International Journal of Research in Engineering and Science (IJRES)
ISSN (Online): 2320-9364, ISSN (Print): 2320-9356
www.ijres.org Volume 6 Issue 8 Ver. I ǁ 2018 ǁ PP. 39-48

Forensic analysis of Telegram Messenger Desktop on MacOS

J. Gregorio[1], B. Alarcos[2], A. Gardel[3]

[*1]*Instituto Universitario de Ciencias Policiales, University of Alcala, Alcala de Henares, Madrid, España*
[2] *Instituto Universitario de Ciencias Policiales, University of Alcala, Alcala de Henares, Madrid, España*
[3] *Instituto Universitario de Ciencias Policiales, University of Alcala, Alcala de Henares, Madrid, España*
Corresponding Author: J. Gregorio

Abstract
This paper proposes a forensic analysis methodology to better evaluate the information that IM applications generate using a desktop version. The methodology has been applied to a particular IM Desktop application "Telegram Messenger" for MacOS retrieving information valuable in a forensic digital analysis. Our evaluation presents the results obtained from a combined analysis using different techniques proposed as a forensic methodology. Additionally, a specific forensic procedure is shown for IM applications installed as desktop suites concerning the cloud remote communication services. The forensic analysis is focused on relevant data extracted from the IM application and related to the user communications that might be interesting in a forensic analysis pursuing the acquisition of digital evidences concerning the commission of criminal acts.

Keywords:*Forensic analysis methodology, Instant Messaging applications, Telegram Messenger, Desktop suites, MacOS.*

Date of Submission: 07-09-2018 Date of acceptance: 24-09-2018

I. INTRODUCTION

The popularity of the Instant Messaging (IM) applications has led to their massive use as a communication system to share multiple data among people (text, images, files, location, etc.) using private/public or individual/group chats. In turn, this new mean of communication provides the criminals with another instrument to commit criminal acts such as cyberbullying, grooming, share pedophile content, insults, threats, coercion, terrorist propaganda, etc.

In the different platforms for smartphone distribution of application such as Apple Store, Google Play, or Windows Store, a large number of IM applications can be founded for example Line, Signal, Tango, Telegram, WickrMe, ooVoo, Viber, Threema, WhatsApp, etc. to name a few [1,2] available for the different mobile OS (iOS, Android, BlackBerry OS, Windows Phone, etc.) [3], [4], [5]. Currently, many of these IM applications gives the user with different possibilities to access the information from smartphones (mobile-client), computer (desktop-client) or web browser (web-client) as the information is usually stored in the cloud.

Additionally, these IM applications might store the information about the application and user data encrypted in the database of the end-user device or even not store any information locally but in the cloud. Some of these IM applications encrypt both the data and communications of the user [6], [7]. Regarding the location of the user data, many of these applications store the information in their own servers rather than using the user device.

This paper presents a methodology for the forensic analysis so the technician can have a detailed view, both functional and forensic views, of the IM applications in order to do a complete in-depth analysis of them. Following the steps shown in the paper the forensic analyst can do a detailed study to know the real information that IM applications gather such as user data or application data, answering relevant questions that arise when

doing the analysis: where are the user data?, why do we not find any messages?, where are the multimedia files?, etc.

In particular, we have focused our paper in the forensic analysis methodology applied to the IM "Telegram Messenger" installed as a desktop application on a MacOS operating system. This is a clear example of IM applications that encrypts the information in the local device and stores user data on the cloud. It is worth noting that at the current time multiple forensic tools such as X-Ways [8], Forensics Explorer [9] or EnCase [10] do not provide support for the new file system of Apple called "APFS" [11]. Other forensic tools are specialized in the artifacts processing (e.g. Belkasoft Evidence Center [12] or Internet Evidence Finder [13]), but they do not support the analysis of the IM desktop application of "Telegram Messenger".

II. RELATED WORK

Concerning the forensic analysis of IM applications, many of the research papers focus their study on the artifacts generated locally by the IM application. Nowadays the forensic analyst should enlarge the study not only processing the static artifacts, because many of the new IM applications make an intensive use of cloud services, storing the information in remote servers not related with the end-user. This is an important reason to search for a new kind of analysis methodologies to provide the technician with all the possible information to do a right and complete forensic analysis.

In the study done by Amine et al. [14], the authors propose a taxonomy for cloud-based applications in particular for Android mobile applications taking into consideration which kind of information is stored in the mobile device. Other relevant work is the paper from Aminnezhad et al. [15] that enumerates the difficulties that a cloud-based forensic analysis faces not only related to technical issues but also with legal matters.

The study from Yusoff et al. [16] analyses the artifacts stored by different IM applications into a mobile device. Its analysis is focused in the artifacts that are generated after accessing the communication system from a web browser. In this case, they provide relevant information about the registers used by the IM "Telegram Messenger" application on a mobile device and web frontend. Similarly, the paper published by Sgaras et al. [17], focuses its analysis in the artifacts study generated from the "Tango" application used on a mobile device, comparing the results obtained with other IM applications. The latter two papers only process the information from a static analysis point of view on two different IM applications. In none of them there are references explaining the analysis methodology that might be used to allow the verification of the obtained information, only present several tests and results on the digital evidences. It is impossible to know how the software will act for other different data. If there is open-source code available, the forensic analysis could take advantage to know better the behavior of the IM application under analysis.

