COMPUTERS & SECURITY XXX (XXXX) XXX

Available online at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

Forensic analysis of communication records of messaging applications from physical memory

Diogo Barradas, Tiago Brito, David Duarte, Nuno Santos, Luís Rodrigues*

INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Portugal

ARTICLE INFO

Article history:
Received 2 May 2018
Accepted 23 August 2018
Available online xxx

Keywords:
Digital forensics
Instant-messaging
Memory forensics
Mobile applications
Web-applications

ABSTRACT

Inspection of physical memory allows digital investigators to retrieve evidence otherwise inaccessible when analyzing other storage media. In this paper, we analyze in-memory communication records produced by instant messaging and email applications, both in desktop web-based applications and native applications running in mobile devices. Our results show that, in spite of the heterogeneity of data formats specific to each application, communication records can be represented in a common application-independent format. This format can then be used as a common representation to allow for general analysis of digital artifacts across various applications. Then, we introduce RAMAS, an extensible forensic tool which aims to ease the process of analysing communication records left behind in physical memory by instant-messaging and email clients.

1. Introduction

Instant-messaging (IM) and email applications such as Facebook's chat and Gmail clients, respectively, are widely used communication services that allow individuals to exchange messages over the Internet. Given the nature of the exchanged data, digital artifacts left by such applications may hold highly relevant forensic value. This is particularly true if, from such artifacts, it is possible to recover communication records of past conversations providing information about the content of exchanged messages, identity of communicating parties, or time-related information.

To assist analysts in recovering such artifacts, we aim to develop a forensic tool for the extraction of conversation records left by *messaging applications* in *physical memory dumps*. Web-based messaging applications run inside a browser and are becoming increasingly popular since users do not need to install them on their computers. Unfortunately, communication records exchanged in web-applications are not stored in desktops' persistent storage. This fact complicates the task of forensic analysts in acquiring digital evidence. In contrast, while native applications (e.g. in mobile devices) may leverage persistent storage, such applications' persistent state can be tampered with or intentionally deleted. By focusing on physical memory analysis, our goal is to complement the functionality of existing forensic tools which focus on the analysis of persistent state, e.g., local logs (Al Mutawa et al., 2011; Yang et al., 2016) or databases (Anglano, 2014; Anglano et al., 2017), and to address a latent need for the analysis of high-level data found lingering in memory dumps (Simon and Slay, 2009).

Although prior work has employed memory forensic techniques on web-messaging applications, existing tools tend to be highly application-dependent. For example, Wong et al. present techniques that allow for the recovery of digital artifacts for the Facebook messaging service (Wong et al., 2011). However, the proposed techniques cannot directly be applied

* Corresponding author.
E-mail addresses: diogo.barradas@tecnico.ulisboa.pt (D. Barradas), tiago.de.oliveira.brito@tecnico.ulisboa.pt (T. Brito), david.duarte@tecnico.ulisboa.pt (D. Duarte), nuno.m.santos@tecnico.ulisboa.pt (N. Santos), ler@tecnico.ulisboa.pt (L. Rodrigues).
https://doi.org/10.1016/j.cose.2018.08.013
0167-4048/

to multiple other applications due to the heterogeneity of data formats implemented by each application. Furthermore, the fact that web-based applications run inside a browser may interfere with the durability of applications' artifacts in memory, for example due to the memory management policy implemented by the browser. Existing works on browser forensics concentrate only on the extraction of browser-specific artifacts (e.g., browsing history, web cache) leaving aside the recovery of application-specific artifacts which may provide evidence (Ohana and Shashidhar, 2013).

Past developments in the use of memory forensic techniques have also been successful in the identification of artifacts produced by a small range of native mobile applications (Nisioti et al., 2017) such as Facebook Messenger. While the extraction of communication records holds in the writing of complex regular expressions, it is unknown whether this technique can be effectively applied in recovering communication records from several other popular messaging applications in the wild.

In this paper we make three key contributions. First, we present a *digital forensics study* aimed to systematically analyze the digital artifacts left in memory by several popular IM and email applications. Our study covers the analysis of artifacts produced by web-applications, when executed on various browsers, and also the analysis of artifacts produced by mobile applications. We successfully identified and retrieved IM communication records from web-applications such as Facebook, Twitter and Skype, as well as email records from Outlook and a generic Roundcube email web-client. Our study helps to characterize how the communication records of web-based messaging applications are typically represented and to identify how browser-specific mechanisms may affect the durability of such records in memory. Our study also allow us to identify and assess the durability of communication records produced by mobile applications such as WhatsApp, Viber or Hangouts (Section 2).

Second, we introduce the design and implementation of a *forensic tool* called RAMAS, which consists of a collaborative and extensible framework for analysis of communication records from volatile memory. RAMAS is able to extract such records from multiple messaging applications and display the extracted records on a user-friendly timeline. RAMAS is designed in a modular fashion so as to accommodate an ever-growing number of applications and allow collaborative development and maintenance of the system by independent forensic analysts. This goal is achieved through the implementation of extraction modules: each module contains a set of (simple) rules that allow RAMAS to extract the records of a specific application and represent them on a common application-independent format (Section 3).

Lastly, we present an *experimental evaluation* of our framework. To this end, we used RAMAS for conducting analysis over the data extracted from memory chips with sizes found in commodity hardware. Our evaluation shows that RAMAS is efficient, e.g., it can process communication records spawning from six different applications, in an 8 GB memory dump, in about three minutes. We also enact a use case for demonstrating the usefulness of our framework's evidence presentation capabilities which may help digital investigators in uncovering sophisticated correlations among evidence from several applications or across memory images (Section 4).

2. Digital forensics study

This section presents the digital forensics study that we carried out in order to assess the existence of communication records in physical memory produced by messaging applications. This study lays the ground for the subsequent development of a forensic tool which enables the automatic extraction of such records.

2.1. *Goals of the forensics study*

More concretely, the goal of this study is to check whether and in which conditions communication records can be obtained from memory dumps. In particular, our research is driven by two key questions:

How are messages represented in memory? The programmers of web-based messaging applications are free to implement them using a range of different technologies. Some design decisions comprise the choice of front-end and back-end programming languages (e.g. Javascript, PHP), others involve selecting the data representation format of communication records (e.g. JSON, XML, binary). In a similar way, the developers of mobile applications have a multitude of technologies available when programming such applications (e.g. native app development, HTML5-based apps). This heterogeneity in data representation and platforms may impact the way communication records are loaded into memory and contribute for the absence of a common model of the structure of communication records among different implementations of browsers, applications, and/or operating systems. Thus, this fact implies that a tool developed for analysing this kind of evidence would exhibit the additional complexity of having to take into account such differences between record structures, even when analysing a single application. We aim to assess whether there is a common model which allows for the interpretation of textual application data lingering in memory.

How long do messages persist in memory? The persistence of in-memory data structures pertaining to a given web-application may be affected by the browser where the application runs. First, we must ascertain whether the *browser runtime environment* imposes limitations on the ability to recover communication records from physical memory. In particular, to provide the interaction with web-applications, web-browsers rely on different layout engines (e.g. Blink, Gecko) which affect the way a browser hosts, renders or executes web content. Furthermore, different *user interactions* may trigger the erasure or replacement of potential evidence in volatile memory. For instance, data pertaining to a given web-application may be evicted shall a user navigate to a different browser tab or terminate her browsing session. Finally, modern browsers implement *private browsing* execution modes which enables users to browse the web while disabling both the browsing history and web cache. This feature is implemented by popular browser

Please cite this article as: Diogo Barradas et al., Forensic analysis of communication records of messaging applications from physical memory, Computers & Security (2018), https://doi.org/10.1016/j.cose.2018.08.013

vendors and is known under different aliases, such as Incognito, InPrivate or PrivateBrowsing. We must evaluate whether the use of private browsing may compromise the existence of digital artifacts lingering in memory.

Assessing the durability of digital artifacts in memory when considering the inspection of mobile applications also presents itself as an interesting endeavour. In fact, since these applications are often executed in resource-constrained devices, it is expected that mobile operating systems apply disparate low-level mechanisms for performing memory management and possibly foster a swift eviction of digital artifacts present in memory. Determining whether communication records can still be recovered after closing an application or blocking the mobile device is another key aspect that must be evaluated.

