a616a6120406361726c6f736c616e6e69737465722074696
56e65732071756520696e73697374697206173c3ad206465206269656e20636f6e206c6f73
20706170657200000015c4b51c010000009d5704fa0900000010000000000000000

Figura 3.28: Localización de tamaño de mensaje

Buscando en la base de datos los primeros 32 bits pasado a little endian y luego a decimal vemos que coincide con la UID de el grupo de chat de Telegram que nos ha enviado el mensaje por lo tanto podemos utilizar esta para saber a que grupo pertenece el mensaje y la uid de la base de datos nos indica quien lo ha escrito.

0x9be63f3f –>0x3f3fe69b –>1061152411

| | | | |
|---|---|---|---|
| 50 | 1056186295 | impresión 3d - imprímete | BLOB |
| **51** | 1061152411 | fail-startup | BLOB |
| 52 | 1061561947 | infosec - es | BLOB |

Figura 3.29: UID de grupo en base de datos cache4.db

9be63f3f = FromUid

9be63f3fd333d25a484a616a616a616a6120406361726c6f736c616e6e69737465722074696
56e65732071756520696e73697374697206173c3ad206465206269656e20636f6e206c6f73
20706170657200000015c4b51c010000009d5704fa0900000010000000000000000

Figura 3.30: Datos con identificación marcada


Figura 3.31: Identificación de los datos

El ultimo valor que nos queda por descifrar es el marcado en verde en la figura. Este coincide con el valor de tiempo EPOCH que también esta en little endian. 0xd333d25a ->0x5ad233d3 ->1523725267

https://www.epochconverter.com/


Figura 3.32: Conversor Epoch

Este valor pertenece a el tiempo en el que el mensaje se ha escrito. El resto de datos son de menos importancia por lo que no vale la pena descifrar ya que no nos aportan nada de información de interés.


Figura 3.33: Estructura de la trama de mensaje

Por lo tanto con esto ya podemos reconstruir los mensajes para poder visualizarlos cronológicamente, podemos saber quien lo ha escrito, conocemos el mensaje, con los UID podemos sacar el grupo al que pertenece sabemos de que tipo son y si son enviados o recibidos y otros datos que nos proporcionan los flags. Estos datos combinados con los que podemos sacar en la propia base de datos nos dará suficiente para una reconstrucción de los mensajes.

# Capítulo 4

# Resultados

En este apartado mostraremos como a partir de el estudio y la metodología que hemos utilizado podemos llegar a crear una herramienta especifica para la aplicación de Telegram, luego se mostrara unos ejemplos de como se podrá sacar información visualizando los datos con nuestra herramienta.

## 4.1 Desarrollo de la Herramienta

Con toda la información anterior somos capaces ahora de construir una herramienta para poder extraer los datos de una forma automatizada. Para el desarrollo de este programa se ha optado por el lenguaje de programación Python. Esta aplicación cuenta con distintas partes que veremos en detalle

### 4.1.1 Adquisición de aplicación del dispositivo

Esto es una opción que es opcional ya que si se ha obtenido los datos de otra forma también se podrán cargar. Para conseguir los datos utilizamos ADB que pondrá el móvil en modo fastboot y cargara una Custom Recovery en RAM por la cual extraemos los datos de la aplicación. Para esto necesitaremos tener la ISO de el Custom Recovery Corespondiente a nuestro dispositivo en la carpeta de TOOLS/TWRP

Listado 4.1: Root y extracción de datos móvil

```
def extractApp():
        hackerPrint("[-] Extracting telegram app data from phone\n", "GOOD", True)
    os.system("""./Tools/platform-tools/adb reboot bootloader""")
    time.sleep(20)
    os.system("""./Tools/platform-tools/fastboot boot ./Tools/TWRP/twrp.img""")
    time.sleep(20)
    os.system('''./Tools/platform-tools/adb pull /data/data/org.telegram.messenger/files/
        cache4.db dump''')
    hackerPrint("[-] DONE!\n", "GOOD", True)
```

### 4.1.2 Adquisición de datos de la base de datos

Con los datos que se han extraído de el dispositivo con nuestro programa nos conectamos a la base de datos cache4.db. Aquí hacemos la adquisición de los datos que se pueden extraer directamente de la base

de datos con las querys de SQL. Esta no nos crea gran complicación ya que es simplemente relacionar los datos de la base de datos y mostrarlos por pantalla.

Listado 4.2: Adquisición de datos de la base de datos

```
def extractContacts(c, conn):
    print("----------=======Extracting Known Contacts=======----------")
    for row in c.execute('SELECT * FROM users WHERE uid IN (SELECT uid FROM contacts)'):
        print(row[1])
        print(row[0])
        print("----------==============================")

def extractBots(c, conn):
    print("----------=======Extracting Bots info=======----------")
    for row in c.execute('SELECT * FROM bot_info'):
        for match in re.finditer(('([\w/]{%s}[\w/]*)' % 1).encode(),row[1]):
            print (match.group(0)),
        print("\n")
        print(row[0])
        print("----------==============================")

def extractBlockedUsers(c, conn):
    print("----------=======Extracting Blocked Users=======----------")
    for row in c.execute('SELECT * FROM contacts WHERE uid IN (SELECT uid FROM blocked_users)'
        ):
        print(row)
        print("----------==============================")

def extractUsers(c, conn):
    print("----------=======Extracting All Users=======----------")
    for row in c.execute('SELECT * FROM users'):
        print(row[1])
        print(row[0])
        print("----------==============================")
```

### 4.1.3 Adquisición de mensajes

Los mensajes guardados en Telegram están serializados como se ha explicado en el capitulo anterior. A la hora de decodificar los mensajes de los chats se hará un puntero que ira recorriendo los datos de el mensaje. Esta leerá el tipo de los primeros 32 bits y los flags que encontramos en los siguientes 32 bits de la trama.

Listado 4.3: Interpretación de Cabecera

```
    if header == 0x44f9b43d:  # type TL_message
        messagetype = "TL_message"
        flags = struct.unpack('<i', trama)[0]
```

Cada flag esta en una posición determinada que hemos podido sacar de el código fuente de Telegram. A partir de aquí iremos actuando haciendo caso a los flags que están activos de los que se han leído. En el caso de los de tipo TL_message hay unos flags que indicaran el tipo de mensaje y sus características por cada flag guardaremos estos datos. Estos flags son:

out: Este flag indica que es un mensaje saliente nuestro mentioned: Este flag indica que hemos sido mencionados en el mensaje media Unread: Esta flag indica que no se ha leído silent: Este flag indica que es un mensaje sin notificación post: Este flag indica que es un post

El resto de flags activara distintas lectura de datos que las iremos guardando en sus variable correspondiente a la vez moviendo el puntero por cada lectura.

Listado 4.4: Interpretación de Flags

```
        if (flags & 2) is not 0x0:
            # Este flag indica que es un mensaje saliente nuestro
            flagsActive.append("out")

        # print(hex(flags & 16))
        if (flags & 16) is not 0x0:
            # Este flag indica que hemos sido mencionados en el mensaje
            flagsActive.append("mentioned")

        # print(hex(flags & 32))
        if (flags & 32) is not 0x0:
            flagsActive.append("media Unread")

        # print(hex(flags & 8192))
        if (flags & 8192) is not 0x0:
            flagsActive.append("silent")

        # print(hex(flags & 16384))
        if (flags & 16384) is not 0x0:
            flagsActive.append("post")

        trama = message[tramacursor:tramacursor + 4]
        tramacursor = tramacursor + 4
        ids = struct.unpack('<i', trama)[0]
        ...
        ...
        # print(hex(flags & 2048))
        if (flags & 32768) is not 0x0:
            flagsActive.append("edit_date")
            tramacursor = tramacursor + 4

        # print(hex(flags & 2048))
        if (flags & 65536) is not 0x0:
            flagsActive.append("post_author")

        # print(hex(flags & 2048))
        if (flags & 131072) is not 0x0:
            flagsActive.append("grouped_id")
```

En el caso de la trama de datos de los mensajes de texto tendremos que ver también el Timestamp, el from_UID, el tamaño de el texto incluyendo el texto. Para esto vamos moviendo el puntero acorde con los datos de tamaños que vamos recibiendo ya que si aquí fallamos el resto de datos estarán descuadrados.