In relation to the collection of data located in the cloud, there are several articles Brunty et al. [18], Huber et al. [19] and Taylor et al. [20] in which their authors carry out a study of different social media networks (e.g. Facebook, etc.) and methods of collecting public information from users using web crawling techniques. Similarly, other articles Ameer et al. [21] and Keyun et al. [22] describe the organization of the data in cloud environments, although no detailed analysis is carried out with respect to the forensic data analysis. In all these papers there are no particular references to the steps that a forensic analyst should do when dealing with IM applications installed on a desktop computer.

Therefore, it is required to have a methodology of forensic analysis based on the combination of different sources of information, that may be used as a way of validate the integrity of the recovered information and provide the technician with added knowledge about the behavior of the application and different artifacts generated. Our proposal is not only focused in processing the data artifacts but provides a complete analysis with a different set of tools such as open knowledgeanalysis, analysis of artifacts (static / dynamic) and analysis of different relevant fragments of source code when available.

III. METHODOLOGY FOR FORENSIC ANALYSIS

The methodology is a combination of 3 different techniques which provide a more complete information and knowledge about how the IM application gathers and process data.

1. Open knowledge: This type of technique relies on any information or documentation available about the IM application such as technical studies, books, related blogs, developer docs, etc. In many cases, the webpage from the developer/manufacturer provides some relevant information [23].

2. Analysis of artifacts: This technique studies the different evidences generated and stored in a digital device. The analysis of these digital traces can be done following different alternatives:

a. Static analysis:The static forensic analysis of artifacts is equivalent to the traditional analysis method, retrieving all the information and traces that the application has stored in a digital evidence such as user's data, registers log, app records, event log, etc. This kind of analysis is based on a forensic image cloned from the digital evidence processed with specific forensic tools to retrieve the valuable information [24], [25].

b. Dynamic analysis: The dynamic forensic analysis is based on the information generated once the digital evidence is executed. In this way, a live analysis of information could be done. This dynamic analysis is called triage. There are 3 different scenarios to do the triage:

   i. Execution of the forensic cloned image:In this scenario, the technician makes use of all the required files and data from the collected digital evidence such as user data files, configuration files, application files, etc. required to execute the IM application. These data and files are obtained from the previous static artifacts analysis. Thus, the IM application is executed on a forensic environment under control, faking the IM application as it were the real end-user executing the original IM application.

   ii. Emulated with virtual machines:In this case, not only the data for the IM application is used but the whole operating system executing it on a virtual machine from the cloned version. Therefore, the original system can be emulated being able to display the live information. There are different forensic tools, commercial and free ones, that are ready to be used by a forensic analyst to virtualize the cloned image such as the ones given in [26] and [27].

   iii. Cloned in original machine:One last option could be the use of the cloned forensic image into the same computer physical device. Sometimes, due to some hardware encryption techniques or complex datafile systems, the original device with a replica of the data must be executed, otherwise the OS refuse to run. Making use of this last resource, the analyst can obtain the results of how it would be the execution of the IM application in the original device, being able to do a live forensic analysis.

3. Source code: In this type of analysis, the forensic technician takes advantage of the source code knowledge. The source code (sometimes fragments of source code) might be provided to the community by the developer based on the different licenses used in it and ultimately by means of inverse engineering [28], [29].

There is a benefit while reviewing the digital evidence for a particular IM application using these techniques as they provide more information about the acquired data, some way of validation and verifying the correctness of the results from different points of view. Current IM applications might have data stored in the cloud, pending requests, etc. so it requires that a dynamic analysis be done.

In next section, we present how to obtain the information and data generated by the IM desktop application of "Telegram Messenger" on a MacOS system, mixing all the three mentioned analysis techniques.

IV. METHODOLOGY FOR FORENSIC ANALYSIS

The following subsections describes the forensic analysis of "Telegram Messenger Desktop" for MacOS done using the techniques in the proposed methodology.

4.1 OPEN KNOWLEDGE ANALYSIS

The open knowledge analysis tries to gather all the relevant information about the IM application, in our case "Telegram Messenger Desktop", using in most of times different open source code information. The search for information should be focused on identifying any piece of documentation that may describe the operation of the application as it is given by the data structure, files, etc. In this particular case, we will focus our work on MacOS.

The official website for the IM Telegram [30] gives anyone a large quantity of information, how data is stored, a schema for data structure, API library, source code, etc. However, there is no detailed information about the management of user data when using Telegram for MacOS.

In the other hand, other websites have relevant information. In the "Unofficial Telegram Wiki" [31], it has been found small pieces of information related with the folder structure that the application generates on MacOS. Another source of information founded was a technical document [32], where an in-depth study about the different artifacts that the Telegram generates in Windows 10.

4.2 ARTIFACTS ANALYSIS.

This section deals with the artifact analysis specific for desktop applications such as encryption data (due to the more powerful capabilities of the desktop computers respect the mobile devices) and data location for user and application information. In these cases, a static forensic analysis provides the technician with few relevant information about the user data being necessary to combine the analysis with a dynamic triage to retrieve user data and information.

4.2.1 Analysis of artifacts. Static analysis.

In the next subsections, we show the results given by the static forensic analysis done processing the artifacts from the desktop application IM "Telegram Messenger" on a "MacOS High Sierra". Due to the features of the new APFS file system, two different forensic suites have been chosen in order to do the static analysis. We have used as analysis tools the native forensic software from the MacOS environment, since, at the time of this study, current forensic tools do not provide the same support for the new "APFS" file system.