2.2. *Analysis of web-based applications*

The ongoing shift from desktop-based to web-based applications can complicate the task of digital investigators in getting access to evidence stored locally (Simon and Slay, 2009). Although browsers may log some data about web-searches or history, communication records exchanged in web-applications are not typically kept on disk. Fortunately, even if significant evidence cannot be successfully obtained from the analysis of static media, it is possible that digital investigators can resort to the inspection of physical memory in order to pinpoint otherwise inaccessible communication records. Next, we detail a study over the ability to retrieve communication records left lingering in memory by messaging web-applications.

2.2.1. *Experimental methodology*

We performed an experimental study of several web-based messaging applications to investigate whether and in which conditions messages can be obtained from memory. In particular, we analyzed digital artifacts concerning IM records for Facebook's chat, Facebook Messenger's chat, Skype, Twitter's Direct Messages, Google Hangouts, WhatsApp, Telegram, and Trillian. We also analyzed communication records of three email web-clients: Outlook, Roundcube, and Gmail.

These applications were tested on different browsers: Google Chrome, Mozilla Firefox, Opera, Microsoft Edge, and Safari. Tests for each browser were conducted for both private and non-private browsing sessions. For our tests, every web-application was used in a freshly generated browser tab. We have conducted our experiments in three widely used desktop operating systems: Windows 10, Ubuntu 16.04, and macOS Sierra. Each operating system was deployed in VirtualBox 5.1.8 virtual machines with 1GB of RAM. We acquired memory dumps from all systems through atomic virtualization-aided methods. Each virtual machine was restarted between separate test runs.

For each individual test, we conducted a predefined set of actions that capture real world usage of each web-application:

- *Instant-Messaging test run:* A user performs login in an IM web-application and sends two messages to a given recipient. The user also browses through existing contacts and inspects the current conversation, as well as past conversations (by navigating back and forth between conversations or by opening different chat windows inside the same browser tab). The aim of this set of actions is to retain relevant data in memory, while simulating a typical use of this kind of applications.
- *Email test run:* A user performs login in her webmail client and browses her inbox and outbox. This allows us to simulate a common use-case for these applications and later check which digital artifacts were retained in memory.

Simulated user interactions: Investigators may miss out the opportunity to get access to the target machine while a suspect is still logged in a given account. We assess four different sequences of actions a user may perform after concluding a session within the web-application, before we have the opportunity of acquiring a memory dump of the system. We consider the following sequences of actions, sorted in an increasing fashion according to their expected *intrusiveness level* (IL) with the artifacts we are concerned with:

- *(IL1)* Logout;
- *(IL2)* Logout and navigate to a webpage in a different browser tab;
- *(IL3)* Logout and navigate to a webpage in the same browser tab;
- *(IL4)* Logout and close the tab/browser.

We contemplate these degrees of *intrusiveness* according to the inner architecture of existing browsers. In all tested browsers, each tab runs in a separate process which is responsible for rendering a page's content. As an example, we refer to the recent Mozilla's Electrolysis project (Mozilla, 2015). In this setting, the browser's parent process manages tabs and the core browser functionality. This architecture brings several advantages, such as providing inter-tab data security, isolate crashes in individual tabs, and maintaining the overall browser's responsiveness (Charlie Reis, 2008). According to this description, we expect *IL1* and *IL2* sequences to be the least intrusive ones, since the tab state after logout is expected to be preserved. In contrast, the *IL3* sequence may affect the content of the memory allocated to the tab where the messaging service was used, possibly causing the eradication of evidence. Lastly, we expect the *IL4* sequence to be the most intrusive one, since the OS shall reclaim the memory associated to the tab/browser process, possibly wiping out any artifact that could have been left behind by the application.

Extraction methodology: To assess whether a given communication record was retained in memory and to pinpoint metadata structures surrounding it, the messages sent in each test run act like keywords for enabling posterior search. To look for these keywords, we begin by extracting the strings contained in the memory dump with the `strings` command-line utility, followed by a `grep` search to match the strings which contain a chosen keyword. Similarly, we attempt to match email records by searching email addresses known to be present in the client's inbox/outbox. A communication record is retrieved if the identified metadata enables for the collection of the tuple ⟨*Timestamp, Author, Recipient, Message*⟩, at least.

Please cite this article as: Diogo Barradas et al., Forensic analysis of communication records of messaging applications from physical memory, Computers & Security (2018), https://doi.org/10.1016/j.cose.2018.08.013

Table 1 – Feasibility of recovering web-based messaging application records from different browsers.

Web-Application	Browser				
	Google Chrome v55.0	Mozilla Firefox v50.0.1	Opera v42.0	Microsoft Edge v38.14393.0.0	Safari v10.1
Facebook	✓	✓	✓	✓	✓
Messenger	✓	✓	✓	✓	✓
Skype	✓	–	✓	✓	✓
Twitter	✓	✓	✓	✓	✓
Hangouts	–	–	–	–	–
WhatsApp	–	–	–	–	–
Telegram	–	–	–	–	–
Trillian	–	–	–	–	–
Outlook	✓	–	✓	✓	✓
Roundcube	✓	✓	✓	✓	✓
Gmail	–	–	–	–	–

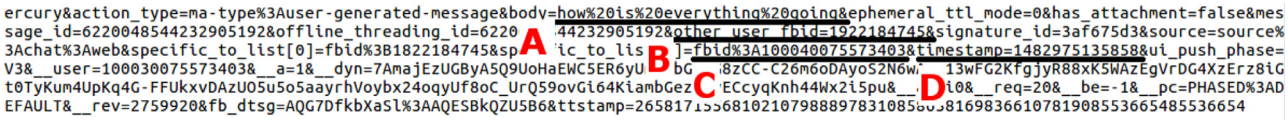

Fig. 1 – Facebook recently sent message data fields.

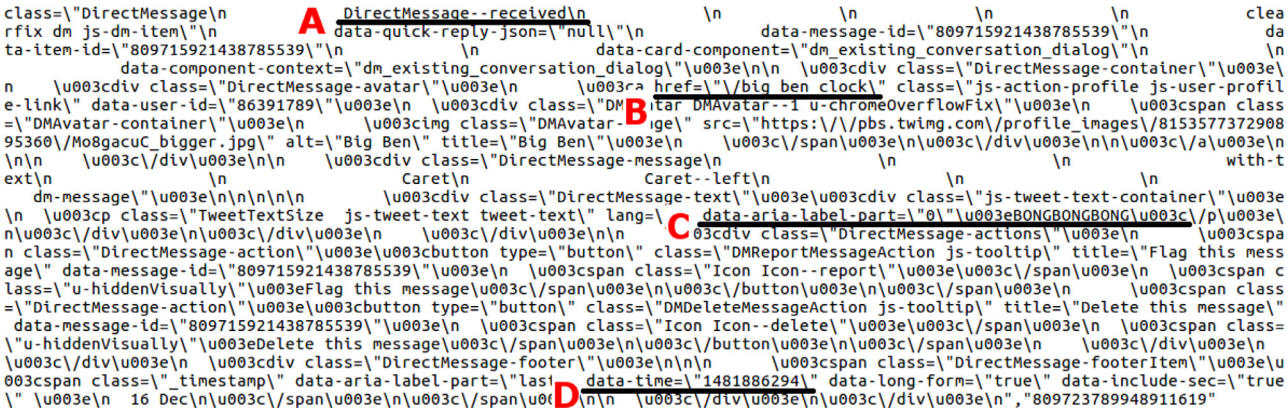

Fig. 2 – Twitter received message data fields.

Below, we present our main findings of this study.

2.2.2. *Message representation*

Table 1 depicts a summary of our analysis for several popular web-applications and web-browsers. Results show that it was not possible to retrieve any structured communication record from Gmail, Hangouts, WhatsApp and Telegram, which leads us to conjecture that these applications may make use of a binary data representation format which can not be directly recovered in the form of strings.

