



universidad
de león

Departamento de Matemáticas

MÁSTER UNIVERSITARIO EN INVESTIGACIÓN EN CIBERSEGURIDAD

Trabajo de Fin de Máster

Extracción y Análisis de Artefactos de Memoria de
la Aplicación Telegram Desktop

Extraction and Analysis of Memory Artifacts from
the Telegram Desktop Application

Autor: Pedro Fernández Álvarez

Tutor Académico: Camino Fernández Llamas

Cotutor: Ricardo Julio Rodríguez Fernández

(Julio, 2021)

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

ALUMNO: Pedro Fernández Álvarez

TUTOR ACADÉMICO: Camino Fernández Llamas

COTUTOR: Ricardo Julio Rodríguez Fernández

TÍTULO: Extracción y Análisis de Artefactos de Memoria de la Aplicación Telegram Desktop

TITLE: Extraction and Analysis of Memory Artifacts from the Telegram Desktop Application

CONVOCATORIA: Julio, 2021

RESUMEN:

Las aplicaciones de mensajería instantánea se han convertido en una manera muy común para comunicarse. Hoy en día, existen multitud de este tipo de aplicaciones, siendo algunas más populares que otras en ciertos países o regiones. A nivel global, Telegram es una de las plataformas de mensajería instantánea más populares, y es el servicio en el cual se centra este proyecto. Concretamente, este trabajo se centra en la aplicación multiplataforma para ordenadores llamada **Telegram Desktop**. En este trabajo se ha elaborado un entorno de análisis forense compuesto por dos herramientas tras estudiar el código fuente de **Telegram Desktop**. Una de las herramientas desarrolladas se encarga de extraer los contenidos de un proceso en un sistema Windows, mientras que la otra está centrada en el estudio de la información presente en memoria RAM relativa al proceso de la aplicación **Telegram Desktop**. El uso conjunto de estas dos herramientas posibilita la obtención de artefactos forenses relacionados con **Telegram Desktop**, como contactos del usuario o contenidos de conversaciones que han tenido lugar, entre otros. La posibilidad de conseguir estos datos resulta de gran ayuda para un analista forense a la hora de resolver un caso, ya que las aplicaciones de mensajería instantánea pueden servir como medio de comunicación para individuos envueltos en algún tipo de acto criminal o para conocer las últimas conversaciones de una posible víctima de un crimen. En ambos casos, el análisis de los contenidos presentes en los equipos de estas personas es vital de cara a esclarecer los hechos ocurridos.

Palabras clave: Análisis forense digital, análisis forense de memoria, mensajería instantánea, **Telegram Desktop**.

Firma del alumno:

VºBº Tutor:

Agradecimientos

Me gustaría agradecer a mis tutores su involucración, sin la cual el presente trabajo no habría sido posible, en especial a Ricardo J. Rodríguez, por toda su ayuda y dedicación a lo largo del proyecto.

Abstract

Instant messaging applications have become a very common way to communicate. Nowadays, there are a multitude of these types of applications, some of which are more popular than others in certain countries or regions. Globally, Telegram is one of the most popular messaging platforms, and it is the service on which this work is focused. Specifically, this work focuses on the cross-platform Telegram application for computers called **Telegram Desktop**. In this work, a forensic analysis environment composed of two tools has been developed after studying the source code of **Telegram Desktop**. One of the tools developed is responsible for extracting the contents of a process on a Windows system, while the other is focused on the study of the information present in RAM memory related to the **Telegram Desktop** application process. The joint use of these two tools makes it possible to obtain forensic artifacts related to **Telegram Desktop**, such as user contacts or content of conversations that have taken place, among others. The ability to obtain these data is of great help to a forensic analyst, since instant messaging applications can serve as a means of communication for individuals involved in some type of criminal act or to know the latest conversations of a possible victim of a crime. In both cases, the analysis of the contents present in the computers of these people is vital in order to clarify the events that occurred.

Índice General

| | |
|--|-------------|
| Agradecimientos | I |
| Abstract | II |
| Índice de Figuras | V |
| Índice de Tablas | VII |
| Glosario de Términos | VIII |
| 1. Introducción | 1 |
| 1.1. Objetivos | 2 |
| 1.2. Alcance | 3 |
| 1.3. Estructura del Trabajo | 5 |
| 2. Conocimientos Previos | 6 |
| 2.1. Telegram | 6 |
| 2.1.1. Aplicación Telegram Desktop | 8 |
| 2.2. Memoria Virtual | 8 |
| 2.3. Mapa de Memoria de un Proceso en Windows | 9 |
| 3. Estado del Arte | 13 |
| 4. Análisis del Código Fuente de Telegram Desktop | 19 |
| 4.1. Diagrama de Clases | 19 |
| 4.2. Diagramas de Secuencia | 23 |
| 5. Entorno de Análisis Desarrollado | 28 |
| 5.1. Herramienta Windows Memory Extractor | 29 |
| 5.2. Herramienta IM Artifact Finder | 31 |
| 5.2.1. Diseño | 32 |

| | |
|--|-----------|
| 5.2.2. Obtención de Artefactos | 35 |
| 6. Experimentos y Evaluación | 39 |
| 6.1. Casos de Prueba | 40 |
| 6.1.1. Categoría <i>Cuentas</i> | 40 |
| 6.1.2. Categoría <i>Conversaciones</i> | 40 |
| 6.1.3. Categoría <i>Usuarios</i> | 42 |
| 6.1.4. Categoría <i>Privacidad</i> | 42 |
| 6.1.5. Categoría <i>Multimedia</i> | 43 |
| 6.1.6. Categoría <i>Bloqueo</i> | 43 |
| 6.1.7. Categoría <i>Sesión</i> | 44 |
| 6.2. Discusión | 44 |
| 6.3. Informes Generados | 46 |
| 7. Conclusiones y Trabajo Futuro | 49 |
| Bibliografía | 51 |
| A. Horas Invertidas | 55 |

Índice de Figuras

| | | |
|------|--|----|
| 1.1. | Plataformas de mensajería instantánea más populares a nivel mundial | 2 |
| 1.2. | Cuota de mercado de los sistemas operativos para ordenador a nivel mundial en el primer semestre de 2020 | 4 |
| 1.3. | Evolución de la cuota de mercado de las versiones de Windows | 4 |
| 2.1. | Localización de páginas en memoria principal y memoria secundaria | 9 |
| 2.2. | Memoria compartida entre varios procesos | 11 |
| 2.3. | Mapa de memoria de un proceso en Windows | 11 |
| 2.4. | Ejemplo de regiones de memoria de una DLL | 12 |
| 4.1. | Diagrama de clases de <code>Telegram Desktop</code> con los elementos de mayor relevancia | 22 |
| 4.2. | Diagrama de secuencia: Ejecución de <code>Telegram Desktop</code> | 25 |
| 4.3. | Diagrama de secuencia: Creación de cuentas | 26 |
| 4.4. | Diagrama de secuencia: Creación de sesiones | 26 |
| 4.5. | Diagrama de secuencia: Creación de historiales y mensajes | 27 |
| 4.6. | Diagrama de secuencia: Creación de conversaciones | 27 |
| 5.1. | Diagrama de alto nivel del entorno de análisis | 29 |
| 5.2. | Ejemplo de salida generada por la utilidad <code>Windows Memory Extractor</code> | 30 |
| 5.3. | Diagrama de clases de la herramienta <code>Windows Memory Extractor</code> | 31 |
| 5.4. | Diagrama de clases de la herramienta <code>IM Artifact Finder</code> | 33 |
| 5.5. | Mensaje enviado a través de <code>Telegram Desktop</code> | 36 |
| 5.6. | Ejemplo de resumen generado por la herramienta <code>IM Artifact Finder</code> | 38 |
| 6.1. | Información obtenida sobre «UsuarioTest TFM» | 46 |
| 6.2. | Información obtenida acerca del usuario correspondiente al bot de GitHub | 46 |
| 6.3. | Conversación individual con «Pedro Fernández» (sin mostrar los mensajes) | 47 |

| | |
|---|----|
| 6.4. Ejemplo de mensaje de texto obtenido | 47 |
| 6.5. Información obtenida sobre un fichero adjunto enviado en un mensaje | 48 |
| 6.6. Información obtenida acerca de una localización geográfica enviada . | 48 |
| A.1. Diagrama de Gantt | 56 |

Índice de Tablas

| | |
|---|----|
| 4.1. Comparativa de las herramientas probadas para realizar ingeniería inversa de código fuente | 21 |
| A.1. Horas invertidas en cada etapa del trabajo | 56 |

Glosario de Términos

Análisis forense digital: Rama del análisis forense que se centra en la investigación y recuperación de información presente en dispositivos digitales.

API: Interfaz en la que se define cómo diferentes sistemas software pueden interactuar entre sí.

Arquitectura cliente-servidor: Estructura de una aplicación software en la cual existen sistemas que proporcionan un servicio, llamados servidores, y sistemas que demandan dicho servicio, llamados clientes.

Artefacto digital: Información obtenida durante un proceso de análisis forense digital que puede ser de valor o no en una investigación forense.

Cifrado de extremo a extremo: Sistema de comunicación en el que únicamente los emisores y los receptores de los mensajes pueden leerlos. Conocido también como cifrado punto a punto.

Evidencia digital: Artefacto digital de utilidad para la resolución de una investigación forense.

Framework: Sistema software que proporciona una funcionalidad genérica y que está diseñado con el objetivo de que pueda ser extendido, de cara a soportar funcionalidades más concretas o específicas.

Git: Sistema de control de versiones de ficheros digitales.

Imagen de disco: Archivo digital que posee todos los contenidos presentes en un disco.

Librería: Funcionalidades implementadas en un determinado lenguaje de programación de las que otros programas pueden hacer uso a través de una interfaz conocida.

Módulo: En un sistema Windows, un módulo representa una imagen (un ejecutable, DLL o controlador) que ha sido cargada por el sistema operativo o por un proceso de usuario.

Patrón de diseño: Solución genérica y reutilizable para resolver un tipo concreto de problema que ocurre con frecuencia en el ámbito del diseño del software.

Proceso: Programa informático en ejecución.

UML: Lenguaje de modelado que proporciona una manera estándar de representar sistemas software.

Unicode: Estándar de codificación de caracteres en un sistema informático. Existen varios sistemas de codificación definidos en el estándar, siendo UTF-8 y UTF-16 dos de ellos.

Capítulo 1

Introducción

Las aplicaciones de mensajería instantánea permiten comunicarse de una manera rápida y cómoda. Hoy en día, una parte notable de la sociedad hace uso de este tipo de aplicaciones para mantener conversaciones. Sin embargo, estas aplicaciones también son usadas en ocasiones como medio para cometer o esclarecer delitos. Es en estos últimos casos cuando el análisis forense de los dispositivos del criminal y de la víctima puede ser de gran ayuda, proporcionando evidencias vitales para la resolución o esclarecimiento del posible crimen acontecido.

Dentro del análisis forense digital existen tres técnicas principales: (i) análisis forense de disco, donde se analiza la información no volátil, la cual sigue presente en el disco (también conocido como memoria secundaria) tras apagar la máquina; (ii) análisis forense de memoria, donde se estudia la información volátil almacenada en la memoria RAM (también llamada memoria principal o memoria física), la cual desaparece al apagar el equipo; y (iii) análisis forense de red, donde se analiza el tráfico de red.

Entre los artefactos digitales presentes en aplicaciones de mensajería instantánea que pueden ser relevantes para la resolución de un caso forense se encuentran, entre otros, mensajes o contactos. Sin embargo, una de las razones que en ocasiones dificulta la obtención de estos artefactos es la existencia de cifrado, tanto en las bases de datos locales de estas aplicaciones como en la información que las aplicaciones reciben y envían a través de la red. Esta ocurrencia provoca que el análisis forense de disco y de red se vean limitados. No obstante, los contenidos presentes en la memoria RAM deben encontrarse descifrados para que la aplicación pueda trabajar con ellos, haciendo que el análisis forense de memoria sea especialmente interesante cuando exista cifrado tanto de los datos almacenados de manera local como de las

comunicaciones. Por este motivo, el presente trabajo se enmarca dentro del análisis forense de memoria.

En la Figura 1.1 se representan las plataformas de mensajería instantánea más populares a nivel global, basándose para ello en el número de usuarios activos de manera mensual. El estudio del que deriva este gráfico ha sido publicado en enero de 2021. En el gráfico se observa que dentro del Top 5 de las plataformas más populares se encuentra Telegram [1], la cual es la plataforma que se analizará en este trabajo.

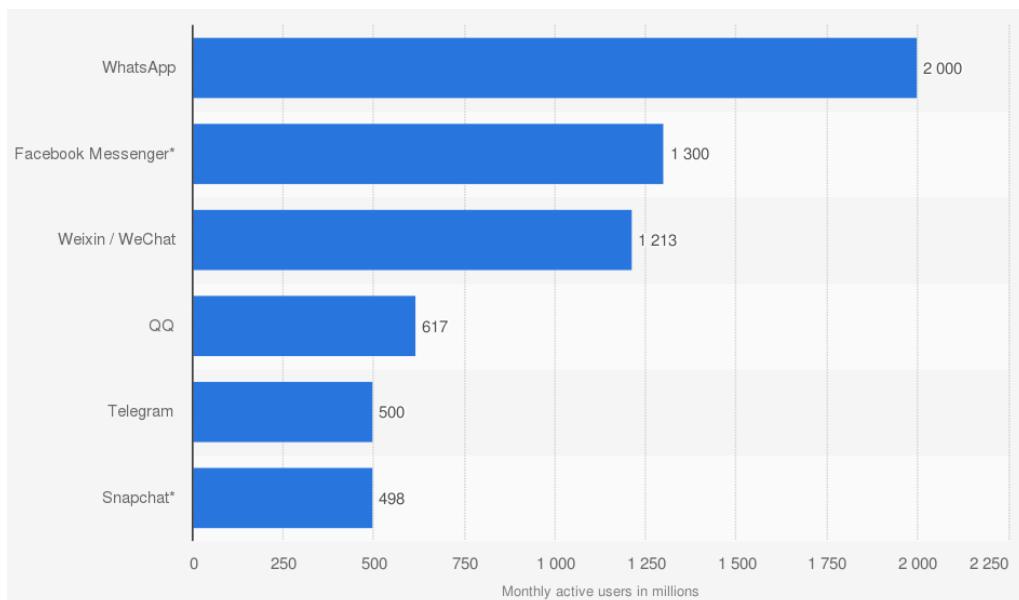


Figura 1.1: Plataformas de mensajería instantánea más populares a nivel mundial
(Fuente: [2])

Una peculiaridad de Telegram respecto a las demás plataformas que aparecen en la Figura 1.1 es que sus aplicaciones oficiales son de código abierto, lo cual facilita su análisis detallado.

1.1. Objetivos

El objetivo general de este trabajo es investigar los contenidos presentes en memoria RAM relativos a la aplicación **Telegram Desktop** de cara a identificar artefactos digitales de interés para una investigación forense.

A continuación se definen los objetivos específicos, los cuales llevarán a la consecución del objetivo general:

1. Analizar el código fuente de la aplicación **Telegram Desktop**.

2. Elaborar una herramienta para extraer los contenidos presentes en la memoria RAM relativos a un proceso determinado que se está ejecutando en un sistema Windows.
3. Desarrollar una herramienta que, recibiendo como entrada los contenidos presentes en memoria RAM pertenecientes a **Telegram Desktop** (extraídos con la herramienta mencionada en el anterior objetivo), se encargue de identificar artefactos de valor para una investigación forense.
4. Evaluar los artefactos digitales que pueden ser obtenidos mediante el uso conjunto de las herramientas mencionadas en los dos previos objetivos.

1.2. Alcance

Es posible hacer uso de Telegram desde diferentes dispositivos y sistemas operativos. No obstante, esta investigación se centra en **Telegram Desktop**, la aplicación oficial de Telegram para ordenadores que se encuentra disponible para varios sistemas operativos. Sin embargo, en este trabajo solamente se trabajará con **Telegram Desktop** para Windows (concretamente, para Windows 10).

La razón por la cual se ha elegido Windows es debido a que es el sistema operativo para ordenadores con más cuota de mercado en la actualidad. En la Figura 1.2 se muestra un gráfico en el que se representa la cuota de mercado a nivel mundial de los sistemas operativos para ordenadores en el primer semestre del año 2020. En ella, se aprecia que el sistema operativo predominante en el sector de los ordenadores es Windows, seguido de macOS y Linux. Adicionalmente, en la Figura 1.3 se observa la evolución en la cuota de mercado de las versiones de Windows desde enero de 2017 hasta marzo de 2021, en la cual se visualiza que la adopción de Windows 10 ha aumentado hasta convertirse en la versión más popular a día de hoy. Concretamente, en marzo de 2021 Windows 10 contaba con un 78,34 % de cuota de mercado frente a otras versiones de Windows.

Por otro lado, este trabajo se centra únicamente en la técnica de análisis forense de memoria. El motivo por el cual se ha decidido centrar el trabajo en el análisis de la información volátil es debido a que, si bien es cierto que **Telegram Desktop** almacena información de manera local (como contactos o mensajes) en una base de datos, dicha base de datos se encuentra cifrada. Adicionalmente, la información que recibe y envía **Telegram Desktop** a través de la red se encuentra también cifrada.