4.2.1.1 Static forensic analysis of artifacts using native software from MacOS.

In this case, the static forensic analysis is done via the use of different native tools that are already installed on MacOS (Finder.app, Terminal.app, DiskImageMounter.app, etc.). The forensic image cloned in a RAW format (file extension ".dmg"), is mounted/added in the forensic computer device as an external device being able to access its contents.

Table 1 enumerates the different artifacts obtained after the application of a static forensic analysis of the IM Telegram such as application, log data, user data, temporal configuration files, multimedia files and socket connection.

Table 1: "Telegram Messenger Desktop" artifacts on "MacOS High Sierra".

Row#	Content	File Name (with extension)	Folder	Description
1	Application	Telegram.app	/Applications/	Application data.
2	Log data	Log.txt	/Users/${user}/Library/Application Support/Telegram Desktop/	Events record.
3	User data	Various files.	/Users/${user}/Library/Application Support/Telegram Desktop/tdata	Encrypted data.
4	Temporal configuration files	data.data, windows.plist, window_1.data	/Users/${user}/Library/Saved Application State/com.tdesktop.Telegram.savedState	Temporal configuration files.
5	Multimedia files	Various files.	/Users/${user}/Downloads (Default)	Multimedia download files.
6	Socket connection	7852aa807d0e61276974ee878396a1c4-{87A94AB0-E370-4cde-98D3-ACC110C5967D}...	/tmp/	Local socket. Regular file

Fig. 1 shows an example content of the "tdata" folder, where user data are stored. The "tdata" folder contains multiple relevant information such as for example the data contained in the files "usertag", "settings1" and "D877F783D5D3EF8C1". Additionally, a subfolder of "tdata" with a similar name than the last recalled

filename except for the last digit, contains other relevant files with an alphanumeric random name composed of 17 characters, and also a file "map1". From the source code analysis, we know the data structure of these files but the information is encrypted.

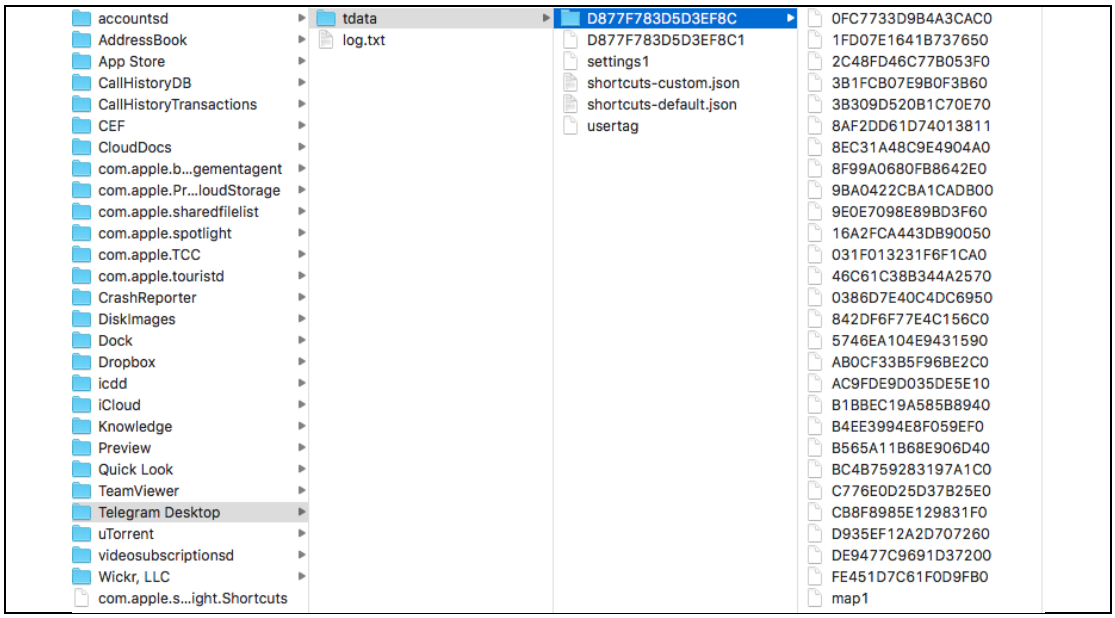


Fig. 1. Content sample for the "Telegram Desktop" folder tdata and subdirectories.

These alphanumeric files (right text column of Fig. 1), stored under the subfolder "D877F783D5D3EF8C", have a correspondence with the messages of the different chats that are created when the user of the application "reads" the first message of a chat.

Analyzing the content of these files, one can observe that they have the same header with a hex data value equal to "0x54444624" ("TDF$" in ASCII). The rest of the content is encrypted. Figure Fig. 2 shows an example of the content of the header extracted from one of these user data files.

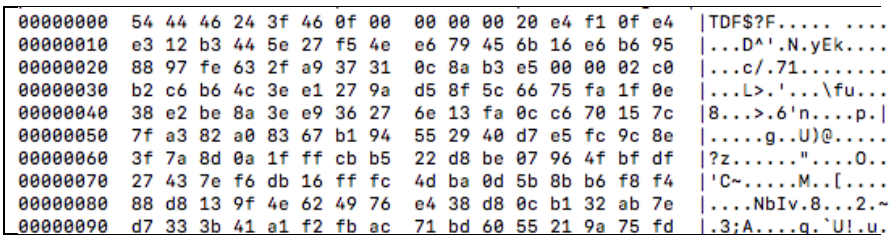


Fig. 2. File header of "settings1".