For all the remaining applications under test we were able to find messages with accompanying high-level metadata structures in the form of strings. For instance, we can observe in Figs. 1 and 2 two digital artifacts left in memory by Facebook chat and Twitter Direct Messages. In these cases, enclosing metadata was found either in the form of JSON or HTML, respectively. Albeit the metadata structures lingering in memory use a different data format and exhibit different fields, both follow a similar model which may be used to reconstruct the tuple ⟨*Timestamp, Author, Recipient, Message*⟩ which we consider a communication record. In the case of Facebook chat, the metadata fields present in Fig. 1 allow us to easily reconstruct the full ⟨*Timestamp, Author, Recipient, Message*⟩ tuple, where 1.A corresponds to *Message*, 1.B to *Author*, 1.C to *Recipient* and 1.D to the message *Timestamp*.

The example in Fig. 2 helps to identify an edge-case on the reconstruction of communication records. While 2.C can be easily matched with *Message* field and 2.D to the message *Timestamp*, 2.B does not provide enough information to state whether the identified Twitter alias corresponds to either *Author* or *Recipient*. However, 2.A clearly specifies the direction of the message. Thus, the record in Fig. 2 suggests it refers to a message where 2.B specifies its *Author*. The message *Receiver* would then consist of the account which the suspect has logged-in to and which can be known either by previously gathered intel or other data structures residing in memory. We

experimentally verified that, for outbound messages, the 2.A field would mark the Twitter Direct Message as *sent*.

Additionally, we found that the structure of communication records remains the same for each web-application across different browsers and operating systems. This observation reinforces our conjecture that the leakage of high-level data into memory is caused by the way web-applications are built, as the structure of high-level records present in memory is not tied to the particular implementation of a browser or an operating system's inner workings.

2.2.3. *Message durability*

We now ascertain how the use of different browsers and operating systems, as well as the execution of intrusive actions affects the recovery of communication records. As shown in Table 1, we retrieved communication logs with all tested browsers. However, we note that for experiment *IL1* we collected a smaller amount of messages when using Firefox, Edge, or Safari rather than when using Chrome or Opera. A possible explanation for this fact is that both Chrome and Opera are based on Chromium's codebase and may share implementation details which favour the preservation of a tab's resources in memory.

Browsers may implement different mechanisms for refreshing the contents of volatile memory, evicting the resources of background tabs and favouring those of the foreground tab. To check whether this fact affects the recovery of communication records, we conducted experiment *IL2*. Our results show that Chrome, Opera and Safari still retain communication records in memory while Firefox and Edge have eliminated all remnants of communication records belonging to the tab where our test ocurred. These results are congruent with experiment *IL1*, where Firefox and Edge were also less amenable to retain artifacts in memory.

Since modern browsers spawn a new process for managing each different tab, our initial intuition was that by closing a tab or by killing the browser, the opportunity to gather communication records would cease to exist. As it stands, for experiment *IL4*, we were not able to retrieve any communication record when performing this experiment over Windows 10. Upon closing a tab or killing the browser process, the operating system swiftly reclaimed the process memory back, thwarting high-level data inspection. Interestingly, when conducting the same experiments over Ubuntu 16.04 and macOS Sierra, we were able to retrieve communication records from memory after executing such intrusive actions.

Opposed to our initial expectations, navigating to different webpages in a given tab has triggered the most changes on volatile memory contents. Albeit we were able to recover a small number of high-level records in Ubuntu 16.04 and macOS Sierra, no records were recovered when conducting experiment *IL3* in Windows 10. In fact, the sequence of actions performed in experiment *IL3* fosters the replacement of older resources kept in memory in favour of more recent data. Thus, possibly useful evidence is discarded more promptly. Taking into account the outcomes of experiments *IL3* and *IL4*, Linux Ubuntu's and macOS Sierra memory management thus seems more favourable to conduct memory analysis in the context of our work rather than Windows 10.

Additionally, we experimentally verified that the use of private browsing in all tested browsers does not affect the collection of targeted high-level data. No visible changes have been observed either in the structure or the amount of communication records recovered in each test. Hence, our findings suggest that some of the privacy preserving properties of private browsing can be nullified through memory forensics.

Lastly, results show that the existence of communication records in memory appears to be loosely dependent on the browser/operating system in use, which leads us to infer that our results arise from the technologies and programming methodologies used by application/browser developers with respect to the loading and presentation of data. Two concrete cases are those comprising the recovery of Skype and Outlook logs, which can be performed for all browsers tested except Firefox when run over Windows 10.

2.3. *Analysis of mobile applications*

In Android, a vast majority of mobile applications logs data into local SQLite databases. In particular, it is not uncommon for instant messaging applications to conveniently store and organize messaging data in these databases. Although tools exist to automatically crawl and reconstruct communication records from SQLite databases (Magnet Forensics, 2014), this technique only succeeds if data is stored unencrypted, is not tampered/erased, or the device under analysis does not leverage full disk encryption.

2.3.1. *Experimental methodology*

We selected a range of popular messaging and email client mobile applications so as to conduct an experimental forensic study. This study has the goal of assessing whether such applications leave traces of communication records in the memory of mobile devices and, if so, if these can be reconstructed in a way that enables digital investigators to extract useful evidence. For this study, we analyzed the digital artifacts produced by a set of IM applications comprising Facebook Messenger, WhatsApp, Viber, Signal, Twitter, Telegram, Hangouts, and Trillian. Moreover, we inspected data produced by the Gmail and Outlook email client applications.

Our experiments were conducted in Android 8.0. We set up an Android emulator, by using Genymotion 2.12.0, in order to mimic a Samsung Galaxy S8 mobile device with 4GB of RAM. Since Genymotion relies on the use of VirtualBox to create emulators, we resorted to the atomic virtualization-aided method for collecting the device memory dump. The emulator was rebooted between experiments. In each experiment we conducted a set of actions aimed at reflecting the typical usage of each mobile application. Namely, we have conducted the *Instant-Messaging test run* and *Email test run* previously described in Section 2.2.1, but in each messaging application mobile counterpart.

Simulated user interactions: The best case scenario for analyzing the communications performed by a suspect using mobile applications is to seize a mobile device unlocked, or one that does not require a PIN code. However, with the increasing

Please cite this article as: Diogo Barradas et al., Forensic analysis of communication records of messaging applications from physical memory, Computers & Security (2018), https://doi.org/10.1016/j.cose.2018.08.013

Table 2 – Feasibility of recovering messaging records from different mobile applications in Android OS.

Application	Messenger	WhatsApp	Viber	Signal	Twitter	Telegram	Hangouts	Trillian	Gmail	Outlook
Recovered Records	✓	✓	✓	–	✓	–	✓	–	✓	–

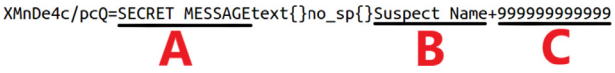

Fig. 3 – Viber received message data fields.

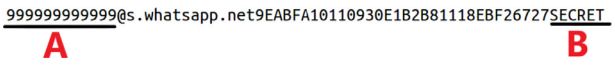

Fig. 4 – WhatsApp received message data fields.

concerns in user privacy, it is unlikely that digital investigators will be able to commonly seize mobile devices with an unlocked screen and manually sweep through the messages exchanged by a suspect.

In our experimetns, we simulate two sequences of actions found to be common usage patterns of mobile applications, prior to the acquisition of a memory dump of the device. These actions are sorted in an increasing fashion according to their expected *intrusiveness level* (IL):

- *(IL1)* Exchange messages/browse email inbox, navigate to Android's home screen and lock the device;
- *(IL2)* Exchange messages/browse email inbox, close the application, navigate to Android's home screen and lock the device.

The presented *intrusiveness* levels allow us to characterize the digital artifacts left in memory after the common use of mobile applications, having these either continuing being run in the background or being shutdown. In the context of our experiment, we expect *(IL2)* to trigger memory management mechanisms which foster the eviction of digital artifacts comprising communication records.

Extraction methodology: The methodology for pinpointing useful digital artifacts in memory and to reconstruct communication record is the same as detailed in Section 2.2.1. Essentially, we extract the strings present in the memory dump and perform a search for keywords present in a given message.