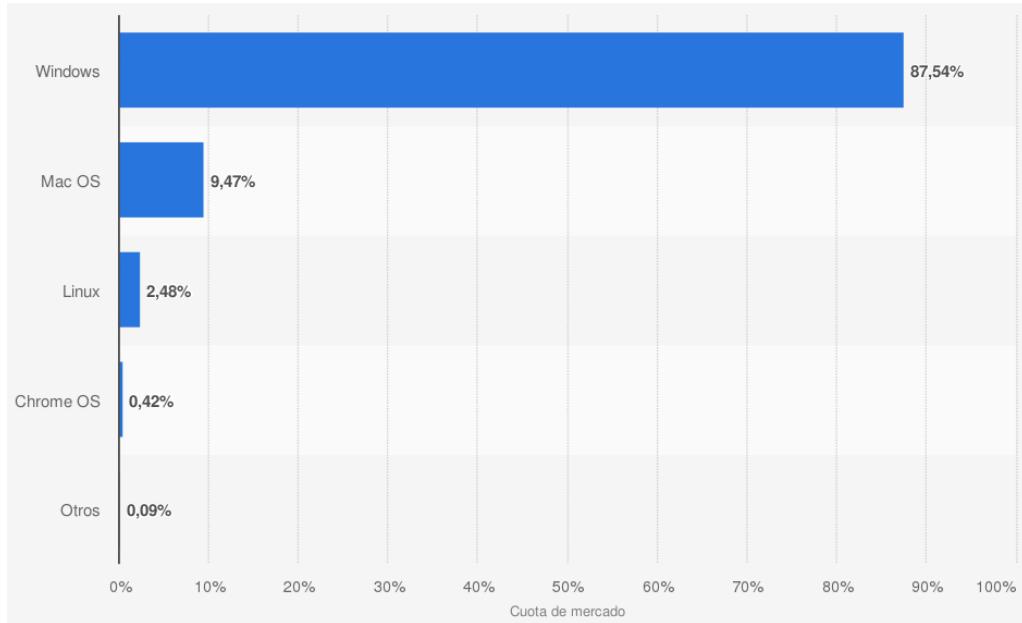


Figura 1.2: Cuota de mercado de los sistemas operativos para ordenador a nivel mundial en el primer semestre de 2020 (Fuente: [3])

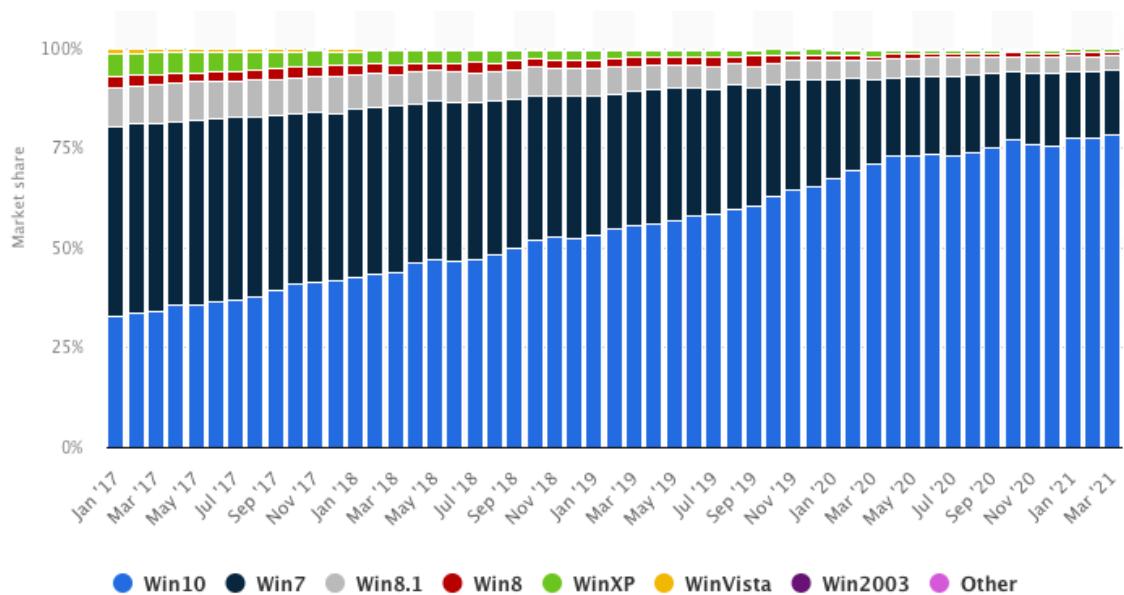


Figura 1.3: Evolución de la cuota de mercado de las versiones de Windows (Fuente: [4])

La aplicación **Telegram Desktop** se actualiza frecuentemente. Como consecuencia, se lanzan nuevas versiones de manera habitual [5]. Con el objetivo de mantener la consistencia a lo largo de la duración del presente trabajo, se trabajará con una versión concreta de **Telegram Desktop**. En particular, se trabajará con la versión 2.7.1, lanzada al mercado el 20 de marzo de 2021.

Por último, las herramientas realizadas en este trabajo se elaboran con fines académicos y de investigación, sin la existencia de fines comerciales ni lucrativos. Ambas herramientas desarrolladas cuentan con la licencia GNU General Public License v3.0 y su código ha sido liberado en GitHub [6], [7].

1.3. Estructura del Trabajo

Tras la presente introducción, se redactan en el capítulo 2 los conocimientos necesarios para comprender correctamente el trabajo elaborado. Posteriormente, en el capítulo 3 se trata el estado actual en lo relativo al análisis forense de plataformas de mensajería instantánea.

En el capítulo 4 se realiza el análisis del código fuente de **Telegram Desktop**. Una vez analizado el código fuente, el desarrollo de las dos herramientas elaboradas se trata en el capítulo 5. Después, en el capítulo 6 se evalúan qué artefactos pueden ser obtenidos mediante el uso de estas herramientas, para posteriormente presentar en el capítulo 7 las conclusiones y el trabajo futuro. Por último, se hace referencia en el anexo A al tiempo invertido para la realización del trabajo.

Capítulo 2

Conocimientos Previos

En este capítulo se presentan los conocimientos requeridos para una adecuada comprensión del trabajo realizado. En primer lugar, se describe la plataforma Telegram. Posteriormente, se define el concepto de memoria virtual, y por último, se explican los contenidos existentes en el mapa de memoria de un proceso que se esté ejecutando en un sistema Windows.

2.1. Telegram

Telegram [8] es un servicio de mensajería instantánea multiplataforma, cuya arquitectura es cliente-servidor. En la actualidad, existen clientes oficiales de Telegram para Android, iOS, Windows, macOS y Linux. Adicionalmente, Telegram también puede ser usado desde un navegador web.

Telegram posee una API pública, a través de la cual es posible acceder a las funcionalidades a las que los clientes o aplicaciones oficiales tienen acceso, lo cual permite que, además de los clientes oficiales, existan también clientes no oficiales desarrollados por terceros. Los clientes oficiales de Telegram son de código abierto, a diferencia del código que se ejecuta en los servidores de Telegram.

De cara a poder usar Telegram, se debe crear una cuenta, para lo cual es necesario proporcionar un número de teléfono. Telegram permite el uso simultáneo de 3 cuentas, y solamente es posible crear una cuenta con cada número de teléfono. Una cuenta de Telegram puede tener asociado un nombre de usuario público, el cual puede ser elegido por el propietario de la cuenta y permite que los usuarios de la plataforma encuentren a otros usuarios realizando una búsqueda por nombre de usuario.

En Telegram, además de conversaciones textuales, pueden realizarse llamadas de voz y videollamadas cifradas de extremo a extremo. En lo relativo a las conversaciones de texto, existen 3 tipos:

- **Conversación individual:** Conversación en la que solamente dos usuarios están involucrados. En Telegram existen dos tipos de conversaciones individuales: las conversaciones normales (conocidas en inglés como *regular chats* o *cloud chats*) y las conversaciones secretas (del inglés *secret chats*). De manera adicional, Telegram permite conversar individualmente con un tipo especial de usuarios, conocidos como bots, los cuales en lugar de ser seres humanos son programas informáticos.
- **Grupo:** Un grupo es una conversación en la que todos los usuarios involucrados pueden enviar mensajes y leer los mensajes que los demás integrantes del grupo han enviado. Existen dos tipos de grupos en Telegram: los grupos normales, los cuales siempre son privados y tienen como máximo 200 integrantes; y los supergrupos (del inglés *supergroup*), los cuales pueden ser tanto privados como públicos y pueden contar con hasta 200.000 integrantes. Adicionalmente, un grupo puede convertirse en un supergrupo.
- **Canal:** Un canal es un tipo especial de grupo en el cual únicamente unos usuarios determinados (conocidos como administradores o publicadores) pueden enviar mensajes, mientras que los demás usuarios del canal (llamados suscriptores) solamente pueden leer los mensajes enviados por los administradores. Un canal puede ser privado o público y puede tener un número ilimitado de suscriptores.

El hecho de que un grupo o canal sea público significa que cualquier usuario de Telegram puede unirse a dicha conversación. En cambio, si la conversación es privada es posible controlar quién puede participar en la conversación.

Los mensajes enviados en conversaciones normales, grupos y canales viajan desde el emisor hasta los servidores de Telegram de manera cifrada, donde se descifran y se envían posteriormente al receptor o receptores, también de manera cifrada [9]. Estos mensajes se almacenan de manera cifrada en los servidores de Telegram. No obstante, Telegram puede descifrarlos, lo cual permite que este tipo de conversaciones puedan ser sincronizadas entre varios dispositivos. Por contra, las conversaciones secretas cuentan con cifrado de extremo a extremo, lo cual significa que Telegram teóricamente no puede descifrar sus contenidos y que estas conversaciones no puedan

ser sincronizadas entre diferentes dispositivos, con lo que estos contenidos solamente están disponibles en el dispositivo emisor y en el dispositivo receptor de los mensajes.

Por otro lado, es posible bloquear con contraseña las aplicaciones de Telegram para que el usuario deba introducir la contraseña que haya configurado previamente al acceder a la aplicación. Esto permite que una persona no autorizada que se haga físicamente con un dispositivo en el cual su propietario cuente con la aplicación de Telegram instalada no pueda acceder a la aplicación si no conoce la contraseña de desbloqueo, añadiendo así un nivel de seguridad adicional.

2.1.1. Aplicación Telegram Desktop

Telegram Desktop [10] es el cliente oficial de Telegram para ordenadores con sistema operativo Windows, macOS o Linux. Una particularidad de Telegram Desktop es el hecho de que actualmente no soporta conversaciones secretas.

En el caso concreto de macOS, existen dos clientes oficiales, uno de ellos es Telegram Desktop, denominado también en este caso Telegram Lite, y el otro es el llamado Telegram para macOS [11], el cual, a diferencia de Telegram Lite, soporta conversaciones secretas y posee integración con características concretas de macOS, como la barra táctil con la que cuentan las versiones de MacBook Pro recientes.

2.2. Memoria Virtual

En los sistemas operativos con soporte para memoria virtual, como Windows, cada proceso tiene su propio espacio privado de direcciones virtuales [12]. Este espacio de memoria es lineal (es decir, las direcciones virtuales son contiguas) y está dividido en secciones del mismo tamaño, llamadas páginas.

No todas las páginas de un proceso están situadas en memoria física. Cuando no existe suficiente espacio en memoria principal para alojar todas las páginas usadas por todos los procesos, algunas de ellas se almacenan en disco (memoria secundaria). De esta manera, cuando los contenidos presentes en las páginas almacenadas en memoria secundaria vuelvan a ser solicitados, dichas páginas pueden volver a cargarse en memoria principal. Este mecanismo de gestión de memoria proporciona una abstracción entre la memoria virtual que el proceso ve y la cantidad real de memoria física disponible en el sistema, ya que el proceso ve un gran espacio de direcciones virtuales privado, sin importar la cantidad de memoria física existente.

Las aplicaciones trabajan con direcciones virtuales, aunque la información se encuentra almacenada en memoria física. Por lo tanto, cuando un proceso solicita un dato almacenado en una dirección virtual, es necesario realizar una traducción de dicha dirección virtual a una dirección física. Esta correspondencia de direcciones virtuales con direcciones físicas es tarea del sistema operativo, el cual se encarga de saber qué páginas está usando cada proceso y de traducir direcciones virtuales a direcciones físicas. Como consecuencia, la aplicación puede operar solamente con direcciones virtuales, sin la necesidad de conocer en qué dirección física está situado el dato solicitado, ya que el sistema operativo se encargará de traducir la dirección virtual a la dirección física correcta.

En la Figura 2.1 se muestran los espacios de direcciones virtuales pertenecientes a 2 procesos, donde se aprecia que las páginas usadas por estos procesos se encuentran o bien en memoria principal o bien en memoria secundaria. De esta manera, varios procesos pueden trabajar internamente con las mismas direcciones virtuales sin preocuparse de que otros procesos también estén trabajando con ellas, ya que el sistema operativo se encargará de traducir las mismas direcciones virtuales de distintos procesos a diferentes direcciones físicas.

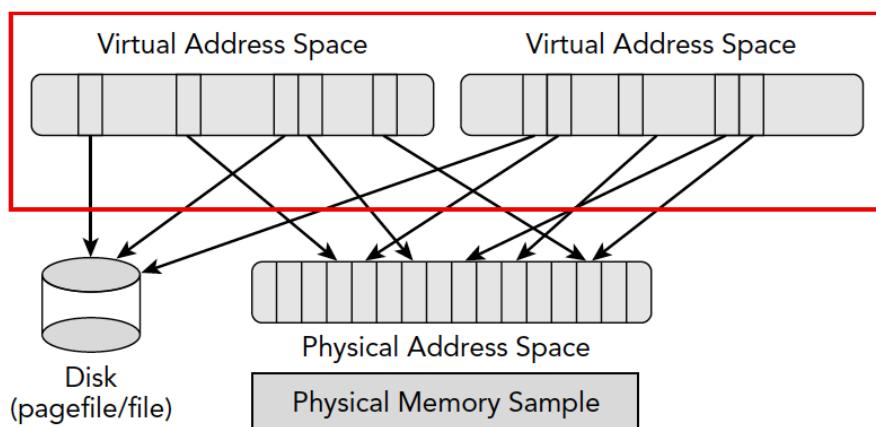


Figura 2.1: Localización de páginas en memoria principal y memoria secundaria

(Fuente: [12], página 21)

2.3. Mapa de Memoria de un Proceso en Windows

Cuando se ejecuta un fichero ejecutable, se crea un proceso. El mapa de memoria de un proceso es la distribución de los contenidos presentes en su espacio privado de direcciones virtuales. En un sistema Windows, las secciones más importantes del mapa de memoria de un proceso son las siguientes:

- **Pila:** En la sección de pila se almacenan de manera temporal los datos relativos a la ejecución de funciones, como entre otros, las variables locales de las funciones. La pila crece y disminuye de tamaño a medida que comienza y finaliza la ejecución de diferentes funciones.
- **Heap:** En esta sección, conocida también como memoria dinámica, se almacenan datos cuyo tiempo de vida puede ser superior al tiempo de vida de una función. Los datos que se almacenan en el heap son controlados por el programador, recayendo en él la responsabilidad de liberar la memoria ocupada por dichos datos de manera adecuada.
- **Imagen de la aplicación:** En esta sección se encuentra el código binario de la aplicación. Adicionalmente, en esta sección también se encuentran datos almacenados, tanto de solo lectura como de lectura y escritura.
- **Librerías dinámicas:** Conocidas en Windows como DLLs (*Dynamic-Link Libraries*), una DLL es un módulo que contiene tanto código como datos. Dicho módulo puede ser usado tanto por una aplicación como por otra DLL [13]. Estos módulos actúan como librerías compartidas entre varios procesos, con el objetivo de que diferentes procesos puedan compartir el código presente en cada DLL. De esta manera, el código de una DLL solamente permanece almacenado en una localización en memoria física, estando dicha localización mapeada en los mapas de memoria de los procesos que usen dicha DLL. A esto se le conoce como memoria compartida, definida como memoria que puede ser accedida desde más de un espacio de direcciones virtuales [12]. En la Figura 2.2 se muestra gráficamente un ejemplo de partición de memoria, donde se aprecia que el proceso A y el proceso B poseen páginas en sus respectivos espacios de direcciones virtuales que se corresponden con las mismas direcciones físicas.

En la Figura 2.3 se representa el mapa de memoria de un proceso en Windows (de manera esquemática y no a escala). Por otro lado, solamente se muestran 2 DLLs (*user32* y *kernel32*), debido a razones de simplicidad. Sin embargo, lo común es que exista un mayor número de DLLs en el mapa de memoria de un proceso.