An in-depth explanation about the meaning of the data contained in all these files is out of the scope of our paper. In fact, the data information obtained after the forensic static analysis of those files for the desktop application does not provide any relevant data about the user and his/her messages.

As we will show later, from the source code knowledge of the application the analyst can extract the required information to validate and understand the data obtained from the analysis. For example, the content of the user datafiles is encrypted data using a private password (shown from the source code file localstorage.cpp;) and the name of the folder corresponds with the MD5 hash restricted to 16 first characters (shown from the source code file utils.cpp).

4.2.2 Analysis of artifacts. Dynamic analysis.

This section describes three different techniques using a "triage" or live analysis of the data generated when the application is executed. The analyst have to execute the IM "Telegram Messenger" application in

order to connect to the servers and retrieve as much relevant data as possible. Section 5.2 presents real examples of this triage.

4.2.2.1 Forensic copy of relevant data.

In this case, the analyst must copy the application and user data obtained in the before static forensic analysis and make the most of them inside a controlled forensic MacOS environment.

For the IM Telegram it is required to use the configuration and data stored by application in "/Applications/Telegram.app" and the user data stored in the folder "/Users/${user}/Library/Application Support/Telegram Desktop/tdata/". These original files must be copied into the controlled forensic environment, emulating the installation and configuration of the application as it were done by the real user. There is no need to obtain the forensic image of the digital evidence.

4.2.2.2 Emulated with virtual machines.

In this type of dynamic analysis of artifacts, the original MacOS forensic image is virtualized. In this way, the analyst can do a live analysis or "triage" of the virtual machine as it were the original computer device. Nowadays the virtualization of an OS image obtained from the forensic cloned copy is a quick option to setup and test because there are many forensics tools such as (Forensics Explorer, OpenVL, Perlustro, etc.) that already provides the capability to emulate the system with the help of standard virtualization software (vmWare, VirtualBox, Parallels, etc.).

It should be noted that the analyst might find certain incompatibilities with the generic hardware that uses the virtualization software or other specific problems to access the system due to passwords to access or to decrypt information, even the existence of a requirement of the original computer to run the system. In these cases, it could be of help for the forensic analysis to mount the cloned copy into the original computer device. This is the third technique described in the next subsection.

4.2.2.3 Cloned image executed in the original machine.

This last scenario makes use of the cloned image but installed into the original computer equipment- In this particular case, the forensic technician uses the original computer device as a work tool plus the image copy of the digital evidence to avoid any hardware issues while preserving the original data and original device. With this technique, the analyst solves any problem with some specific hardware required to access the data or any other condition that makes useless to emulate the OS in another computer device.

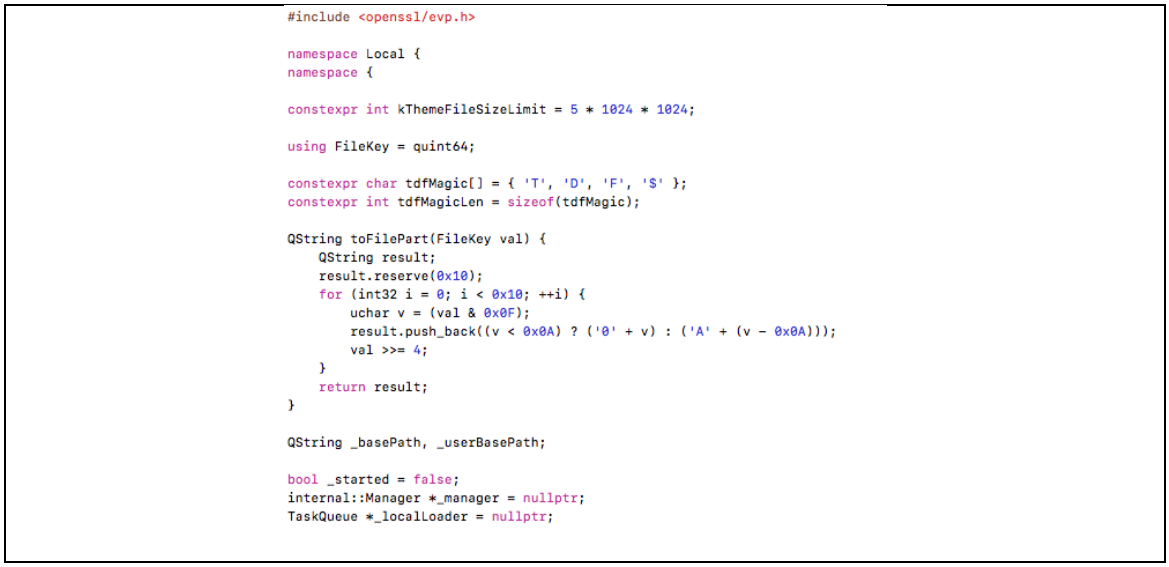
In section 5, we present a practical example about how to obtain the user data for the IM "Telegram Messenger" desktop application on MacOS mixing the static analysis and dynamic analysis of all the artifacts stored in the cloned forensic image, plus the particular features that this information retrieval involves.

4.3 SOURCE CODE ANALYSIS.

Finally, the technician might require the information obtained from the analysis of the source code of the IM application, focusing specifically in those, which can provide relevant information for the analysis. The developer website gives some details and several source code files for different OS installations. The source code for the "Telegram Messenger" desktop application on MacOS is coded in the "C/C++" programming language. The analysis of some fragments or lines extracted from the source code should be focused mainly in those files that might contain some definitions or functions that gives some clue or information about the how the application manages the user data files.