Listado 4.5: Interpretación de trama de datos de mensajes

```
        trama = message[tramacursor:tramacursor + 4]
        tramacursor = tramacursor + 4
        date = struct.unpack('<I', trama)[0]

        trama = message[tramacursor:tramacursor + 4]
        tramacursor = tramacursor + 4
        #tramaMsg = binascii.hexlify(trama)
        fromuid = struct.unpack('<i', trama)[0]
        c.execute('SELECT name FROM chats WHERE uid=%s' % fromuid)
        chatsstream = c.fetchall()
        for rowss in chatsstream:
            usernameChats = rowss[0]
        c.execute('SELECT name FROM users WHERE uid=%s' % fromuid)
        chatsstream = c.fetchall()
        # conn.commit()
        for rowss in chatsstream:
            usernameUsers = rowss[0]
        trama = message[tramacursor:tramacursor + 4]
        tramacursor = tramacursor + 4
        timestamp = struct.unpack('<i', trama)[0]
        trama = message[tramacursor:tramacursor + 1]
        tramacursor = tramacursor + 1
        bytes = struct.unpack('<b', trama)[0]
        # bytes = int.from_bytes(trama[0:9], byteorder='little')
        size = bytes
        trama = message[tramacursor:tramacursor + bytes]
        tramacursor = tramacursor + bytes
        messagetext = str(trama)

        # trama de mensaje que queda por decodificar
        trama = message
        tramaMsg = binascii.hexlify(trama)

        # print(hex(flags & 512))
        if (flags & 512) is not 0x0:
            # Si es media imprimimos el mensaje hay tambi n datos de la ubicaci n de el
                archivo guardado hay que intentar sacar esto tambi n
            flagsActive.append("media")
```

### 4.1.4 Almacenamiento de datos para la visualización

Los datos los almacenaremos en Elastic Search Elasticsearch puede ser usado para buscar todo tipo de documentos. La búsqueda es escalable y casi en tiempo real, soportando multi-tenencia y es perfecto para la utilización que le vamos a dar.

Para poder usarlo necesitamos Java Runtime Environemnt Necesitamos descargar la aplicación de Elastic Search /elasticsearch-6.2.4/bin$ ./elasticsearch con esto la ejecutamos y nos creara un servidor en localhost:9200

Cada uno de los datos irán guardados con una categoría poder identificarlos Ejemplo:

interactuamos con la base de datos desde Python y almacenaremos todos los datos extraídos de la aplicación de Telegram para que se pueda hacer búsquedas y análisis de los datos extraídos.

Para poder facilitar la visualización de los datos utilizaremos Kibana Kibana es una herramienta open-source perteneciente a Elastic, que nos permite visualizar y explorar datos que se encuentran indexados en ElasticSearch.

/kibana-6.2.4-linux-x86_64/bin$ ./kibana Para instalar Kibana nos descargamos los archivo y ejecutamos el servidor una vez ejecutado la de ElasticSearch. Esto nos abrirá un servidor en la siguiente dirección http://localhost:5601

## 4.2 La Herramienta

Aquí ya tenemos la herramienta completada por lo que podemos ya visualizar los datos de Telegram. En este programa tenemos varias opciones en la que vamos extrayendo los datos que se han comentado anteriormente. A continuación podemos ver que todo el trabajo anterior ha servido para poder obtener la información que antes era ilegible a algo que podemos utilizar para un análisis forense

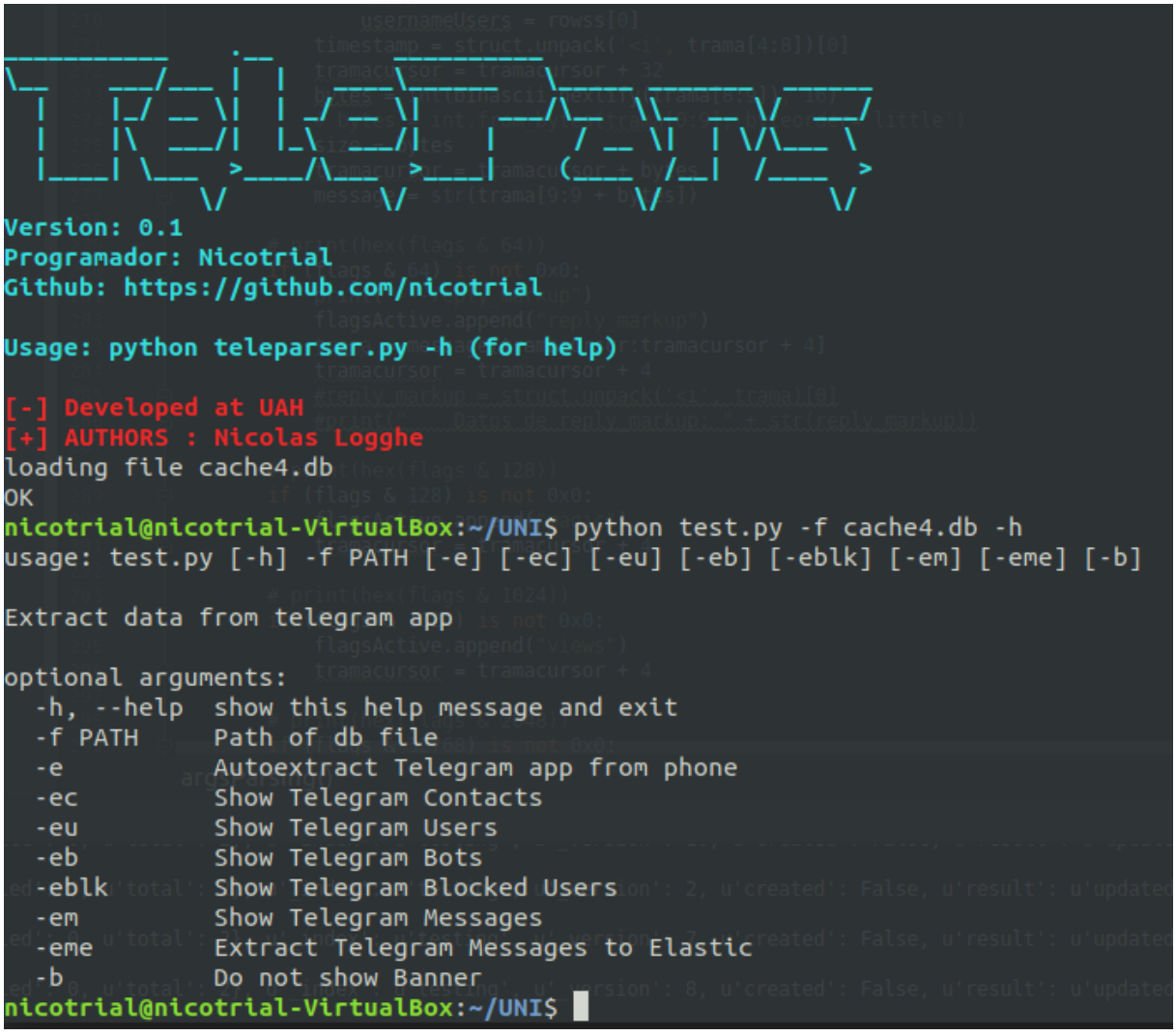



Figura 4.1: Telepars menú de inicio

## 4.3 Extracción de datos directos

Como se ha mencionado anteriormente podemos extraer todos los datos que están directamente en la aplicación. Aquí tenemos una extracción de los contactos que están agregados por el propio usuario desde su lista e contactos.