Cada una de las secciones descritas previamente se divide a su vez en regiones de memoria. Una región de memoria está formada por un conjunto de páginas, siendo el tamaño de cada región dependiente del número de páginas por las que está compuesta. Por otro lado, cada región de memoria cuenta con unos permisos concretos. A modo de ejemplo, en la Figura 2.4 se aprecian las regiones pertenecientes

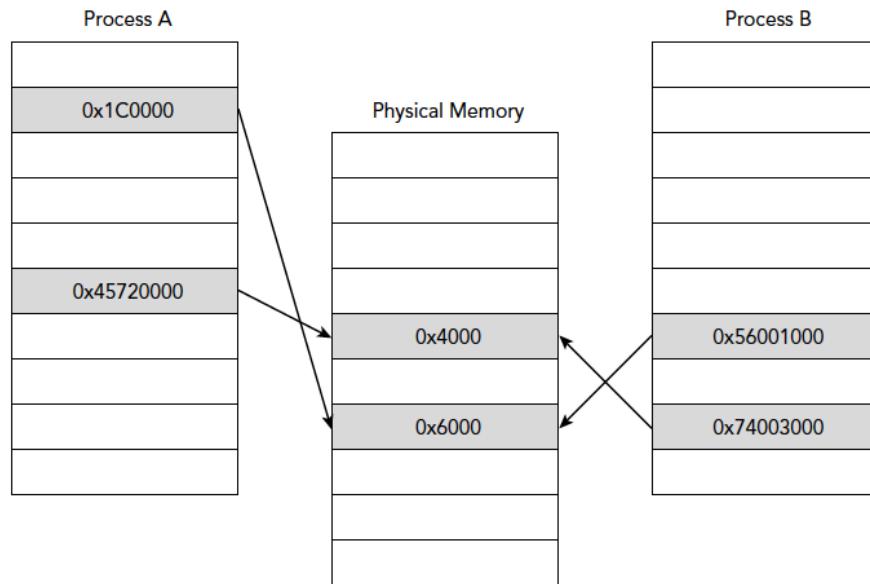


Figura 2.2: Memoria compartida entre varios procesos (Fuente: [12], página 22)

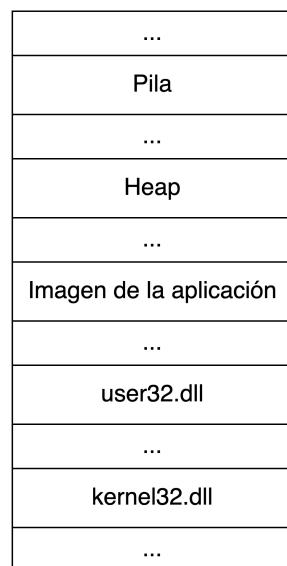


Figura 2.3: Mapa de memoria de un proceso en Windows

a la DLL *user32*, donde se observan 5 regiones de memoria, cada una con unos permisos determinados. De cara a hacer mención a algunos de ellos, se observa que la segunda región (la cual comienza en la dirección 0x7ffd6d141000) posee permisos de lectura (R) y ejecución (X), mientras que la tercera solamente cuenta con permisos de lectura.

| Base address | Type | Size | Protection | Use |
|----------------|---------------|----------|------------|--------------------------------|
| 0x7ffd6d140000 | Image | 1.664 kB | WCX | C:\Windows\System32\user32.dll |
| 0x7ffd6d140000 | Image: Commit | 4 kB | R | C:\Windows\System32\user32.dll |
| 0x7ffd6d141000 | Image: Commit | 576 kB | RX | C:\Windows\System32\user32.dll |
| 0x7ffd6d1d1000 | Image: Commit | 132 kB | R | C:\Windows\System32\user32.dll |
| 0x7ffd6d1f2000 | Image: Commit | 8 kB | RW | C:\Windows\System32\user32.dll |
| 0x7ffd6d1f4000 | Image: Commit | 944 kB | R | C:\Windows\System32\user32.dll |

Figura 2.4: Ejemplo de regiones de memoria de una DLL

Capítulo 3

Estado del Arte

El presente capítulo contiene el estado actual relativo al análisis forense de plataformas de mensajería instantánea. Para su redacción, se ha decidido seguir un orden principalmente cronológico de cara a reflejar la manera en que ha avanzado la investigación en este campo durante los últimos años.

En primer lugar, en [14] se analizan y comparan los artefactos generados en *smartphones* Android relativos a conversaciones normales y a conversaciones privadas realizadas a través de las plataformas Telegram, KakaoTalk y Line. En este caso, se analizan los datos presentes en el almacenamiento interno (o memoria interna) de estos dispositivos, es decir, en la memoria no volátil. Adicionalmente, también se realiza análisis forense de red, estudiando para ello el tráfico generado por estas aplicaciones. En cuanto al análisis de tráfico, se descubrió que las 3 aplicaciones se comunicaban con sus respectivos servidores de manera cifrada, sin importar la modalidad de conversación. Sin embargo, estas aplicaciones presentaban diferencias y algunas deficiencias en lo relativo al almacenamiento interno de los mensajes. En el caso de Telegram y Line, los mensajes de los dos tipos de diálogos estudiados se almacenaban de manera no cifrada en sus respectivas bases de datos. Por contra, KakaoTalk guardaba los mensajes de ambos tipos de conversaciones de manera cifrada, salvo el último mensaje, el cual se almacenaba en texto plano.

En [15] se investiga la aplicación de Telegram en *smartphones* Android, centrándose en el análisis de los artefactos presentes en la memoria interna, donde se ha identificado la presencia de la base de datos relativa a la aplicación sin cifrar. En dicha base de datos se ha encontrado información acerca del usuario de la aplicación, de sus contactos, y rutas de archivos enviados y recibidos. Adicionalmente, también se ha identificado información sobre las conversaciones mantenidas, siendo posible

la reconstrucción de las mismas, tanto de las conversaciones individuales, como de los grupos y los canales.

Una investigación similar a la anterior (pero proporcionando numerosos detalles adicionales) es [16]. Un método importante mencionado en ella y no explícitamente en [14] ni en [15] es el análisis de código fuente. En este estudio, además de haber identificado la información extraída de Telegram en [14] y [15], se ha descubierto también información acerca de las llamadas de voz, fotos de perfil del usuario y de sus contactos, y varios datos adicionales, como el número de participantes en un grupo, o si un canal o grupo es privado o público.

En [17] se sigue la misma línea que en los dos artículos previamente mencionados. Sin embargo, en lugar de realizar el análisis sobre el sistema operativo Android, se realiza sobre Windows Phone, el cual está discontinuado a día de hoy. En este caso, se han encontrado mensajes recibidos y enviados relativos a los diferentes tipos de conversaciones, así como información del propio usuario y de sus contactos. También se han podido reconstruir las conversaciones, centrándose en este caso en los detalles de las conversaciones uno a uno, y no de grupos o canales. Al igual que en [16], se ha analizado el código fuente de Telegram para descubrir la manera en la que se almacenaba la información, la cual no se encontraba cifrada, e interpretarla así adecuadamente.

Otro ejemplo en el que el sistema operativo analizado se encuentra actualmente discontinuado es [18], en el cual se analiza el sistema operativo Firefox OS en *smartphones*. Las plataformas analizadas en este caso son Telegram, OpenWapp y Line. Como mención adicional, la plataforma OpenWapp se encuentra discontinuada también hoy en día. En cuanto a técnicas forenses, este es el primer artículo encontrado en el que además de analizar la memoria no volátil se analiza también la memoria volátil (memoria RAM). A la hora de analizar el almacenamiento interno en el caso de OpenWapp solamente se pudo obtener en este estudio el número de teléfono del usuario. En el resto de aplicaciones no se pudo obtener este dato ni tampoco las conversaciones de la memoria interna. En cambio, se encontró el número de teléfono del usuario para las 3 aplicaciones al analizar los contenidos de la memoria RAM. Por otra parte, únicamente se han podido identificar mensajes en la memoria volátil relativos a la aplicación OpenWapp.

Tanto en [19] como en [20] se analiza la aplicación WeChat para *smartphones* Android, centrándose en los datos presentes en el almacenamiento interno del dispositivo. La base de datos en la que esta aplicación guarda las conversaciones y

contactos se encontraba cifrada. No obstante, en estas dos investigaciones se estudió el algoritmo de cifrado de dicha base de datos, consiguiendo descifrarla en ambas.

En cuanto a tabletas (concretamente, el modelo iPad) se estudia en [21] su almacenamiento interno de cara a obtener los artefactos relativos a la aplicación Kik en iOS, los cuales estaban presentes en una base de datos no cifrada. Por su parte, las fotos de perfil del usuario y de sus contactos, así como los archivos enviados y recibidos a través de chats tampoco se encontraban cifrados. En la base de datos hallada se han encontrado los datos de los contactos del usuario. Por otro lado, se han conseguido recuperar los contenidos de los mensajes, y tras ello reconstruir las conversaciones, tanto individuales como grupales. En lo relativo a las conversaciones grupales, se han conseguido identificar varios datos adicionales, como quién era el administrador o los usuarios que tenían la entrada prohibida al grupo.

A diferencia de las anteriores mencionadas, una investigación que pone el foco completamente en el análisis de la memoria volátil es [22], donde se analizan las plataformas Messenger, WhatsApp y Viber en *smartphones* Android. Como resultados de este estudio, se han encontrado contenidos de conversaciones en la memoria RAM, no solamente relativos a diálogos recientes, sino también de conversaciones llevadas a cabo incluso 16 meses atrás. En las 3 aplicaciones analizadas se encontraron mensajes en la memoria RAM en texto plano.

El almacenamiento interno en *smartphones* Android vuelve a ser el objetivo en [23], pero este caso centrado en las aplicaciones Messenger y Hangouts. En ambas aplicaciones se han podido reconstruir las conversaciones (salvo en el caso de las conversaciones secretas de Messenger), ya que los datos se almacenaban de manera no cifrada en la memoria interna del dispositivo. También se han conseguido localizar los archivos enviados y recibidos a través de ambas plataformas. Además, ambas aplicaciones almacenaban la localización geográfica desde la cual cada mensaje fue enviado, en forma de coordenadas geográficas.

En [24] el foco es el mismo que en [23], aunque se estudian de manera adicional las plataformas WhatsApp y Line. En las 4 aplicaciones se han podido encontrar datos acerca de conversaciones y de los contactos, almacenados en sus respectivas bases de datos, todas ellas sin cifrar. Adicionalmente, se hace referencia a que existe otra base de datos relativa a WhatsApp, que es en realidad una copia de seguridad de la original que sí se encuentra cifrada. No obstante, este estudio no se ha centrado en el análisis de dicha base de datos cifrada.

En [25] se analiza la plataforma Telegram, concretamente, la aplicación **Telegram Desktop** para ordenadores macOS. El estudio hecho en este caso se centra en la información localizada en el disco duro del dispositivo. En primer lugar, se ha logrado conocer el lugar donde se localizan los artefactos de Telegram tras consultar fuentes públicas. Sin embargo, la base de datos donde se almacenan las conversaciones de forma local se encontraba cifrada y en este caso no se ha podido descifrar. No obstante, los autores de este estudio han podido recuperar de otro modo los mensajes. Para ello, la acción que han hecho es copiar la aplicación de Telegram y los datos con los cuales trabaja (cifrados) a un entorno forense desde una imagen de disco forense y ejecutar la aplicación desde dicho entorno. Al realizarlo, la aplicación ha descifrado los mensajes y los investigadores han podido verlos en la interfaz de la propia aplicación. Tras ello, a la hora de proporcionar conexión a Internet a la aplicación, los mensajes nuevos almacenados en la nube también se han recuperado. Sin embargo, una limitación de este estudio es que no se detalla en ningún momento la configuración de seguridad de la aplicación bajo la cual se han conseguido recuperar los mensajes.

En relación con el anterior artículo, se encuentra [26], donde se analizan las plataformas WeChat, Telegram, WhatsApp y Viber en *smartphones* Android. Este estudio se centra en el almacenamiento interno, mencionando que las bases de datos locales relativas a Telegram, Viber y WeChat se encuentran cifradas y la copia de seguridad de la base de datos original de WhatsApp también se encuentra cifrada, como se había mencionado en [24]. Hasta ahora, se han mencionado investigaciones donde se había descubierto que Telegram no poseía su base de datos cifrada. Sin embargo, en este estudio se analiza una versión más reciente que en las otras investigaciones, descubriendo que esta vez sí se encuentra cifrada. En este estudio se hace referencia a la técnica para descifrar la base de datos de WeChat presentada en [20], mencionando que ese proceso ya no es reproducible. Sin embargo, en este estudio también se ha conseguido descifrar la base de datos con un nuevo método. En el caso de Telegram, los autores de este estudio no han podido descifrar la base de datos, a diferencia de la base de datos que actuaba como copia de seguridad de WhatsApp. En cuanto a Viber, aunque se creía que la base de datos estaba completamente cifrada, se han conseguido ver en ella mensajes sin cifrar.

En [27] se analizan los datos que residen en la memoria volátil para recuperar datos de conversaciones cuando las bases de datos están cifradas. En esta investigación se han analizado, tanto en ordenadores con sistema operativo macOS y Windows, las aplicaciones Skype, WhatsApp, iMessage (en este caso solo en macOS), Viber y

Messenger. Para todas estas plataformas y ambos sistemas operativos se han conseguido recuperar mensajes intercambiados de la memoria RAM. De manera adicional, de Skype también se ha conseguido recuperar el nombre de usuario, su contraseña, y fechas de intercambio de mensajes. En WhatsApp se ha identificado el número de teléfono del usuario. En lo relativo a Viber, se han recuperado números de secuencia, los cuales permiten ordenar los mensajes. A su vez, en Messenger se han encontrado también fechas de envío de mensajes, junto al identificador del emisor y del receptor. Por último, en el caso de iMessage, se han hallado las direcciones de correo electrónico de los usuarios que forman parte de las conversaciones.

Siguiendo la línea de [26], en [28] se analizan los servicios KakaoTalk, NateOn y QQ en ordenadores Windows, donde estas plataformas almacenan los mensajes en bases de datos cifradas. En este estudio se mantiene el foco en el análisis del disco, de cara intentar descifrar estas bases de datos y comprobar cómo de seguras son. Para ello, se han estudiado los algoritmos de cifrado y descifrado que emplean estas aplicaciones sobre sus bases de datos mediante procesos de ingeniería inversa. En los casos de KakaoTalk y NateOn se han conseguido descifrar sus bases de datos, lo cual no ha sido posible en el caso de QQ.

En [29] se analizan, en primer lugar, los artefactos de diferentes plataformas presentes en la memoria volátil de ordenadores con sistema operativo Windows, macOS y Ubuntu. Sin embargo, una peculiaridad con respecto a anteriores estudios incluidos es que en este no se analizan las aplicaciones de escritorio, sino la versión web de las propias plataformas. Concretamente, se analizan los servicios Trillian, Messenger, Hangouts, Skype, WhatsApp y Telegram, cada uno de ellos en los siguientes navegadores web: Google Chrome, Mozilla Firefox, Opera, Microsoft Edge y Safari. Como resultados, se han identificado datos relativos a comunicaciones para todos los sistemas operativos y navegadores en los casos de Messenger y Skype (en este caso menos en Mozilla Firefox en Windows). Sin embargo, en el resto de plataformas no se ha encontrado esta información en ningún navegador ni sistema operativo. Además, en este estudio se analiza también la memoria RAM en *smartphones* Android, focalizándose en este caso en Viber, Signal y las mismas aplicaciones mencionadas anteriormente excepto Skype. En este caso, se han hallado datos acerca de diálogos en Messenger, WhatsApp, Viber y Hangouts, pero no en las 3 restantes analizadas.

Continuando con el estudio de la memoria volátil, en [30] se proporciona un análisis de la plataforma Hangouts en ordenadores Windows. Los logros alcanzados en este caso han sido el hecho de encontrar en la memoria RAM mensajes, correos

electrónicos tanto de emisores como de receptores, y sus nombres. Sin embargo, las fechas en la que los mensajes fueron enviados no se han podido identificar.

En cuanto al artículo más reciente analizado, en [31] se investiga la aplicación Kik para *smartphones* Android, estudiando tanto la memoria volátil como la no volátil. Mediante el empleo de ambas técnicas se han conseguido encontrar datos pertenecientes a conversaciones y a contactos.

Para concluir, cabe mencionar que, tras realizar la revisión de la literatura actual incluida en este capítulo, no se han encontrado investigaciones relativas a los contenidos presentes en memoria RAM que genera la aplicación Telegram Desktop, temática en la que se centra el presente trabajo.

Capítulo 4

Análisis del Código Fuente de Telegram Desktop

En el actual capítulo se realiza el análisis del código fuente de **Telegram Desktop** [32], en concreto, de la versión 2.7.1 [33]. El hecho de que el código fuente de **Telegram Desktop** se distribuya de manera libre permite que pueda ser estudiado, de cara tanto a entender mejor el comportamiento del programa como a comprender la manera en la cual se gestionan y almacenan los datos con los que trabaja la aplicación, información imprescindible para posteriormente conocer cómo obtenerlos e interpretarlos.

El lenguaje de programación utilizado para el desarrollo de **Telegram Desktop** es C++. Adicionalmente, **Telegram Desktop** hace uso de **Qt** [34], un *framework* escrito también en C++ que permite el desarrollo de aplicaciones multiplataforma para ordenadores, dispositivos móviles y sistemas embebidos. **Qt** puede ser empleado bajo una licencia gratuita o bajo una comercial. Mediante el uso de **Qt**, el código de **Telegram Desktop** no se centra en los detalles de interfaz de cada sistema operativo, sino que delega dicha responsabilidad en **Qt**.