In particular, we have parsed the source code file with name "localstorage.cpp" that can be accessed in the folder "tdesktop-dev\tdesktop-dev\Telegram\SourceFiles\storage\" once the source code is downloaded, which contains different functions, for example those functions that controls the reading and writing operations of user data files.

Fig. 3 shows a fragment of the source code given in the "localstorage.cpp" file in which the variable "tdfMagic" it can be shown together with its corresponding value "TDF$" (hexadecimal header "0x54444624"). This value has been previously identified in the header of the different data files located in the "tdata" folder of the IM application.



```
#include <openssl/evp.h>

namespace Local {
namespace {

constexpr int kThemeFileSizeLimit = 5 * 1024 * 1024;

using FileKey = quint64;

constexpr char tdfMagic[] = { 'T', 'D', 'F', '$' };
constexpr int tdfMagicLen = sizeof(tdfMagic);

QString toFilePart(FileKey val) {
    QString result;
    result.reserve(0x10);
    for (int32 i = 0; i < 0x10; ++i) {
        uchar v = (val & 0x0F);
        result.push_back((v < 0x0A) ? ('0' + v) : ('A' + (v - 0x0A)));
        val >>= 4;
    }
    return result;
}

QString _basePath, _userBasePath;

bool _started = false;
internal::Manager *_manager = nullptr;
TaskQueue *_localLoader = nullptr;
```

Fig. 3. Variable "tdfMagic". Source code file: "localstorage.cpp"

Additionally, Fig. 4 shows a fragment of the source code given in the "localstorage.cpp" file with the following relevant functions: "writeData", "writeEncrypted" and "prepareEncrypted". These functions are responsible of writing the information encrypted in the data files of the application.

```
bool writeData(const QByteArray &data) {
    if (!file.isOpen()) return false;

    stream << data;
    quint32 len = data.isNull() ? 0xffffffff : data.size();
    if (QSysInfo::ByteOrder != QSysInfo::BigEndian) {
        len = qbswap(len);
    }
    md5.feed(&len, sizeof(len));
    md5.feed(data.constData(), data.size());
    dataSize += sizeof(len) + data.size();

    return true;
}
static QByteArray prepareEncrypted(EncryptedDescriptor &data, const MTP::AuthKeyPtr &key = LocalKey) {
    data.finish();
    QByteArray &toEncrypt(data.data);

    // prepare for encryption
    uint32 size = toEncrypt.size(), fullSize = size;
    if (fullSize & 0x0F) {
        fullSize += 0x10 - (fullSize & 0x0F);
        toEncrypt.resize(fullSize);
        memset_rand(toEncrypt.data() + size, fullSize - size);
    }
    *(uint32*)toEncrypt.data() = size;
    QByteArray encrypted(0x10 + fullSize, Qt::Uninitialized); // 128bit of sha1 - key128, sizeof(data), data
    hashSha1(toEncrypt.constData(), toEncrypt.size(), encrypted.data());
    MTP::aesEncryptLocal(toEncrypt.constData(), encrypted.data() + 0x10, fullSize, key, encrypted.constData());

    return encrypted;
}
bool writeEncrypted(EncryptedDescriptor &data, const MTP::AuthKeyPtr &key = LocalKey) {
    return writeData(prepareEncrypted(data, key));
}
void finish() {
    if (!file.isOpen()) return;
```

Fig. 4. Source code of functions related with the encrypted writing of data. Source code file: "localstorage.cpp"

Finally, Fig. 5 shows a fragment of the same source code file "localstorage.cpp" in which the source code of the function "createLocalKey" can be analyzed. Doing this analysis, we have concluded that this source code manages the creation of local passwords (variable localKey) that later will be used in the encryption of the data files, using a key derivation function [33].

```
auto OldKey = MTP::AuthKeyPtr();
auto SettingsKey = MTP::AuthKeyPtr();
auto PassKey = MTP::AuthKeyPtr();
auto LocalKey = MTP::AuthKeyPtr();

void createLocalKey(const QByteArray &pass, QByteArray *salt, MTP::AuthKeyPtr *result) {
    auto key = MTP::AuthKey::Data { { gsl::byte{} } };
    auto iterCount = pass.size() ? LocalEncryptIterCount : LocalEncryptNoPwdIterCount; // dont slow down for no password
    auto newSalt = QByteArray();
    if (!salt) {
        newSalt.resize(LocalEncryptSaltSize);
        memset_rand(newSalt.data(), newSalt.size());
        salt = &newSalt;

        cSetLocalSalt(newSalt);
    }

    PKCS5_PBKDF2_HMAC_SHA1(pass.constData(), pass.size(), (uchar*)salt->data(), salt->size(), iterCount, key.size(),
        (uchar*)key.data());

    *result = std::make_shared<MTP::AuthKey>(key);
}
```

Fig. 5. Management functions to manage the creation of local passwords/keys. Source code file "localstorage.cpp".

Therefore, it is clear to see that the analysis of the source code of the "Telegram Messenger Desktop" for MacOS, provides the forensic technician with a large quantity of information about the operation and usage of the application and how this application manages the user data. Despite the fact that it is a good technique to obtain more information, it does not offer any straight method to obtain the user messages content because these messages are cyphered/encrypted.

V. STATIC AND DYNAMIC ARTIFACTS ANALYSIS. COPY FORENSIC

This section shows how to apply the described techniques in the previous sections. The dynamic analysis has led to the acquisition of user data stored in the cloud. The static analysis has obtained the user and application information and knowing this information we have been able to obtain the user messages using the IM desktop application inside a controlled forensic environment.