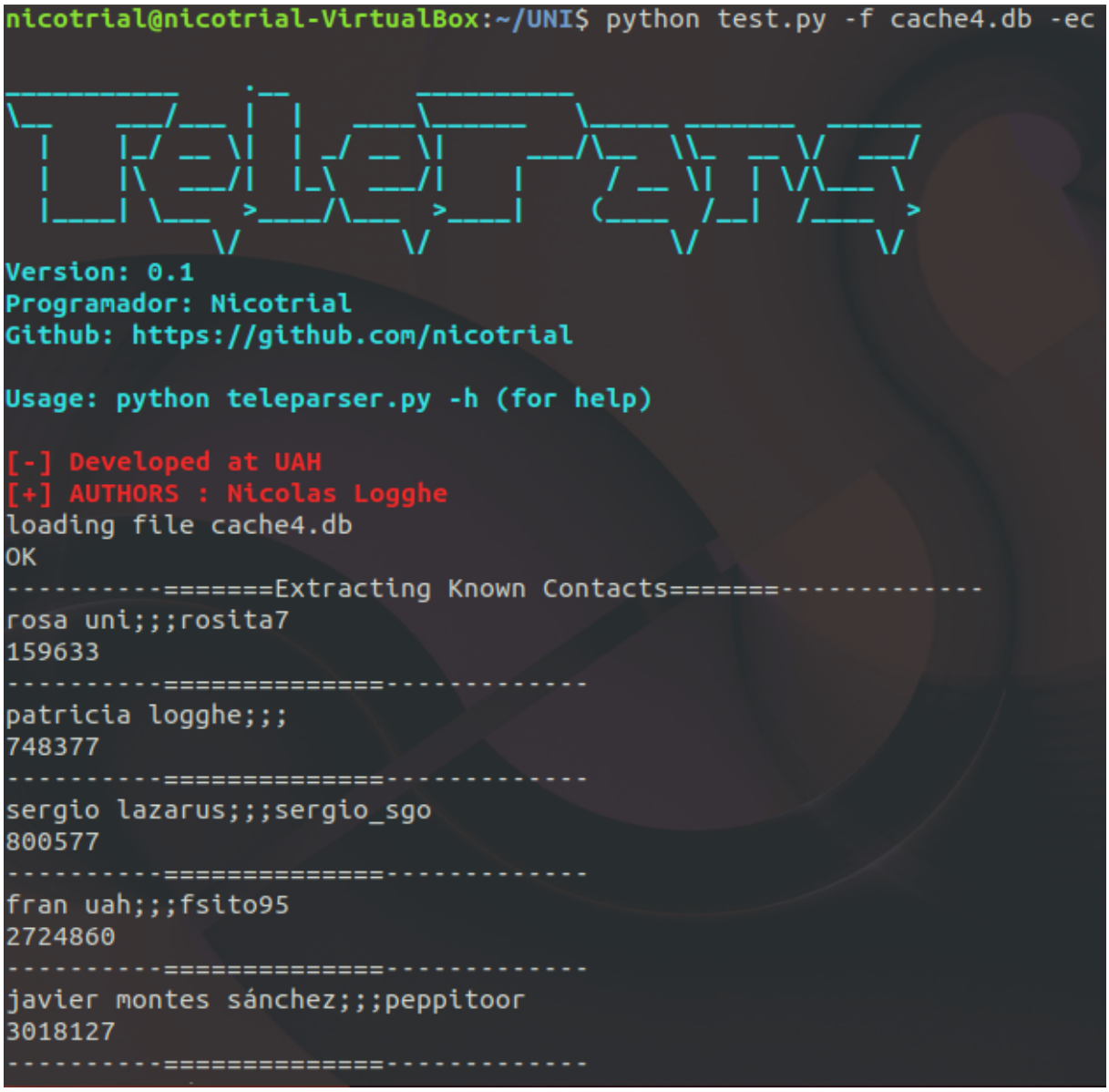



Figura 4.2: Extracción de contactos conocidos

Podemos observar que tenemos los mensajes con su tiempo y q quien van dirigido, incluso podemos ver el nombre de el grupo al que esta asignado. Toda esta información nos servirá para poder reconstruir los chats en un análisis Forense de este dispositivo y poder sacar la conversación íntegramente.

```
----------============------------------
HeaderHEX: 0x44f9b43d
HeaderType: TL_message
Flags: ['fromu_id', 'regularMessage']
Ids: 129996
TramaMensaje: c4ce9a0fe74f745a22457374652076657261 6e6f2061206e6f732076616d6f7320612
06c6120706c6179610000000000
MessageSize: 34
Message: Este verano a nos vamos a la playa
Date: 3134252475
TimeStamp: 2018-02-02 12:47:51
Fromuid: 261803716
fwd_from_name:
fwd_id_name: josie log;;;
UsernameUsers:
UsernameChats: familia loggify
----------============------------------
```

Figura 4.3: Extracción de mensajes de texto

Podemos observar que tenemos un listado completo de todos los usuarios que tiene registrado el dispositivo en la aplicación Android y no solo los que están guardados en la agenda y podemos ver los bots registrados con su descripción.

```
---------=============-------------
carlos aldama sainz;;;peritoinformaticoforense
17309421
---------=============-------------
eml;;;emalox
17572520
---------=============-------------
patxi50;;;patxi50
17940102
---------=============-------------
chadesx64💜🔥🔵🍔🏺🐉🦉🔱💜🇬🎆 infraworld king💜🔥🔵🍔🏺🐉🦉🔱💜🇬🎆;;;rochkx64
17987914
---------=============-------------
eloi eco9;;;eco_9
18305525
---------=============-------------
ezequiel iriso;;;hwisdown
18512686
---------=============-------------
```

Figura 4.4: Extracción de usuarios de Telegram

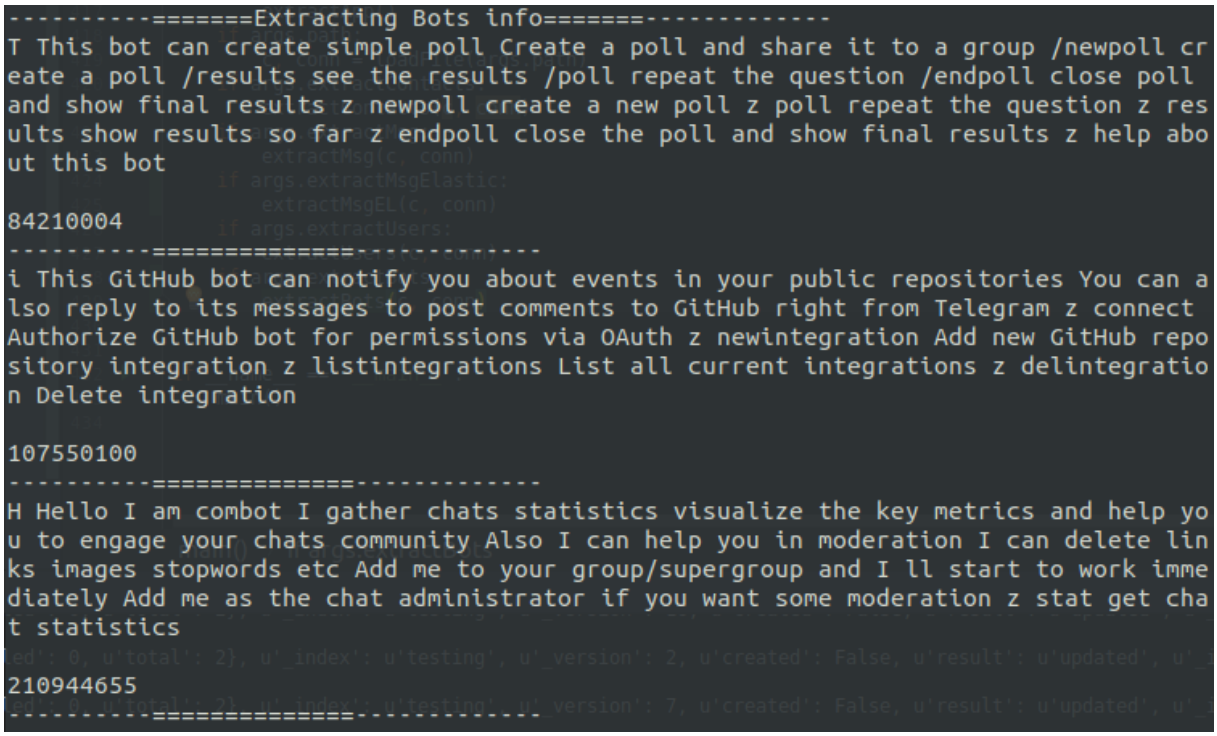



Figura 4.5: Extracción de BOTS de Telegram

## 4.4 Visualización de datos e interpretación

El fin de este trabajo no es la explicación de la interpretación de los datos pero si es importante que lo mencionemos para tener una idea de el potencial que nos puede dar una herramienta que nos permita colocar los datos de manera que podamos visualizarlos y relacionarlos de otras formas. Como hemos mostrado anteriormente tenemos los datos directos los mensajes los contactos pero a partir de estos datos podemos sacar mucha mas información y que puede ser de igual o mas importancia que los anteriores.

En nuestra herramienta almacenamos todos los datos que extraemos con una base de datos Elastic-Search. Esto nos permitirá utilizar el Plugin de Kibana para poder hacer búsquedas y visualizaciones sencillas que nos proporcionaran inteligencia apartar de estos datos.

Una vez que tenemos los datos en la base de datos podremos buscar todos los mensajes que pertenecen a un mismo usuario.

| | | |
|---|---|---|
| # | DiaSemana | 7 |
| # | HoraDia | 21 |
| t | _id | 1525037988000 |
| t | _index | testing |
| # | _score | - |
| t | _type | telegram |
| ⏲ | date | April 29th 2018, 23:39:48.000 |
| t | msg.Date | 2645671021 |
| t | msg.Flags | ['out', 'fromu_id', 'media'] |
| t | msg.FromUid | 3197124 |
| t | msg.FwdFromName | |
| t | msg.FwdIdName | nicotrial log;;; |
| t | msg.Header | 0x44f9b43d |
| t | msg.HeaderType | TL_message |
| t | msg.Id | 132438 |
| t | msg.MsgSize | 0 |
| t | msg.MsgText | Hay un ejemplo en la página web ya hecho |
| t | msg.MsgTrama | c4c83000a43be65a2948617920756e20656a656d706c6f20656e206c612070c3a167696e6120776562207961206865636866f00002063ed3d00000000 |
| t | msg.UsernameChats | |
| t | msg.UsernameUsers | carlos catedra;;;carlos |

Figura 4.6: Numero total de mensajes almacenados

En esta caso buscamos el usuario nicotrial en la que podemos ver que tiene un total de 11,058 mensajes

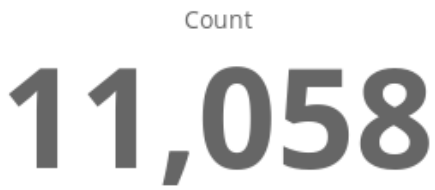



Figura 4.7: Datos en la base de datos ElasticSearch

Podemos visualizar la cantidad de mensajes y la utilización que ha hecho este usuario en la vida entera de la aplicación.