4.1. Diagrama de Clases

En primera instancia, se llevó a cabo un proceso de análisis manual mediante la búsqueda de palabras clave en el código fuente, como *Message* o *User*, de cara a identificar clases que representasen elementos de valor para un caso forense, como mensajes o usuarios, respectivamente. Mediante esta técnica se consiguieron identificar y relacionar una serie de clases relevantes. No obstante, debido a que **Telegram**

Desktop es una aplicación de notable envergadura, su análisis manual implica la necesidad de invertir una gran cantidad de tiempo. Por ello, tras identificar una serie de clases de aparente interés, se decidió optar por realizar un análisis automático del código fuente.

De cara a realizar el análisis automático, primero se ha hecho una búsqueda y comparativa de herramientas UML que soporten la ingeniería inversa de código fuente C++. En este caso, la ingeniería inversa de código fuente se refiere al hecho de generar uno o varios diagramas UML a partir de código fuente. Finalmente, se han seleccionado 3 herramientas que ofrecen esta posibilidad: **Visual Paradigm** [35], **StarUML** [36] y **BOUML** [37]. En la Tabla 4.1 se muestra la comparativa de las 3 herramientas mencionadas anteriormente.

Tras trabajar con las herramientas seleccionadas, los mejores resultados en cuanto a la obtención de un diagrama de clases total de la aplicación se han conseguido con **Visual Paradigm**. En el caso de **StarUML**, los resultados no han sido buenos, ya que presentaba muchos fallos a la hora de identificar clases. En cuanto a **BOUML**, la identificación de clases era correcta, pero el hecho de que el diagrama deba generarse de manera manual supone un inconveniente cuando el número de clases es alto, como ocurre en el caso de **Telegram Desktop**. Adicionalmente, a la hora de analizar todo el código con **BOUML**, ocurrían errores y se tenía que ir analizando por partes.

El diagrama de clases global generado automáticamente con **Visual Paradigm** no se incluye en el presente documento debido a que es demasiado grande. Por ello, se ha decidido insertar dentro de la carpeta compartida [38] entregada como parte del trabajo. Para hacerse una idea del volumen de dicho diagrama (archivo llamado *DiagramaClasesTelegramDesktop.pdf*), de cara a ver los contenidos de cada clase de manera nítida debe emplearse la mayor cantidad de ampliación que la herramienta **Adobe Acrobat Reader DC** [39] permite (6400 %).

El hecho de obtener el diagrama de clases global ha permitido la identificación de nuevas clases de interés, lo cual ha ayudado a ampliar el diagrama de clases realizado previamente de manera manual. Como consecuencia, se ha elaborado un diagrama de clases parcial en el cual solamente se incluyen las clases que se consideran de mayor relevancia, representando algunos de sus métodos y atributos de mayor interés (es decir, para cada clase no se incluyen todos sus métodos ni atributos, de cara a simplificar el diagrama y reflejar en él lo que se considera más importante). Dicho diagrama se muestra en la Figura 4.1.

Tabla 4.1: Comparativa de las herramientas probadas para realizar ingeniería inversa de código fuente

| | Visual Paradigm | StarUML | BOUML |
|--|---|--|---|
| Licencia | Comercial y Community Edition (versión gratuita) | Comercial | Gratuita |
| Posibilidad de evaluación gratuita | Sí | Sí | — |
| Sistemas operativos | Windows, Linux, macOS | Windows, Linux, macOS | Windows, Linux, macOS |
| Open source Ingeniería inversa de código fuente | No Lenguajes Java, C++, Ada, PHP, Python y Objective-C soportados de manera nativa (solamente en la versión comercial) | No Lenguajes C++, C# y Java soportados a través de extensiones <i>open source</i> | No Lenguajes C++, Java y PHP soportados de manera nativa |
| Generación automática de diagramas | Sí | Sí | No, la creación de diagramas debe ser manual |

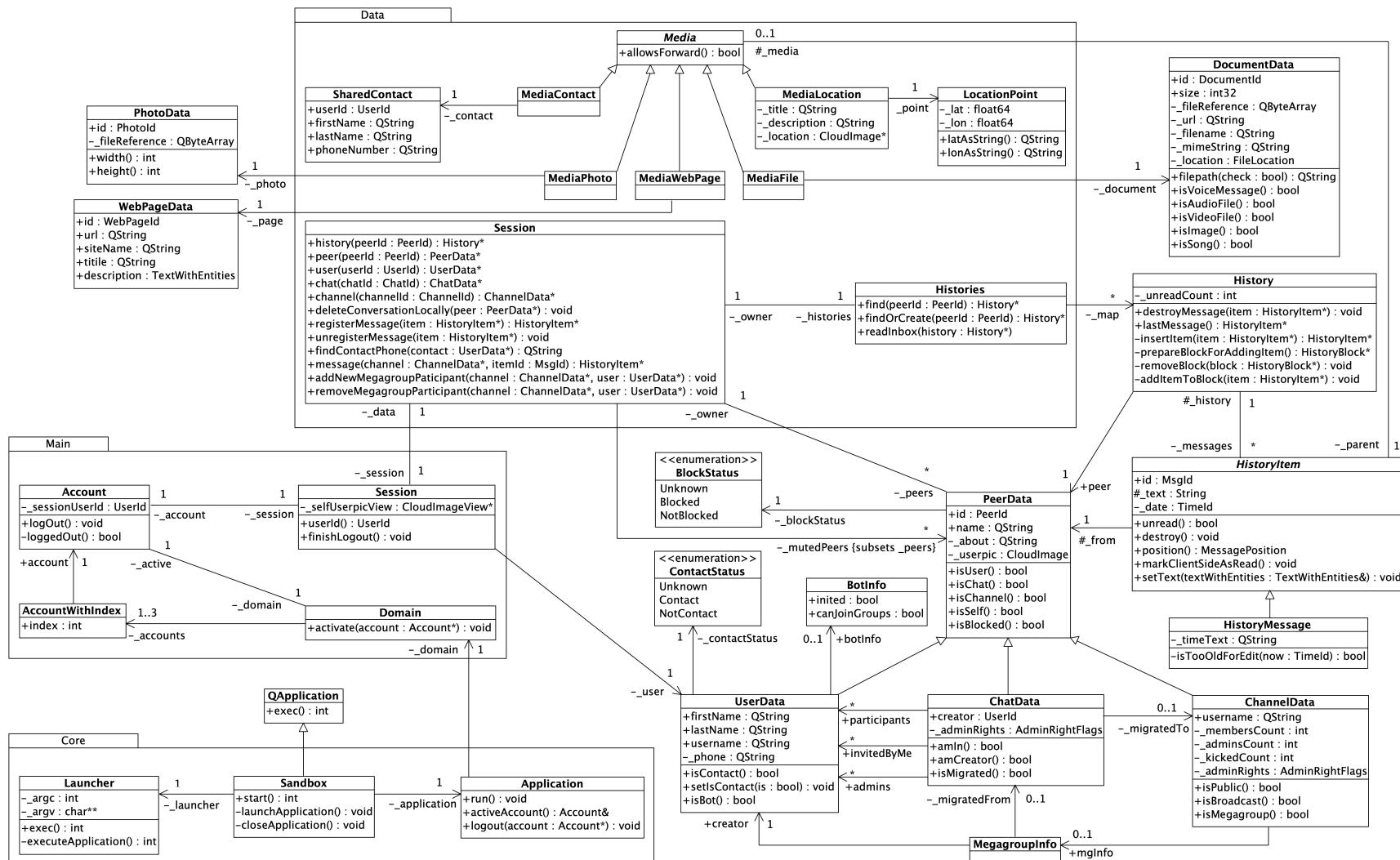


Figura 4.1: Diagrama de clases de Telegram Desktop con los elementos de mayor relevancia

La clase *PeerData* representa una conversación. La subclase *UserData* representa una conversación individual, mientras que la subclase *ChatData* se corresponde con un grupo normal. Por último, *ChannelData* representa tanto un canal como un supergrupo.

Por otro lado, existe la clase *HistoryMessage*, que modela un mensaje tanto enviado como recibido, pudiendo tener un adjunto asociado o ser un mensaje multimedia, representado mediante la propiedad *_media*. Los posibles elementos multimedia o adjuntos que pueden ser transferidos a través de **Telegram Desktop** se modelan como subclases de *Media* y pueden ser, entre otros, coordenadas geográficas, contactos compartidos, o archivos, representándose en el diagrama exclusivamente los de mayor interés.

La clase *History* representa el historial de una conversación concreta y permite relacionar un conjunto de mensajes con una conversación. Por otra parte, se aprecia en la clase *Domain* el hecho de que puede haber un máximo de hasta 3 cuentas, pero únicamente se puede trabajar con una a la vez (atributo *_active*).

En el diagrama se aprecia que cada cuenta (clase *Account*) posee una sesión, representada por la clase *Session* ubicada dentro del espacio de nombres (o *namespace*) llamado *Main*. Esta clase se denota por tanto como *Main::Session*, para diferenciarla de la clase *Data::Session*, situada en el *namespace Data*. En la clase *Main::Session* existe una referencia a un objeto *UserData*, el cual es el propietario de la cuenta. La clase *UserData* representa también a un usuario de Telegram, además de representar una conversación individual con un usuario.

Para concluir la descripción de los componentes más importantes del diagrama, cabe mencionar la clase *Sandbox*, la cual hereda de *QApplication* (clase perteneciente al *framework Qt*). En la clase *Sandbox* se encuentra el bucle principal de gestión de eventos, en el cual se procesan y gestionan, entre otros, los eventos generados por el usuario al interactuar con la interfaz gráfica de la aplicación.

4.2. Diagramas de Secuencia

Una vez identificadas las clases de interés, se han estudiado los momentos en los cuales se instancian, puesto que para encontrar objetos de estas clases en la memoria RAM primero deben ser creados por la aplicación. De cara a representar la creación de objetos, se han elaborado manualmente una serie de diagramas de secuencia.

En la Figura 4.2 se representa la ejecución de `Telegram Desktop`. La creación de cuentas se modela en la Figura 4.3 y la creación de sesiones en la Figura 4.4. Por otro lado, la creación tanto de historiales como de mensajes se incluye en la Figura 4.5. Por último, se muestra en la Figura 4.6 la creación de conversaciones.

De cara a reducir la complejidad de los diagramas de secuencia, en algunas partes se han incluido anotaciones con tres puntos para indicar operaciones cuya representación en los diagramas no se considera vital para este trabajo. Adicionalmente, cabe mencionar que la clase `HistoryMessage` posee diversos constructores que se encargan de la creación de objetos de las diferentes subclases de `Media`, dando valor así al atributo `_media`.