2.3.2. *Message representation*

Table 2 summarizes the results of obtaining structured communication records from the diverse mobile applications under test. Our results show that we are able to obtain structured communication records from the majority of messaging applications under test, including Facebook Messenger, WhatsApp, Viber, Twitter, and Google Hangouts, as well as from Gmail email client. Contrary to the aforementioned applications, Signal, Telegram, Trillian, and the Outlook email client were found not to load into memory a structured representation of data which allowed for the extraction of communication records.

Figs. 3 and 4 depict the digital artifacts pertaining to Viber and WhatsApp, respectively, found to be lingering in memory. In both cases, metadata was found not to be enclosed inside JSON or HTML alike structures, but represented in an application-dependent data structure. This observation is also true for the remaining applications with the exception of the Gmail email client where, interestingly, the communication records metadata was enclosed within HTML artifacts. This observation may be explained due to the fact that the Android Gmail application includes an HTML rendering engine.

In contrast to the in-memory representation of web-based applications explored in Section 2.2, the results of our study show that the digital artifacts left in memory by native mobile applications are substantially less detailed. For instance, in Fig. 3, we can observe that Viber's communication records are represented in fairly small strings, where metadata fields have no telltale delimiters. Even in the absence of such delimiters, 3.A can be easily matched with the *Message* field, field 3.B can be matched to the message's *Author* username, and field 3.C contains the *Author* phone number (obscured in our figure). The inclusion of a *Timestamp* field in Viber's digital artifact is not clearly identifiable. In a similar way, the digital artifacts produced by WhatsApp only allow for the extraction of partial communication records. We can observe in Fig. 4 that identifiable fields consist of the *Author* field composed by a phone number (4.A), and by the *Message* field (4.B).

The previous examples show that, as opposed to the majority of metadata-rich digital artifacts produced by web-based applications, native mobile applications generate significantly shortened digital artifacts. While these artifacts may be insufficient to reconstruct the full ⟨*Timestamp, Author, Recipient, Message*⟩ tuple, these can still convey useful information which digital investigators may be able to correlate with other sources of data.

In addition, our analysis reveals that it is possible to retrieve communication records from several applications running on mobile devices, as opposed to these same applications' web-based counterparts. For instance, we were able to retrieve communication records from WhatsApp, Hangouts, and Gmail mobile versions, while such extraction was not possible in the respective web-applications (Section 2.2).

2.3.3. *Message durability*

In our study, we determined that it is possible to retrieve communication records when the mobile application is either open (*IL1*) or closed (*IL2*) after locking the mobile device. Similarly to our experiments with Linux Ubuntu 16.04 and macOS Sierra, our findings suggest that Android's memory management policies also do not swiftly evict application-related data from memory when applications are closed. As shown in Section 2.3.2, this fact allows for the retrieval

Please cite this article as: Diogo Barradas et al., Forensic analysis of communication records of messaging applications from physical memory, Computers & Security (2018), https://doi.org/10.1016/j.cose.2018.08.013

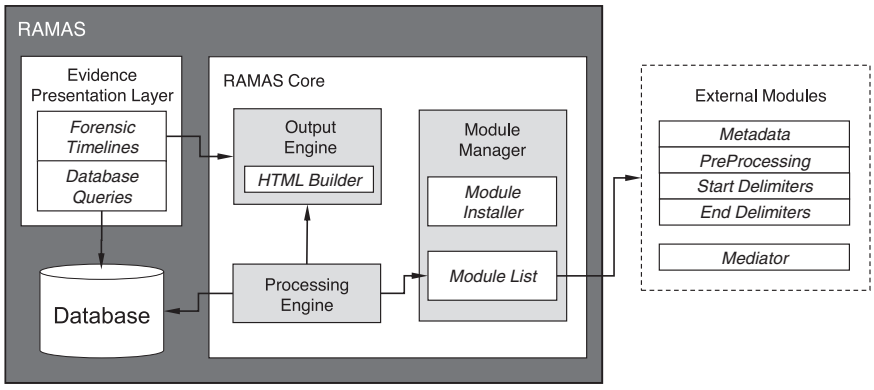

Fig. 5 – RAMAS architecture.

of communication records pertaining to multiple messaging applications.

2.4. *Lessons learned*

The outcomes of our study indicate that the recovery of communication records from physical memory is a viable path for uncovering otherwise ephemeral evidence. We were able to retrieve high-level data from some of the most popular messaging applications used today, which motivates the need to develop specialized tools to recover evidence for the unfolding of digital investigation cases. Unfortunately, the heterogeneity of data structures found in memory significantly increases the burden of digital investigators with the task of building and maintaining a wealth of pattern matching expressions to automatically extract available evidence from memory dumps.

Given this fact, we suggest that a useful tool for the inspection of communication records lingering in memory should provide the ability to easily generate and maintain such matching expressions. In the next section, we describe the design and implementation of a novel forensic framework with the goal of recovering communication records from physical memory. Our framework aims at alleviating digital investigators from the hindrances associated to the generation and maintenance of complex regular expressions which can then be used for the extraction of evidence from digital artifacts.

3. RAMAS framework

This section presents RAMAS, a forensic tool for the extraction of communication records from memory dumps[1].

3.1. *Design goals*

We built RAMAS according to four design goals:

Simple design of extraction modules: The design of new extraction modules (composed of string matching patterns) should be easy to perform. Practitioners should be able to contribute to the RAMAS framework without the need of being familiar with a particular programming language.

Simple sharing/update of extraction modules: Modules should be easily upgradable, without the need to change RAMAS' core functionality. It should be straightforward for practitioners to share and advertise newly developed modules.

Organized case management: The number of seized machines in computer forensics cases may scale to large numbers. To avoid extra work in managing data pertaining to several cases, RAMAS should provide an integrated way to organize the collected memory images and store the retrieved high-level data.

Simple inspection of results: RAMAS should provide a simple interface to visualize extracted communication records, support queries on the recovered data, and aid investigators in disclosing complex correlations among pieces of evidence.

3.2. *Architecture*

We implemented a RAMAS prototype for Linux. Our prototype was written in Python and leverages a SQLitev3 database for holding memory analysis results. Fig. 5 shows the several components that implement the core functionality of the system, the management of extraction modules, and the evidence presentation layer. Additionally, we deployed RAMAS as a GUI desktop application as a further effort in making the system attractive to users without compromising any of its functionality (Section 3.6).

Our framework is based on a carving approach to retrieve matching records present in memory images. RAMAS matches communication records by exploring their structure, retrieving all meaningful data held between well defined delimiters. Upon the collection of digital artifacts containing communication records, RAMAS processes the metadata associated to these artifacts to isolate the sole components which convey useful evidence to the investigator. This allows for the discard of application dependent fields which are useless for an high-level inspection of records, such as application versions. Furthermore, other kinds of data forming the structure of a

[1] RAMAS stands for "RAM Analysis System".

```
this.add_message_row(6501,{"subject":"Bank assault plan","fromto":"<span class=\"adr\"><span title=\"burglar@example.com\"
 class=\"rcmContactAddress\">Billy the Kid<\/span><\/span>","date":"2017-01-12 20:21","size":"12 KB"},{"seen":1,"ml":1,"ct
ype":"text\/plain","mbox":"INBOX"},false);
```

Fig. 6 – Roundcube inbox entry data fields.

record can also be discarded. A relevant example comprises verbose HTML wrapper artifacts that compose the structure of the communication record but offer no value from the point of view of the analysis (see Fig. 2).

Analysis in RAMAS is conducted by running a selection of the available extraction modules over the strings obtained from a given memory dump. After analysis, RAMAS stores the recovered data in a database and offers a simplified visualization of the recovered records, organized by modules.

3.3. *Extraction modules*

The forensic study conducted in Section 2 reveals that the internal representation of messaging clients is different for each application. Thus, the absence of a common internal representation for diverse applications imply that it is hard to devise a single regular expression to retrieve all relevant high-level data. Additionally, considering the ever-growing number of messaging applications, there is not an obvious number of extraction modules that should be integrated into RAMAS before-hand. Thus, users should be able to developnew extraction modules without the need for changing the system's core or to be familiar with a specific programming language. We used Python's `ConfigParser` configuration files for implementing extraction modules. These easily allow us to define groups of values which suffice for delimiting an existing record, as well as individual fields within the record.