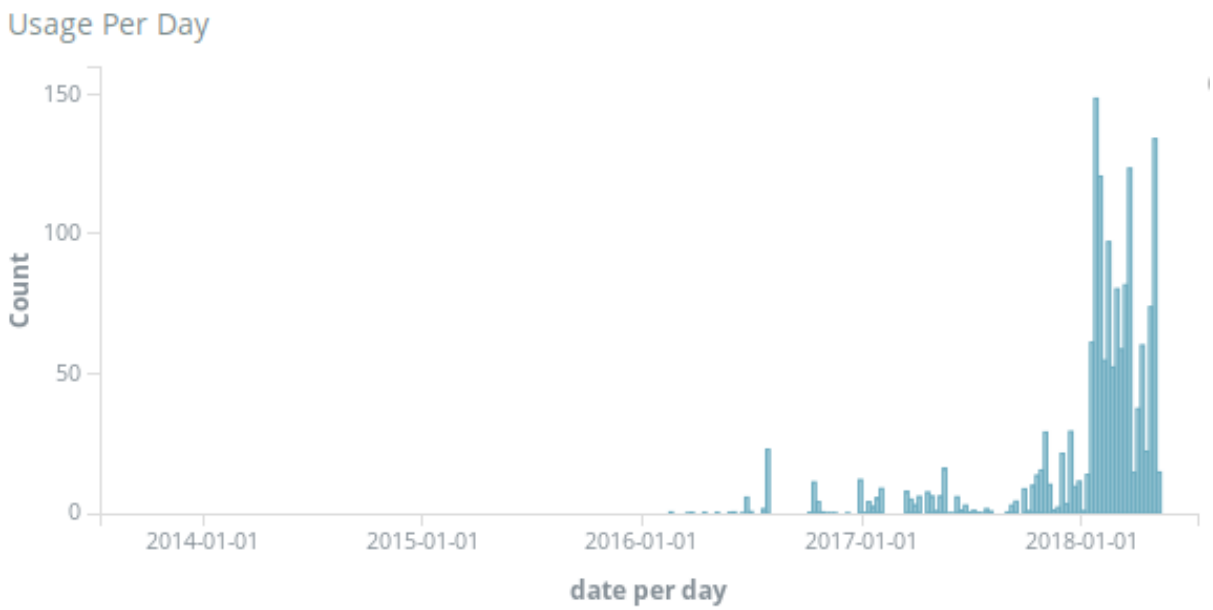



Figura 4.8: Visualización de utilización

Si dividimos en mensajes por día de semana podemos ver el patrón de uso de este usuario podemos sacar conclusiones de que los fines de semana y por la mañana no suele tener disponibilidad para hablar puede indicar su horario de trabajo o alguna actividad esto puede que sea un dato de gran importancia en un caso.

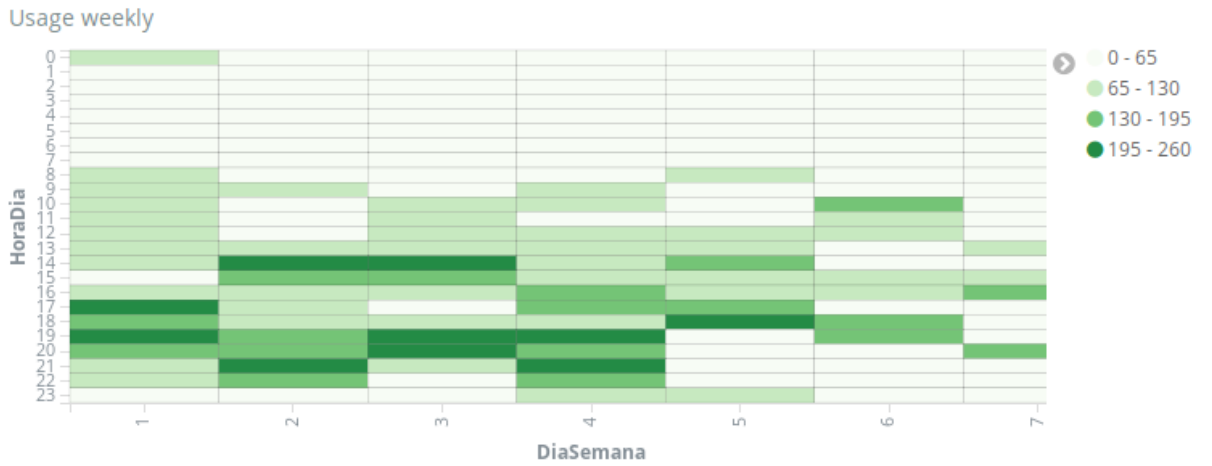



Figura 4.9: Mensajes por día de semana

Aquí podemos visualizar el uso que le da a la aplicación durante el día. Indirectamente de aquí se podría sacar información sobre la ubicación de esta persona viendo el horario en el que no hay actividad y en el que si ya que indica que es de noche y esta durmiendo.

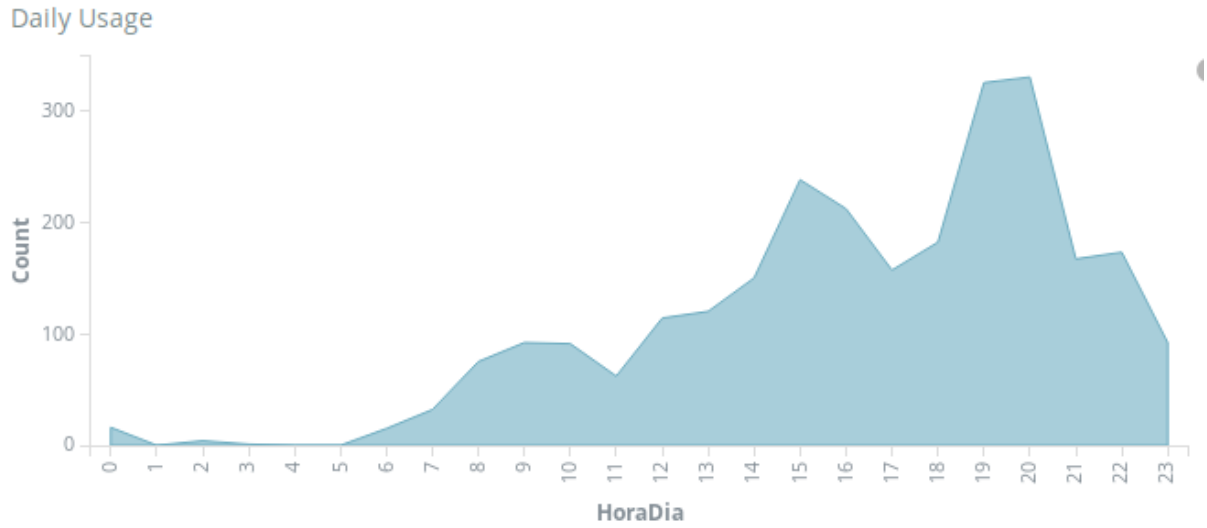



Figura 4.10: Uso aplicación durante el día

En esta figura podemos visualizar los contactos y los grupos con los que mas interactúa el usuario

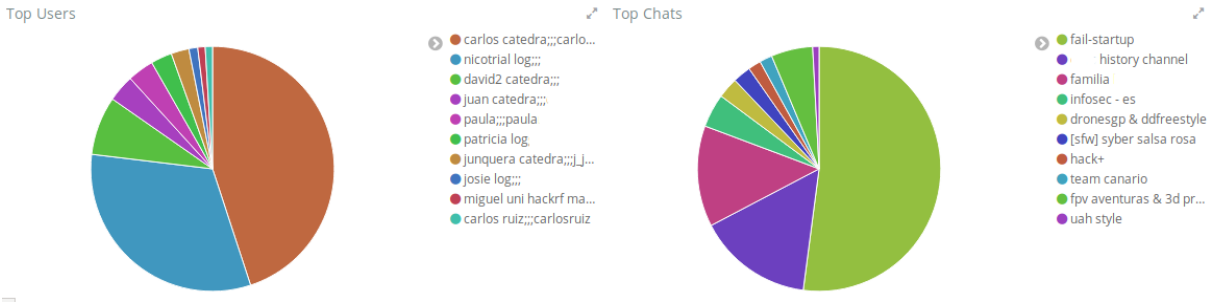



Figura 4.11: Contactos y los grupos con los que mas interactúa el usuario

# Capítulo 5

# Conclusiones y trabajo futuro

## 5.1 Conclusión

Podemos ver que con esta ejemplo se puede sacar una metodología siguiendo estos pasos adaptados a cada situación y se puede realizar una herramienta para la extracción de datos de los dispositivos móvil de Android y que puede ser de gran utilidad para un análisis Forense. Hay millones de dispositivos distintos y de aplicaciones y con el tiempo estas van cambiando por lo que no se puede depender de que exista una sola herramienta que lo pueda hacer todo así que es importante tener estos conocimientos y estas habilidades para poder crear tus propias herramientas en un Analisis forense.

## 5.2 Trabajo futuro

Esta trabajo ha servido como demostración para crear una herramienta que a día de hoy es inexistente y que sirve para el apoyo en un Análisis Forense móvil. La aplicación de Telegram es un aplicación muy compleja en la que cuenta con mas de un millón de lineas de código. Con este ejemplo apenas se a abordado un 1 % de lo que es podría extraer con un análisis en profundidad y con un tiempo menos limitado.

Uno de los trabajos futuro seria hacer un análisis de la aplicación de Telegram con mas profundidad para poder interpretar los cientos de datos que aun han quedado como desconocidos y poder utilizar estos datos para ver si se puede sacar mas información.