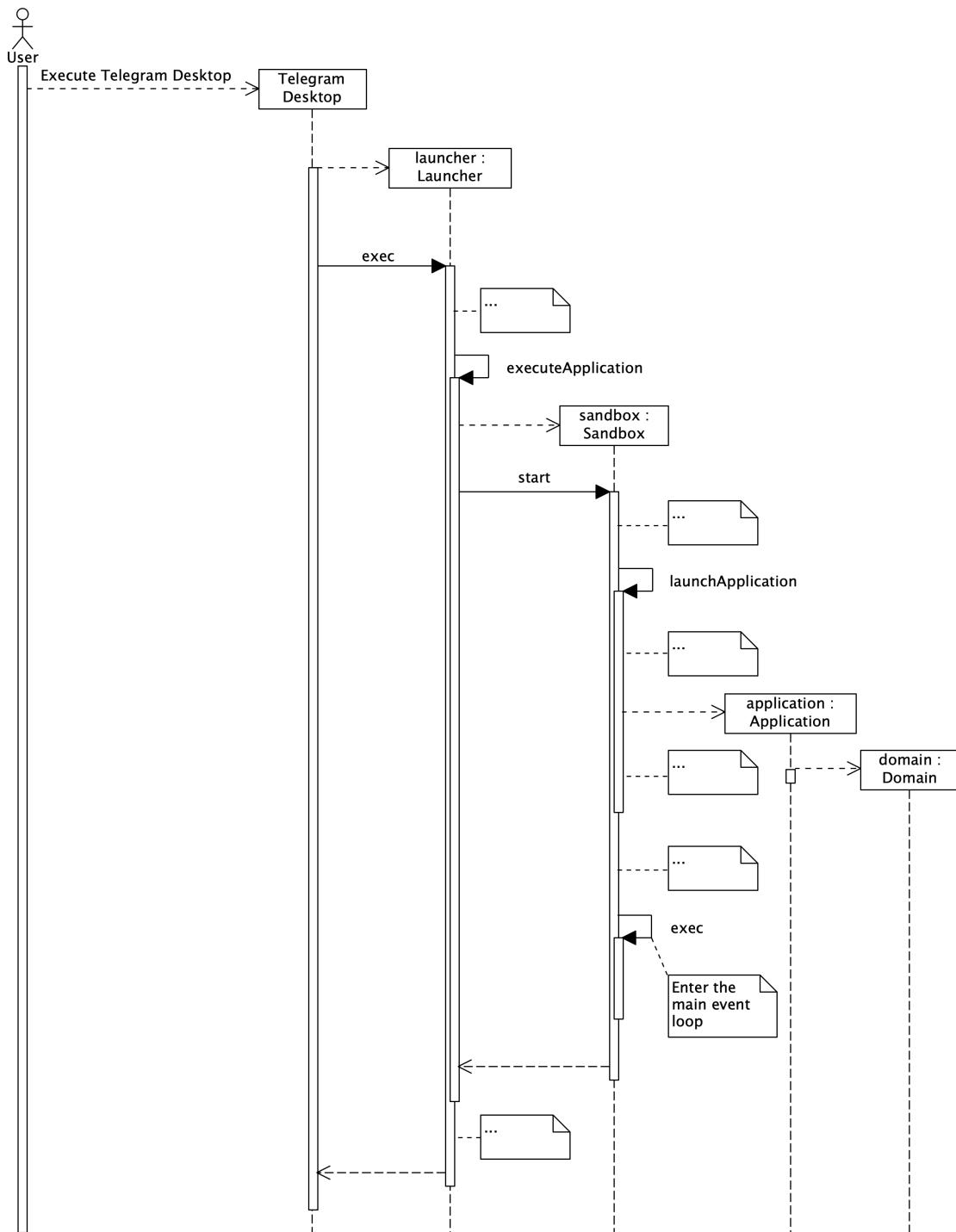


Figura 4.2: Diagrama de secuencia: Ejecución de Telegram Desktop

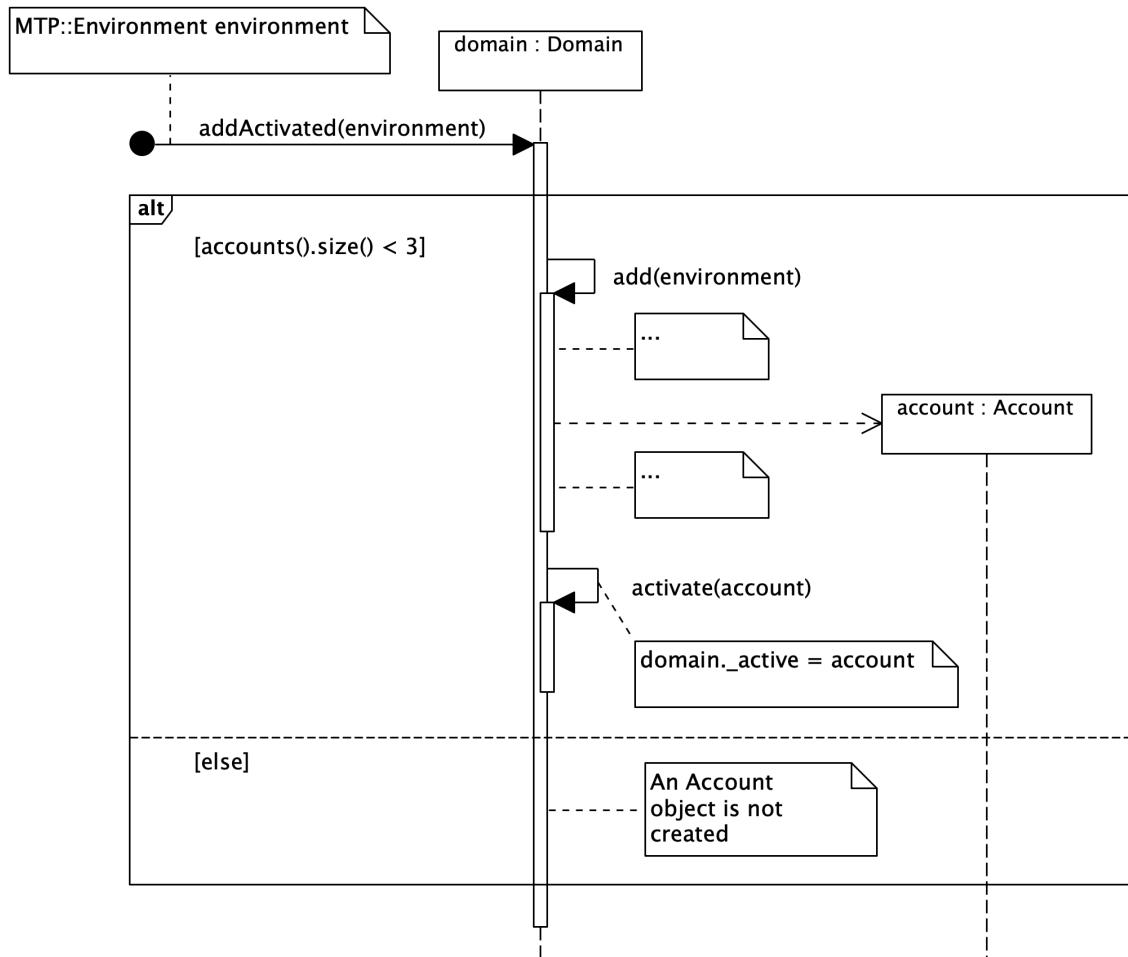


Figura 4.3: Diagrama de secuencia: Creación de cuentas

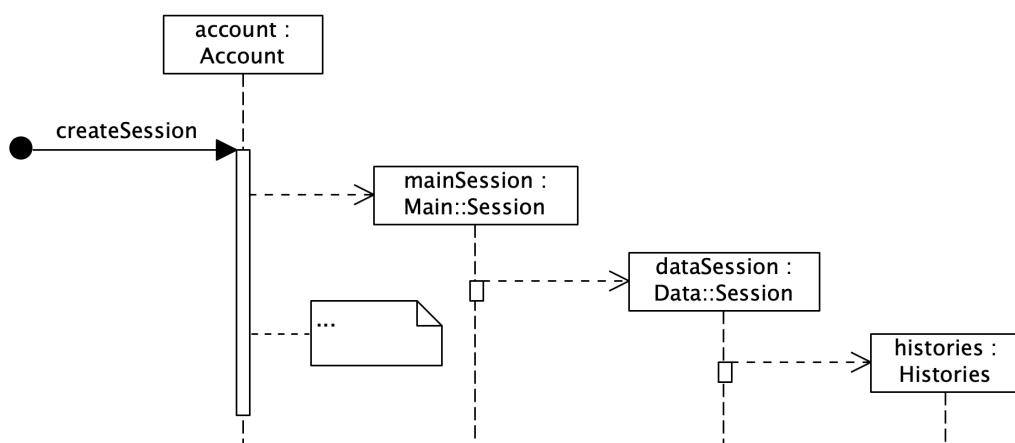


Figura 4.4: Diagrama de secuencia: Creación de sesiones

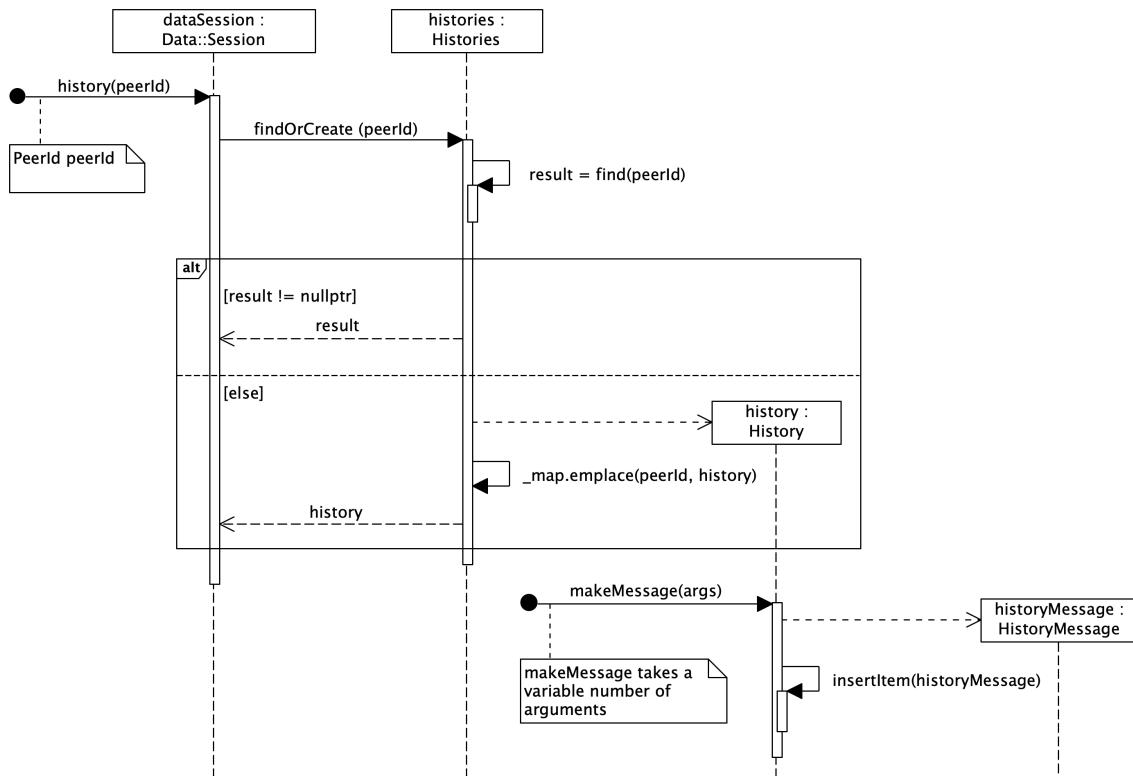


Figura 4.5: Diagrama de secuencia: Creación de históricos y mensajes

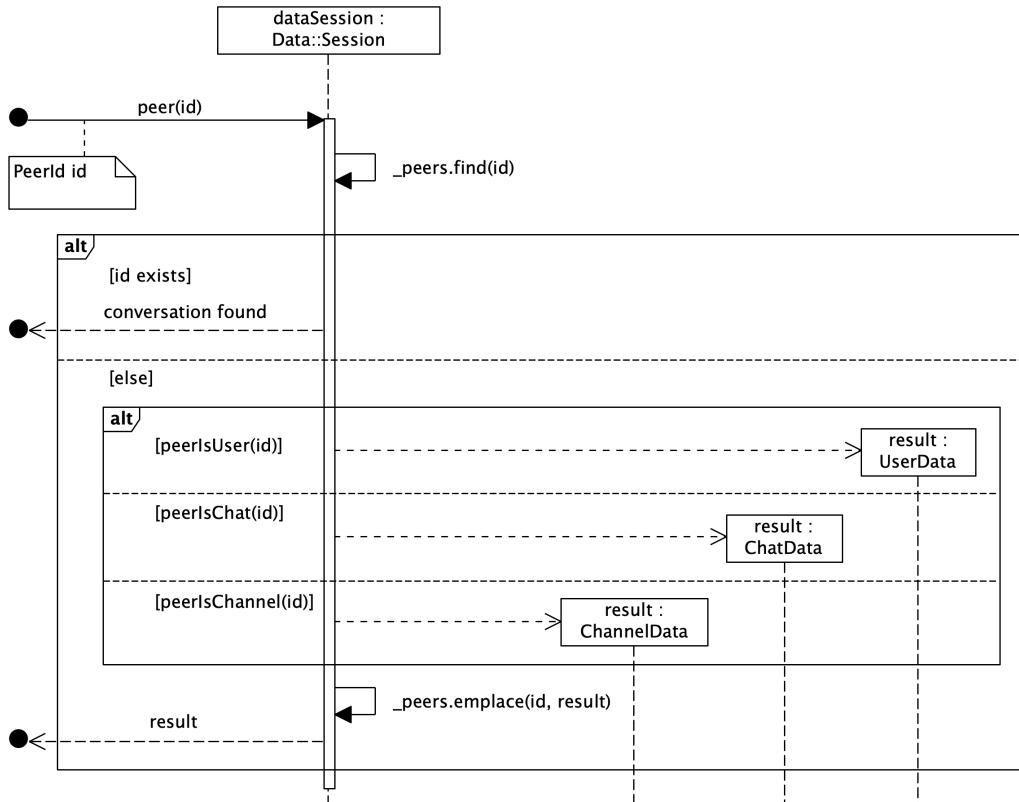


Figura 4.6: Diagrama de secuencia: Creación de conversaciones

Capítulo 5

Entorno de Análisis Desarrollado

En el presente capítulo se aborda el desarrollo del entorno de análisis dedicado a la obtención de artefactos presentes en la memoria RAM de la aplicación **Telegram Desktop**.

De cara a analizar los contenidos que se encuentran en la memoria RAM de un sistema Windows es necesario, en primera instancia, obtener dichos contenidos. Para ello, existen hoy en día diferentes herramientas, siendo **ProcDump** [40] una de las más conocidas. No obstante, en este entorno de análisis se requiere que, dado un volcado de memoria RAM, se pueda identificar de manera abierta y programática la localización con la que se corresponden las direcciones virtuales de los procesos. En los volcados obtenidos con **ProcDump** esto es posible mediante el uso del depurador de Windows **WinDbg** [41]. Sin embargo, para tal fin, se necesitan tener instalados los símbolos de Windows relativos a la versión del sistema operativo del cual proviene el volcado, ya que existen ciertos elementos en los datos extraídos propietarios de Microsoft que sin su comprensión dificulta un análisis adecuado de dichos datos.

El programa **Process Hacker** [42] permite, entre otras cosas, observar los contenidos presentes en la memoria RAM de los procesos que se encuentran en ejecución en un sistema Windows. Adicionalmente, permite volcar los contenidos de cada región de memoria perteneciente a un proceso determinado, generando ficheros cuya nomenclatura es *direcciónVirtual_tamaño*. Si se dispone de todos estos ficheros correspondientes a todas las regiones de memoria de un proceso con la nomenclatura mencionada, es posible saber el fichero donde se encuentra una dirección virtual dada y la localización concreta dentro de dicho fichero con la que se corresponde. De esta manera, es posible acceder al elemento almacenado en una dirección virtual. Una consecuencia de lo anterior es que *se puede navegar a través de la memoria*

RAM desde un objeto A hacia un objeto B, ya que si en el objeto A se encuentra almacenada la dirección de memoria del objeto B, si se ha localizado de alguna manera el objeto A, también se localizará el objeto B.

Sin embargo, un inconveniente identificado en **Process Hacker** es que no se ha encontrado la manera de obtener los ficheros individuales relativos a varias regiones de memoria de manera automática. La versión de la herramienta actual requiere crearlos manualmente de uno en uno, lo cual no es viable en un proceso forense ya que conlleva la inversión de una gran cantidad de tiempo. Como no se ha encontrado ninguna utilidad que realice esta tarea de manera automática, se ha decidido elaborar una herramienta que se encargue de esta tarea. Esta herramienta se ha denominado **Windows Memory Extractor** y se describe en más detalle en la siguiente sección.

Una vez que se han obtenido los ficheros correspondientes a las regiones de memoria de **Telegram Desktop** se indicará la localización de estos ficheros a la segunda herramienta elaborada en este trabajo, llamada **Instant Messaging Artifact Finder** (abreviado como **IM Artifact Finder**), que se encarga de analizar los contenidos presentes en ellos y generar un informe en el que se refleje la información de los artefactos obtenidos. Estas dos herramientas elaboradas componen el entorno de análisis. En la Figura 5.1 se aprecia un diagrama de alto nivel de este entorno. En las dos próximas secciones se explica en detalle cada herramienta desarrollada.

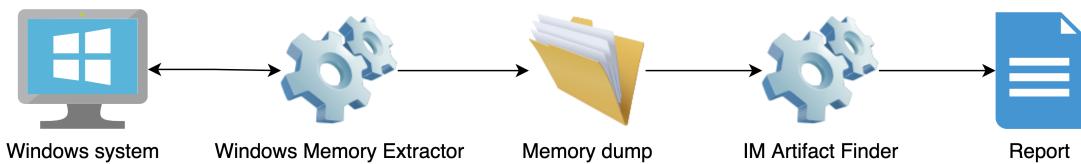


Figura 5.1: Diagrama de alto nivel del entorno de análisis

5.1. Herramienta Windows Memory Extractor

Windows Memory Extractor es una herramienta de línea de comandos escrita en C++ que realiza llamadas a la API de Windows para obtener los contenidos de la memoria RAM solicitados por el usuario. La herramienta es portable para entornos Windows, por lo tanto, no es necesario instalarla en un sistema para usarla. De esta manera, un analista forense puede tenerla en su memoria USB y ejecutarla desde dicha ubicación, evitando así contaminar más de lo estrictamente necesario el sistema que se está analizando.

Para usar la herramienta es necesario, al menos, proporcionarle el identificador de un proceso (PID) del que extraer las regiones de memoria. Por defecto, las regiones que se extraen son aquellas cuyos permisos no son de ejecución. No obstante, es posible indicar como argumento opcional las protecciones con las que deben contar las regiones que se desean extraer [43]. Adicionalmente, se le puede indicar a la herramienta el nombre de un módulo (por ejemplo, el nombre de una DLL) para extraer solamente las regiones relativas a dicho módulo.

La herramienta dispone de otra serie de argumentos opcionales que amplían su funcionalidad. Toda la documentación y ejemplos sobre su uso pueden encontrarse en el repositorio de GitHub [6]. El código fuente de esta herramienta se va a liberar bajo licencia GNU/GPLv3 en aras de la ciencia abierta.

La salida producida al ejecutar la herramienta es un directorio en el cual se encuentran ficheros con extensión *.dmp*. La nomenclatura usada para este directorio es *PID_Día-Mes-Año_Hora-Minuto-Segundo_UTC*, de cara a identificar de manera única cada extracción de memoria realizada. En cada fichero *.dmp* se encuentran los contenidos presentes en una región de memoria diferente. Al igual que se ha mencionado previamente en este capítulo, cada fichero *.dmp* se ha nombrado incluyendo primero la dirección virtual en la que comienza la región, seguida de un separador (carácter ‘_’) y del tamaño de dicha región, estando ambos valores en formato hexadecimal. Adicionalmente, en el directorio creado se encuentra un archivo de texto en el cual se listan todos los ficheros *.dmp* generados, junto a sus respectivos hashes SHA-256 y la protección de memoria con la que contaba la región correspondiente a cada fichero. En la Figura 5.2 se muestra un ejemplo de salida generada por la herramienta tras extraer los contenidos de una DLL de un proceso.

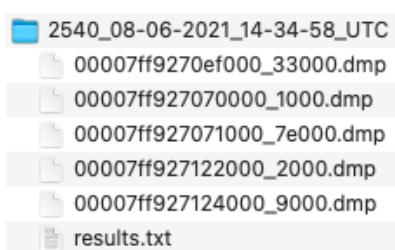


Figura 5.2: Ejemplo de salida generada por la utilidad Windows Memory Extractor

En cuanto a la estructura de la aplicación, se muestra en la Figura 5.3 su diagrama de clases. La clase *ArgumentManager* se encarga de validar y almacenar los argumentos introducidos por el usuario. Por su parte, la clase *MemoryExtractionManager* se encarga de la extracción de los contenidos de la memoria RAM (basándose

para ello en los argumentos introducidos) y de la generación de la salida descrita previamente.

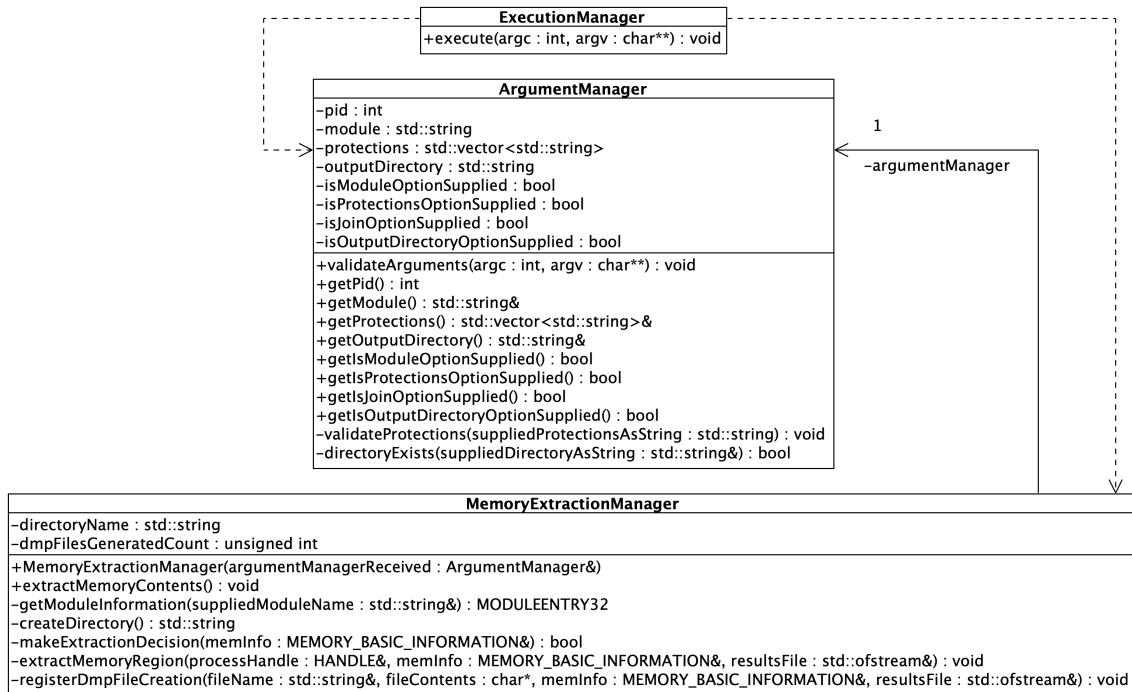


Figura 5.3: Diagrama de clases de la herramienta Windows Memory Extractor

5.2. Herramienta IM Artifact Finder

Una vez que se ha desarrollado la herramienta Windows Memory Extractor y se ha analizado el código fuente de **Telegram Desktop**, se dispone de la posibilidad de generar extracciones de memoria en el formato deseado y de los conocimientos necesarios para buscar artefactos en ellas. Por lo tanto, en este momento ya es posible desarrollar la herramienta **IM Artifact Finder**.