5.1 STATIC ARTIFACTS ANALYSIS. OBTAINING DATA FROM FORENSICS IMAGE.

This step of the forensic analysis process the static artifacts from a forensic image to obtain the required information to do a forensic clone of the relevant data located in the folder "/Applications/Telegram.app", and also clone the user data stored in the folder "/Users/{USER}/Library/Application Support/Telegram Desktop/".

In this particular case, the static analysis of artifacts has been done from the data generated on an IM "Telegram Messenger" desktop application (v1.1.23) running a "MacOS High Sierra" (v12.13) on a "MacBook Pro" hardware. We have followed the next steps:

- Clone the forensic image. We have used a liveCD with the "Zero Live" DEFT Toolkit [34], taking into consideration that the "FileVault" encryption system should not be active.
- Process the artifacts stored in the forensic image. In our particular case, because this new MacOS has the "APFS" new filesystem we have used as forensic tools the standard tools to read and search for data currently available in this new MacOS environment. Thus, the forensic image has been read using the native tools DiskImageMounter.app and Finder.app.
- Copy the relevant datafiles for later use in the dynamic analysis.

Figures Fig. 6 and Fig. 7 show the execution of the native tool DiskImageMounter.app, so it is possible to mount and read the data and files contained in the forensic image "image_27_09_2017.dmg" inside a forensic environment based on MacOS. The forensic image is mounted as "Untitled".

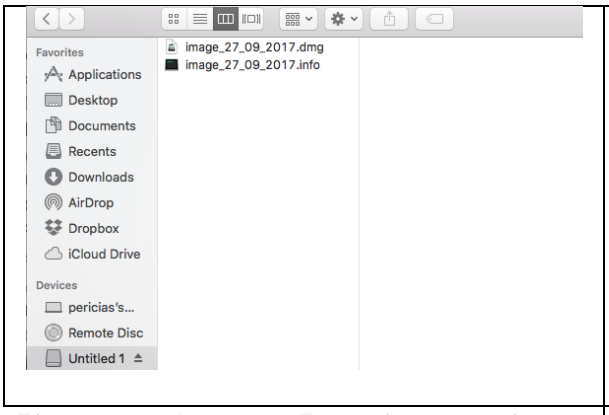


Fig. 6. Forensic image: "Image_27_09_2017.dmg" stored in the hard disk "Untitled 1".

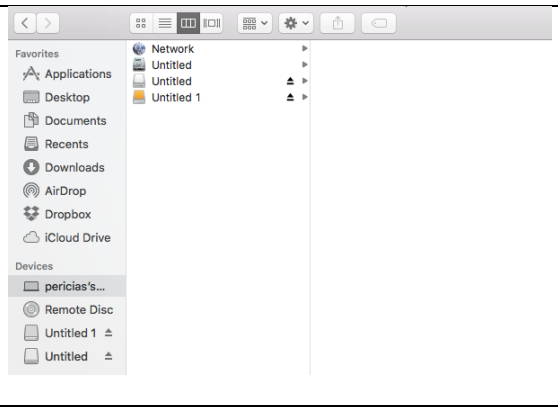


Fig. 7. External device "Untitled" mounted from the forensic image named "Image_27_09_2017.dmg".

Once mounted, the content of this new external drive "Untitled" ("image_27_09_2017.dmg") can be analyzed by different tools, for example the native tools of MacOS "Finder.app" or "Terminal.app".

5.2 DYNAMIC ARTIFACTS ANALYSIS. OBTAINING USER DATA FROM STATIC ARTIFACTS ANALYSIS

This section shows the procedure to obtain cloud information of a Telegram desktop user. Legal aspects regarding the location of data and other legal issues are out of the scope of our paper. The procedure starts from the data copied in the static artifacts analysis done in the section before.

The application data contained in "Untitled/Applications/Telegram.app" ("image_27_09_2017.dmg"), is copied into our monitored forensic environment, creating the application "/Applications/Telegram.app" (forensic copy) together with the user data in the following application folder "Untitled/Users/{USER}/Library/Application Support/Telegram Desktop/" ("image_27_09_2017.dmg"), creating the folder "/Users/{FORENSIC_USER}/Library/Application Support/Telegram Desktop/" (forensic copy).

Fig. 8 shows a list of applications stored in the system folder "/Applications/" in our monitored forensic environment where it is shown the cloned application "Telegram.app" (data from the forensic image).

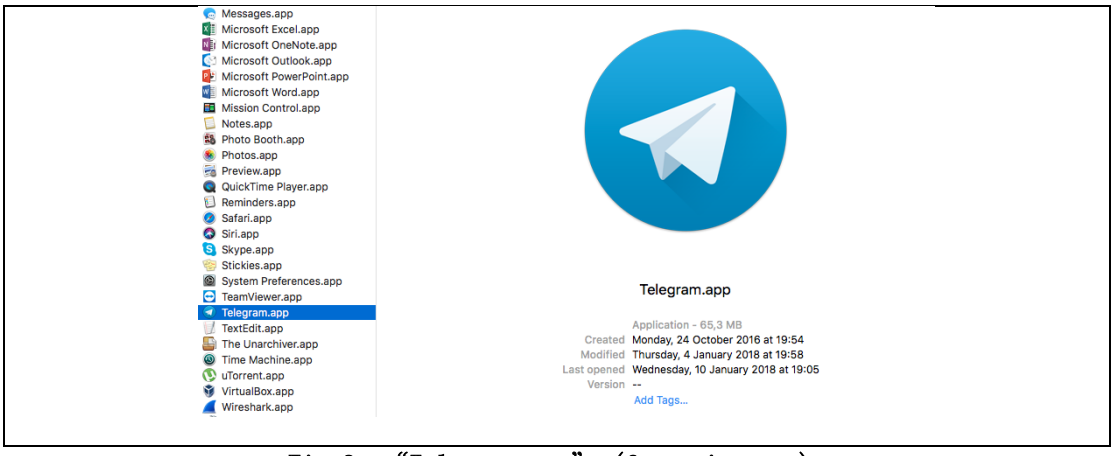


Fig. 8. "Telegram.app" (forensic copy).