Otro trabajo futuro seria hacer esta metodología estandarizada que fuera aplicable en cualquier aplicación y en la que se abarquen todas los posibles caminos que se puedan a llegar a tomar para poder hacer una extracción de datos de una aplicación o dispositivo en un análisis forense de la manera mas efectiva posible.

# Bibliografía


[1] "twilo:how consumers use messaging," https://www.twilio.com/learn/commerce-communications/how-consumers-use-messaging.

[2] "Db browser sqlite explorer," https://telegram.org//.

[3] "whatsapp," https://www.whatsapp.com//.

[4] "wechat," https://web.wechat.com//.

[5] "snapchat," https://www.snapchat.com//.

[6] "facebook," https://www.facebook.com//.

[7] "line," https://line.me/en//.

[8] "Wappalyzer," https://github.com/AliasIO/Wappalyzer.

[9] "whatsappviewer," https://andreas-mausch.de/whatsapp-viewer/.

[10] "Android," https://es.wikipedia.org/wiki/Android.

[11] "Android debug bridge (adb)," https://developer.android.com/studio/command-line/adb.

[12] "Android sdk," https://developer.android.com/studio/releases/platform-tools).

[13] "Teamwin twrp," https://twrp.me/).

[14] "Magnet acquire," https://www.magnetforensics.com/magnet-acquire/.

[15] "Autopsy," https://www.sleuthkit.org/autopsy/.

[16] "Telegram github," https://github.com/DrKLO/Telegram.

[17] "Sqlite," https://www.ecured.cu/SQLite.

[18] "Db browser sqlite explorer," https://sqlitebrowser.org/.

[19] "Sqlite write ahead log," http://www.sqlite.org/wal.html.

[20] "Github telegram," https://github.com/DrKLO/Telegram/.

[21] "Codigo fuente telegram decodificacion de mensajes," https://github.com/DrKLO/Telegram/blob/4ebcbf61cd1ce04685c82f93b962d1094c5ffac2/TMessagesProj/src/main/java/org/telegram/tgnet/TLRPC.java.

[22] "Ascii-to-hex," https://www.rapidtables.com/convert/number/ascii-to-hex.html.

[23] "Ubuntu," https://www.ubuntu.com/.

[24] "Pycharm," https://www.jetbrains.com/pycharm/.

[25] "Artículo de Wikipedia de ," https://es.wikipedia.org/wiki/LaTeX [Castellano].

[26] "Página principal de ShareLatex," https://www.sharelatex.com.

[27] "Página principal de Python," https://www.python.org/.

[28] "Página principal de GitHub," https://github.com/.

# Apéndice A

# Manual de usuario

## A.1 Instalación

Telepars se enstala de el sisignte modo

- Instalar Python3
- Instalar ElasticSearch
- Instalar Kibana
- Clonar Repositorio de GitHub

```
git clone https://github.com/nicotrial/TeleParse
```

## A.2 Utilizacion

```
python teleparser.py -h
Abrira ayuda con todas las opciones disponibles
  -h, --help  show this help message and exit
  -f PATH     Path of db file
  -e          Autoextract Telegram app from phone
  -ec         Show Telegram Contacts
  -eu         Show Telegram Users
  -eb         Show Telegram Bots
  -eblk       Show Telegram Blocked Users
  -em         Show Telegram Messages
  -eme        Extract Telegram Messages to Elastic
  -b          Do not show Banner
```

Primero debemos de obtener los archivos de Telegram.

Si no los tenemos podemos extraerlos con la aplicación automaticamente.

Para esto hay que poner el dispositivo móvil en modo debug y luego enchufar por un cable usb al ordenador.

Antes de nada descargar en TWRP.me la versión de recovery para tu dispositivo y poner en la carpeta de ./TOOLS/TWRP

Ejecutar python teleparse.py -e

Esto el dispositivo se reiniciara solo y nos devolverá la base de datos cache4.db

A partir de aquí podemos extraer la información con las opciones

```
  -ec         Show Telegram Contacts
  -eu         Show Telegram Users
  -eb         Show Telegram Bots
  -eblk       Show Telegram Blocked Users
  -em         Show Telegram Messages

Ejemplo python teleparse.py -f cache4.db -eu
```

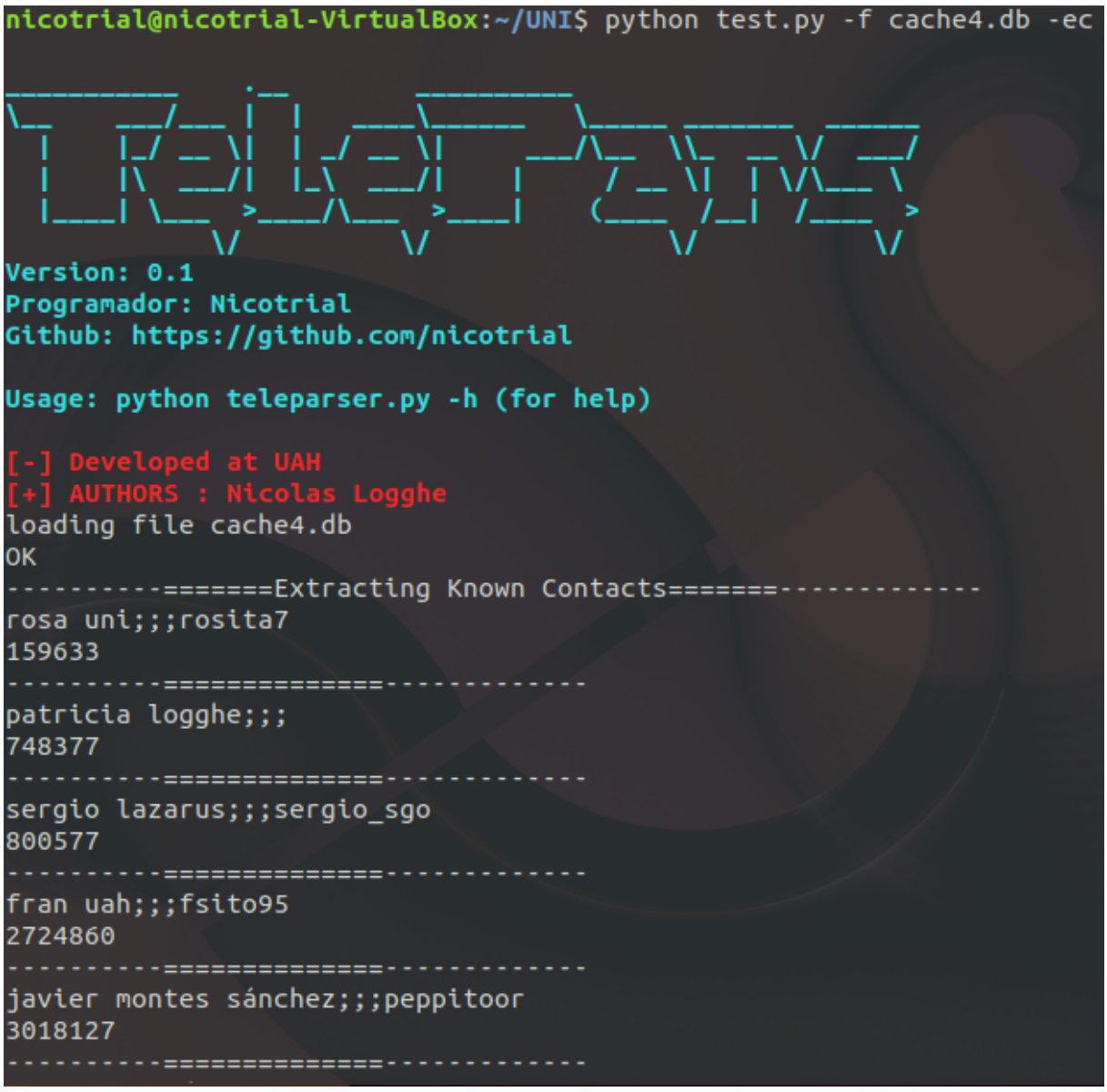



Figura A.1: Extracción de contactos conocidos

Para extraer los datos a la base de datos elasticsearch tenemos la opcion

```
  -eme        Extract Telegram Messages to Elastic
```

Recordar que debemos iniciar Elasticsearch y Kibana con anterioridad

```
Ejemplo python teleparse.py -f cache4.db -eme
```

Con esta podremos visualizar los datos desde Kibana en la dirección http://localhost:5601