La utilidad **IM Artifact Finder**, desarrollada sobre Python, se ha diseñado para que sea usada como una herramienta independiente o como una librería. De cara a ser empleada como herramienta independiente, su uso se realiza a través de la línea de comandos. En cuanto a la funcionalidad como librería, el objetivo es en un futuro próximo subirla al repositorio oficial de paquetes Python [44].

Por otro lado, esta herramienta no se ha confeccionado únicamente para analizar **Telegram Desktop**, sino que se ha desarrollado como un *framework* que pueda ser extensible a otras aplicaciones de mensajería instantánea. La intención es liberar el

código fuente de esta herramienta para que pueda ser ampliada y mejorada por la comunidad.

Esta utilidad requiere dos argumentos obligatorios, siendo el primero de ellos la ruta en la que se encuentra el directorio creado por la herramienta **Windows Memory Extractor** y el segundo argumento el nombre de la plataforma de mensajería instantánea de la cual se ha realizado el volcado de memoria. Con esta información, la herramienta intentará obtener artefactos presentes en el volcado de memoria y modelar dichos artefactos en forma de objetos. En caso de hacer uso de la utilidad desde la línea de comandos, se generará un informe con la información presente en dichos objetos. Si se emplea como librería, la creación del informe dependerá de si el usuario de la librería desea generararlo o no.

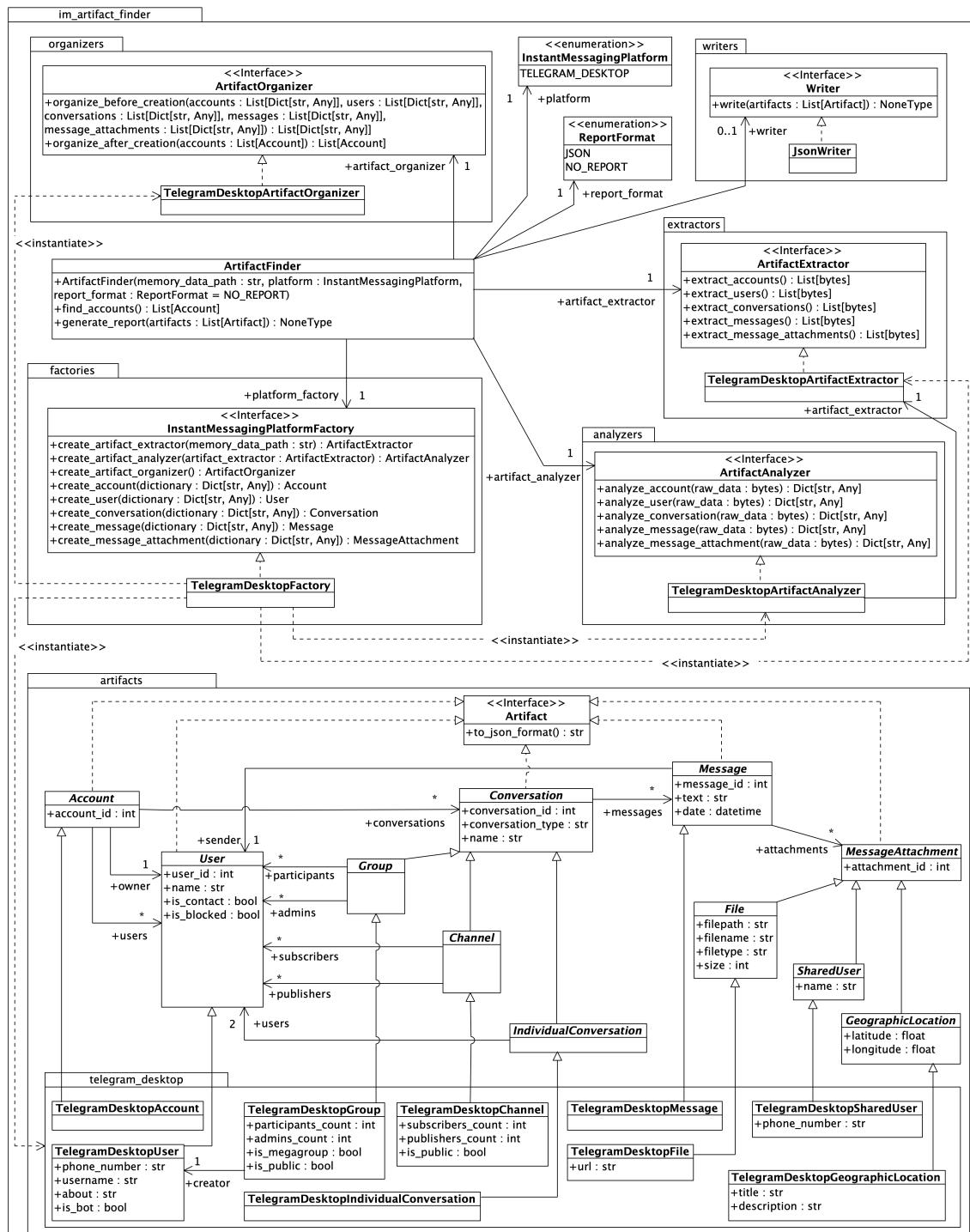
Por defecto, se usa el formato JSON para la generación del informe. A pesar de que actualmente es el único formato soportado, en el diseño se ha tenido en cuenta la posibilidad de soportar formatos adicionales en el futuro.

En el desarrollo de esta herramienta también se ha hecho uso de un repositorio en GitHub [7], donde se ha liberado el código fuente también bajo licencia GNU/GPLv3 y en el cual se encuentra información acerca de cómo usarla.

Adicionalmente, cabe mencionar que **IM Artifact Finder** acepta un argumento opcional para indicarle un directorio temporal al que se copiarán los contenidos del volcado de memoria antes de comenzar el análisis. Este directorio temporal se eliminará al finalizar el análisis. Esta funcionalidad puede ser de utilidad por razones de rendimiento (por ejemplo, en caso de que el directorio temporal se encuentre situado en la memoria RAM en lugar de en el disco duro).

5.2.1. Diseño

El diagrama de clases de **IM Artifact Finder** se muestra en la Figura 5.4. En él se aprecia que en el modelo de objetos que se ha elaborado de cara a representar los diferentes tipos de artefactos, situado en la parte inferior del diagrama, todos ellos implementan la interfaz *Artifact*. Se ha pretendido que este modelo fuese lo más general posible, de cara a poder representar en él los elementos comunes a todas las aplicaciones de mensajería instantánea. Con este propósito en mente, se han definido una serie de clases abstractas, las cuales representan los artefactos presentes en todas las plataformas. De cara a representar los detalles de cada plataforma individual, para cada plataforma soportada se debe crear un conjunto de clases concretas que hereden de las clases abstractas genéricas.

**Figura 5.4:** Diagrama de clases de la herramienta IM Artifact Finder

Obsérvese que los objetos de las clases modeladas no poseen comportamiento como tal, sino que solamente cuentan con una serie de atributos. En este caso no es necesario que tengan métodos, puesto que únicamente se pretende que almacenen la información identificada en memoria. Es también responsabilidad de cada clase concreta que implementa la interfaz *Artifact* el hecho de representarse a sí misma en los formatos soportados, es decir, las clases abstractas no implementan los métodos de representación definidos en la interfaz *Artifact*, sino que la responsabilidad recae en las clases concretas.

En el diseño de este *framework* se ha hecho uso de dos patrones de diseño: el patrón *Strategy* y el patrón *Abstract Factory*. El patrón *Strategy* [45] se emplea para definir una familia de algoritmos y hacer que los algoritmos de dicha familia sean intercambiables, permitiendo que el algoritmo concreto pueda variar de manera independiente al cliente que lo usa. En el caso del *framework* elaborado, cada implementación de la interfaz *Writer* es una estrategia diferente de generación de informes, existiendo una estrategia por cada formato soportado. Para cada formato de informe al que se desee dar soporte deberá existir una clase que implemente la interfaz *Writer* y en el método *write()* de dichas clases se invocará, para cada objeto recibido, al método de representación del formato correspondiente, para obtener la representación deseada de cada objeto. De esta forma, se simplifica el soporte de nuevos formatos de informes.

De cara a generar objetos concretos que representen artefactos, primero es necesario extraer los datos correspondientes a dichos artefactos de un volcado de memoria, mediante la búsqueda de patrones en dicho volcado, de lo cual se encargan las implementaciones de la interfaz *ArtifactExtractor*. Tras haber extraído los datos de los artefactos, es competencia de las clases que implementan la interfaz *ArtifactAnalyzer* el analizar dichos datos. Antes de crear los objetos que representan a los artefactos, puede que sea necesario una previa organización de la información descubierta en el análisis de los mismos. Las operaciones para organizar la información, tanto antes de la creación de objetos como después (en caso de ser necesario) están declaradas en la interfaz *ArtifactOrganizer*. Una vez que los artefactos se han extraído, su información ha sido analizada y posteriormente organizada adecuadamente, es posible la creación de objetos que modelan dichos artefactos, que es a su vez responsabilidad de las clases que implementan la interfaz *InstantMessagingPlatformFactory*. De esta manera, existe una separación de responsabilidades.

Una instancia de *ArtifactFinder* debe tener acceso a una implementación de cada una de las 4 interfaces descritas en el párrafo anterior. Sin embargo, un aspecto

clave a tener en cuenta es que dichos 4 objetos deben pertenecer todos a la misma plataforma de mensajería instantánea para evitar que, por ejemplo, se use la técnica de extracción de artefactos relativa a **Telegram Desktop** mientras que se emplea la técnica de análisis de WhatsApp. De cara a provocar que dichos 4 objetos sean todos de la misma plataforma se ha empleado el patrón *Abstract Factory* [45], mediante el cual se define una interfaz para crear familias de objetos relacionados. En este caso, la interfaz es *InstantMessagingPlatformFactory*. Las clases que implementan esta interfaz se conocen como factorías, las cuales se definen como clases encargadas de crear objetos. Para cada plataforma que sea soportada por **IM Artifact Finder** deberá existir su propia factoría y, dependiendo de la plataforma que se indique como argumento a *ArtifactFinder*, se creará la factoría que corresponda. Posteriormente, desde *ArtifactFinder* se invocarán los métodos de creación de dicha factoría para crear los objetos necesarios, que serán siempre relativos a la misma plataforma.

Mediante el empleo de los dos patrones de diseño descritos anteriormente, la clase *ArtifactFinder* siempre depende de interfaces. Como consecuencia, *ArtifactFinder* se abstrae de los detalles para obtener artefactos de cada plataforma, ya que son las clases de cada propia plataforma las que se encargan de dichos detalles. Desde *ArtifactFinder* únicamente se invocan métodos declarados en interfaces, que son implementadas por las clases concretas de cada servicio de mensajería instantánea soportado.

Realizar el diseño de esta forma implica que el *framework* pueda soportar diferentes plataformas de manera sencilla. Para soportar un nuevo servicio de mensajería instantánea se deberán implementar las interfaces *ArtifactExtractor*, *ArtifactAnalyzer*, *ArtifactOrganizer* e *InstantMessagingPlatformFactory*, además de extender las clases abstractas que implementan la interfaz *Artifact*.

5.2.2. Obtención de Artefactos

En este apartado se describe el proceso que sigue la herramienta para identificar artefactos presentes en volcados de memoria del proceso de **Telegram Desktop**. Para ello, se hace referencia en múltiples ocasiones a la Figura 4.1, incluida en el capítulo anterior.

En lo relativo a la identificación de usuarios, se han buscado patrones de números de teléfono en un volcado de memoria dado. En este caso, lo que se está intentando encontrar son los contenidos de la propiedad *_phone* de la clase *UserData*, cuyo tipo es *QString* [46] (clase perteneciente a **Qt**). Debido a que la clase *QString* es

empleada en numerosas ocasiones por **Telegram Desktop** para almacenar cadenas de texto, se ha estudiado la manera en la que se estructura la información de dicha clase, descubriendo que sigue un patrón muy definido. En **IM Artifact Finder** se ha incorporado la funcionalidad para conocer si en una dirección virtual concreta se encuentra almacenado un objeto *QString*. Por otro lado, los caracteres del texto que se almacena en un objeto *QString* se almacenan en memoria en formato UTF-16. Una vez encontrados números de teléfono codificados en UTF-16 y comprobado que son objetos de tipo *QString*, se procede al análisis de localizaciones concretas en los alrededores de la dirección virtual de *_phone*, en busca de más objetos *QString*, los cuales serán, entre otros, el nombre de usuario (atributo *username* de *UserData*) o su nombre completo (propiedad *name* de *PeerData*). Esto ha permitido identificar objetos *UserData* en un volcado de memoria.

Una limitación presente en esta estrategia de búsqueda de usuarios es que si un usuario A de **Telegram** no comparte su número de teléfono con un usuario B, el número de teléfono del usuario A no se encontrará en un volcado de memoria extraído del equipo del usuario B. Sin embargo, se ha descubierto que existen usuarios que se almacenan de manera contigua en memoria, aunque ello no sucede para todos los usuarios. Como consecuencia, conociendo el tamaño de un objeto *UserData* se puede analizar si para los objetos *UserData* identificados existen otros objetos del mismo tipo contiguos, siendo así posible encontrar usuarios adicionales.

En cuanto a la estrategia para recuperar mensajes, se ha observado en la interfaz de usuario de **Telegram Desktop** que junto a cada mensaje enviado y recibido se refleja el momento en el cual dicho mensaje ha sido enviado, como se aprecia en la Figura 5.5. Este dato corresponde con la propiedad *_timeText* de la clase *HistoryMessage*. Como consecuencia, la manera en la que se han encontrado objetos de tipo *HistoryMessage* en un volcado de memoria ha sido a través de la búsqueda de patrones horarios.

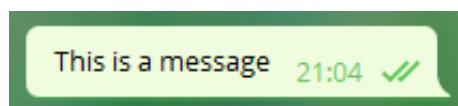


Figura 5.5: Mensaje enviado a través de **Telegram Desktop**

Con la intención de clasificar cada mensaje encontrado en su conversación correspondiente, se ha descubierto la localización de la propiedad *_history* de *HistoryItem*, de cara a obtener el objeto *History*, y en él se ha localizado la propiedad *peer*, para tener así acceso a la conversación relativa al historial. De esta forma se han podido identificar conversaciones y conocer qué mensajes pertenecen a cada una de ellas.

Por otra parte, para descubrir quién envió cada mensaje, se analizan los contenidos presentes en el atributo `_from` de `HistoryItem`, permitiendo también descubrir más usuarios que no hayan sido encontrados con los métodos descritos previamente.

En lo relacionado con mensajes multimedia, en caso de que la propiedad `_media` de `HistoryMessage` contenga un puntero nulo, significa que el mensaje no es de tipo multimedia. En caso contrario, será necesario acudir a la localización en el volcado de memoria correspondiente a la dirección virtual almacenada en el atributo `_media` para estudiar los contenidos del objeto almacenado en esa dirección, con objeto de conocer qué tipo de elemento se envió en el mensaje e información acerca de él. En este trabajo se ha puesto el foco en contactos compartidos, ficheros, y coordenadas geográficas, ya que se han considerado los elementos adjuntos más importantes desde un punto de vista forense.

Con el propósito de descubrir con qué cuenta está relacionado cada usuario y conversación descubiertos, se ha estudiado dónde se almacena la propiedad `_owner` en un objeto `PeerData` para desde él navegar hacia el objeto `Data::Session`, del cual se ha analizado su propiedad `_session`, siendo posible así llegar hasta el objeto `Main::Session`. Aquí se ha identificado su propiedad `_user`, para finalmente obtener el objeto `UserData` que representa al poseedor de la cuenta.

Además de los datos encontrados previamente empleados en la identificación de los principales tipos de artefactos (cuentas, conversaciones, usuarios, mensajes y elementos adjuntos en mensajes), para cada tipo de artefacto se han identificado localizaciones de más propiedades, de cara a obtener más datos sobre ellos. Un ejemplo de ello es la propiedad `_date` en `HistoryMessage`, con el objetivo en este caso de saber en qué fecha fue enviado cada mensaje.

Una vez que se encuentra de manera manual por primera vez un artefacto en un volcado de memoria, de cara a identificar la localización concreta de una propiedad dentro de dicho artefacto, es de vital importancia en este caso el estudio del código fuente. Adicionalmente, es de gran ayuda conocer información acerca del objeto o artefacto que se está analizando. A modo de ejemplo, en el momento de identificar manualmente un objeto de tipo `UserData` mediante su número de teléfono, es muy útil conocer el nombre del usuario al que pertenece dicho número de teléfono. Tras descubrir de manera manual la localización de una propiedad dentro de un objeto, ya es posible automatizar el proceso, sabiendo que dicha propiedad siempre se encontrará en el mismo lugar dentro de objetos del mismo tipo.