With the user data stored in the folder "Telegram Desktop", now it is possible to execute the forensic copy application "/Applications/Telegram.app", which will show the graphical user interface so the analyst has access to the chat information and messages of the user.

Once enabled internet connection to the forensic environment the forensic copy application "/Applications/Telegram.app" retrieves the user messages from the cloud to be displayed in the local computer. Fig. 9 shows the user messages (the contents are intentionally blanked). The analyst obtains not only the user data but any message sent until this execution instant.

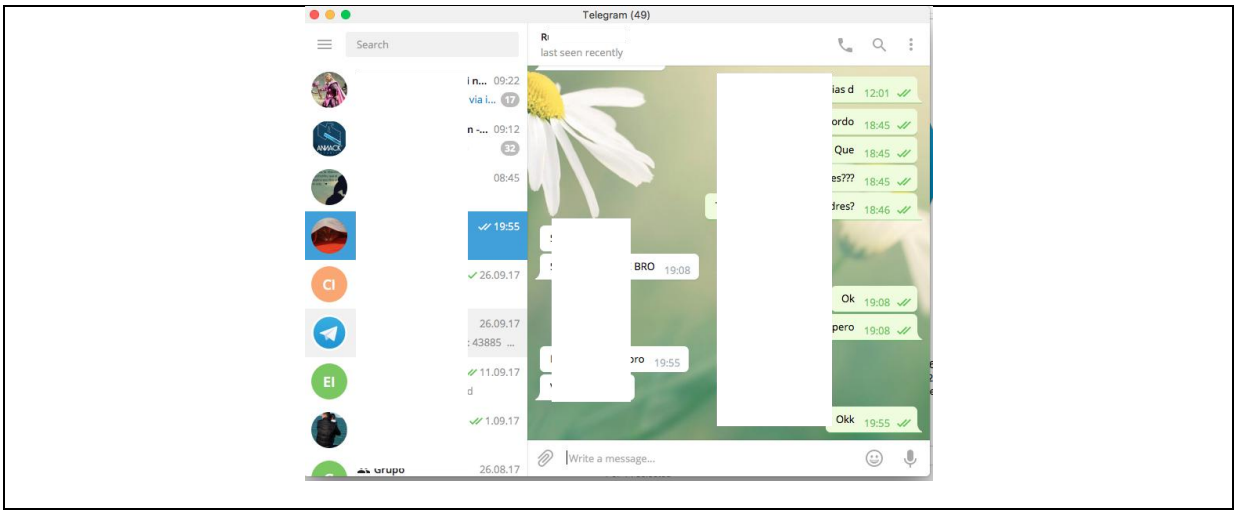


Fig. 9. Cloud data from the cloned IM application in the forensic environment "/Applications/Telegram.app" (forensic copy).

VI. CONCLUSION

After carrying out the different types of analysis presented in the paper, as current IM applications for their desktop versions include encryption systems the forensic analysis is many times impractical. Additionally, several IM applications do not store user's data locally but in the cloud. The proposed methodology for the IM forensic analysis provides a combination of different techniques thus validating the integrity of the data obtained. It also provides the specialist with a general and forensic knowledge on how and what data can be obtained from a particular IM application.

We have reached to the conclusion that for the forensic analysis of the desktop versions of IM applications, the specialist must carry out a preliminary study of the application through the open knowledge and source code to obtain information regarding its operation. Afterwards, a static and dynamic analysis of artifacts must be performed, validating the information obtained from the different data sources.

Although, as the paper have presented, certain characteristics of IM desktop applications require to access the cloud data to obtain the information related to the user. Therefore, the specialist needs to keep in mind both the chain of custody of the digital evidence and the information, preserving at any time that the data is obtained with the appropriate guarantees, documenting the operations carried out and maintaining the inalterability of the information.

VII.REFERENCES

[1]. "Number of mobile phone messaging app users worldwide from 2016 to 2021 (in billions)". Available at, https://www.statista.com/statistics/483255/number-of-mobile-messaging-users-worldwide/.
[2]. "Most popular mobile messaging apps worldwide as of January 2017" Available at, https://www.statista.com/statistics/258749/most-popular-global-mobile-messenger-apps/.
[3]. "Communication App in Google Play". Available at, https://play.google.com/store/apps/category/COMMUNICATION/collection/topselling_free.
[4]. "Social App in Apple Store". Available at, https://itunes.apple.com/us/genre/mac-social-networking/id12016?mt=12.