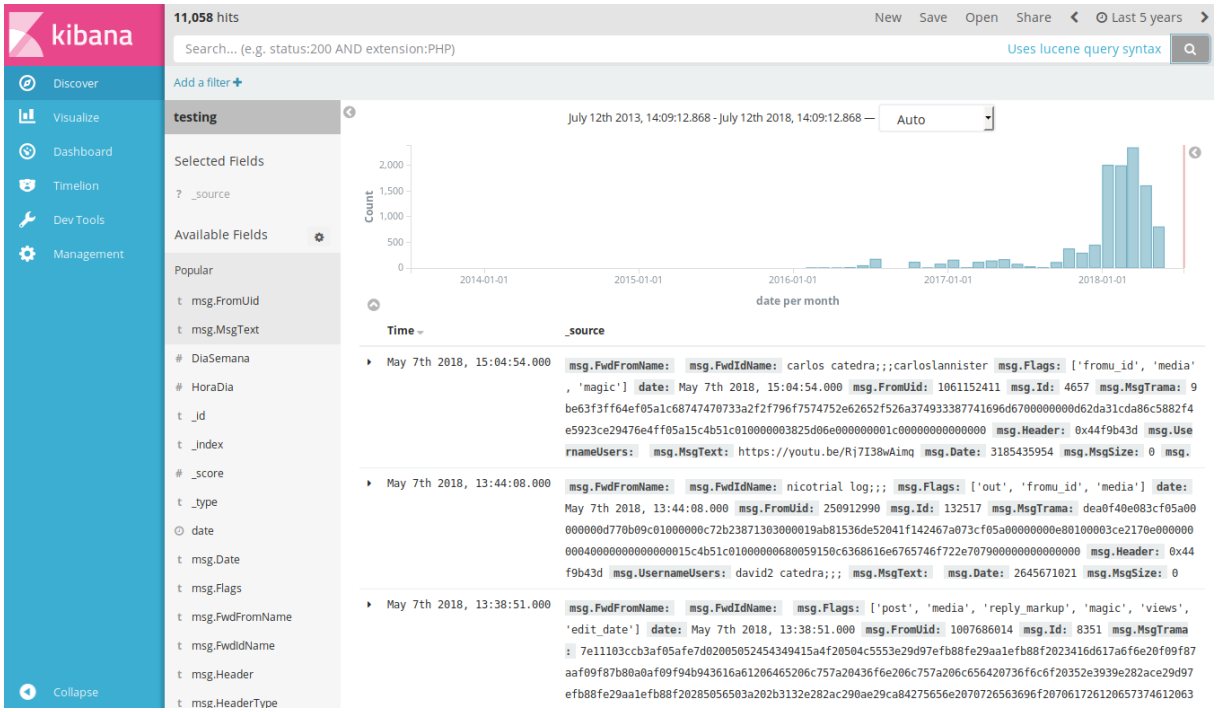



Figura A.2: Numero total de mensajes almacenados

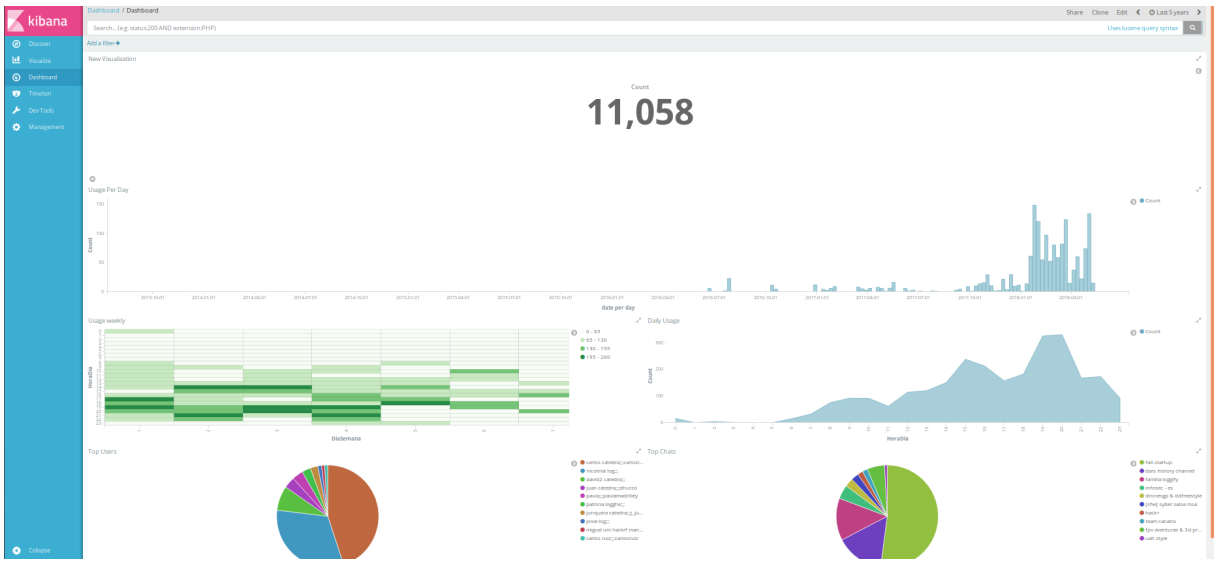



Figura A.3: Numero total de mensajes almacenados

# Apéndice B

# Herramientas y recursos

Las herramientas necesarias para la elaboración del proyecto han sido:

- Ordenador portátil
- Sistema operativo Linux Ubuntu [23]
- Entorno de desarrollo Pycharm [24]
- SDK Android [**?**]
- Dispositivo Moviles con Android LaTeX[25]
- Plataforma ShareLaTeX[26]
- Lenguaje de programación Python [27]
- Servicio de repositorios GIT GitHub [28]
- Base de datos Elasticsearch y Kibana [**?**]
- Software DBrowser [**?**]
- Software Autopsy [**?**]
- Software Magnet Aquire [**?**]
- Café y paciencia [**?**]

# Apéndice C

# Planos, diagramas y tablas

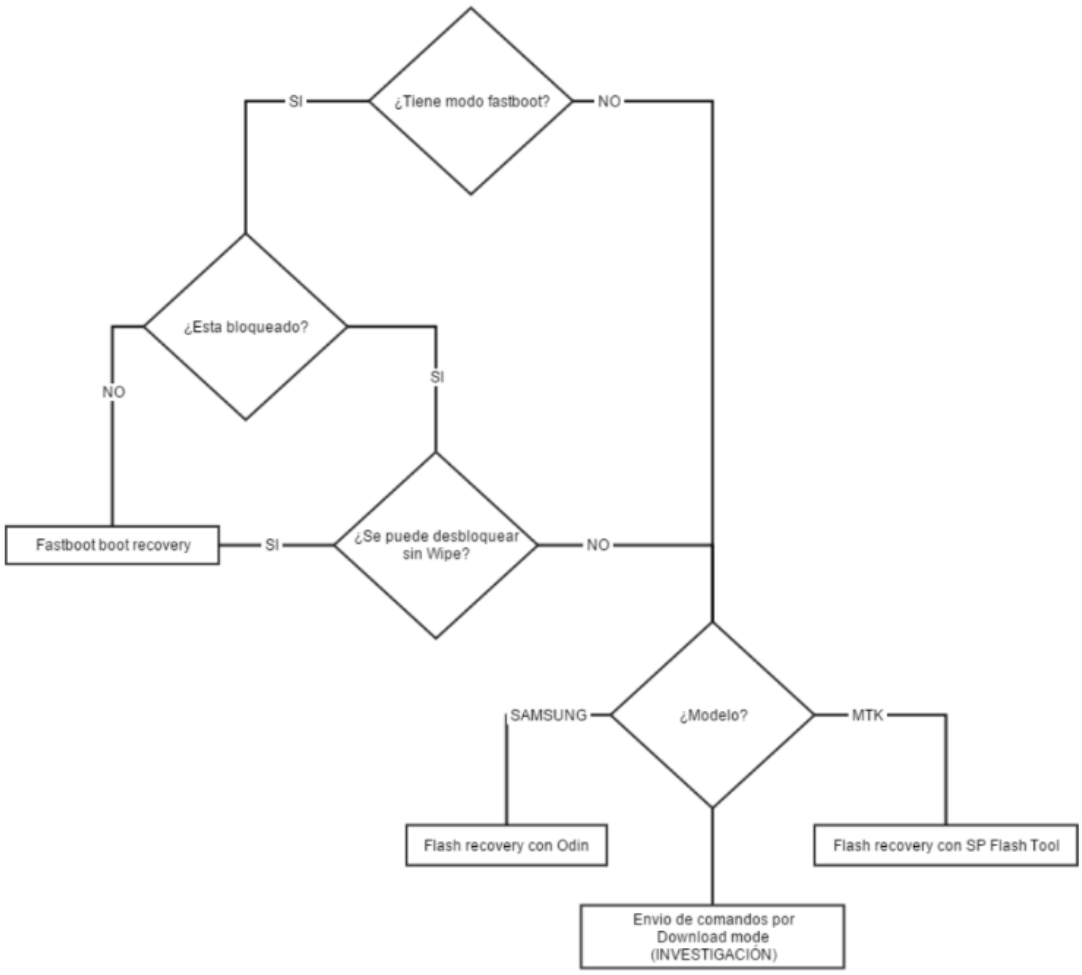



Figura C.1: Flujograma flasheo de Custom Recovery

| | | |
|---|---|---|
| # | DiaSemana | 7 |
| # | HoraDia | 21 |
| t | _id | 1525037988000 |
| t | _index | testing |
| # | _score | - |
| t | _type | telegram |
| ⊙ | date | April 29th 2018, 23:39:48.000 |
| t | msg.Date | 2645671021 |
| t | msg.Flags | ['out', 'fromu_id', 'media'] |
| t | msg.FromUid | 3197124 |
| t | msg.FwdFromName | |
| t | msg.FwdIdName | nicotrial log;;; |
| t | msg.Header | 0x44f9b43d |
| t | msg.HeaderType | TL_message |
| t | msg.Id | 132438 |
| t | msg.MsgSize | 0 |
| t | msg.MsgText | Hay un ejemplo en la página web ya hecho |
| t | msg.MsgTrama | c4c83000a43be65a2948617920756e20656a656d706c6f20656e206c612070c3a167696e6120776562207961206865636866f00002063ed3d00000000 |
| t | msg.UsernameChats | |
| t | msg.UsernameUsers | carlos catedra;;;carlos |