Para concluir, la herramienta también genera un documento de texto en el que se incluye un resumen acerca de los artefactos que se han conseguido recuperar. En la Figura 5.6 se muestran los contenidos de un ejemplo de resumen. El resumen comienza mencionando el número de cuentas identificadas. Tras ello, para cada cuenta se muestra su propietario, el número de conversaciones, mensajes y usuarios encontrados relacionados con la cuenta, e información acerca del mensaje más reciente recuperado en dicha cuenta. La posesión de un resumen de este estilo permite a un analista forense conocer la cantidad de información identificada antes de acudir al informe completo. En el próximo capítulo se dan más detalles acerca de este informe completo.

```
Number of accounts found: 2

Information about account 1:
  Owner: Pedro Fernández
  Number of conversations found: 5
  Number of messages found: 89
  Number of users found: 50
  The most recent message recovered was sent at 2021-05-24T23:33:30+00:00 in the conversation with UsuarioTest TFM

Information about account 2:
  Owner: UsuarioTest TFM
  Number of conversations found: 3
  Number of messages found: 58
  Number of users found: 6
  The most recent message recovered was sent at 2021-05-27T10:26:32+00:00 in the conversation with GitHub
```

Figura 5.6: Ejemplo de resumen generado por la herramienta IM Artifact Finder

Capítulo 6

Experimentos y Evaluación

En este capítulo se evalúa el entorno de análisis descrito en el capítulo anterior. En primer lugar, se incluyen los casos de prueba definidos y los resultados obtenidos. Posteriormente, se presenta una discusión acerca de las implicaciones de estos resultados. Por último, se incluyen detalles acerca de los informes generados por la herramienta **IM Artifact Finder**.

Los experimentos se han llevado a cabo en 2 máquinas virtuales con sistema operativo Windows 10 de 64 bits, una de ellas con la edición Education (versión 20H2) y otra con la edición de prueba Enterprise (versión 1809). Adicionalmente, la evaluación se ha hecho con la versión 2.7.1 de **Telegram Desktop** para 64 bits instalada en las máquinas.

Por otro lado, para realizar los experimentos se han utilizado dos cuentas de **Telegram**. Una de ellas es la cuenta personal del autor de este trabajo. Adicionalmente, se ha creado una cuenta cuyo único propósito es realizar pruebas en este proyecto. El nombre completo del propietario de esta cuenta de prueba es «UsuarioTest TFM» y su nombre de usuario es «UsernameTestTFM». La cuenta de prueba mencionada ha sido configurada para que el número de teléfono no sea compartido con ningún otro usuario.

De cara a llevar a cabo un caso de prueba determinado se ha interactuado de manera manual con la aplicación **Telegram Desktop** para realizar las acciones correspondientes a dicho caso. Posteriormente, se ha realizado un volcado del proceso de **Telegram Desktop** con la herramienta **Windows Memory Extractor**. Tras ello, se ha ejecutado la herramienta **IM Artifact Finder** proporcionándole como entrada la ruta del volcado obtenido. Por último, se ha analizado de manera manual el

informe generado por la utilidad **IM Artifact Finder**. Estos pasos se han seguido para cada caso de prueba definido.

6.1. Casos de Prueba

Con el propósito de evaluar el entorno de análisis desarrollado se han definido 17 casos de prueba, clasificados en una de las siguientes 7 categorías: *cuentas, conversaciones, usuarios, privacidad, multimedia, bloqueo y sesión*. Cabe destacar que no se han apreciado diferencias en los resultados obtenidos entre las dos ediciones de Windows 10 evaluadas. A continuación se describen los casos de prueba y los resultados, agrupados por categorías.

6.1.1. Categoría *Cuentas*

Caso de prueba I: «Cuenta única». *Obtener información sobre el propietario de la cuenta, cuando solamente existe una cuenta.*

Se han logrado identificar los siguientes datos acerca del propietario de la cuenta: identificador, nombre completo, número de teléfono, y, en caso de contar con él, su nombre de usuario.

Caso de prueba II: «Varias cuentas». *Encontrar información acerca de los propietarios de las cuentas añadidas a la aplicación, en el caso de existir múltiples cuentas.*

Se ha conseguido conocer el número de cuentas añadidas en la aplicación, para cada una de las cuales se han identificado los mismos datos acerca de sus respectivos propietarios que en el **Caso de prueba I**. Las pruebas relacionadas con este caso se han realizado con las 2 cuentas mencionadas previamente.

6.1.2. Categoría *Conversaciones*

Caso de prueba III: «Conversaciones totales». *Identificar las conversaciones existentes. Nótese que en este caso de prueba no se pretenden obtener los mensajes de dichas conversaciones.*

Solamente se han podido conocer las conversaciones a las que el usuario ha accedido, bien para leer sus contenidos o para enviar y recibir mensajes. Las conversaciones a las que el usuario no ha entrado de manera explícita no se han encontrado en memoria RAM, y por tanto no se han podido identificar. Para cada conversación

encontrada se puede distinguir si se trata de una conversación individual, un grupo o un canal. Además, en las conversaciones individuales se pueden conocer los dos participantes, y en el caso de los grupos se ha conseguido conocer un subconjunto de los participantes (concretamente, aquellos que hayan enviado mensajes en el grupo). De manera adicional, se ha conseguido saber el nombre tanto de los grupos como de los canales identificados. Por último, cada conversación identificada se puede relacionar con la cuenta a la que pertenece.

Caso de prueba IV: «Conversaciones accedidas recientemente». *Identificar las conversaciones que han sido accedidas de manera reciente, bien debido a que un usuario ha enviado algún mensaje a través de ellas o porque las ha abierto para leer sus contenidos.*

Las conversaciones accedidas por el usuario de manera reciente se han podido identificar, ya que son las que se encuentran cargadas en memoria RAM.

Caso de prueba V: «Reconstrucción de conversaciones». *Tratar de identificar en los contenidos de la memoria RAM aquellos datos necesarios para reconstruir conversaciones. Este caso se centra en mensajes de texto, no en mensajes multimedia.*

Es posible la reconstrucción de conversaciones en las que el usuario ha entrado, tanto en el caso de conversaciones individuales como en el caso de grupos y canales. Concretamente, se han podido recuperar los mensajes de texto normales, las respuestas a mensajes y los mensajes reenviados. Los mensajes editados, sin embargo, no han sido encontrados. En el caso de los mensajes multimedia, se ha identificado el hecho de haber sido enviados, conociendo el texto de los mismos. Sin embargo, la obtención de los datos acerca de los elementos multimedia transferidos es responsabilidad de otros casos de prueba definidos (**casos de prueba XII, XIII y XIV**).

Caso de prueba VI: «Mensajes eliminados». *Eliminar mensajes tanto enviados como recibidos, los cuales desaparecerán de la interfaz de usuario, y comprobar si tras ello se encuentran presentes en memoria RAM.*

En este caso no se han podido recuperar mensajes enviados ni recibidos tras eliminarlos.

Caso de prueba VII: «Conversaciones eliminadas». *Comprobar si tras eliminar una conversación se pueden recuperar contenidos relativos a ella de la memoria RAM.*

Tras eliminar una conversación, es posible recuperar parte de ella. Sin embargo, los contenidos recuperados no son completamente fiables, ya que no siempre son correctos y en ocasiones se encuentran incompletos.

6.1.3. Categoría *Usuarios*

Caso de prueba VIII: «Lista de contactos». *Obtener la lista de contactos relativos a cada cuenta.*

Se han logrado detectar los usuarios que comparten su número de teléfono con el propietario de cada cuenta y también algunos de los usuarios que no lo comparten. Además, para cada usuario recuperado es posible saber con qué cuenta está relacionado. Tras abrir Telegram Desktop, no es necesario realizar ninguna interacción con la aplicación para encontrar usuarios en memoria RAM. Por otro lado, se ha podido diferenciar entre los usuarios que son contactos y los que no. Por último, para cada usuario es posible conocer su identificador, su nombre completo, su número de teléfono (en caso de que lo comparta), su nombre de usuario (en caso de que lo posea) y si se trata de un bot o no.

Caso de prueba IX: «Contactos eliminados». *Comprobar si tras eliminar un contacto se puede encontrar información sobre él en memoria RAM.*

En todas las pruebas realizadas se han recuperado los contactos eliminados. Adicionalmente, una vez recuperados, se indica que ya no son contactos.

Caso de prueba X: «Usuarios bloqueados». *Identificar si un usuario se encuentra bloqueado por otro usuario.*

Además de los datos mencionados en el **Caso de prueba VIII**, es posible identificar si un usuario está bloqueado o no. Sin embargo, para que esto pueda ser detectado, el usuario debe estar envuelto en una conversación individual cargada previamente en memoria RAM.

6.1.4. Categoría *Privacidad*

Caso de prueba XI: «Privacidad del número de teléfono». *En caso de que el usuario A no comparta su número de teléfono con el usuario B, comprobar si en*

los contenidos de la memoria RAM del equipo del usuario B puede identificarse el número de teléfono del usuario A.

Si un usuario no comparte su número de teléfono, no ha sido posible encontrar dicho número de teléfono.

6.1.5. Categoría *Multimedia*

Caso de prueba XII: «Archivos». *Obtener información sobre los archivos que han sido tanto enviados como recibidos.*

Cuando se adjunta un fichero en un mensaje, se ha conseguido recuperar el nombre del archivo y su tipo. Además, si se envían varios ficheros adjuntos en el mismo mensaje, es posible recuperar dicha información para cada fichero transferido.

Caso de prueba XIII: «Contactos compartidos». *Conocer los contactos que han sido compartidos con otros usuarios.*

Se ha sido capaz de recuperar el nombre y el número de teléfono de los contactos compartidos.

Caso de prueba XIV: «Localizaciones geográficas». *Obtener localizaciones geográficas transferidas. Nótese que a través de Telegram Desktop no pueden enviarse localizaciones geográficas, pero sí pueden recibirse.*

Se han recuperado la latitud y la longitud de cada localización geográfica transferida. Además, también es posible la recuperación de cualquier información adicional de la localización (como el nombre o dirección del lugar).

6.1.6. Categoría *Bloqueo*

Caso de prueba XV: «Bloqueo de la aplicación». *Investigar la información que se encuentra en memoria RAM tras bloquear la aplicación.*

En las pruebas llevadas a cabo se ha observado que se puede recuperar la misma información de la memoria RAM, sin importar que la aplicación esté bloqueada o no.

Caso de prueba XVI: «Contraseña de desbloqueo». *Tratar de encontrar en memoria RAM la contraseña de desbloqueo de la aplicación.*

Tras desbloquear la aplicación se ha identificado la contraseña en memoria de manera manual, almacenada en UTF-8. No obstante, dicha contraseña deja de estar en memoria tras un corto periodo de tiempo. En este caso, no se ha podido recuperar la contraseña de desbloqueo de manera automática y fiable, a pesar de que se encontrase presente en el volcado de memoria realizado.

6.1.7. Categoría Sesión

Caso de prueba XVII: «Cierre de sesión». *Investigar los contenidos presentes en memoria RAM una vez que se ha cerrado sesión en Telegram Desktop.*

Ha sido posible identificar artefactos tras cerrar sesión. Sin embargo, los artefactos recuperados no son todos los que se obtienen antes de cerrar sesión, sino solamente un subconjunto. Por otro lado, la recuperación de los mismos no es completamente fiable, ya que en ocasiones los datos recuperados son incorrectos o incompletos.

6.2. Discusión

El hecho de que mediante el entorno de análisis elaborado se pueda recuperar de la memoria RAM la información mencionada previamente implica que, aunque dicha información se encuentre cifrada en el disco del equipo y las comunicaciones con los servidores de Telegram estén cifradas, un analista forense podrá tener la posibilidad de recuperar artefactos valiosos de la memoria volátil de un ordenador incautado, siempre que se encuentre encendido.

La recuperación de información acerca del propietario o propietarios de las cuentas que se encontraban en uso dentro de Telegram Desktop podría ayudar a identificar a quién pertenece un equipo incautado. De manera adicional, identificar usuarios relacionados con cada cuenta y diferenciar si son contactos puede proporcionar indicios sobre las personas con las que se relacionaba un sospechoso, además de permitir identificar nuevos individuos posibles a investigar. Por otro lado, debe existir una razón de peso normalmente para que un contacto sea bloqueado por otro. Por lo tanto, conocer este hecho puede proporcionar detalles relevantes a un analista forense acerca de la relación personal entre ambos usuarios.

La posibilidad de reconstruir una conversación cronológicamente, sabiendo la fecha en la que cada mensaje fue enviado, quién fue el emisor, quién o quiénes fueron los receptores y qué se envió en cada mensaje puede ser de vital relevancia para la resolución de un caso forense. Adicionalmente, conocer grupos o canales en los que

un sujeto está presente puede revelar información acerca de sus intereses, ya que, por ejemplo, el hecho de que se encuentre en un grupo en el que se comparta material polémico o ilegal puede ser un descubrimiento importante en una investigación.

En cuanto a los mensajes en los que no se envía solamente texto, la opción de conocer los contactos que han sido compartidos en las conversaciones puede resultar de utilidad, ya que si un sospechoso recibe un contacto de otro usuario, cabe pensar que estos 3 individuos puedan estar relacionados entre sí. En lo relativo a las localizaciones geográficas, poseer estos datos puede ser relevante de cara a conocer lugares concretos donde un sospechoso haya podido estar o que le resulten de interés. Por otra parte, los archivos que se transfieren a través de **Telegram Desktop**, por defecto, se almacenan en un directorio dentro de la carpeta de descargas del sistema, aunque esta localización puede ser modificada. Sin embargo, conocer los nombres de los archivos transferidos y su tipo puede resultar beneficioso en un potencial posterior análisis forense de disco, de cara a localizar estos archivos y conocer sus contenidos.

Un aspecto a destacar en las pruebas realizadas es el hecho de que sea posible la recuperación de artefactos de manera completa y fiable cuando la aplicación está bloqueada, ya que en la supuesta situación de que un analista forense pueda interactuar con el ordenador de un sospechoso que todavía esté encendido, y aunque la aplicación se encuentre bloqueada, existirá la posibilidad de recuperar datos de valor a los que no se tendría acceso salvo que se conozca la contraseña de desbloqueo de **Telegram Desktop**.

Por otro lado, aunque la recuperación de conversaciones después de ser eliminadas y la obtención de artefactos tras cerrar sesión no sea completamente confiable, una gran parte de los datos recuperados en las pruebas realizadas sí eran correctos, lo cual también puede resultar útil ya que, aunque un analista forense tuviese permitido interactuar con la aplicación de **Telegram Desktop** de un equipo incautado encendido, no podría acceder a dichos datos a través de la interfaz de usuario.

En relación con la privacidad del número de teléfono si un usuario decide no compartirlo, tras los experimentos llevados a cabo, no se ha encontrado en los datos obtenidos correspondientes a dicho usuario su número de teléfono. Este hecho es coherente con la expectativa de privacidad del usuario.

De manera adicional a lo mencionado anteriormente, conocer las conversaciones a las que un usuario ha accedido recientemente podría ser relevante a la hora de saber

qué estaba haciendo o con quién se estaba comunicando un sospechoso momentos previos a la sucesión de unos determinados hechos.