Listing 1 provides the extraction module for recovering Roundcube's inbox email records, depicted in Fig. 6.

```
[Info]
Name: roundcubeInbox
Description: Roundcube's inbox record fetcher.

[PreProcess]
Keyword: add_message_row

[StartDelimiter]
Record: this.add_message_row
Content: {"subject":"
Author: title="
Date: date":"

[EndDelimiter]
Record:;
Content: ","from"
Author: " class
Date: ",

[Mediator]
Date: yyyy-mm-dd hh:mm
```

Listing 1 – Extraction Module for Roundcube inbox records.

The first section of a module is named `Info` and contains general information about the module, namely its name and a short description. The second section is called `PreProcess` and contains a single `Keyword` parameter, the content of which shall be used to restrict the pool of strings upon which record matching will be applied. In our example, we retain all strings which contain the substring `add_message_row`, since this keyword appears in the metadata structure of the records we aim to extract. The two next sections declare the start and end delimiters of the whole record as well as the individual fields that should be extracted and recorded in RAMAS' database. As RAMAS attempts to sequentially match the declared record fields, the corresponding delimiters must be declared in the order they appear within the record. We note that this restriction must be enforced by the module developer. Lastly, the `Mediator` section contains a hint on how to interpret the date representation so as to convert it to a common representation shared amongst all modules.

Finally, writing a module for describing the communication records produced by a given application is fairly simple. As presented in Listing 1, the module for extracting Roundcube inbox records requires a total of 17 lines of code.

Module update: Application providers are free to update the internal representation of data. To ensure the retrieval of communication records, investigators must update extraction modules accordingly. Hence, the update of extraction modules should be straightforward. Due to the modular design of RAMAS, the task of updating a module only comprises changes on the module itself, not affecting any other component of RAMAS' core functionality. Similarly to the development of new modules, updates are performed by changing the required fields on the configuration file. This process may encompass the addition/removal of some metadata field that has became available/deprecated, or it may just involve the minor updates in fields already declared in the configuration file.

Module sharing: To avoid repeated work due to concurrent endeavours by digital investigators, we envision the creation of a centralized repository for helping a module's creator and other practitioners to update existing modules while keeping track of changes. Additionally, RAMAS can check the repository on start-up and update installed modules to their most recent version. While the implementation of such a repository is deferred to future work, the codebase resulting from our work already allows for the manual install and update of extraction modules.

3.4. *Dealing with heterogeneous data*

Due to the heterogeneity between applications, building a database of all recovered records represents a challenge. We

Please cite this article as: Diogo Barradas et al., Forensic analysis of communication records of messaging applications from physical memory, Computers & Security (2018), https://doi.org/10.1016/j.cose.2018.08.013

refer to classical problems in data integration. Firstly, we encounter a *schema-matching* problem, where the same concept may be identified in different applications by differently named fields. Secondly, we may encounter a *semantic-matching* problem, where even fields with the same name can refer to data with distinct underlying meanings. Moreover, the metadata available in some application may be richer than that available in others. Thus, the identification of the minimal set of fields that can be successfully used for unveiling correlations between collected evidence is crucial for allowing RAMAS to execute data integration routines for providing better query support to digital investigators.

The fields extracted by each module can represent semantically equivalent metadata, although it is named differently in distinct applications, e.g., a message timestamp may be identified by fields with different names (*date* vs *time*). To offer a single database schema which supports queries over records extracted with different modules, RAMAS performs data integration to provide users a unified view of the communication records coming from different sources. Each configuration file declares the mapping between the heterogeneous schema of different modules and the global RAMAS database schema.

RAMAS still faces a semantic integration problem where, for different modules, the same concept may express different meanings. As an example, for any two different modules, *author* can represent a user's name in some application and an application-dependent identifier on another. Even with such a limitation, RAMAS is able to maintain a global timeline of all application activity conducted by a suspect. When ordering records which use different representations for time, the configuration file can contain a hint on how to convert a particular timestamp representation to a common representation such as UNIX time.

3.5. *Memory dump processing pipeline*

The strategy we followed in the creation of modules ensures a flexible independence between these and the core functionality of RAMAS. We now describe how RAMAS processes a memory dump with base in such modules. We assume that RAMAS receives as input a file containing strings, instead of the raw memory image acquired by first responders. To produce such a file, investigators may resort to the standard command-line utility `strings` which finds and prints text strings embedded in binary files.

The strings file obtained from the raw memory image can be analyzed simultaneously by a set of modules. To this end, RAMAS dispatches a batch of modules to a pool of threads. Threads filter the initial list of strings obtained from the memory image, generate regular expressions by gluing together the delimiters of the fields declared in each module, and perform the actual evidence extraction work.

The initial pre-processing of the strings file with a keyword is justified by performance reasons. We discuss in Section 4 the advantages of this preliminary filtering step. After filtering the strings dump, the backend of RAMAS applies a regular expression over the remaining strings, further restricting the existing digital artifacts to the strings comprising a full communication record. In this second processing stage, RAMAS introduces a countermeasure against the injection of delimiters in the content of messages. If RAMAS allowed for such an injection, a miscreant would be able to inject an end-delimiter of a message among the written text and hide incriminating messages after this artificial delimiter. To thwart this attack on the inner workings of our system, RAMAS applies a find-and-replace method on each individual record. RAMAS searches for the start delimiter of the message field of a record starting from the far left and scans the end delimiter of the same field from the far right. When found, these delimiters are replaced by a pseudo-random nonce which will now act as start/end delimiter for the message field. Thus, the only way an attacker would have to interfere with the correct recovery of the message content would be to guess this nonce and place it before his incriminating message.

In the third processing stage, RAMAS builds a second regular expression which scans every remaining record and retains data according to each of the declared fields in the module configuration file. Lastly, RAMAS saves the obtained records in a database for further analysis. Details over the evidence presentation layer are detailed next.

3.6. *Evidence presentation layer*

Upon completing the analysis of the high-level data contained in a selected memory dump, RAMAS updates its global database schema, adding the newly discovered ⟨*Timestamp, Author, Recipient, Message*⟩ tuples. This global schema allows investigators to pose queries at the database, enabling the correlation of evidence obtained from the use of several modules over a single memory image, or across different memory images pertaining to a case. A motivating example of such a scenario is presented in Section 4.1.

Our desktop application includes a GUI for providing access to the tool's functionalities, allowing investigators to conduct a typical investigation workflow, including the management of cases, indexing memory dumps, and check the results through simplified forensic timelines. We deployed RAMAS as a desktop application by employing the QT framework and corresponding PyQt bindings for operability with our existing Python codebase. Forensic timelines are rendered through the HTML rendering engine available in PyQt so as to enable investigators to inspect the generated timelines without the need to resort to a separate web-browser.

We evaluated the performance of RAMAS by conducting several experiments over the time it takes to complete a forensic analysis. Particularly, we tested the impact of the number of extraction modules applied, as well as the impact of the memory image size on the forensic analysis performance.

In our experiments, we have exchanged messages on four web-applications in order to conduct forensic analysis: Facebook chat, Skype, Twitter and Roundcube. Each web-application was operated in a different tab of Google Chrome running over a Windows 10 virtual machine. For the experiments reported in this section, we used the sequence of user actions with intrusiveness level *IL1*.

The analysis of communication records using RAMAS was conducted on a 64-bit Linux Ubuntu 16.04 LTS virtual machine equipped with four 2.6GHz Intel i7-6700HQ virtual CPUs, and 4GB of RAM.

Please cite this article as: Diogo Barradas et al., Forensic analysis of communication records of messaging applications from physical memory, Computers & Security (2018), https://doi.org/10.1016/j.cose.2018.08.013

Table 3 – Time elapsed (in seconds) in extracting printable characters out of differently sized memory images.

Raw memory	Strings file size	Time elapsed
1 GB	82 MB	27.205 s
2 GB	157 MB	53.148 s
4 GB	255 MB	108.459 s
8 GB	233 MB	193.539 s

3.7. *Measuring analysis time*

Finding printable characters: The strings file used as input for RAMAS is indeed smaller than the total size of the raw memory image under analysis. Table 3 depicts the time spent in reducing a raw memory dump to its printable characters in a default execution of the `strings` utility. We note that `strings` can be further instructed to ignore sequences of characters less than a given size constant, which may further reduce the time elapsed during the pre-processing step.