Figura C.2: Numero total de mensajes almacenados

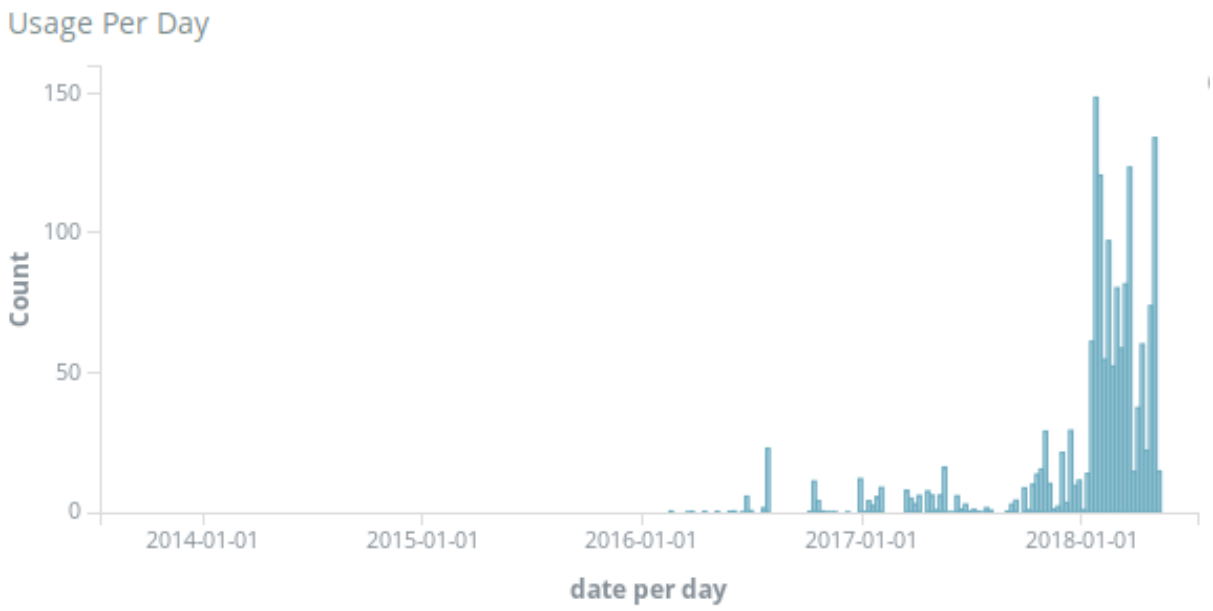



Figura C.3: Visualización de utilización

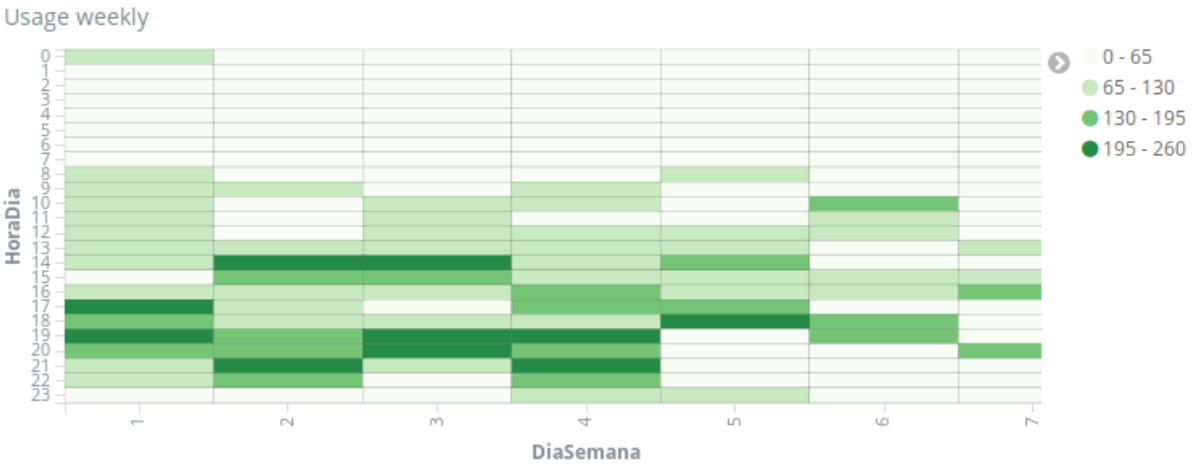



Figura C.4: Mensajes por día de semana

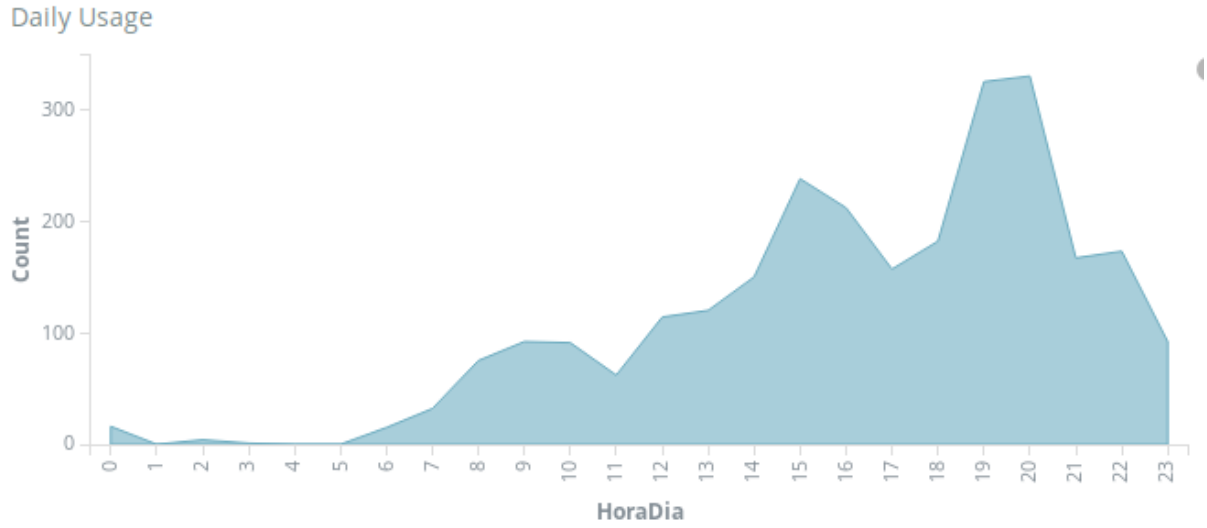



Figura C.5: Uso aplicación durante el día

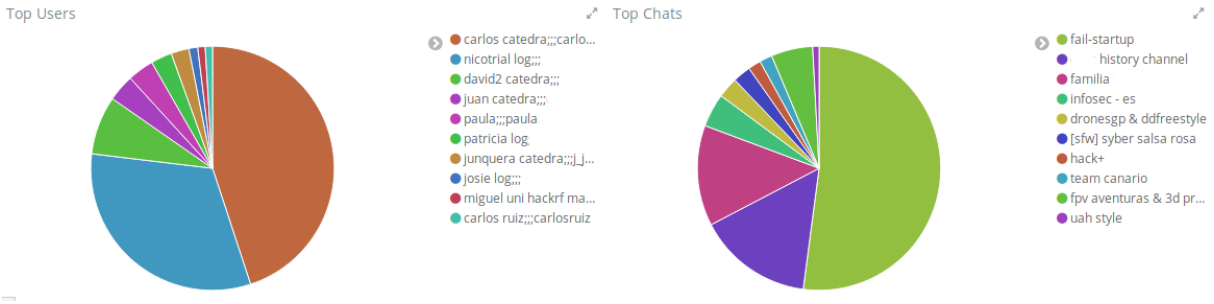



Figura C.6: Contactos y los grupos con los que mas interactúa el usuario


Figura C.7: Estructura de la trama de datos del mesages


Figura C.8: Estructura de la trama de del mensaje

# Apéndice D

# Fragmentos de código

Listado D.1: Root y extracción de la aplicación de telegram con adb

```
def extractApp():
    hackerPrint("[-] Extracting telegram app data from phone\n", "GOOD", True)
    os.system("""./Tools/platform-tools/adb reboot bootloader""")
    time.sleep(20)
    os.system("""./Tools/platform-tools/fastboot boot ./Tools/TWRP/twrp.img""")
    time.sleep(20)
    os.system('''./Tools/platform-tools/adb pull /data/data/org.telegram.messenger/files/
        cache4.db dump''')
    hackerPrint("[-] DONE!\n", "GOOD", True)
```

Listado D.2: Coneccion con la base de datos de teelgram

```
def loadFile(file):
    print("loading file " + file)
    try:
        conn = sqlite3.connect(file)
        c = conn.cursor()
        print("OK")
    except Exception as e:
        raise e
    return c, conn
```