6.3. Informes Generados

De cara a proporcionar detalles acerca de los informes que genera la aplicación IM Artifact Finder se ha decidido no mostrar números de teléfono, por razones de privacidad. No obstante, ello no impide mostrar extractos de diferentes informes generados. El primer extracto se muestra en la Figura 6.1, donde se aprecia la información obtenida sobre el usuario llamado «UsuarioTest TFM». En la Figura 6.2 se muestra la información relativa a otro usuario, en este caso, el bot de GitHub.

```
{  
    "user_id": 1680408400,  
    "name": "UsuarioTest TFM",  
    "is_contact": true,  
    "is_blocked": false,  
    "phone_number": null,  
    "username": "UsernameTestTFM",  
    "about": null,  
    "is_bot": false  
},
```

Figura 6.1: Información obtenida sobre «UsuarioTest TFM»

```
{  
    "user_id": 107550100,  
    "name": "GitHub",  
    "is_contact": false,  
    "is_blocked": false,  
    "phone_number": null,  
    "username": "GitHubBot",  
    "about": null,  
    "is_bot": true  
}
```

Figura 6.2: Información obtenida acerca del usuario correspondiente al bot de GitHub

En lo relativo a las conversaciones, se encuentran en la Figura 6.3 parte de los datos recuperados de una conversación, donde se aprecia que es una conversación individual con el usuario «Pedro Fernández», aunque no se muestran los mensajes de dicha conversación. Un ejemplo de mensaje de texto normal enviado por «UsuarioTest TFM» se muestra en la Figura 6.4, en la que se aprecia que no existe ningún adjunto ya que no es un mensaje multimedia. Por su parte, se representa en la Figura 6.5 un mensaje en el que se ha enviado un fichero de texto como adjunto. Por

último, se observa en la Figura 6.6 la información que se ha podido recuperar acerca de una localización geográfica enviada. Concretamente, en este caso se aprecia que dicha localización se corresponde con el Museo de León.

```
"conversations": [
  {
    "conversation_id": 916093840,
    "conversation_type": "Individual conversation",
    "name": "Pedro Fernández",
    "messages": [
```

Figura 6.3: Conversación individual con «Pedro Fernández» (sin mostrar los mensajes)

```
{
  "message_id": null,
  "text": "This is a normal message",
  "date": "2021-06-18T18:09:14+00:00",
  "sender": {
    "user_id": 1680408400,
    "name": "UsuarioTest TFM",
    "is_contact": true,
    "is_blocked": false,
    "phone_number": null,
    "username": "UsernameTestTFM",
    "about": null,
    "is_bot": false
  },
  "attachments": null
},
```

Figura 6.4: Ejemplo de mensaje de texto obtenido

```
{  
    "message_id": null,  
    "text": "",  
    "date": "2021-06-18T18:11:05+00:00",  
    "sender": {  
        "user_id": 1680408400,  
        "name": "UsuarioTest TFM",  
        "is_contact": true,  
        "is_blocked": false,  
        "phone_number": null,  
        "username": "UsernameTestTFM",  
        "about": null,  
        "is_bot": false  
    },  
    "attachments": [  
        {  
            "attachment_id": null,  
            "filepath": null,  
            "filename": "Note.txt",  
            "filetype": "text/plain",  
            "size": null,  
            "url": null  
        }  
    ]  
},
```

Figura 6.5: Información obtenida sobre un fichero adjunto enviado en un mensaje

```
{  
    "message_id": null,  
    "text": "",  
    "date": "2021-06-18T18:19:08+00:00",  
    "sender": {  
        "user_id": 1680408400,  
        "name": "UsuarioTest TFM",  
        "is_contact": true,  
        "is_blocked": false,  
        "phone_number": null,  
        "username": "UsernameTestTFM",  
        "about": null,  
        "is_bot": false  
    },  
    "attachments": [  
        {  
            "attachment_id": null,  
            "latitude": 42.59849852178585,  
            "longitude": -5.571721718367973,  
            "title": "Museo de León",  
            "description": "Pl. Santo Domingo, 8"  
        }  
    ]  
},
```

Figura 6.6: Información obtenida acerca de una localización geográfica enviada

Capítulo 7

Conclusiones y Trabajo Futuro

En el presente capítulo se abordan las conclusiones del proyecto elaborado y se hace referencia al trabajo futuro que puede ser realizado de cara a la mejora y ampliación del mismo.

El uso de aplicaciones de mensajería instantánea ha aumentado de manera notable durante la última década. En este trabajo se ha puesto el foco en la plataforma de mensajería instantánea Telegram, una de las más populares a nivel global, centrándose de manera específica en su versión disponible para ordenadores, conocida como **Telegram Desktop**.

En este proyecto se ha elaborado un entorno de análisis destinado a la obtención de artefactos forenses presentes en la memoria RAM del proceso de **Telegram Desktop** en sistemas Windows. El entorno de análisis está formado por dos herramientas, llamadas **Windows Memory Extractor** e **IM Artifact Finder**. La utilidad **Windows Memory Extractor** se encarga de obtener los contenidos solicitados por el usuario presentes en la memoria RAM de un proceso determinado en un sistema Windows. Dichos contenidos extraídos son posteriormente analizados con la herramienta **IM Artifact Finder** de cara a identificar en ellos artefactos forenses. Las herramientas desarrolladas están liberadas de manera gratuita para que los profesionales del análisis forense hagan libremente uso de ellas y con el objetivo de que cualquier persona pueda contribuir a la mejora de ambas. Cabe destacar el hecho de que la herramienta **Windows Memory Extractor** también puede ser empleada como herramienta independiente, con la intención de extraer los contenidos presentes en la memoria RAM que un determinado usuario necesite.

Haber elaborado la herramienta **Windows Memory Extractor** y analizar el código fuente de **Telegram Desktop** han sido dos pilares básicos previos y fundamentales al comienzo de la implementación de la utilidad **IM Artifact Finder**, ya que mediante el análisis de código fuente se ha conseguido saber la manera en que la aplicación organiza la información en memoria RAM. Del mismo modo, la estructura de los datos extraídos por **Windows Memory Extractor** permite conocer la localización correspondiente a direcciones virtuales, lo cual posibilita la navegación a través de la información recolectada.

Los resultados obtenidos tras la evaluación del entorno de análisis han proporcionado una serie de artefactos relacionados con **Telegram Desktop** cuya relevancia puede ser notable de cara a la resolución de un caso forense. Entre otros, se ha conseguido obtener información acerca de los usuarios de la aplicación y de sus contactos, se han reconstruido conversaciones, y se han podido recuperar determinados contenidos tras haber sido eliminados deliberadamente, tras haber bloqueado la aplicación e incluso tras haber cerrado sesión.

Por último, cabe recalcar en el caso concreto de **Telegram Desktop** la importancia del estudio de los contenidos que se almacenan en la memoria volátil, puesto que tanto los datos presentes en el disco como los transmitidos desde y hacia los servidores de Telegram se encuentran cifrados. No obstante, **Telegram Desktop** necesita descifrar dichos contenidos para operar con ellos, y el lugar en el que estarán presentes los datos descifrados será la memoria RAM.

Una de las limitaciones de este trabajo es que solamente se ha analizado la versión 2.7.1 de **Telegram Desktop**. Por tanto, posibles versiones futuras de esta aplicación implicarán adaptaciones de **IM Artifact Finder**. Otra de las limitaciones de este proyecto es que únicamente se ha trabajado con sistemas Windows. Como la aplicación **Telegram Desktop** se encuentra disponible para otros sistemas operativos, se considera relevante el análisis futuro de **Telegram Desktop** en dichos sistemas. Por último, se pretende que el *framework* elaborado sea extendido en el futuro para soportar aplicaciones de mensajería instantánea adicionales a **Telegram Desktop**.

Bibliografía

- [1] *Telegram*, [Online: <https://telegram.org/>], accedido el 1 de junio de 2021, 2021.
- [2] *Plataformas de mensajería instantánea más populares a nivel mundial*, [Online: <https://www.statista.com/statistics/258749/most-popular-global-mobile-messenger-apps/>], accedido el 1 de junio de 2021, abril de 2021.
- [3] *Cuota de mercado de los sistemas operativos para ordenador a nivel mundial en el primer semestre de 2020*, [Online: <https://es.statista.com/estadisticas/576870/cuota-de-mercado-mundial-de-los-sistemas-operativos/>], accedido el 1 de junio de 2021, julio de 2020.
- [4] *Evolución de la cuota de mercado de las versiones del sistema Windows*, [Online: <https://www.statista.com/statistics/993868/worldwide-windows-operating-system-market-share/>], accedido el 1 de junio de 2021, abril de 2021.
- [5] *Versiones existentes de Telegram Desktop*, [Online: <https://github.com/telegramdesktop/tdesktop/releases>], accedido el 1 de junio de 2021, 2021.
- [6] *Repositorio en GitHub de la herramienta Windows Memory Extractor*, [Online: <https://github.com/pedrofdez26/windows-memory-extractor>], accedido el 30 de junio de 2021, 2021.
- [7] *Repositorio en GitHub de la herramienta Instant Messaging Artifact Finder*, [Online: <https://github.com/pedrofdez26/instant-messaging-artifact-finder>], accedido el 30 de junio de 2021, 2021.
- [8] *Documentación de Telegram*, [Online: <https://telegram.org/faq>], accedido el 2 de junio de 2021, 2021.
- [9] *Política de privacidad de la plataforma Telegram*, [Online: <https://telegram.org/privacy>], accedido el 2 de junio de 2021, agosto de 2018.
- [10] *Telegram Desktop*, [Online: <https://desktop.telegram.org/>], accedido el 2 de junio de 2021, 2021.

- [11] *Telegram para macOS*, [Online: <https://macos.telegram.org/>], accedido el 2 de junio de 2021, 2021.
- [12] M. H. Ligh, A. Case, J. Levy y A. Walters, *The Art of Memory Forensics: Detecting Malware and Threats in Windows, Linux, and Mac Memory*, 1.^a ed. Wiley Publishing, 2014, cap. 1, ISBN: 1118825098.
- [13] Microsoft Docs, *Dynamic-Link Libraries*, [Online: <https://docs.microsoft.com/en-us/windows/win32/dlls/dynamic-link-libraries>], accedido el 3 de junio de 2021, mayo de 2018.
- [14] G. B. Satrya, P. T. Daely y S. Y. Shin, «Android forensics analysis: Private chat on social messenger,» en *2016 Eighth International Conference on Ubiquitous and Future Networks (ICUFN)*, 2016, págs. 430-435.
- [15] G. B. Satrya, P. T. Daely y M. A. Nugroho, «Digital forensic analysis of Telegram Messenger on Android devices,» en *2016 International Conference on Information & Communication Technology and Systems (ICTS)*, 2016, págs. 1-7.
- [16] C. Anglano, M. Canonico y M. Guazzone, «Forensic analysis of Telegram Messenger on Android smartphones,» *Digital Investigation*, vol. 23, págs. 31-49, 2017, ISSN: 1742-2876.
- [17] J. Gregorio, A. Gardel y B. Alarcos, «Forensic analysis of Telegram Messenger for Windows Phone,» *Digital Investigation*, vol. 22, págs. 88-106, 2017, ISSN: 1742-2876.
- [18] M. Yusoff, A. Dehghantanha y R. Mahmod, «Forensic Investigation of Social Media and Instant Messaging Services in Firefox OS: Facebook, Twitter, Google+, Telegram, OpenWapp, and Line as Case Studies,» en *Contemporary Digital Forensic Investigations of Cloud and Mobile Applications*, K.-K. R. Choo y A. Dehghantanha, eds., Syngress, 2017, cap. 4, págs. 41-62, ISBN: 978-0-12-805303-4.
- [19] L. Zhang, F. Yu y Q. Ji, «The Forensic Analysis of WeChat Message,» en *2016 Sixth International Conference on Instrumentation & Measurement, Computer, Communication and Control (IMCCC)*, 2016, págs. 500-503.
- [20] S. Wu, Y. Zhang, X. Wang, X. Xiong y L. Du, «Forensic analysis of WeChat on Android smartphones,» *Digital Investigation*, vol. 21, págs. 3-10, 2017, ISSN: 1742-2876.
- [21] K. M. Ovens y G. Morison, «Forensic analysis of Kik messenger on iOS devices,» *Digital Investigation*, vol. 17, págs. 40-52, 2016, ISSN: 1742-2876.
- [22] A. Nisioti, A. Mylonas, V. Katos, P. D. Yoo y A. Chryssanthou, «You can run but you cannot hide from memory: Extracting IM evidence of Android apps,»

- en *2017 IEEE Symposium on Computers and Communications (ISCC)*, 2017, págs. 457-464.
- [23] N. Scrivens y X. Lin, «Android Digital Forensics: Data, Extraction and Analysis,» en *Proceedings of the ACM Turing 50th Celebration Conference - China*, 2017, págs. 1-10, ISBN: 9781450348737.
- [24] H. Zhang, L. Chen y Q. Liu, «Digital Forensic Analysis of Instant Messaging Applications on Android Smartphones,» en *2018 International Conference on Computing, Networking and Communications (ICNC)*, 2018, págs. 647-651.
- [25] J. Gregorio, B. Alarcos y A. Gardel, «Forensic analysis of Telegram Messenger Desktop on macOS,» *International Journal of Research in Engineering and Science*, vol. 6, n.º 8, págs. 39-48, 2018.
- [26] K. Rathi, U. Karabiyik, T. Aderibigbe y H. Chi, «Forensic analysis of encrypted instant messaging applications on Android,» en *2018 6th International Symposium on Digital Forensic and Security (ISDFS)*, 2018, págs. 1-6.
- [27] R. D. Thantilage y N. A. Le Khac, «Framework for the Retrieval of Social Media and Instant Messaging Evidence from Volatile Memory,» en *2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, 2019, págs. 476-482.
- [28] J. Choi, J. Yu, S. Hyun y H. Kim, «Digital forensic analysis of encrypted database files in instant messaging applications on Windows operating systems: Case study with KakaoTalk, NateOn and QQ messenger,» *Digital Investigation*, vol. 28, S50-S59, 2019, ISSN: 1742-2876.
- [29] D. Barradas, T. Brito, D. Duarte, N. Santos y L. Rodrigues, «Forensic analysis of communication records of messaging applications from physical memory,» *Computers & Security*, vol. 86, págs. 484-497, 2019, ISSN: 0167-4048.
- [30] A. Kazim, F. Almaeeni, S. A. Ali, F. Iqbal y K. Al-Hussaeni, «Memory Forensics: Recovering Chat Messages and Encryption Master Key,» en *2019 10th International Conference on Information and Communication Systems (ICICS)*, 2019, págs. 58-64.
- [31] A. M. Al-Rawashdeh, Z. A. Al-Sharif, M. I. Al-Saleh y A. S. Shatnawi, «A Post-Mortem Forensic Approach for the Kik Messenger on Android,» en *2020 11th International Conference on Information and Communication Systems (ICICS)*, 2020, págs. 079-084.
- [32] *Repositorio de Telegram Desktop en Github*, [Online: <https://github.com/telegramdesktop/tdesktop>], accedido el 7 de junio de 2021, 2021.

- [33] *Código fuente de la versión 2.7.1 de Telegram Desktop*, [Online: <https://github.com/telegramdesktop/tdesktop/releases/tag/v2.7.1>], accedido el 7 de junio de 2021, marzo de 2021.
- [34] *Qt*, [Online: <https://www.qt.io/>], accedido el 7 de junio de 2021, 2021.
- [35] *Visual Paradigm*, [Online: <https://www.visual-paradigm.com/>], accedido el 7 de junio de 2021, 2021.
- [36] *StarUML*, [Online: <https://staruml.io/>], accedido el 7 de junio de 2021, 2021.
- [37] *BOUML*, [Online: <https://www.bouml.fr/>], accedido el 7 de junio de 2021, 2021.
- [38] *Enlace a la carpeta compartida*, [Online: <https://drive.google.com/drive/folders/1KL9GWzt0IlnF1cVHrnCc9tvU4jKvF3TS?usp=sharing>], accedido el 30 de junio de 2021.
- [39] *Adobe Acrobat Reader DC*, [Online: <https://get.adobe.com/reader/>], accedido el 7 de junio de 2021, 2021.
- [40] Microsoft Docs, *ProcDump*, [Online: <https://docs.microsoft.com/en-us/sysinternals/downloads/procdump>], accedido el 9 de junio de 2021, septiembre de 2020.
- [41] Microsoft Docs, *WinDbg*, [Online: <https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/debugger-download-tools>], accedido el 9 de junio de 2021, mayo de 2020.
- [42] *Process Hacker*, [Online: <https://processhacker.sourceforge.io/>], accedido el 9 de junio de 2021, 2021.
- [43] Microsoft Docs, *Protecciones de memoria en Windows*, [Online: <https://docs.microsoft.com/en-us/windows/win32/memory/memory-protection-constants>], accedido el 9 de junio de 2021, julio de 2020.
- [44] *PyPI*, [Online: <https://pypi.org/>], accedido el 11 de junio de 2021, 2021.
- [45] E. Gamma, R. Helm, R. Johnson y J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. USA: Addison-Wesley Longman Publishing Co., Inc., 1995, ISBN: 0201633612.
- [46] *Clase QString*, [Online: <https://doc.qt.io/qt-5/qstring.html>], accedido el 14 de junio de 2021, 2021.

Anexo A

Horas Invertidas

La realización de este trabajo se ha dividido en varias etapas, descritas a continuación:

- **Etapa 1:** Adquisición de los conocimientos requeridos para llevar a cabo el trabajo.
- **Etapa 2:** Análisis del código fuente de `Telegram Desktop` y elaboración de diagramas.
- **Etapa 3:** Tras conocer la manera en la que se estructura la información relativa a `Telegram Desktop` en memoria RAM, se han definido los casos de prueba que se desean realizar, previamente a la elaboración de las herramientas que componen el entorno de análisis.
- **Etapa 4:** Realizar la arquitectura del entorno de análisis y el diseño de la herramienta `IM Artifact Finder`.
- **Etapa 5:** Elaboración de la herramienta `Windows Memory Extractor`.
- **Etapa 6:** Implementación de la herramienta `IM Artifact Finder` incluyendo las clases concretas relativas a `Telegram Desktop`.
- **Etapa 7:** Evaluación del entorno de análisis desarrollado. Esta etapa ha sido transversal a la elaboración de las dos herramientas que componen el entorno.
- **Etapa 8:** Escritura del presente documento.

En la Tabla A.1 se aprecian las horas que han sido invertidas en cada una de las etapas definidas anteriormente. La suma de las horas invertidas en cada etapa, junto con las 15 horas que han sido empleadas en reuniones, proporciona un tiempo total

invertido de 768 horas. Adicionalmente, se muestra en la Figura A.1 el diagrama de Gantt del proyecto.

Tabla A.1: Horas invertidas en cada etapa del trabajo

| Etapas | Horas |
|---------|-------|
| Etapa 1 | 95 |
| Etapa 2 | 135 |
| Etapa 3 | 25 |
| Etapa 4 | 55 |
| Etapa 5 | 90 |
| Etapa 6 | 185 |
| Etapa 7 | 40 |
| Etapa 8 | 128 |



Figura A.1: Diagrama de Gantt