Table 3 also shows us a counter-intuitive result comprising the amount of printable characters found in 4GB and 8GB raw memory images. Interestingly, the size of the strings file obtained from the 4 GB memory image (255 MB) was larger than that obtained from saving the strings contained in the 8GB memory image (233 MB). We verified our results by acquiring six new memory images using the same procedure. For all the acquisitions, the amount of printable characters found in a 4 GB memory image was larger than that found in an 8 GB memory image.

4. Evaluation

Varying memory image size: We studied the impact of the memory image size on the elapsed time to complete a forensic analysis with RAMAS. In this test, we fixed the use of a single module while we vary the size of the memory dump (respectively, the size of the produced strings file). Without loss of generality, we selected a module for identifying Facebook communication records depicted in Fig. 1 and extract the full ⟨*Timestamp, Author, Recipient, Message*⟩ tuple.

Fig. 7 presents the elapsed time for five executions of our test without employing the pre-processing stage described in Section 3.5. Conversely, Fig. 8 presents the elapsed time for a similar experiment in which we employ pre-processing.

In a general way, our results suggest that the size of the memory image negatively affects the time it takes to complete a forensic analysis. The results depicted in Fig. 7 show that omitting a pre-processing step over the initial high-level data artifacts slows down the analysis time considerably. For an 8GB memory dump, scanning for evidence lasts for about eight seconds, threefold the time it takes for analyzing the strings contained in a 1GB raw memory dump.

Fig. 8 shows that pre-processing the initial strings file brings a whole lot of improvement to the performance of the forensic analysis conducted by RAMAS. Indeed, reducing the set of strings that a module must be matched against to the data related to the search domain drastically decreases the

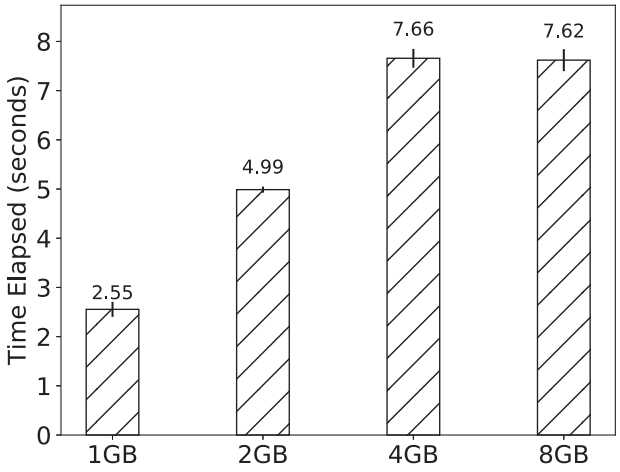

Fig. 7 – Time elapsed on the forensic analysis of differently sized memory images with Facebook recent messages module - no pre-processing.

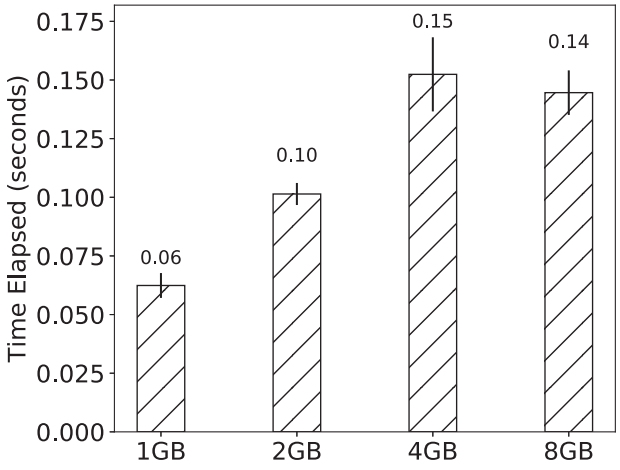

Fig. 8 – Time elapsed on the forensic analysis of differently sized memory images with Facebook recent messages module - with pre-processing.

analysis elapsed time. This performance improvement is of several orders of magnitude, being noticeable when analysing the biggest memory images under test.

The attentive reader may notice the similarity in the time spent while analysing a 4GB and an 8GB memory image, either applying pre-processing or not. In fact, the analysis performed over the strings collected from the 4GB memory image is slower than the analysis for the strings of an 8 GB memory image. This is due to the fact that, albeit the 4 GB raw image is smaller than the 8 GB raw image, we were able to find a larger amount of strings in the former.

Varying the number of modules: A different experiment aims to understand the impact of the number of applied modules on the elapsed time to complete a RAMAS analysis. In this test, we fixed the size of the memory dump in 8 GB, a reasonable amount of RAM widely deployed in commodity

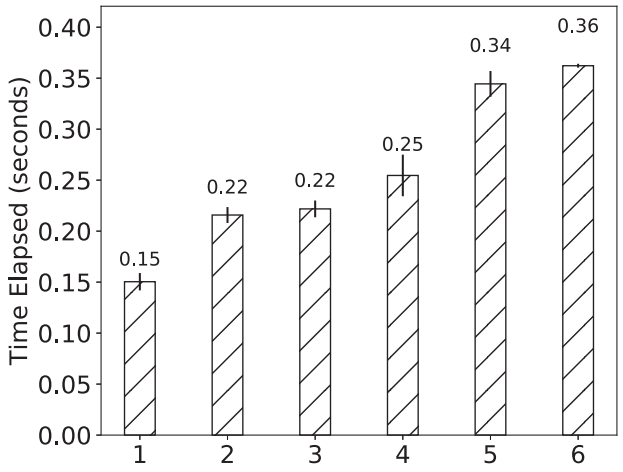

Fig. 9 – Time elapsed on the forensic analysis of an 8GB memory image with increasing number of modules.

```
#Records recovered with Twitter Direct
    Messages module
1482949701, S_3, S_1, I have revised the plan,
     lets do it at 4pm.
1482949712, S_3, S_1, Ok boss, I'll tell
    Pinkman.

#Records recovered with Skype messages module
1482949730, S_1, S_2, Hey, Heisenberg has
    revised the plan, lets rob the bank at 4pm

#Global Timeline
-------------------------------------------
1482949701, S_3, S_1, Twitter, I have revised
    the plan, lets do it at 4pm.
1482949712, S_3, S_1, Twitter, Ok boss, I'll
    tell Pinkman.
1482949730, S_1, S_2, Hey, Heisenberg has
    revised it, we strike at 4pm.
```

Listing 2 – Records gathered by several extraction modules.

hardware. For conducting this experiment, we attempted the extraction of communication records from all of the different Facebook, Skype and Roundcube Inbox artifacts we identified in our forensic study in Section 2. Accordingly, we developed the corresponding extraction modules for each of the corresponding web-application's digital artifacts, spawning a total of six modules. Fig. 9 presents the average elapsed time for five executions of our test.

The results of our experiments show that running several modules in parallel results in a sub-linear time for the completion of the forensic analysis. In fact, an analysis comprised of six modules has completed in nearly double the time of the time spent conducting an analysis with a single module. Our study suggests that the most expensive operation consists in the extraction of high-level data in the form of strings. When compared to this preliminary effort, the actual forensic analysis of the obtained data is several orders of magnitude smaller, even when applying multiple extraction modules. This suggests that RAMAS can be deemed a practical tool for aiding digital investigators as its performance allows for evidence to be quickly obtained, raging from seconds to a few minutes, depending on the size of each memory image.

In this experiment, we recovered 17 Roundcube Inbox, 1 Skype and 26 Facebook records, where 5 of the latter were duplicates and 3 were corrupted (application-level data was included beyond a partial message itself). We discuss possible causes for the corruption of recovered records in Section 4.2.

4.1. *Use case*

We enacted a use case for showing the benefits of maintaining a global database schema which investigators can query in order to unveil more sophisticated correlations among data fetched from different modules/memory images. An example of such a scenario is the identification of a chain of command in a criminal organization. Let us assume that law-enforcement suspects one individual (S_1) to be involved in a criminal organization. While investigating this case, law-enforcement storms through S_1's household and acquires the volatile memory of the hardware operated by the suspect. Upon analysing the memory image in search of ⟨*Timestamp, Author, Recipient, Message*⟩ message tuples, investigators recover the chat records depicted in Listing 2.