Listado D.3: Extracción de datos de la base de datos

```
def extractBots(c, conn):
    print("----------=======Extracting Bots info=======-------------")
    for row in c.execute('SELECT * FROM bot_info'):
        for match in re.finditer(('([\w/]{%s}[\w/]*)' % 1).encode(),row[1]):
            print (match.group(0)),
        print("\n")
        print(row[0])
        print("----------================================")

def extractBlockedUsers(c, conn):
    print("----------=======Extracting Blocked Users=======-------------")
    for row in c.execute('SELECT * FROM contacts WHERE uid IN (SELECT uid FROM blocked_users)'
        ):
        print(row)
        print("----------================================")

def extractUsers(c, conn):
    print("----------=======Extracting All Users=======-------------")
    for row in c.execute('SELECT * FROM users'):
        print(row[1])
        print(row[0])
        print("----------==============================")

def extractContacts(c, conn):
    print("----------=======Extracting Known Contacts=======-------------")
    for row in c.execute('SELECT * FROM users WHERE uid IN (SELECT uid FROM contacts)'):
        print(row[1])
        print(row[0])
        print("----------==============================")
```

Listado D.4: Interpretación de flags de tipo de datos

```
def decodeMsg(c, conn, message):
    usernameUsers = ""
    usernameChats = ""
    flagsActive = []
    fromuid = 0
    tramaMsg = ""
    timestamp = 0.0
    size = 0
    fwd_from_name = ""
    fwd_id_name = ""



    # solo pillamos el 5 ya que esta es el que contiene el datastream de telegram
    # print (row[5])
    tramacursor = 0
    trama = message[tramacursor:tramacursor + 4]
    tramacursor = tramacursor + 4
    header = struct.unpack('<i', trama)[0]
    # Aqui vamos viendo la cabeceras de los tadastream
    if header == 0x44f9b43d:  # type TL_message
        # Aqui la cabecera es de un mensjae y vamos viendo los flags que tiene activo en cada
            uno de estas vamos sacando los datos correspondientes si esta activo
        messagetype = "TL_message"
        trama = message[tramacursor:tramacursor + 4]
        tramacursor = tramacursor + 4
        flags = struct.unpack('<i', trama)[0]

        if (flags & 2) is not 0x0:
            # Este flag indica que es un mensaje saliente nuestro
            flagsActive.append("out")

        # print(hex(flags & 16))
        if (flags & 16) is not 0x0:
            # Este flag indica que hemos sido mencionados en el mensaje
            flagsActive.append("mentioned")

        # print(hex(flags & 32))
        if (flags & 32) is not 0x0:
            flagsActive.append("media Unread")

        # print(hex(flags & 8192))
        if (flags & 8192) is not 0x0:
            flagsActive.append("silent")

        # print(hex(flags & 16384))
        if (flags & 16384) is not 0x0:
            flagsActive.append("post")
```

Listado D.5: Interpretación de flags y busqueda de datos en base de datos

```
#
trama = message[tramacursor:tramacursor + 4]
tramacursor = tramacursor + 4
ids = struct.unpack('<i', trama)[0]

# print(hex(flags & 16))
if (flags & 256) is not 0x0:
    flagsActive.append("fromu_id")
    trama = message[tramacursor:tramacursor + 4]
    tramacursor = tramacursor + 4
    from_id = struct.unpack('<i', trama)[0]
    c.execute('SELECT name FROM users WHERE uid=%s' % from_id)
    usersstream = c.fetchall()
    # conn.commit()
    for rowss in usersstream:
        fwd_id_name = rowss[0]

# print(hex(flags & 16))
if (flags & 4) is not 0x0:
    flagsActive.append("fwd_from")
    trama = message[tramacursor:tramacursor + 4]
    tramacursor = tramacursor + 4
    fwd_from = struct.unpack('<i', trama)[0]
    c.execute('SELECT name FROM users WHERE uid=%s' % fwd_from)
    usersstream = c.fetchall()
    # conn.commit()
    for rowss in usersstream:
        fwd_from_name=rowss[0]

# print(hex(flags & 2048))
if (flags & 2048) is not 0x0:
    flagsActive.append("via_bot_id")
    trama = message[tramacursor:tramacursor + 4]
    tramacursor = tramacursor + 4
    via_bot_id = struct.unpack('<i', trama)[0]
    #print("    Datos de via_bot_id=" + str(via_bot_id))

# print(hex(flags & 8))
if (flags & 8) is not 0x0:
    flagsActive.append("replay_to_msg_id")
    trama = message[tramacursor:tramacursor + 4]
    tramacursor = tramacursor + 4
    reply_to_msg_id = struct.unpack('<i', trama)[0]
    #print("    Datos de reply_to_msg_id: " + str(reply_to_msg_id))

trama = message[tramacursor:tramacursor + 4]
tramacursor = tramacursor + 4
date = struct.unpack('<I', trama)[0]
```

Listado D.6: Interpretación de el mensaje de texto

```
trama = message[tramacursor:tramacursor + 4]
tramacursor = tramacursor + 4
#tramaMsg = binascii.hexlify(trama)
fromuid = struct.unpack('<i', trama)[0]
c.execute('SELECT name FROM chats WHERE uid=%s' % fromuid)
chatsstream = c.fetchall()
for rowss in chatsstream:
    usernameChats = rowss[0]
c.execute('SELECT name FROM users WHERE uid=%s' % fromuid)
chatsstream = c.fetchall()
# conn.commit()
for rowss in chatsstream:
    usernameUsers = rowss[0]
trama = message[tramacursor:tramacursor + 4]
tramacursor = tramacursor + 4
timestamp = struct.unpack('<i', trama)[0]
trama = message[tramacursor:tramacursor + 1]
tramacursor = tramacursor + 1
bytes = struct.unpack('<b', trama)[0]
# bytes = int.from_bytes(trama[0:9], byteorder='little')
size = bytes
trama = message[tramacursor:tramacursor + bytes]
tramacursor = tramacursor + bytes
messagetext = str(trama)

# trama de mensaje que queda por decodificar
trama = message[tramacursor:]
tramaMsg = binascii.hexlify(trama)

# print(hex(flags & 512))
if (flags & 512) is not 0x0:
    # Si es media imprimimos el mesaje hay tambien datos de la ubicacion de el
        alrchivo guardado hay que intentar sacar esto tambien
    flagsActive.append("media")
```

Listado D.7: Interpretación de flags y texto

```
        # print(hex(flags & 64))
        if (flags & 64) is not 0x0:
            print("---reply_markup")
            flagsActive.append("reply_markup")
            trama = message[tramacursor:tramacursor + 4]
            tramacursor = tramacursor + 4
            #reply_markup = struct.unpack('<i', trama)[0]
            #print("    Datos de reply_markup: " + str(reply_markup))

        # print(hex(flags & 128))
        if (flags & 128) is not 0x0:
            flagsActive.append("magic")
            tramacursor = tramacursor + 4

        # print(hex(flags & 1024))
        if (flags & 1024) is not 0x0:
            flagsActive.append("views")
            tramacursor = tramacursor + 4

        # print(hex(flags & 2048))
        if (flags & 32768) is not 0x0:
            flagsActive.append("edit_date")
            tramacursor = tramacursor + 4

        # print(hex(flags & 2048))
        if (flags & 65536) is not 0x0:
            flagsActive.append("post_author")

        # print(hex(flags & 2048))
        if (flags & 131072) is not 0x0:
            flagsActive.append("grouped_id")

        texto = (message.replace("\00", "")).decode('utf-8', 'ignore')
        return([hex(header),messagetype,str(flagsActive),str(ids),str(tramaMsg),size,str(texto
            ).strip(),date,timestamp,fromuid,fwd_from_name,fwd_id_name,usernameUsers,
            usernameChats])
    else:
        messagetype = "Other"
        return([hex(header), messagetype, "", "", "", 0, "", 0.0,0.0,"","","","",""])
```

Listado D.8: Mostrar datos por consola

```
def extractMsg(c, conn):
    c.execute('SELECT * FROM messages')
    messagestream = c.fetchall()
    for row in messagestream:
        data = decodeMsg(c, conn, row[5])
        #print(data)
        print("HeaderHEX: " + str(data[0]))
        print("HeaderType: " + str(data[1]))
        print("Flags: " + str(data[2]))
        print("Ids: " + str(data[3]))
        print("TramaMensaje: " + str(data[4]))
        print("MessageSize: " + str(data[5]))
        print("Message: " + str(data[6]))
        print("Date: " + str(data[7]))
        print("TimeStamp: " + time.strftime('%Y-%m-%d %H:%M:%S', time.localtime(float(data[8])
            )))
        print("Fromuid: " + str(data[9]))
        print("fwd_from_name: " + str(data[10]))
        print("fwd_id_name: " + str(data[11]))
        print("UsernameUsers: " + str(data[12]))
        print("UsernameChats: " + str(data[13]))
        print("----------==========----------")
    print("done")
```

Universidad de Alcalá
Escuela Politécnica Superior



ESCUELA POLITECNICA
SUPERIOR

Universidad
de Alcalá