[5]. "Social App in Windows Store". Available at, https://www.microsoft.com/en-us/store/top-free/apps/mobile?target=apps..social.
[6]. Warren G. Kruse II, Jay G. Heiser. "Computer Forensics: Incident Response Essentials". Pearson Education, 26 September. 2001, Pages 416.
[7]. NedaaB.. Al Barghuthi 1 and Huwida Said. "Social Networks IM Forensics: Encryption Analysis". Journal of Communications Vol.8, No. 11, November 2013.
[8]. X-Ways Software Technology AG. X-Ways Forensics: Integrated Computer Forensics Software. Available at, http://x-ways.net/forensics/index-m.html; 2017.
[9]. OpenText. Encase Forensics. Available at, https://www.guidancesoftware.com/encase-forensic?cmpid=nav_r; 2017.
[10]. GetData Software Company. Forensic Explorer. Available at, http://www.forensicexplorer.com/; 2017.
[11]. Apple File System Guide. Available at, https://developer.apple.com/library/content/documentation/FileManagement/Conceptual/APFS_Guide/Features/Features.html; 2017.
[12]. Oxygen Forensics, Inc. Oxygen Forensics. Available at, http://www.ogygen-forensics.com/en/; 2017.
[13]. Magnet Forensics, Inc. Mobile Forensics. Available at, https://www.magnetforensics.com/mobile-forensics/; 2017.
[14]. M. Amine Chelihi, A. Elutilo, I. Ahmed, C. Papadopoulos, A. Dehghantanha, "An Android Cloud Storage Apps Forensic Taxonomy", Pages 285-305, Chapter 15, (Elsevier) Contemporary Digital Forensic Investigations Of Cloud And Mobile Applications, 2017.
[15]. Aminnezhad, Asou&Dehghantanha, Ali & Abdullah, MohdTaufik&Damshenas, Mohsen. (2013). "Cloud Forensics Issues and Opportunities. International Journal of Information Processing and Management". 4. 76-85. 10.4156/ijipm.vol4.issue4.9.
[16]. M. N. Yusoff, A. Dehghantanha, R. Mahmod. "Forensic Investigation of Social Media and Instant Messaging Services in Firefox OS: Facebook, Twitter, Google++, Telegram, OpenWapp and Line as Case Studies". Contemporary Digital Forensic Investigations of Cloud and Mobile Applications, Chapter: 4, Publisher: Elsevier, Pages.41-62.
[17]. Sgaras, Christos&Kechadi, Tahar& Le-Khac, Nhien-An. "Forensics Acquisition and Analysis of Instant Messaging and VoIP Applications". Conference: 6th IAPR International Workshop on Computational Forensics (IWCF 2014), At Stockhom, Sweeden, Volume: LNCS 8915. 10.1007/978-3-319-20125-2_16.
[18]. Brunty, J., Miller, L., Helenek, K., 2014. "Social media investigation for law enforcement". Routledge.
[19]. Huber, M., Mulazzani, M., Leithner, M., Schrittwieser, S., Wondracek, G., Weippl, E., "Social snapshots: Digital forensics for online social networks". In: Proceedings of the 27th annual computer security applications conference. ACM, 2011, Pages 113–122.
[20]. Taylor, M., Haggerty, J., Gresty, D., Almond, P., Berry, T., "Forensic investigation of social networking applications". Network Security 2014 (11), Pages 9–16.
[21]. Ameer Pichan, Mihai Lazarescu, SieTengSoh, Cloud forensics: Technical challenges, solutions and comparative analysis, Digital Investigation, Volume 13, 2015, Pages 38-57.
[22]. KeyunRuan. "Cybercrime and Cloud Forensics: Applications for Investigation Processes". University College Dublin, Ireland, December 2012, Pages 348, ISBN13: 9781466626621.
[23]. Telegram Messenger developer web page. Available at, https://core.telegram.org/mtproto.
[24]. ISO/IEC 27037:2015. "Information technology -- Security techniques -- Guidelines for identification, collection, acquisition, and preservation of digital evidence", 15 November 2012.
[25]. ENFSI. "ENFSI-BPM-FIT-01. Best Practice Manual for the Forensic Examination of Digital Technology". Available at, "http://www.enfsi.eu/sites/default/files/documents/enfsi-bpm-fit-01_1.pdf". Version 1 – November 2015.
[26]. Brett Shavers. "Virtual Forensics. A Discussion of Virtual Machines Related to Forensics Analysis". Available at, http://www.forensicfocus.com/downloads/virtual-machines-forensics-analysis.pdf.
[27]. NanniBassetti. "Imm2Virtual: A Windows GUI To Virtualize Directly From Disk Image File". Available at, https://articles.forensicfocus.com/2017/10/06/imm2virtual-a-windows-gui-to-virtualize-directly-from-disk-image-file/.
[28]. Source code Telegram Desktop. Available at, https://github.com/telegramdesktop/tdesktop.
[29]. Source code Telegram for MacOS. Available at, https://github.com/overtake/TelegramSwift.
[30]. Telegram Messenger. Available at, https://telegram.org.
[31]. Unofficial Telegram Wiki. Available at, https://telegram.wiki/#telegram_desktop.
[32]. Christian Oertle. "Anwendungsanalyse des Messengers Telegram Desktop (Version 0.9.15) unter Windows 10". Available at http://docplayer.org/28398743-Anwendungsanalyse-des-messengers-telegram-desktop-version-unter-windows-10.html. 2016.
[33]. "PKCS5_PBKDF2_HMAC_SHA1". Available at, http://openssl.cs.utah.edu/docs/crypto/PKCS5_PBKDF2_HMAC.html.
[34]. "DEFT Zero". Available at, http://www.deftlinux.net/2017/02/13/deft-zero-2017-1-ready-for-download/.