This example sequence of messages allows investigators to draw the following conclusions:

- S_1 was a surrogate in the organization. Investigators now have digital evidence about the fact.
- S_2 represents a person in the organization that authorities were not aware of. He was not the head of the criminal organization, but authorities now have a new lead to follow.
- S_3 seems to be the organization's mastermind. Moreover, Heisenberg appears to be his name.

Although we present a simple example, we expect the number of records to be recovered in actual settings to be much larger and harder to digest. By analysing modules' output in an isolated fashion, investigators could not be able to directly reach the conclusion presented in the last bullet. The connection between the name and organization rank can only be established by joining and contextualizing the evidence recovered from both modules. Albeit this example suffices to show the need for a global timeline for correlating data across modules, the benefits of maintaining a single global timeline can be further noticeable when attempting to correlate data obtained from several different memory images, collected in the scope of a single case.

4.2. *Limitations and future work*

Miscreants which are knowledgeable about the analysis procedure conducted by RAMAS may refresh the browser tabs they use to conduct illegal activity often. This behavior may cause communication records to be replaced/evicted, eliminating traces of criminal activity.

Due to the nature of volatile memory, metadata structures may be only partially available when scanning for evidence. If RAMAS is unable to find the start/end delimiters of a record, it

will either ignore a partial record or capture data beyond the expected limits, respectively. To overcome this limitation, an improved version of RAMAS may attempt to match the end of a record and backtrace to fetch existing metadata, as well as use heuristics for the record expected size and limit overruns in adjacent artifacts.

Albeit RAMAS supports the integration of new extraction modules, digital investigators are still required to write such modules for a range of existing applications. It is possible that an improved version of RAMAS can leverage a corpus of existing messaging record structures to learn models which could automatically scan memory dumps and identify record fields where telltale delimiters exist (e.g. `timestamp`, `msgTime`). While such an approach is promising for the inspection of applications which load verbose content into memory, it is unclear how such a method will perform in identifying records composed by minimal field delimiters (e.g., Viber's `text{}no_sp{}` depicted in Fig. 3). Evaluating the possibility of reliably retrieving communication records in an automatic fashion is an interesting direction for future work.

5. Related work

This section describes related literature regarding memory collection procedures and the analysis of high-level data present in physical memory.

5.1. *Memory acquisition*

The proper collection of evidence is a crucial step in any computer forensics investigation. Typical computer investigations targeted the hard drive of a suspect's machine, ignoring all of the data kept in volatile memory. Today, the analysis of both static media and physical memory allows digital investigators to have a better picture of the original state of the system and recover otherwise ephemeral data (Vömel and Freiling, 2012). Although the choice of a memory acquisition procedure should take into account the particularities of the case at hand, there are three main memory acquisition methods used by first responders, which are based on hardware, software and virtualization (Vömel and Freiling, 2011).

Hardware-based methods can take advantage of DMA (Direct Memory Access) requests through hardware cards and allow for the collection of memory content while having little impact in the system. However, the target system must be equipped with specialized hardware, and this technique may lead to system crashes which poses reliability issues in memory collection. A different hardware-based method relies on the remanence of data in DRAM and SRAM chips, whose contents are kept even after several seconds of power being lost. While memory contents fade gradually over time, it is possible to quickly replug the same RAM chips into another computer system and acquire a memory dump (Halderman et al., 2009). A similar technique can be applied on mobile devices by flashing a recovery image responsible for dumping RAM contents after performing a cold boot (Müller and Spreitzenbarth, 2013).

Virtualization-based methods allow for a sound extraction of memory by either collecting a file where the physical memory of the virtual machine is kept or by dumping it with the virtualization software. Clearly, this is only interesting if malicious activity is being conducted on top of a virtual machine. Nonetheless, the growing importance of Internet-hosted services is expected to impose a partial shift on the focus of digital investigations to virtual machines.

Software-based methods for dumping physical memory are widely available to digital investigators. However, some of these tools require special access privileges and cannot generally offer a full copy of the memory at a given time. Kernel level acquisition tools overcome some of the limitations imposed by user level collection tools, but are still unable to provide a completely atomic memory image due to the activity of concurrent processes. Despite this shortcoming, software-based collection is often applicable in practice since it does not require a specific system configuration to be set in advance nor does it rely on specialized hardware.

5.2. *Memory analysis*

Contrary to the analysis of static media, the analysis of physical memory typically presents a harder challenge due to the lack of a completely deterministic organization. It should be noted that many of the efforts dedicated to the analysis of physical memory focus on the recognition and inspection of OS-dependent low-level memory structures (Simon and Slay, 2009; Vömel and Freiling, 2011). Conversely, high-level memory inspection strategies are typically bound to the search of strings in the acquired dump.

Analysis approaches based on the search of strings containing pertinent keywords related to a case exhibits several drawbacks. In the one hand, investigators may be presented with thousands of matching records when analyzing large amounts of data (Beebe and Dietrich, 2007). To make it worse, only few of those records may be directly related to the case itself. In the other hand, if the terms contained in memory slightly deviate from the keyword list used by the investigator, some evidence will fail to be recognized.

A previous approach (Beebe et al., 2011) has addressed some of these drawbacks by improving string searching through neural networks which learn a list of terms related to the case beforehand. Analysis results are ordered according to their relevance, allowing the investigator to inspect the most pertinent data first. Although the prototype yields good results with respect to the obtained recall, the neural network takes a non-negligible time to learn the terms and is not able to provide any context about the matched keywords.

A different technique based on regular expressions has been successfully applied in order to extract evidence from strings accompanied by syntactically structured metadata (Nisioti et al., 2017; Simon and Slay, 2010; Wong et al., 2011; Yang et al., 2016). This metadata provides context about a given artifact, allowing investigators to reason about the contextual relevance of the data (Raghavan, 2013). For instance, providing that the metadata contains a timestamp, investigators can build a timeline and reconstruct a sequence of actions took forth by a suspect.

However, this line of research presents several drawbacks. Firstly, prototypes are developed independently without enabling regular expression sharing or providing evidence visualization interfaces; extraction capabilities are merely seen as

Please cite this article as: Diogo Barradas et al., Forensic analysis of communication records of messaging applications from physical memory, Computers & Security (2018), https://doi.org/10.1016/j.cose.2018.08.013

Table 4 – Comparison of RAMAS with different memory forensic tools. * Tool requires plugins to provide the functionality.

Forensic tool	Memory acquisition	Multiple OS target	Live analysis	Extensibility	Investigation workflow	Open-source	High-level analysis
RAMAS	–	✓	–	✓	✓	✓	✓
Volatility	–	✓	–	✓	–	✓	-
Redline	–	–	–	–	✓	–	✓
Memoryze	✓	–	✓	–	–	–	-
FATkit	–	✓	–	✓	–	–	-
VAD Tools	–	–	–	–	–	✓	-
EnCase	✓	✓*	–	✓	✓	–	✓
Rekall	✓	✓	✓	✓	–	✓	-
IEF	✓	✓	–	–	✓	–	✓

proofs-of-concept. Secondly, string matching works only as long as regular expressions match the syntax of the application artifacts found lingering in memory. Since an artifact's structure may change due to application implementation decisions, investigators must take the burden of updating regular expressions accordingly. In fact, this observation is seconded by our findings in Section 2, where Facebook chat's digital artifacts were found to be different than those previously documented (Wong et al., 2011). Lastly, previous regular expressions developed for the extraction of communication records lingering in memory are significantly complex (Nisioti et al., 2017). RAMAS aims to tackle the aforementioned issues by fostering the collaboration of digital investigators on maintaining a platform for the analysis of high-level data residing in memory, through the sharing of simple to build record extraction modules.

To better lay RAMAS in the space of existing memory forensic analysis tools, Table 4 depicts a comparison of several properties exhibited by our system with those of well-known memory forensic software. We can observe that RAMAS presents itself as the only open-source tool that provides an investigation workflow (including case management and evidence visualization) and is extensible, while focusing on the recovery of high-level data. Comparatively, although EnCase can be extended through plugins, it remains a proprietary and expensive tool, while Rekall focuses on low-level analysis and fails to provide a proper investigation workflow.

6. Conclusion

This paper described a forensic study over the digital artifacts left behind in memory by popular web and mobile applications in current desktop and mobile devices. Our study concludes that it is possible to retrieve communication records from IM/email applications in various system configurations.

Motivated by the findings above, we have introduced RAMAS, a framework for the extraction of instant-messaging and email client data from volatile memory. Our evaluation suggests that RAMAS can efficiently extract and report the existence of communication records. RAMAS code is publicly available and has been released as an open-source tool[2].

[2] RAMAS repository - https://www.tiagolb.github.io/CSF/.

Acknowledgments

This work was supported by Fundação para a Ciência e Tecnologia (FCT) via projects PTDC/EEI-SCR/1741/2014 (Abyss), SFRH/BSAB/135236/2017, and UID/CEC/50021/2013 (INESC-ID), and by Instituto Superior Técnico, Universidade de Lisboa.

REFERENCES

Al Mutawa N, Al Awadhi I, Baggili I, Marrington A. Forensic artifacts of Facebook's instant messaging service. Proceedings of the international conference for internet technology and secured transactions, 2011.

Anglano C. Forensic analysis of whatsapp messenger on android smartphones. Digital Investig 2014;11(3):201–13.

Anglano C, Canonico M, Guazzone M. Forensic analysis of telegram messenger on android smartphones. Digital Investig 2017;23:31–49.

Beebe N., Dietrich G. A new process model for text string searching; New York, NY: Springer New York. p. 179–191. doi:10.1007/978-0-387-73742-_12.

Beebe NL, Clark JG, Dietrich GB, Ko MS, Ko D. Post-retrieval search hit clustering to improve information retrieval effectiveness: two digital forensics case studies. Decis Support Syst 2011;51(4):732–44.

Charlie R. Multi-process architecture. https://blog.chromium.org/2008/09/multi-process-architecture.html; 2008. Accessed: 2018-04-29.

Halderman JA, Schoen SD, Heninger N, Clarkson W, Paul W, Calandrino JA, Feldman AJ, Appelbaum J, Felten EW. Lest we remember: cold-boot attacks on encryption keys. Commun ACM 2009;52(5):91–8.

Magnet F. The rise of mobile chat apps: recovering evidence from Kik messenger, WhatsApp & BBM. https://www.magnetforensics.com/mobile-forensics/the-rise-of-mobile-chat-apps-recovering-evidence-from-kik-messenger-whatsapp-bbm/; 2014. Accessed: 2018-04-29.

Mozilla. Electrolysis - Mozilla Wiki. https://wiki.mozilla.org/Electrolysis; 2015. Accessed: 2018-04-29.

Müller T, Spreitzenbarth M. Frost: Forensic recovery of scrambled telephones. In: Proceedings of the international conference on applied cryptography and network security. Springer; 2013. p. 373–88.

Nisioti A, Mylonas A, Katos V, Yoo PD, Chryssanthou A. You can run but you cannot hide from memory: extracting IM evidence of Android apps. In: Proceedings of the IEEE symposium on computers and communications; 2017. p. 457–64.

Ohana DJ, Shashidhar N. Do private and portable web browsers leave incriminating evidence?: A forensic analysis of residual artifacts from private and portable web browsing sessions. EURASIP J Inf Secur 2013;2013(1):6 https://jis-eurasipjournals.springeropen.com/articles/10.1186/1687-417X-2013-6.

Raghavan S. Digital forensic research: current state of the art. CSI Trans ICT 2013;1(1):91–114 doi:10.1007/s40012-012-0008-7.

Simon M, Slay J. Enhancement of forensic computing investigations through memory forensic techniques. In: Proceedings of the international conference on availability, reliability and security. IEEE; 2009. p. 995–1000.

Simon M, Slay J. Recovery of Skype application activity data from physical memory. In: Proceedings of the international conference on availability, reliability, and security. IEEE; 2010. p. 283–8.

Vömel S, Freiling FC. A survey of main memory acquisition and analysis techniques for the windows operating system. Digital Investig 2011;8(1):3–22.

Vömel S, Freiling FC. Correctness, atomicity, and integrity: defining criteria for forensically-sound memory acquisition. Digital Investig 2012;9(2):125–37.

Wong K, Lai ACT, Yeung JCK, Lee WL, Chan PH. Facebook forensics. https://www.fbiic.gov/public/2011/jul/facebook_forensics-finalized.pdf; 2011. Valkyrie-X Security Research Group, Accessed: 2018-04-29.

Yang TY, Dehghantanha A, Choo KKR, Muda Z. Windows instant Messaging App forensics: Facebook and Skype as case studies. PloS ONE 2016;11(3):e0150300.

Diogo Barradas is a Ph.D. student of Information Systems and Computer Engineering at Instituto Superior Técnico, Universidade de Lisboa. He received his BSc. (2014) and M.Sc. (2016) from the same institution. His research interests include network security and privacy, with particular emphasis on statistical traffic analysis and Internet censorship circumvention. He conducts his research at the Distributed Systems Group at INESC-ID Lisboa.

Tiago Brito is a Ph.D. student of Information Systems and Computer Engineering at Instituto Superior Técnico (IST) and a researcher of the Distributed Systems Group at INESC-ID Lisboa. His focus has been on cyber-security and privacy, which correspond to his biggest interests in this field of study. He is also interested in mobile computing, distributed systems and computer forensics. Additionally, he is a co-founding member of Security Team at Téecnico (STT), a security oriented team for solving cyber-security challenges at Capture The Flag (CTF) events.

David Duarte is a M.Sc. student of Engineering Systems and Computer Engineering at Instituto Superior Técnico, Universidade de Lisboa, where he also received is BSc. (2013) in Engineering Systems and Computer Engineering. He currently works at the Infrastructure Team at the IT department of Instituto Superior Técnico. His main interests covers designing and implementing reactive web applications, with security as a first class citizen and designing new infrastructures for organizations based on code.

Nuno Santos is an Assistant Professor of the Computer and Information Systems Department at Instituto Superior Técnico, University of Lisbon, and a research member of the Distributed System Group at INESC-ID Lisbon. His research interests span the areas of security and trusted computing. He finished his Ph.D. in 2013 at Max-Planck Institute for Software Systems / Saarland University, Germany. During his PhD, he built several systems aimed at improving trust in cloud, enterprise, and mobile platforms. He developed novel policy-based trusted execution runtimes that take advantage of trusted computing hardware, namely Trusted Platform Module (TPM) in the context of cloud infrastructures, and ARM TrustZone in the context of mobile devices. His work has resulted in the publication of multiple peer-reviewed articles in journals and conferences, such as ASPLOS, USENIX Security, PETS, and Middleware. He was the recipient of a best paper award in Middleware'07.

Luís Rodrigues graduated (1986), has a Master (1991) and a Ph.D. (1996) in Electrical and Computer Engineering, by the Instituto Superior Técnico (IST), Universidade de Lisboa. He is a Professor at Departamento de Engenharia Informática, Instituto Superior Téecnico, Universidade de Lisboa. From 1996 to July 2007 he served at the Departmento de Informática, Faculdade de Ciências (Faculty of Sciences), Universidade de Lisboa. He initiated his academic career at the Electrotechnic and Computers Engineering Department of Instituto Superior Téecnico de Lisboa (IST) in 1989. From 1986 to 1996 he was a member of the Distributed Systems and Industrial Automation Group at INESC. From 1997–2007, he was a (founding) member of the LASIGE laboratory at University of Lisbon, first as a member of the Navigators group and later as the leader of the Distributed Algorithms and Network Protocols group. He served as Director of the LASIGE in 2004–2005 and he served in the board of directors of INESC-ID Lisboa from 2010 to 2017. From July 2007 he is a member of the Distributed Systems Group at INESC-ID Lisboa. His current interests include faulttolerant distributed systems, concurrency, replicated data management, cloud computing, dynamic networks, information dissemination, and autonomic computing. He has more than 200 publications in these areas. He is co-author of two books on distributed computing.

Please cite this article as: Diogo Barradas et al., Forensic analysis of communication records of messaging applications from physical memory, Computers & Security (2018), https://doi.org/10.1016/j.cose.2018.08.013